Hotlink Assignment

Dissertation
zur Erlangung des Doktorgrades
der Fakultät für Angewandte Wissenschaften
der Albert-Ludwigs-Universität
Freiburg im Breisgau

A thesis submitted for
the doctoral degree at the
Faculty for Applied Sciences of the
Albert-Ludwigs-Universität
Freiburg im Breisgau

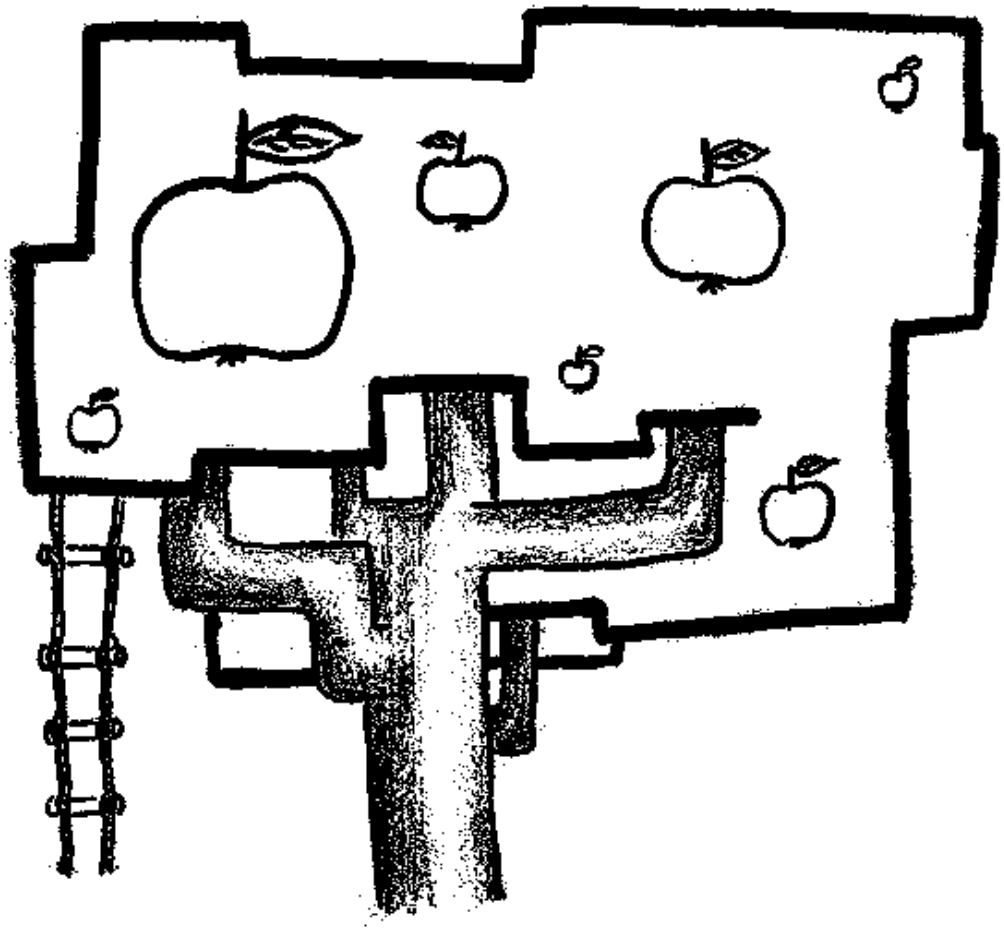
Tobias Jacobs

Albert-Ludwigs-Universität Freiburg im Breisgau
Fakultät für Angewandte Wissenschaften

Dekan: Prof. Dr. Hans Zappe

Referentin: Prof. Dr. Susanne Albers
Koreferent: Prof. Dr. Klaus Jansen

Tag der Promotion: 8. Mai 2009



Zusammenfassung

Die vorliegende Dissertation behandelt kombinatorische Optimierungsprobleme, die mit dem Erweitern von Websites durch zusätzliche Hyperlinks zusammenhängen.

Das Einfügen ggf. speziell hervorgehobener zusätzlicher Links, so genannter *Hotlinks*, ist ein Verfahren zur Optimierung von Websites. Ein Vorteil dieses Verfahrens ist, dass es nicht destruktiv ist, da die ursprüngliche Struktur der Site erhalten bleibt. Hotlinks können - abhängig von den Interessen der Benutzer - so zugeordnet werden, dass auf beliebtere Seiten schneller zugegriffen werden kann. Auf diese Weise wird die erwartete Anzahl an ,,Klicks“ minimiert und der Datenverkehr gleichzeitig verringert.

Eine hierarchisch aufgebaute Website kann formal als *gewichteter Baum* $T = (V, E, \omega)$ beschrieben werden, wobei (V, E) ein Baum mit Wurzel $r \in V$ ist. Die *Gewichtsfunktion* $\omega : V \to \mathbb{R}_0^+$ ordnet jedem Knoten eine Zugriffswahrscheinlichkeit zu. Ein *Hotlink Assignment* ist eine Menge $A \subset V \times V$ von zusätzlichen Kanten, welche für die Benutzer Abkürzungen darstellen. Aus Gründen der Übersichtlichkeit ist nur eine gewisse Anzahl K ausgehender Hotlinks pro Seite erlaubt. Ein Hotlink Assignment ist für einen gegebenen gewichteten Baum optimal, wenn es unter allen zulässigen Assignments die erwartete Länge des Pfades von der Wurzel r zu einem Knoten im Baum minimiert. Hierbei betrachten wir nicht den kürzesten Pfad, sondern gehen davon aus, dass jeder Hotlink auf dem Weg von der Wurzel zum Zielknoten durch die Benutzerin oder den Benutzer unmittelbar benutzt wird.

Nach einer ausführlichen Einführung in die Problemstellung befassen wir uns zunächst mit der Berechnungskomplexität der optimalen Lösung. Hierbei zeigen wir, dass es NP-vollständig ist, zu entscheiden, ob für einen gegebenen Baum ein Hotlink Assignment existiert, welches eine bestimmte erwartete Pfadlänge erreicht. Dies gilt selbst für den Fall, dass maximal ein Hotlink von jedem Knoten ausgehen darf und dass nur die Blätter positive Zugriffswahrscheinlichkeit haben. Das Ergebnis wurde in [Jac08b] veröffentlicht.

Im darauffolgenden Kapitel identifizieren wir eine praxisrelevante Einschränkung des Lösungsraumes, unter deren Berücksichtigung das Hotlink-Assignment-Problem in Polynomialzeit gelöst werden kann. Es handelt sich hierbei um die Anforderung, dass Hotlinks nur auf die Blätter des Baumes weisen dürfen. Dieses Ergebnis wurde ebenfalls in [Jac08b] veröffentlicht.

Im weiteren Verlauf der Dissertation geben wir Algorithmen an, die in Polynomialzeit Näherungslösungen mit konstanten Approximationsfaktoren berechnen. Der Greedy-Algorithmus fügt immer den Hotlink in den Baum ein, welcher momentan die größte Verbesserung erzielt. Wir zeigen, dass mit diesem Verfahren mindestens die Hälfte der maximal möglichen Verbesserung

erreicht wird. Des Weiteren geben wir ein Approximationsschema (PTAS) an, welches Lösungen mit beliebigem Näherungsgrad in Polynomialzeit berechnet, wobei der Grad des Polynoms vom gewünschten Näherungsgrad abhängt. Letzterer bezieht sich auch hier auf die maximal mögliche Verbesserung. Eine alternative Zielsetzung ist es, die erwartete Pfadlänge möglichst gut zu approximieren. Hierfür präsentieren wir einen Polynomialzeitalgorithmus mit Approximationsfaktor 2. Diese drei Näherungsalgorithmen und deren Analyse wurden in [Jac07] veröffentlicht.

Im letzten Kapitel der Dissertation präsentieren wir die Ergebnisse einer ausführlichen experimentellen Studie. Neben den oben erwähnten Algorithmen werden hier auch die in [DL05] und [DL06] vorgeschlagenen Strategien evaluiert. Des weiteren geben wir eine neuartige heuristische Methode zum Einfügen von Hotlinks an, die in der Praxis exzellente Ergebnisse erzielt. Die Experimente basieren auf zwei Testdatensätzen. Der erste Datensatz besteht aus Baum-Instanzen, welche die Struktur der Websites großer Universitäten repräsentieren. Der zweite Datensatz beinhaltet synthetische Instanzen, die von einem probabilistisch arbeitenden Algorithmus erzeugt wurden. Letzterer wurde bei dieser Studie erstmals eingesetzt. Die Experimente zeigen, dass die heuristische Methode und der Greedy-Algorithmus in der Praxis die besten Lösungen berechnen und dass bei Einschränkung des Lösungsraumes auf Blätter durchaus vergleichbar gute Ergebnisse erzielt werden. Auf der anderen Seite schneiden Algorithmen, die auf Approximationsfaktoren bezüglich der erwarteten Pfadlänge abzielen, in den Experimenten deutlich schlechter ab. Die Studie wurde in [Jac08a] veröffentlicht.

Abstract

This thesis treats combinatorial optimization problems that arise from the task to enhance web sites with additional hyperlinks.

The goal of inserting additional hyperlinks called *hotlinks* is to optimize web sites. An advantage of this approach is that it is non-destructive, i.e. the original site structure is preserved. Hotlinks are typically assigned according to the interests of the users, such that popular pages can be accessed especially fast. This both minimizes user interaction and reduces web traffic.

A hierarchically structured web site can be formalized as a *weighted tree* $T = (V, E, \omega)$, where (V, E) is a tree rooted at $r \in V$. The *weight function* $\omega : V \to \mathbb{R}_0^+$ assigns an access probability to each node. A *hotlink assignment* is a set $A \subset V \times V$ of additional shortcut edges. For reasons of clearness, only a certain maximum number K of hotlinks is allowed to leave each node. A hotlink assignment is optimal for T if it minimizes the expected length of the path from r to some other node. Instead of considering the shortest path, we assume that the user immediately takes any hotlink on the path to her or his destination node.

After a detailed introduction to the problem, we first address the computational complexity of the optimal solution. We show that it is NP-complete to decide whether there exists a hotlink assignment for a given weighted tree achieving a given expected path length. This even holds true in the case where only one hotlink is allowed to leave each node and only the leaves can have a positive access probability. The result has been published in [Jac08b].

Subsequently, we identify a restriction of the solution space that is relevant in practice and allows for a polynomial time optimal algorithm. Namely, we study the model where hotlinks may only point to the leaves of the tree. This result has also been published in [Jac08b].

Returning our attention to the original hotlink assignment problem, we give a number of polynomial time algorithms that compute solutions guaranteeing constant approximation factors. Greedy always inserts a hotlink currently achieving the greatest improvement or *gain*. We prove that this algorithm achieves at least one half of the optimal solution's total gain. Furthermore, we give an approximation scheme (PTAS) which computes solutions with an arbitrary approximation ratio in polynomial time, where the degree of the polynom depends on the desired ratio. Here the ratio also corresponds to the gain. An alternative approach is to approximate the expected path length as good as possible. To this end, we present a polynomial time 2-approximation. These algorithms and their analyses have been published in [Jac07].

In the last chapter of the thesis, we present the results of an extensive

experimental study. Besides the abovementioned algorithms, the assignment methods proposed in [DL05] and [DL06] are also included in the study. Moreover, we propose a new heuristic that achieves excellent results in practice. Our experiments are based on two data sets. One set contains tree instances representing the structure of large university web sites. The other data set consists of synthetic instances generated by a new random construction method. The experiments show that in practice our heuristic method and GREEDY achieve the best results, and, in terms of solution quality, assignments which are optimal under the restriction that hotlinks only point to leaves are comparable to the best assignments not meeting that restriction. On the other hand, algorithms tailored to approximate the expected path length perform considerably worse in the experiments. The study has been published in [Jac08a].

Contents

Chapter 1

Introduction

This thesis is on algorithms for hotlink assignment. Hotlink assignment denotes a special kind of graph augmentation, where additional edges are inserted into a graph in order to decrease the length of certain paths. The eponymous and most typical application are web graphs, with nodes representing web pages and edges representing hyperlinks. Here a hotlink is a specially highlighted shortcut link. Hotlinks aim to reduce the amount of interaction required for accessing popular pages. This does not only increase user-friendliness, but also reduces web traffic and server load.

The focus of this thesis is on trees. Regarding web graphs, we believe that this is no severe restriction. Users typically enter a web site via the *home page*. This special page can be considered as the root of the tree. If the site is well structured, the path users take to reach their destination page will be the shortest possible one. Thus, from our point of view it suffices to consider the shortest path tree rooted at the home page.

1.1 Minimizing Human Interaction

The value of information is always closely related to its accessibility. Due to the extensive growth of the Internet as a huge information source, the task of organizing the content of web sites in an effective way is becoming increasingly important. Clearly, the worst case scenario of users looking for a specific piece of information like a needle in a haystack is to be avoided.

To our knowledge, there are two main approaches for making contents accessible in a convenient way. One is the usage of search engines, and the other is to organize the information in a hierarchical structure. Modern web sites typically offer both methods.

1.1.1 Search Engines

On search engines, user interaction is basically divided into two steps. In the first step, the user enters one ore more keywords. She or he is then presented a list of pages that she or he is possibly looking for. In the second step of interaction, the user chooses one of these pages. An overview of a specific area can be obtained by performing only the first step.

The main advantage of this approach is that, if the right keywords are known to the user, the search effort is reduced to the two steps just described. State-of-the-art implementation of search engines allow answer times that are negligible. Ranking algorithms (cf. e.g. [BP98]) are employed to order the resulting page list by decreasing relevance.

One drawback is that the user has to know the right keyword. This may become a problem when there are several synonyms of one expression. It is also possible that typing errors occur in the query, or in the page the user is looking for. There are however approaches to the development of search engines that can handle those kinds of problems.

Another drawback is that the page lists generated by search engines are unstructured, making it hard to obtain a clear overview of the available contents. For example, if a user is interested in car stereos, entering the keyword "car stereo" to the search engine of an online store will result in a long list of radios, CD-players, speakers, and other accessories, ordered by sales rank.

As a conclusion one can say that search engines certainly are a highly useful tool for information retrieval, but they can never be a substitute for web indexes.

1.1.2 Hierarchical Indexes

Large web sites typically make their contents available via a hierarchical index. In order to be helpful, the index has to fulfill certain requirements.

First, its hierarchy has to be such that users always know which hyperlink will lead them towards their destination. This requirement is only met if the structure represents some semantic hierarchy of the content. For example, a user looking for an overview of a specific type of car stereo at an online store might subsequently click hyperlinks labeled "HiFi", "Car HiFi", "CD Players", and "Models with mp3 support".

Second, the number of hyperlinks on any index page must be somehow limited, as the cognitive effort for choosing an item from a list rapidly increases with the list size.

These requirements imply that there are not many degrees of freedom

for the design of a hierarchical index. In [PE00], Perkowitz and Etzioni propose an algorithm called *PageGather* for automatically generating index pages based on correlating user access patterns. However, they point out that new index pages should always be approved by the webmaster before being inserted.

1.1.3 Access Frequencies

The interests of the users of a web site are usually highly correlated. They typically follow a Zipf distribution, where about 80% of all requests correspond to about 20% of the contents [Pit99]. Therefore, it is advisable to take access frequencies into account when designing web indexes. Placing popular pages close to the home page will pay off in terms of user interaction, web traffic, and web server load. Access patterns can easily be read off from the web server log files.

One difficulty is that the requirements outlined in Section 1.1.2 still have to be met. Moreover, the interests of users are likely to change rapidly over time, e.g. flash crowds may occur. Therefore, some adaptivity of the web structure is desirable. Redesigning the index from scratch is certainly not an option as this would be more confusing than helpful for the users.

In this thesis we address a non-destructive approach for improving the structure of web sites. Via a minor enhancement that can be inserted and updated automatically, the expected amount of user interaction can be significantly reduced.

1.1.4 Hotlinks

A *hotlink* is an additional hyperlink on a web page, i.e. hotlinks do not belong to the original site structure. In practice, it is advisable that the hotlinks are somehow specially highlighted or appear at a special position on the pages. The maximum number of hotlinks on a web page can be fixed to some constant, or it can be specified by the web designer for each page individually. A set of hotlinks for the web site meeting these restrictions can then be assigned automatically. Such a hotlink set is called a *hotlink assignment (HLA)*. The assignment can be updated regularly in case of changing access patterns.

It is straightforward to measure the quality of a hotlink assignment as the expected number of "clicks" that are necessary to reach a web page. The expectation is calculated on the basis of the access frequencies. The task of finding a hotlink assignment for a given web site that is optimal with

respect to this measure is called the *Hotlink Assignment Problem*. This thesis addresses algorithmic aspects of the problem.

1.2 Formal Model

In this section we give a formal definition of the Hotlink Assignment Problem. Like mentioned in the beginning of this chapter, a web site can be modeled as a directed graph $G = (V, E)$, where nodes represent pages and edges represent hyperlinks. The home page of the web site is some special node $r \in V$ that will be called the *root*.

Clicking a hyperlink corresponds to traversing an edge from one node to another. It is assumed that every user starts from the root and traverses a path towards some destination node. A hotlink assignment $A \subseteq V \times V$ is a set of additional shortcut edges.

1.2.1 User Behavior

Before going into details about the quality measure for hotlink assignments, we have to be clear about how users behave in a graph that is enhanced with hotlinks. There are two models of user behavior that have been considered in literature.

The first model assumes that users always take the shortest path in the enhanced graph $(V, E \cup A)$. At first glance this might seem natural, as users try to avoid any unnecessary effort. However, such a behavior requires that users know the entire graph $(V, E \cup A)$. Such an assumption is unrealistic at least for A because the hotlinks do not belong to the fixed graph structure. This is why that model has later been named the *clairvoyant user model* [GKMP03].

In this work we consider the more realistic *greedy user model*. Users follow the shortest path in G, and immediately use any hotlink that takes them closer to their destination node. The example in Figure 1.1 shows that this can lead to suboptimal behavior.

1.2.2 Weighted Trees

As already mentioned in the beginning of the chapter, we restrict our attention to rooted trees in this thesis. A rooted tree $T = (V, E)$ is a directed graph that contains a unique node $r \in V$ such there is exactly one path from r to each node $v \in V$. This special node r is called the *root* of T.

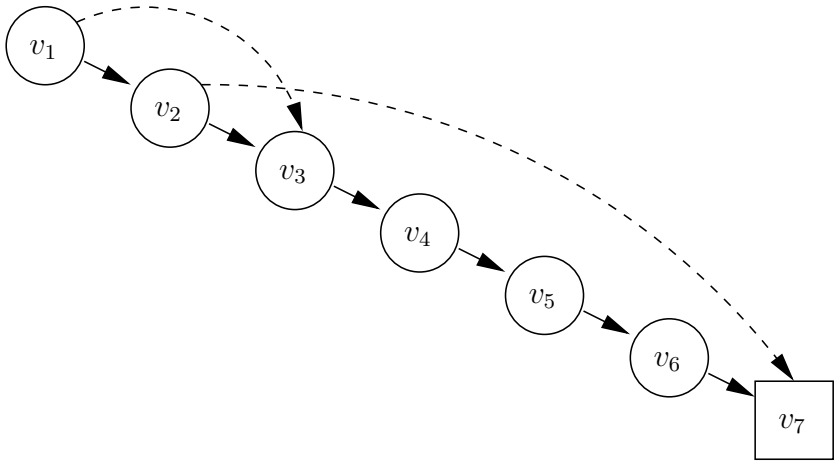


Figure 1.1: A clairvoyant user would traverse the path v_1, v_2, v_7, while a greedy user traverses $v_1, v_3, v_4, v_5, v_6, v_7$. In the greedy user model, the hotlink (v_2, v_7) is obsolete.

We use the following terminology: For $u, v \in V$ with $u \neq v$, node v is a *descendant* of u if the path from r to v contains u. We say that u is an *ancestor* of v if v is a descendant of u, and v is a *child* of u if $(u, v) \in E$. Nodes having no children are called *leaves*, while nodes having children are *internal nodes*.

A *weighted tree* is a triple (V, E, ω), where (V, E) is a rooted tree and $\omega : V \to \mathbb{R}_0^+$ is a function assigning a non-negative weight to each node. As unweighted or unrooted trees are not considered in this thesis, we do not always explicitly mention that a tree is rooted and weighted.

The weights can be interpreted as access frequencies or access probabilities. In the latter case, of course, they have to sum up to 1. However, as we will see in Subsection 1.2.4, the objective functions are linear in the weights. This means that the weights can be scaled arbitrarily, so from an algorithmic point of view it makes no difference whether or not they constitute a probability distribution.

In literature about hotlink assignment it is often assumed that only leaves have non-zero weights, i.e. there is a strict separation between navigation pages and content pages. We will adopt that restriction only in Chapter 3 and 5.

1.2.3 Feasibility

A straightforward consequence of the greedy user assumption is that, for any hotlink (u, v) in a reasonable assignment, v is a descendant of u. Furthermore,

there are no *crossing* hotlinks, i.e. there is no hotlink from an ancestor of u to a node on the path between u and v. Otherwise, the hotlink (u, v) would never be taken by a greedy user (cf. Figure 1.1).

Throughout this thesis we will consider only hotlink assignments that do not contain such unreasonable hotlinks. Due to the greedy user assumption, this is no restriction. On the other hand, whenever a greedy user takes a hotlink (u, v), we know that there is no hotlink from one of the nodes bypassed by the hotlink to a descendant of v. This means that (u, v) also would have been taken by a clairvoyant user. So the greedy user's path is always the shortest one and, consequently, the explicit greedy user assumption is not needed any more.

Definition 1.1 *A hotlink assignment A for a weighted tree T is called feasible, if, for any hotlink $(u, v) \in A$, u is an ancestor of v and there is no hotlink in A from an ancestor of u to v or a node bypassed by (u, v).*

Clearly, if there was no restriction concerning the number of hotlinks that are allowed to leave a node, then any reasonable hotlink would start in the root. This is however not desirable, as the number of hyperlinks on a concise web page must be somehow limited.

In the most common formulation of the Hotlink Assignment Problem, only one hotlink is allowed to start in each node. A natural generalization is to say that a K-hotlink assignment is a feasible assignment where each node is the source of up to K hotlinks. We will also consider the model where the number of outgoing hotlinks is specified individually for each node and hotlinks are only allowed to end in leaves.

1.2.4 Performance Measures

The *weighted path length* or simply *path length* of a hotlink assignment A for a tree $T = (V, E, \omega)$ rooted at r is defined as

$$p(A, T) = \sum_{v \in V} \omega(v) \mathrm{dist}^A(r, v) \ ,$$

where $\mathrm{dist}^A(u, v)$ is the length of the shorted path between u and v in $(V, E \cup A)$. In the preceding subsection we have seen that, if the assignment is feasible, this is equal to the number of edges and hotlinks a greedy user has to traverse on her or his way from u to v.

The *K-Hotlink Assignment Problem* denotes the task of finding an optimal K-hotlink assignment for a given weighted tree. An assignment A is

optimal for T if it minimizes the path length among all K-hotlink assignments for T. This is equivalent to saying the assignment maximizes the *gain*

$$g(A,T) = p(\emptyset,T) - p(A,T) \ ,$$

which is the improvement achieved by A. Clearly, a hotlink assignment achieves a minimum path length if and only if it maximizes the gain. Despite equivalence with respect to optimal solutions, the problem formulations are not equivalent when we are interested in *approximation ratios*.

Let Alg be a hotlink assignment algorithm, and let $\textsc{Alg}(T)$ be the assignment computed by Alg for the tree T. Let Opt be an optimal hotlink assignment algorithm. Following the common definition of approximation algorithms, we say that Alg is a *c-approximation in terms of the path length*, if

$$\frac{p(\textsc{Alg}(T),T)}{p(\textsc{Opt}(T),T)} \leq c \text{ for any tree } T \ .$$

Respectively, we say that Alg is a *c-approximation in terms of the gain*, if

$$\frac{g(\textsc{Opt}(T),T)}{g(\textsc{Alg}(T),T)} \leq c \text{ for any tree } T \ .$$

Note that both the path length and the gain are linear in the node weights. This implies that the approximation ratios are invariant to weight scaling, which is why there is no need to specially consider the case when the weights constitute a probability distribution.

1.3 Applications

The eponymous application of hotlink assignment is the enhancement of web sites. If the weights of the nodes are access probabilities, then the path length of a hotlink assignment is the expectation of the number of hyperlinks a user has to click in order to reach her or his destination page.

The straightforward approach for estimating such a probability distribution is to take past access frequencies into account. The distribution could be updated regularly in order to actualize the hotlink assignment respectively. Due to the linearity of the objective function, the hotlink assignment algorithm can work directly on the frequencies, i.e. they do not have to be scaled.

Hotlink assignments can be *global* or *personal*. In the global case, one assignment is provided for all users of a web site. Such an assignment should be based on a global estimate of access probabilities. If the web server has

the possibility to recognize its individual users (e.g. by web cookies or a login interface), it is also imaginable that each user sees a personal hotlink assignment based on her or his interests.

Past access patterns are not the only possible way to obtain weights of the nodes. It is also thinkable to exploit domain specific knowledge. For example, when a new product comes into the market, it is likely that many users will be interested in it. Or, in the case of personal hotlink assignments, the interests of individual users can be estimated based on similarities as regards content. For example, a user having read a number of poems by a certain author will probably be interested in further literature by the same author. For global hotlink assignment, online stores would supposably consider sales ranks rather than access frequencies.

The concept of hotlink assignment can be applied in a number of additional scenarios. In principle, any hierarchical structure where access probabilities can be estimated and where restructuring the whole hierarchy is not possible could be enhanced with hotlinks. Examples are knowledge bases, file system browsers and large menus of computer applications.

A further application has been discovered by Bose, Krizanc, Langerman, and Morin [KLM02]. Asymmetric communication is characterized by a server and a client, where the bandwidth with which the server can send packets to the client is much larger than vice versa. The objective is to speed up the data transmission from the client to the server by exploiting the bandwidth of the other direction. It is assumed that the communication is based on an alphabet Σ that is encoded in binary, and the server has access to a probability distribution of Σ. It computes a hotlink assignment for the binary tree representing the encoding of Σ. During the transmission of a letter, client and server keep track of the prefix that has already been sent. In each round, the server presents all edges and hotlinks that leave the tree node associated with the current prefix, and the client returns the index of the edge or hotlink that leads to the maximal prefix of the letter it wants to transmit. Unlike approaches using Huffman encoding, only the server needs to have access to the probability distribution and the hotlink assignment, and the assignment can be updated whenever the probability distribution changes. Although this is not explicitly mentioned by Bose et. al., the tree traversal carried out during the transmission of a letter corresponds to the behavior of a greedy user.

1.4 Related Work

To our knowledge, the concept of assigning hotlinks has first been proposed by Perkowitz and Etzioni in [PE97]. The first paper describing algorithms for hotlink assignment has been published in 2000 by Bose et. al. [BKK+00]. Since then, a considerable amount of research has been spent on the related combinatorial optimization problems.

In the following. when considering a hotlink (u, v), we say that u is the *hotparent* of v, and v is a *hotchild* of u.

1.4.1 Bookmark Assignment

A *Bookmark assignment* is a set of k hotlinks starting in the source node of a web graph. In [CKPM02], Czyzowicz et. al. show that calculating an optimal bookmark assignment for a directed acyclic graph is NP-hard.

Regarding trees, they also discover that the problem is equivalent to the placement of at most k proxies in tree networks with a single server. The latter problem is described as a server and a number of clients that are associated with nodes in a tree. The server node can be considered as the tree root. Instead of accessing the server, clients can alternatively connect to a proxy node. The total number of edges on the paths between clients and server (or the next proxy on the path to the server) is to be minimized.

Polynomial time algorithms for solving the proxy placement problem for trees have been proposed in [LDGS98, KRS00, JLH+00, LS04]. These algorithms work in the presence of node weights and/or edge weights. In [CKPM02], an $O(k)$ time algorithm is given for the special case of complete binary trees with node weights and $k \leq \sqrt{n+1}$, where n is the tree size.

In this thesis, bookmark assignment algorithms will be an ingredient of a greedy algorithm for the K-Hotlink Assignment Problem.

1.4.2 Entropy

In [BKK+00], Bose et. al. point out a close connection between weighted trees and the encoding of an alphabet (cf. Section 1.3). For the moment, let us assume that only the leaves of a tree T carry weights, and that these weights constitute a probability distribution[1] $\mathbf{p} = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. Let further Δ be the *degree* of the tree, i.e. the maximum number of children of any tree node.

We can associate every edge of T with a letter from a Δ-ary alphabet, such that any two edges leaving the same tree node carry unequal letters.

[1] In order to avoid confusion with the path length, we use the bold $\mathbf{p}$ for probabilities.

Then, T can be interpreted as a Δ-ary encoding of its leaves: Each leaf l is encoded with the letters associated with the edges on the path from the root r to l.

As only the leaves of T have weights, the code is prefix-free, i.e. no code word is the prefix of another code word. The weighted path length of the tree, $p(\emptyset, T)$, corresponds to the expected word length of the code.

Shannon's theorem [Abr63] states that the expected word length of a prefix code is at least $\frac{H(p)}{\log \Delta}$, where

$$H(\mathbf{p}) = \sum_{i=1}^{m} \mathbf{p}_i \log \frac{1}{\mathbf{p}_i}$$

is the *entropy* of the probability distribution. Translated into our terms, this means that

$$p(\emptyset, T) \geq \frac{H(\mathbf{p})}{\log \Delta} \ .$$

It is easy to observe that the same holds for general directed graphs[2] as the weighted path length does not change if we extract a *shortest path tree* by deleting all edges that do not belong to a shortest path. As a consequence, any assignment A of at most K hotlinks to each node of a Δ-ary tree will result in a path length of at least

$$p(A, T) \geq \frac{H(\mathbf{p})}{\log(\Delta + K)} \ .$$

This *entropy bound* holds both in the clairvoyant and greedy user model. It can easily be generalized to the case of arbitrary weights. Let $\omega(T)$ be the sum of all leaf weights. We define the probability of the ith leaf l_i as $\mathbf{p}_i = \omega(l_i)/\omega(T)$. The entropy bound is then

$$p(A, T) \geq \omega(T) \frac{H(\mathbf{p})}{\log(\Delta + K)} \ .$$

In order to simplify notation, we omit the factor $\omega(T)$ in the rest of Section 1.4.

If we also allow internal nodes to carry non-zero weights, then the corresponding code is not prefix-free any more, so the entropy bound does not hold directly. However, by relocating the weight of each internal node to an additional leaf child of that node, we obtain a $(\Delta+1)$-ary tree, and the path to any weighted node is increased by at most 1. This corresponds to adding

[2] In directed graphs, we can consider those nodes as leaves that have no outgoing edges.

a termination symbol to a coding alphabet. As Shannon's Theorem holds for the modified tree, we can conclude that

$$p(A,T) \geq \frac{H(\mathbf{p})}{\log(\Delta + K + 1)} - 1$$

for the original tree T.

Note that the entropy bound would still hold if an optimal $(\Delta + K)$-ary tree was constructed from scratch (e.g. by Huffman encoding [Huf52]) instead of enhancing the existing tree with hotlinks. Thus, it might seem that this lower bound is rather untight. However, in the following subsections we will see that any tree can be enhanced with only one hotlink leaving each node such that a path length of $O(H(\mathbf{p}))$ is achieved.

1.4.3 NP-Hardness for DAGs

For the clairvoyant user model, Bose et. al. prove in [BKK+00] that the 1-Hotlink Assignment Problem is NP-hard for directed acyclic graphs, even if all leaves have the same weight. The proof is a reduction from 3-Set Cover [Kar72].

Gerstel et. al. claim in [GKL+07] that the proof can be augmented to prove also NP-hardness in the greedy user model. We are, however, convinced that this is not true. In our opinion, it is not even clear how the greedy user model can be extended to DAGs. Shortest paths in DAGs are not necessarily unique, so paths taken by greedy users are not determined by the graph structure.

This observation holds in particular with regards to the class of problem instances considered in the NP-hardness proof of Bose et. al. For each leaf in these graphs, users can possibly choose among several shortest paths, and they do not know which of them is shortened by a hotlink.

1.4.4 Clairvoyant User Model

In [BKK+00], besides giving the entropy bound and the NP-hardness proof, Bose et. al. also present a number of hotlink assignment methods for full binary trees with special probability distributions of the leaves. In [FKW01], Fuhrmann et. al. give algorithms for assigning K hotlinks to full Δ-ary trees.

To our knowledge, the first hotlink assignment algorithm for general trees has been proposed by Kranakis et. al. in [KKS04]. Their algorithm assigns one outgoing hotlink to each node, and achieves a maximum path length of $\frac{H(\mathbf{p})}{\log(\Delta+1)-(\Delta/(\Delta+1))\log\Delta} + \frac{\Delta+1}{\Delta}$. We note that this guarantee also holds in the

greedy user model because the algorithm only computes hotlink assignments that satisfy our definition of feasibility given in Section 1.2.3.

The first constant factor approximation for the K-Hotlink Assignment Problem has been presented by Matichin and Peleg in [MP03]. The authors show that a natural greedy strategy achieves the gain of an optimal hotlink assignment up to the constant factor of 2. The analysis holds for general graphs.

1.4.5 Greedy User Model

The greedy user model has first been explicitly considered by Gerstel et. al. [GKMP03] and Pessoa et. al. [PLdS04a] (called "obvious navigation assumption" in the latter work). These papers independently report on the same dynamic programming algorithm for the computation of optimal assignments of at most K hotchildren to each node. The runtime of their algorithm is exponential in the tree depth. Thus, for balanced trees, where the depth is logarithmic in the number of nodes, it runs in polynomial time.

The algorithm can be adapted such that is computes hotlink assignments that minimize the length of the *longest* path. That version runs in polynomial time even on arbitrary trees.

The following approximation algorithms have been proposed for the 1-Hotlink Assignment Problem.

Like for the clairvoyant user model, Matichin and Peleg also were the first authors who have presented a constant factor approximation for the greedy user model. The algorithm proposed in [MP07] guarantees to achieve at least one half of the optimal solution's gain. This is attained by calculating the best assignment of hotlinks that bypass exactly one node.

In [DL05], Douïeb and Langerman give a hotlink assignment algorithm that guarantees a maximum path length of $3H(\mathbf{p})$. Recall that the path length of any hotlink assignment is at least $\frac{H(\mathbf{p})}{\log \Delta}$, so the latter algorithm is a constant factor approximation for trees of constant degree Δ. It also is the only known hotlink assignment algorithm that runs in linear time. In addition, the authors propose an efficient data structure for maintaining hotlinks when nodes are added, deleted, or weights are modified.

In [DL06], Douïeb and Langerman improve upon their previous work by presenting a method that guarantees a path length of $1.141H(\mathbf{p}) + 1$. They propose a sophisticated implementation of the improved algorithm guaranteeing a worst case runtime of $O(n \log n)$, and it is shown by reduction from sorting that this is asymptotically optimal.

1.4.6 Experimental Results

A number of experimental papers related to hotlink assignment have appeared.

In an early work [CKK+01], Czyzowicz et. al. evaluate greedy-like hotlink assignment methods. Their experiments are based both on real and synthetic tree instances.

Kranakis et. al. [KKM02] have developed a software tool for assigning hotlinks to web sites that is empowered with one of the greedy algorithms. In [KKM03], the same authors evaluate a similar greedy approach for optimizing the data transfer of web servers.

The first comparative study has been conducted by Czyzowicz et. al. in [CKK+03]. The study shows empirically that the algorithm given in [KKS04] is outperformed by greedy-like methods. The result is confirmed in [PLdS04b] and [GKL+07].

In the latter papers, Gerstel et. al. also report on the performance of an implementation of their optimal hotlink assignment algorithm. It turns out that optimal solutions can be computed for trees having hundred thousands of nodes, as long as the depth is moderate.

1.5 Outline of the Thesis

In this thesis we contribute a number of new insights, both theoretical and practical, into the Hotlink Assignment Problem for trees. We exclusively consider the greedy user model, as we believe that it is much more realistic than the clairvoyant model.

It has been an open questions for several years if there is hope to develop an efficient and optimal hotlink assignment algorithm for arbitrary trees. In Chapter 2 we answer that question negatively. By a reduction from the well-known 3-Set Cover Problem we show that the Hotlink Assignment Problem is NP-hard, even if only one hotlink is allowed to start in each node and only leaves have non-zero weights. This means that optimal hotlink assignments for arbitrary trees cannot be computed in polynomial time unless P=NP.

In Chapter 3 we identify a restricted problem version that is computationally tractable. In our opinion, the restriction arises from practice. In many applications where hotlinks are actually used, they only point directly to leaves. Examples are the product recommendations of amazon.com and similar sites, suggestions for possible completions of typed-in prefixes in web browsers, or suggestions for frequently used functions in menus of computer applications. The probable reason for such a restriction is that users would

find it confusing to end up on another navigation page after following a hotlink.

We present a polynomial time optimal algorithm for that problem version. While the set of possible outputs of our algorithm is restricted, the set of possible inputs is rather general. The algorithm accepts not only a tree and a weight function as parts of the input, but also a function $K : V \to \mathbb{N}_0$ specifying the maximum number of outgoing hotlinks for each node individually.

We turn our attention back to the original Hotlink Assignment Problem in Chapter 4, where we present a number of algorithms that substantially improve upon the best approximation ratios previously known for both the path length and the gain.

The natural greedy algorithm Greedy, which always adds a hotlink achieving the greatest gain, has exhibited the best performance among the approximation algorithms studied experimentally in [CKK+01, CKK+03, GKL+07][3]. We show that Greedy is a 2-approximation in terms of the gain for the K-Hotlink Assignment Problem. Furthermore, we prove the existence of a polynomial time approximation scheme (PTAS) in terms of the gain for the K-Hotlink Assignment Problem. For the 1-Hotlink Assignment Problem, we present the first algorithm that yields a constant approximation factor in terms of the path length for trees of unbounded degree. The resulting ratio is 2. Our approximation algorithms all work in the model where also internal nodes can have non-zero weights.

The analyses of the approximation algorithms are based on three operations for the modification of hotlink assignments. We believe that these operations are of independent interest, as they provide general insights into the problem. For example, a basic fact exploited in the analysis of the greedy algorithm in [MP03] is that, in the clairvoyant user model, new hotlinks never increase the length of any shortest path. Surprisingly, this observation holds also true in the greedy user model, if some further adjustments are made to preserve the assignment's feasibility.

In Chapter 5 we report on an extensive experimental study of all recent hotlink assignment algorithms. We have implemented the methods proposed by Douïeb and Langerman [DL05, DL06], and all algorithms developed in Chapter 3 and 4. Moreover, we present an additional hotlink assignment method PMin that is easy to implement and turns out to perform excellently in practice. Our experiments are conducted with both real and synthetic tree instances. The synthetic trees have been generated by a new random construction method that produces more realistic instances than the method

[3] Greedy is called *recursive* in [CKK+01, CKK+03] and *greedyBFS* in [GKL+07].

used in previous experiments.

The main conclusion from our experimental results is that algorithms tailored to approximate the gain achieve the best results in practice. Another finding is that, in practice, it is not a severe restriction if hotlinks may only point to leaves.

1.6 Notation and Terms

This section summarizes the terms and notational concepts that are used throughout the chapters of this thesis. Some of them have already been utilized in the preceding sections of Chapter 1.

1.6.1 Hotlinks

When considering a hotlink (u, v), we use the following expressions: Node u is the *hotparent* of v, while v is a *hotchild* of u. The hotlink *starts* in its *source* u and *ends* in its *sink* v.

1.6.2 Abbreviations

For the sake of simplicity, we write $v \in T$ for a node v belonging to T. As the tree under consideration will most often be clear from the context, we then also write $p(A)$ and $g(A)$ instead of $p(A, T)$ and $g(A, T)$, and $\mathrm{dist}(u, v) = \mathrm{dist}^{\emptyset}(u, v)$ denotes the distance between nodes u and v in the original tree.

1.6.3 Trees and Subtrees

We recapitulate and formalize the terms introduced in Section 1.2.2. Let $u, v \in T, u \neq v$.

Node v is a *descendant* of u, if the path from r to v contains u. The set of all descendants of u is denoted $\mathrm{desc}(u)$. We say that u is an *ancestor* of v, if v is a descendant of u. The set of all ancestors of v is denoted $\mathrm{anc}(v)$. Finally, v is a *child* of u if there is an edge from u to v, and the set of u's children is denoted $\mathrm{ch}(u)$. The term *parent* is defined respectively, and the parent of v is $\mathrm{par}(v)$.

Throughout this thesis, subscripts of tree identifiers are reserved for denominating subtrees. We denote by T_v the maximal subtree of T rooted at v. For any set V' of nodes, let $T \setminus V'$ be the tree obtained from T by omitting all maximal subtrees rooted at a node in $V \cap V'$. Let further $T_{v,A} = T_v \setminus \{v' \in \mathrm{desc}(v) \mid \exists (u, v') \in A : u \in \mathrm{anc}(v)\}$ be the maximal

subtree rooted at v, where the maximal subtrees rooted at the hotchildren of v's ancestors are omitted. Finally, for any subtree T' of T, we define $A|T' = \{(u, v) \in A \mid u, v \in T'\}$.

Chapter 2

Complexity

In this chapter we address the computational complexity of the Hotlink Assignment Problem. The main result is that it is NP-hard to compute the optimal solution to a given problem instance, even if only one hotlink is allowed to start in each node and only the leaves carry non-zero weights. We explicitly show the NP-completeness of the following decision problem:

Definition 2.1 *Given a weighted tree T and a real number α, the Hotlink Assignment Decision Problem is to decide whether there exists a hotlink assignment A for T with $p(A) \leq \alpha$ and any node having at most one hotchild.*

This chapter is organized as follows: In Section 2.1 we specify the NP-hard problem *3-Set Cover (X3C)* we reduce from. In Section 2.2-2.6, the weighted tree instance corresponding to an instance of X3C is defined, and the main idea of the reduction is outlined. The formal proof of equivalence is presented in Section 2.7. Section 2.8 concludes the chapter.

2.1 3-Set Cover

Definition 2.2 (Exact Cover by 3-Sets, X3C) *Given some set C with $|C| = 3k$, $k \in \mathbb{N}$, and a collection D of 3 element subsets of C, the problem X3C is to decide whether there is a sub-collection $D' \subseteq D$, such that each element of C occurs in exactly one member of D'.*

Problem X3C is well-known to be NP-hard [Kar72]. We will use the following additional terminology: The elements of C are $c_1, \ldots, c_n$, so $n = 3k$ is the size of C. The size of D is m, and its members are denoted $D_1, \ldots, D_m$, with $D_i = \{d_i^1, d_i^2, d_i^3\}$ for $1 \leq i \leq m$.

We assume an order among the elements in the subsets, i.e. $x < y < z$ for any $D_i = \{d_i^1, d_i^2, d_i^3\}$ with $d_i^1 = c_x, d_i^2 = c_y, d_i^3 = c_z$. Furthermore, we assume that each element of C is contained in at least one member of D. These assumptions clearly preserve the hardness of the problem.

The version of X3C just described is the starting point of our reduction to the Hotlink Assignment Decision Problem. In the following sections we show how an instance (T, α) corresponding to an instance (C, D) of X3C can be constructed.

In this chapter, index i will always serve to specify some subset D_i or associated entity, while index j will always specify an element c_j of C or associated entity. Consequently, $1 \leq i \leq m$ and $1 \leq j \leq n$ will always hold.

2.2 Tree Structure

Throughout this chapter, T will be the tree *corresponding* to an X3C instance. We first give the structure of T. For each $D_i \in D$ we construct a subtree T^i of depth 1. The root of T^i is denoted as r_i and the eight leaves are $\pi_i, \bar{\pi}_i, \sigma_i^1, \bar{\sigma}_i^1, \sigma_i^2, \bar{\sigma}_i^2, \sigma_i^3$, and $\bar{\sigma}_i^3$. For $x \in \{1, 2, 3\}$, σ_i^x and $\bar{\sigma}_i^x$ will correspond to element $d_i^x \in D_i$.

Another part of T is the path P defined by the nodes $\{p_i \mid 1 \leq i \leq m\} \cup \{\bar{p}_i \mid 1 \leq i \leq m\}$ and the edges $\{(p_i, \bar{p}_i) \mid 1 \leq i \leq m\} \cup \{(\bar{p}_i, p_{i-1}) \mid 2 \leq i \leq m\}$. Additionally, for $1 \leq i \leq m$ there are nodes s_i, a_i, and edges $(\bar{p}_i, s_i)$ and (s_i, a_i).

The final important part of T is the path Q. The set of nodes in Q is $\{q_j \mid 1 \leq j \leq n\} \cup \{\bar{q}_j \mid 1 \leq j \leq n\} \cup \{t_j \mid 1 \leq j \leq n\} \cup \{b_j \mid 1 \leq j \leq n\}$. The edges are $\{(q_j, \bar{q}_j) \mid 1 \leq j \leq n\} \cup \{(\bar{q}_j, q_{j-1}) \mid 2 \leq j \leq n\} \cup \{(\bar{q}_j, t_j) \mid 1 \leq j \leq n\} \cup \{(t_j, b_j) \mid 1 \leq j \leq n\}$.

P and Q are connected via the edge $(\bar{p}_1, q_n)$. Between Q and the subtrees T^i, there are two extra nodes u_1 and u_2. Although the existence of those nodes is not necessary, they avoid the occurrence of a special case at one point of the analysis. The construction of T is completed by inserting the edges $(\bar{q}_1, u_2), (u_2, u_1)$ and (u_1, r_i) for $1 \leq i \leq m$. The corresponding tree is depicted in Figure 2.1.

2.3 Terminology

For $1 \leq i \leq m$, the set $\{p_i, \bar{p}_i\}$ is denoted by P_i. Respectively, $\{q_j, \bar{q}_j\} = Q_j$ for $1 \leq j \leq n$. Recall that there is a subtree T^i and a set P_i for each subset $D_i \in D$, and there is a set Q_j for each element $c_j \in C$. From a very abstract

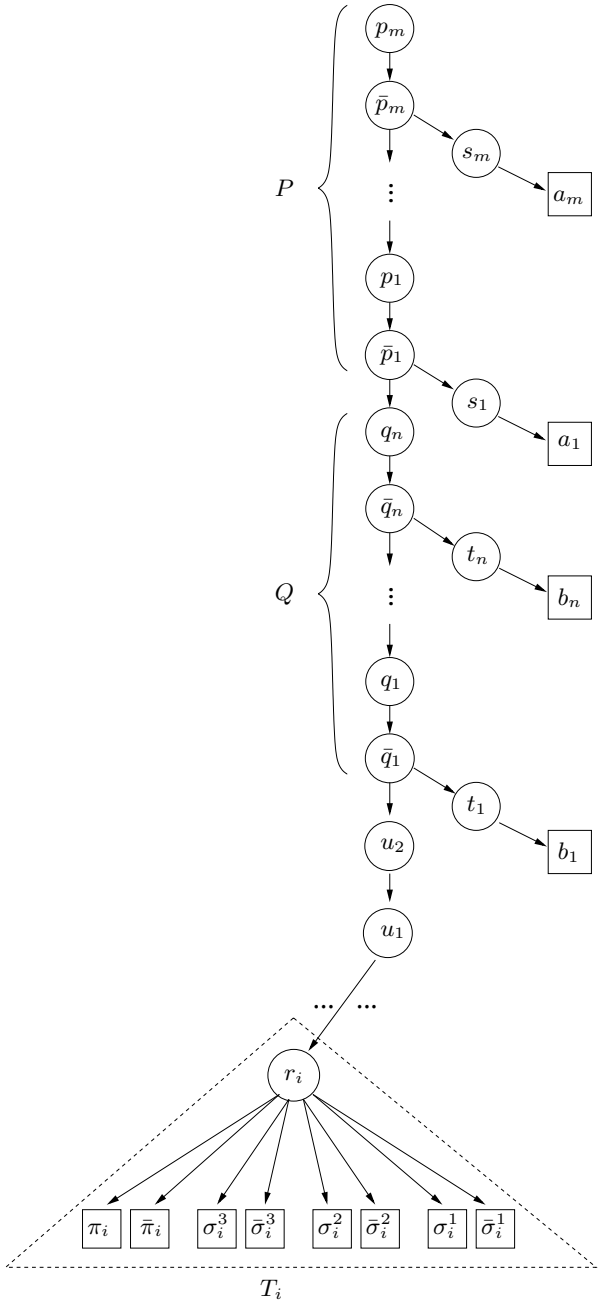


Figure 2.1: The tree corresponding to an instance of the 3-Set Cover Problem

point of view, c_j being covered by D_i will be translated into hotlinks from Q_j pointing into T^i.

We call the hotlink assignment $B = \{(p_i, r_i) \mid 1 \leq i \leq m\} \cup \{(\bar{p}_i, a_i) \mid 1 \leq i \leq m\} \cup \{(\bar{q}_j, b_j) \mid 1 \leq j \leq n\}$ the *basic assignment* for T. The *surplus* of a hotlink assignment A for T is defined as the improvement upon the basic assignment, i.e. $p(B) - p(A)$.

Instead of explicitly specifying a threshold path length α for the decision problem, we will define a certain surplus value $\beta = p(B) - \alpha$ and claim that a surplus of at least β can be achieved if and only if there is a solution to the instance of 3-Set Cover.

We now introduce two different concepts, which correspond to two different points of view from which hotlink assignments for T can be described. The concept of being *open* or *closed* refers to the configuration of the hotlinks that start in a P_i or Q_j. Conversely, the notion of *development* describes the hotlinks that end in a subtree T^i.

We start with the latter concept. Fix any T^i and let $d_i^1 = c_{j_1}, d_i^2 = c_{j_2}$ and $d_i^3 = c_{j_3}$ in the X3C instance.

We say that T^i is *undeveloped*, if there is a hotlink (p_i, r_i) (Figure 2.2a). Observe that in B each T^i is undeveloped.

From the status of being undeveloped, T^i is *developed to* j_3 by replacing (p_i, r_i) with (p_i, π_i), $(\bar{p}_i, \bar{\pi}_i)$, and (q_{j_3}, r_i) (Figure 2.2b).

For $x \in \{2, 3\}$, from the status of being developed to j_x, T^i is developed to j_{x-1} by replacing (q_{j_x}, r_i) with (q_{j_x}, σ_i^x), $(\bar{q}_{j_x}, \bar{\sigma}_i^x)$, and $(q_{j_{x-1}}, r_i)$.

From the status of being developed to j_1, T^i becomes *fully developed* by replacing (q_{j_1}, r_i) with (q_{j_1}, σ_i^1) and $(\bar{q}_{j_1}, \bar{\sigma}_i^1)$ (Figure 2.2c).

Observe that, if T^i is fully developed, then it contains all hotchildren of P_i, Q_{j_1}, Q_{j_2}, and Q_{j_3}. Therefore, there is a solution to an instance of X3C if and only if there is a hotlink assignment for the corresponding tree T where exactly k subtrees $T^{i_1}, \ldots, T^{i_k}$ are fully developed.

We proceed specifying the converse point of view. For a given hotlink assignment A, we say that P_i is *closed* if $(p_i, r_i), (\bar{p}_i, a_i) \in A$. This means in particular that T^i is undeveloped. In fact, each P_i is closed in the basic assignment B. Let $d_i^3 = c_j$. We say that P_i is *open subject to* j if $(p_i, \pi_i), (\bar{p}_i, \bar{\pi}_i) \in A$.

Respectively, we say that Q_j is *closed* if $(q_j, r_i), (\bar{q}_j, b_j) \in A$, where r_i is the root of some T^i with $d_i^x = c_j$ for some $x \in \{1, 2, 3\}$ in the X3C instance. We say that Q_j is *open*, if $(q_j, \sigma_i^x), (\bar{q}_j, \bar{\sigma}_i^x) \in A$ with $d_i^x = c_j$ in the X3C instance. In case of $x \in \{2, 3\}$ and $d_i^{x-1} = c_{j'}$, we say that Q_j is *open subject to* j'.

Note that, in general, knowing that a P_i or Q_j is closed or open does not necessarily mean that the corresponding subtree is correctly developed. P_i

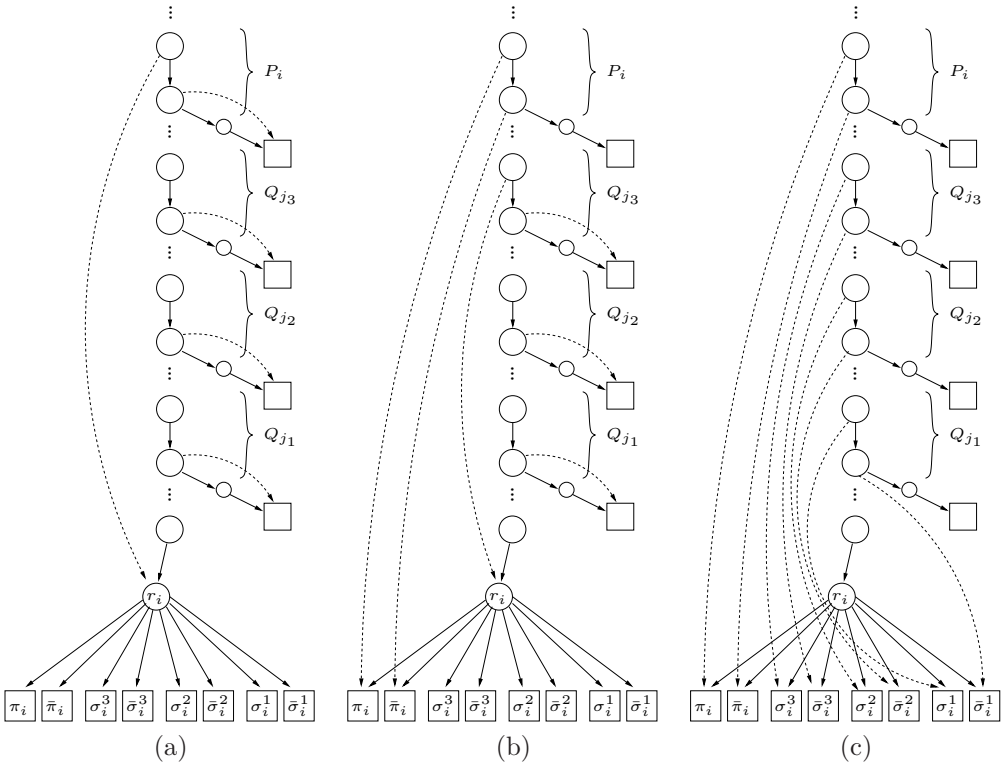


Figure 2.2: Subtree T^i representing the subset $\{c_{j_1}, c_{j_2}, c_{j_3}\} = D_i \in D$, as it is (a) undeveloped, (b) developed to j_3, and (c) fully developed. Accordingly, P_i is closed in (a), P_i is open subject to j_3 and Q_{j_3} is closed in (b) and P_i, Q_{j_3}, Q_{j_2}, and Q_{j_1} are open in (c).

or Q_j being closed or open only is a statement about a subset of the hotlinks pointing into that subtree.

2.4 Outline of the Proof

We define

$$\beta = 3\sum_{j=1}^{n} W_j + k \ , \tag{2.1}$$

where W_j will be defined in the following section. We claim that a surplus of β can be achieved if and only if there is a solution to the X3C instance. The proof of this claim has the following structure:

1. We define the weights such that, for $1 \leq j \leq n$, developing a subtree to j is rewarded with a surplus of $3W_j$. Furthermore, for the final step to the full development of a subtree, an extra surplus of 1 can be gained.

2. We prove that if an optimal hotlink assignment for T achieves a reward of at least β, then each P_i and each Q_j is either open or closed, and for each $j = 1, \ldots, n$ there is exactly one P_i or $Q_{j'}$ that is open subject to j.

3. We show that the preceding property implies that the transformation from B to an optimal assignment A^* can be described as a sequence of subtree developments. This makes it easy to reason about the surplus of A^*.

4. We finally prove that a surplus of β can be achieved if and only if there is a solution to the X3C instance. While the "if"-direction directly follows from the weight assignment, one can show that the converse direction is implied by the existence of the transformation procedure.

The second and third proposition will be proved simultaneously by induction over j. It is also shown that the weights of T can be represented by a number of bits polynomial in the size of the X3C instance, which completes the reduction from X3C.

2.5 Weight Assignment

As mentioned at the beginning of the chapter, internal nodes of T have no weights. We assign weights to the leaves of T in an iterative manner.

There will be $m + n$ different weight classes $W_1, \ldots, W_{n+m}$. Let $h = 2(n + m) + 3$ be the depth of T. We begin by setting $W_1 = h$. We assign $\omega(b_1) = 2W_1$ and $\omega(\sigma_i^x) = \omega(\bar{\sigma}_i^x) = 2W_1 + 1$ for each $d_i^x = c_1$ in the X3C instance.

For a fixed j, $2 \leq j \leq n$, assume that $W_{j'}$ and the weights of all $b_{j'}$ and all σ_i^x and $\bar{\sigma}_i^x$ with $d_i^x = c_{j'}$ are given for any $j' < j$. We denote the set of those leaves as L_{j-1}. We define

$$W_j = 2h \sum_{v \in L_{j-1}} \omega(v) \; + \; 1 \; . \tag{2.2}$$

As a consequence, W_j is more than twice the total weighted path length to all leaves lighter than W_j.

We assign $\omega(b_j) = 2W_j$ and, for any $d_i^x = c_j$, we assign $\omega(\bar{\sigma}_i^x) = 2W_j + 1$. Furthermore, $\omega(\sigma_i^x) = 2W_j + w_i^{x-1}$, where w_i^{x-1} is chosen as follows:

In case of $x \in \{2, 3\}$, let $d_i^{x-1} = c_{j'}$. We choose w_i^{x-1} such that the reward (i.e. decrease in path length) for deleting the hotlink $(\bar{q}_j, b_j)$ and then developing T^i to j' is $3W_{j'}$. When deleting $(\bar{q}_j, b_j)$, the length of the path to b_j increases by one. During the development, the path to σ_i^x is shortened by one, the length of the path to $\bar{\sigma}_i^x$ does not change, and the path length to all σ_i^y and $\bar{\sigma}_i^y$ with $y < x$ is increased by $\mathrm{dist}(q_j, q_{j'})$. Thus, the overall weighted path length decreases by

$$w_i^{x-1} - \mathrm{dist}(q_j, q_{j'}) \cdot \sum_{1 \leq y < x} \left(\omega(\sigma_i^y) + \omega(\bar{\sigma}_i^y) \right) \; .$$

So, in order to provide the desired reward, we choose

$$w_i^{x-1} = \mathrm{dist}(q_j, q_{j'}) \cdot \sum_{1 \leq y < x} \left(\omega(\sigma_i^y) + \omega(\bar{\sigma}_i^y) \right) \; + \; 3W_{j'} \; . \tag{2.3}$$

In case of $x = 1$, we choose $w_i^{x-1} = 1$ so that fully developing T^i achieves an extra reward of 1, which can be shown by a similar calculation.

As we have assumed that in 3-Set Cover the subsets are ordered, the values of $\omega(\sigma_i^y)$ and $\omega(\bar{\sigma}_i^y)$ have already been assigned for $y < x, d_i^x = c_j$, so our weight-assignment is well-defined.

It remains to specify the weights of $\pi_i, \bar{\pi}_i$ and a_i for $1 \leq i \leq m$, which is done with respect to similar objectives: For $i = 1, \ldots, m$, W_{n+i} must be greater than two times the weighted path length to all leaves lighter than W_{n+i}, and the reward for deleting $(\bar{p}_i, a_i)$ and then developing T^i to j' (with $d_i^3 = c_{j'}$ in the X3C instance) must be exactly $3W_{j'}$.

Let $L_n = \{\sigma_j^x, \bar{\sigma}_j^x, b_j \mid 1 \leq j \leq n, x \in \{1,2,3\}\}$. Assume that, for a fixed i, $W_{n+i'}$ and the weight of any $\pi_{i'}, \bar{\pi}_{i'}, a_{i'}$ for $1 \leq i' < i$ is already given. We assign

$$W_{n+i} = 2h \cdot \Big(\sum_{v \in L_n} \omega(v) + \sum_{i'=1}^{i-1} \big(\omega(\pi_{i'}) + \omega(\bar{\pi}_{i'}) + \omega(a_{i'})\big) \Big) + \ 1 \ , \qquad (2.4)$$

$\omega(a_i) = 2W_{n+i}$, $\omega(\bar{\pi}_i) = 2W_{n+i} + 1$ and $\omega(\pi_i) = 2W_{n+i} + w_i^3$. Let $d_i^3 = c_{j'}$. Then, due to the same argumentation as above, we choose

$$w_i^3 = \mathrm{dist}(p_i, q_{j'}) \cdot \sum_{1 \leq x \leq 3} \big(\omega(\sigma_i^x) + \omega(\bar{\sigma}_i^x)\big) \ + \ 3W_{j'} \ .$$

2.6 Problem Size

The number of nodes in the tree T corresponding to an instance of 3-Set Cover is $O(h)$. Therefore, $W_1 = h$ and $W_y = O(h \cdot h \cdot W_{y-1})$ for $2 \leq y \leq n+m$. Thus, $W_{m+n} = O(h^{O(h^2)})$, and we need at most $O(h) \log(O(h^{O(h^2)})) = O(h^3 \log h)$ bits to represent the weights of T, which is polynomial in the size of the X3C instance.

2.7 Proof of Equivalence

Lemma 2.1 *Let A^* be an optimal hotlink assignment for T. Then P_i is either closed or open in A^* for $i = 1, \ldots, m$.*

Proof. The claim is proved by induction over decreasing values of i, where the argumentation naturally includes the base case $i = m$. For any fixed i assume that the lemma has already been proven for $i' > i$. As a consequence, no path from the root of T to any leaf $\pi_{i'}$, $\bar{\pi}_{i'}$, or $a_{i'}$ with $i' > i$ contains p_i or any descendant of p_i in P and Q. We restrict our attention to the subtree T_{p_i,A^*}, which contains all leaves of T except for $\pi_{i'}$, $\bar{\pi}_{i'}$, and $a_{i'}$ with $i' > i$. As A^* is optimal, it is clear that $A^*|T_{p_i,A^*}$ is optimal for T_{p_i,A^*} in particular.

Observe that the only leaves in T_{p_i,A^*} that have a weight greater than W_{n+i} are a_i, π_i, and $\bar{\pi}_i$. Equation 2.4 ensures that the total weighted path length to all other leaves is smaller than $W_{n+i}/2$.

In any hotlink assignment where P_i is closed or open, the path length in

T_{p_i,A^*} is at most

$$\begin{aligned} & \max\{ \quad 2\omega(\pi_i) + 2\omega(\bar{\pi}_i) + 2\omega(a_i) + W_{n+i}/2 \ , \\ & \qquad\quad \omega(\pi_i) + 2\omega(\bar{\pi}_i) + 3\omega(a_i) + W_{n+i}/2 \quad \} \\ = \ & \max\{ \quad 12.5W_{n+i} + 2w_i^3 + 2, 12.5W_{n+i} + w_i^3 + 2\} \\ = \ & \qquad\quad 12.5W_{n+i} + 2w_i^3 + 2 < 13.5W_{n+i} \ . \end{aligned}$$

By systematically considering all possible configurations of the hotlinks starting in p_i and $\bar{p}_i$, we show that any but the open or closed configuration results in a suboptimal hotlink assignment. In particular, A^* is already suboptimal when $p(A^*|T_{p_i,A^*}) \geq 13.5W_{n+i}$.

First, we address the case where the hotchild of p_i is a node different from a_i, r_i, π_i, $\bar{\pi}_i$, or does not exist. One can easily show that then two of the three leaves a_i, π_i, and $\bar{\pi}_i$ have a distance of at least 3 from p_i. This means that the total path length to those leaves is at least 7 and thus $p(A^*|T_{p_i,A^*}) \geq 14W_{n+i}$.

If the hotchild of $\bar{p}_i$ is any other node than a_i, r_i, π_i, $\bar{\pi}_i$, or does not exist, then one can show by similar arguments that the total path length to a_i, π_i, and $\bar{\pi}_i$ is at least 7, which contradicts the optimality of A^* for the same reason.

It remains to consider the cases with the hotchildren of p_i and $\bar{p}_i$ being a_i, r_i, π_i, or $\bar{\pi}_i$. If $(p_i, r_i) \in A^*$, then we have just shown that $(\bar{p}_i, a_i) \in A^*$, i.e. P_i is closed.

If $(p_i, \pi_i) \in A^*$ and $(\bar{p}_i, a_i) \in A^*$, then the assignment can be improved by at least 1 if we replace $(\bar{p}_i, a_i)$ with $(\bar{p}_i, \bar{\pi}_i)$.

If $(p_i, \bar{\pi}_i) \in A^*$, then the assignment can be improved by interchanging the hotparents of π_i and $\bar{\pi}_i$, or, if π_i has no hotparent in A^*, by replacing $(p_i, \bar{\pi}_i)$ with (p_i, π_i).

If $(p_i, a_i) \in A^*$ then, as we know that A^* contains either $(\bar{p}_i, r_i)$, $(\bar{p}_i, \pi_i)$, or $(\bar{p}_i, \bar{\pi}_i)$, the assignment can be improved by interchanging the hotchildren of p_i and $\bar{p}_i$. □

Lemma 2.2 *Let A^* be an optimal hotlink assignment for T achieving a surplus of at least β. Then, for each $j = 1, \ldots, n$, Q_j is either closed or open and there is exactly one P_i or $Q_{j'}$ that is open subject to j.*

For proving this and another lemma we define a procedure with which the basic assignment B can be transformed into an optimal assignment A^*. Assume for a fixed j that Lemma 2.2 has already been proven for $j' > j$, i.e. for each $j' = j+1, \ldots, n$, Q_j is either closed or open and there is exactly one P_i or $Q_{j''}$ that is open subject to j'. The transformation proceeds as follows:

(1) For $j' = n, \ldots, j+1$, let T^i be the unique subtree containing hotchildren of the unique P_i or $Q_{j''}$ that is open subject to j' in A^*. Delete the hotlink $(\bar{p}_i, a_i)$ or $(\bar{q}_{j''}, b_{j''})$ and develop T_i to j'. This switches P_i or $Q_{j''}$ from closed to open, and $Q_{j'}$ becomes closed.

(2a) For $i = m. \ldots, 1$, if P_i is open subject to some j'' in A^* and has not been switched to open in Step (1), delete the hotlink $(\bar{p}_i, a_i)$ and develop the subtree T^i to j''. Observe that $j'' \leq j$.

(2b) For $j' = n, \ldots, j+1$, if $Q_{j'}$ is open in A^* and has not been switched to open in Step (1), let T^i be the subtree containing hotchildren of $Q_{j'}$ in A^*. Delete the hotlink $(\bar{q}_{j'}, b_{j'})$. If $Q_{j'}$ is open subject to some j'' in A^*, then develop T^i to j'' (it also holds here that $j'' \leq j$), otherwise fully develop T^i.

(3) Modify the outgoing hotlinks of q_j and its descendant nodes such that the resulting assignment is A^*.

Lemma 2.3 *The transformation sequence described above is well-defined and transforms B into A^*.*

Proof. For $j' = n, \ldots, j$, let $A^{j'}$ be the temporary hotlink assignment obtained after Step (1) has been iterated for index $j'+1$ and before it is iterated for index j'. In particular, A^n equals B and A^j is the assignment resulting from Step (1). We show the following invariants to hold:

I_1: For each $n \geq j'' > j' \geq j''' \geq 1$ where $Q_{j''}$ is open subject to j''' in A^*, it holds that $Q_{j''}$ is closed in $A^{j'}$ and some T^i is developed to j'' in $A^{j'}$.

I_2: For each $n \geq j'' > j'$, it holds that in $A^{j'}$ the hotchild of $q_{j''}$ belongs to the same subtree T^i as it does in A^*.

We use induction over decreasing values of j' to prove both the invariants and the soundness of Step (1). Clearly, the invariants hold for $A^n = B$.

Fix a $j' > j$ and assume that the invariants hold for $A^{j'}$. The precondition of the lemma states that there is a unique P_i or $Q_{j''}$ that is open subject to j'. In case of P_i, the fact that there is only one index P_i can be open subject to implies that P_i is still closed and T^i is undeveloped in $A^{j'}$, so the transformation step from $A^{j'}$ to $A^{j'-1}$ is applicable. In case of $Q_{j''}$, the applicability follows from I_1.

Now we show that the invariants hold for $A^{j'-1}$. As $Q_{j'}$ becomes closed by the development to j', it is easy to observe that I_1 is established for $A^{j'-1}$. For showing that I_2 is established as well, it suffices to prove that

after Step (1) has been applied for index j', the hotchild of $q_{j'}$ belongs to the same subtree T^i in $A^{j'-1}$ and A^*.

Let T^i be the subtree containing the hotchild of $q_{j'}$ in A^*. From the precondition of the lemma we know that $Q_{j'}$ is either closed or open in A^*. It follows that $d_i^x = c_{j'}$ for some $x \in \{1, 2, 3\}$.

P^i must be open in A^*, because if it were closed, then no descendant of p_i could have a hotchild in T^i. If $x = 3$, then P_i is open subject to j' in A^*, so T^i is developed to j' in Step (1) and I_2 follows.

Otherwise, P_i is open subject to some index $j'' > j'$ in A^*. T^i has been developed to j'' in a previous iteration of Step (1), so in $A^{j'}$ the node $q_{j''}$ has a hotchild in T^i. From the induction hypothesis regarding I_2 we know that the hotchild of $q_{j''}$ in A^* also belongs to T^i.

Therefore, we know that there is at least one index $j'' > j'$ such that in A^* the hotchildren of j' and j'' are in the same subtree T^i. Now consider j'' to be the smallest possible index satisfying that property. We know that $Q_{j''}$ is either open or closed, but it must be open subject to some j''' because otherwise a hotlink from $q_{j'}$ into T^i could not exist.

It is not possible that $j''' > j'$, as this would violate the minimality property of j''. It is either not possible that $j''' < j'$, because then there could be no $d_i^x = c_{j'}$. So $j''' = j'$ must hold, i.e. $Q_{j''}$ is open subject to j' in both A^* and $A^{j'}$.

From the induction hypothesis regarding I_2 follows that the hotchild of $q_{j''}$ belongs to T^i also in $A^{j'}$, so Step (1) effectuates that the hotchild of $q_{j'}$ belongs to T^i in $A^{j'-1}$ as well. This proves I_2 for $A^{j'-1}$.

Now the remaining claims of the lemma follow straightforwardly. After Step (1) has been completed, the invariants hold for A^j. Therefore, Step (2a) and (2b) are well-defined. Step (2) might result in some nodes from Q having more than one hotchild, but this cannot be the case for $Q_{j'}$, $j' > j$. After Step (2) has been applied, the open/closed status of the ancestor nodes of q_j is as in A^*, and, because of I_2, the hotchildren also belong to the same subtrees as in A^*. This means that also Step (3) is well-defined, as it suffices in fact to modify only the hotlinks of q_j and its descendants for obtaining A^*. □

Proof of Lemma 2.2. We divide the lemma into three claims.

> *Claim 1:* For $j = 1, \ldots, n$, there is at least one P_i or $Q_{j'}$ that is open subject to j.
>
> *Claim 2:* For $j = 1, \ldots, n$, there is at most one P_i or $Q_{j'}$ that is open subject to j.

Claim 3: For $j = 1, \ldots, n$, Q_j is either closed or open.

We prove the claims by induction over decreasing values of j. The argumentation naturally includes the base case $j = n$. Fix a j and assume that the lemma has already been shown for $j' > j$. From Lemma 2.3 we know that A^* can be obtained from B by the above transformation procedure.

Claim 1: From the weight assignment follows that the surplus achieved in Step (1) is exactly

$$\sum_{j'=j+1}^{n} 3W_{j'} \ .$$

After Step (2), the hotchildren of P and $Q_{j'}$, $j' > j$, are already as in A^*.

If we assume for the sake of contradiction that no P_i or $Q_{j'}$ is open subject to j, then each pair of deletion and development in Step (2) improves the assignment by at most $3W_{j-1}$, so the total improvement of that step is at most

$$(m + n) \cdot 3W_{j-1} < 3hW_{j-1} < 0.5W_j \ ,$$

as we have assumed that in the 3 Set Cover instance there is at least one $d_i^x = c_j$, thus there are at least 3 nodes u with $W_{j-1} \leq \omega(u) < W_j$ in T, and Equation 2.2 guarantees that their total weight is less than $\frac{1}{2h} W_j$.

The only leaf of weight greater than W_j whose path to is possibly affected by Step (3) is b_j. The path length to b_j can only increase by one, which happens if the hotlink $(\bar{q}_j, b_j)$ is replaced with (q_j, b_j). So the surplus achieved in Step (3) is less than

$$2W_j + h \sum_{v \in Lj-1} \omega(v) < 2.5W_j \ .$$

Summing up the surplus terms results in a total surplus of A^* strictly less than $3 \sum_{j'=1}^{n} W_{j'}$, a contradiction.

Claim 2: Consider a P_i or $Q_{j'}$ that is open subject to j in A^*. W. l. o. g. we assume that it is P_i that is open subject to j. Let $c_j = d_i^x$ in the X3C instance. For deleting $(\bar{p}_i.a_i)$ and developing T^i to j in Step (2), a reward of $3W_j$ is gained. Then, in Step (3), the hotlinks leading to r_i, σ_i^x and $\bar{\sigma}_i^x$ are possibly reassigned.

In the following, we show that the hotchild of q_j in A^* must be a node from T^i. Claim 3 then follows from the fact that q_j can only have one hotchild. For contradiction, we assume that the hotchild of q_j does not belong to T^i and show that this always causes A^* to be suboptimal.

First, assume that also the hotchild of $\bar{q}_j$ is neither r_i, σ_i^x, nor $\bar{\sigma}_i^x$. Then the total length of the paths to σ_i^x and $\bar{\sigma}_i^x$ has increased by at least 3 in

Step (3)[1]. This means that the weighted path length to those two leaves has increased by more than $6W_j$. The other leaves in T^i affected by Step (3) are all lighter than W_j and so the weighted path length to them has decreased by at most $W_j/2$ in Step (3). Altogether, the weighted path length to the leaves in T^i has increased by at least $5.5W_j$ in Step (3), which is more than the reward gained for developing T_i to j in Step (2). Therefore, one would obtain a better assignment than A^* by keeping P_i closed in Step (2).

We now know that a hotparent of b_j can only be q_j. But then, as both σ_i^x and $\bar{\sigma}_i^x$ are heavier than b_j, A^* would be improved by interchanging the hotchildren of q_j and $\bar{q}_j$.

Therefore, b_j has no hotparent at all in A^*. The hotlink $(\bar{p}_j, b_j)$ must have been deleted in Step (3), and so the length of the path to b_j has increased by one. As we still assume that the hotchild of q_j does not belong to T^i, the total length of the paths to σ_i^x and $\bar{\sigma}_i^x$ has increased by at least 1 during Step (3). As the weight of b_j, σ_i^x, and $\bar{\sigma}_i^x$ is at least $2W_j$ each, the weighted path length to those leaves has increased by at least $4W_j$ in Step (3). The other leaves in T^i affected by Step (3) are all lighter than W_j and so the weighted path length to them has decreased by at most $W_j/2$ in that step. Altogether, the weighted path length to the leaves in T^i and b_j has increased by at least $3.5W_j$ in Step (3), which is more than the reward gained for developing T_i to j in Step (2). Therefore, one would obtain a better assignment than A^* by keeping P_i closed in Step (2) and not deleting $(\bar{p}_j, b_j)$ in Step (3).

Claim 3: From Claim 1 and 2 we already know that there is a unique P_i or $Q_{j'}$ that is open subject to j in A^*, say w.l.o.g. P_i. Let $c_j = d_i^x$ in the X3C instance. Similar to the proof of Lemma 2.1, we consider the subtree T_{q_j,A^*} and the partial assignment $A^*|T_{q_j,A^*}$, which must be optimal for T_{q_j,A^*}. The only leaves heavier than W_j in that subtree are b_j, σ_i^x, and $\bar{\sigma}_i^x$.

This is the same situation as in the proof of Lemma 2.1. With W_{n+i}, p_i, π_i, $\bar{\pi}_i$, and a_i respectively replaced with W_j, q_j, σ_i^x, $\bar{\sigma}_i^x$, and b_j, one can prove that Q_j not being closed or open leads to a contradiction to the optimality of A^*. □

Lemma 2.4 *There is a solution to the instance (C, D) of X3C if and only if there is a hotlink assignment for the corresponding tree T achieving a surplus of β.*

Proof. "⇒": Let $D' = D_{i_1}, \ldots, D_{i_k}$ be a solution to (C, D), i.e. the elements of D' are pairwise disjoint. Equivalently, each element $c_j \in C$ is contained in exactly one $D_i \in D'$.

[1]For this to hold also in case of j=1 the extra nodes u_1, u_2 are required.

Starting with the basic assignment B, we construct a suitable hotlink assignment for T. For each $D_i \in D'$, apply the following transformation: Let $d_i^1 = c_{j_1}, d_i^2 = c_{j_2}$ and $d_i^3 = c_{j_3}$. Delete the hotlink $(\bar{p}_i, a_i)$ and develop T^i to j_3. Then, for $x = 3, 2$, delete the hotlink $(\bar{q}_{j_x}, b_{j_x})$ and develop T^i to j_{x-1}. Finally, delete $(\bar{q}_{j_1}, b_{j_1})$ and fully develop T^i. This causes P_i, Q_{j_3}, Q_{j_2}, and Q_{j_1} to be open.

During the transformation, a subtree is developed to each $j = 1, \ldots, n$. So for those developments, a surplus of $\sum_{j=1}^{n} 3W_j$ is gained. Furthermore, k subtrees are fully developed, each time gaining an extra surplus of 1. Therefore, the total surplus of the resulting hotlink assignment is exactly β.

"$\Leftarrow$" If a hotlink assignment A for T achieves a surplus of at least β, then so does an optimal hotlink assignment A^* for T. Therefore, due to Lemma 2.2, each P_i and each Q_j is either closed or open in A^*, and for each $j = 1, \ldots, m$ there is exactly one P_i or $Q_{j'}$ that is open subject to j.

Now we consider again the transformation procedure whose soundness has been proven in Lemma 2.3. From to Lemma 2.2 follows that we can set $j = 0$. Then A^* already results from Step (1). As the surplus of A^* is at least $\beta = 3\sum_{j=1}^{n} W_j + k$, the procedure must include a development to j for each $j = 1, \ldots, m$ and k times the step to the full development of a subtree.

Before a subtree T^i can be fully developed, that subtree must have been developed to three different indices j_1, j_2, j_3. Moreover, these index sets of fully developed subtrees have to be pairwise disjoint. Therefore, there are k different subtrees having disjoint index sets. This corresponds to the existence of a solution to the X3C instance. □

We are now ready to prove the main theorem of this chapter.

Theorem 1 *The Hotlink Assignment Decision Problem is NP-Complete.*

Proof. The path length of any hotlink assignment can be calculated in polynomial time, i.e. the problem is in NP.

In Section 2.2 and 2.5 we have given a construction method for a weighted tree corresponding to an instance of 3-Set Cover, which is NP-hard [Kar72]. The construction can be performed in polynomial time. In particular, the size of the corresponding tree is polynomial in the size of the X3C instance, which has been shown in Section 2.6. Lemma 2.4 states that there is a solution to the X3C instance if and only if there is a hotlink assignment for the corresponding tree achieving a path length of $p(B) - \beta$, where β is given in Equation 2.1, and B has been defined in the beginning of Subsection 2.3. □

2.8 Remarks

In this chapter we have proven that the Hotlink Assignment Problem is NP-hard. This holds true in the common model where hotlinks can end in any node.

The intractability of the problem even remains under certain restrictions concerning the sinks of hotlinks. Consider the case when hotlinks may only end in nodes having only leaf children. If there is a solution to an X3C instance, then the hotlink assignment proposed for the corresponding tree in this chapter satisfies that restriction. If there is no solution to the X3C instance, then no hotlink assignment achieves a surplus of β, which also includes all feasible assignments in the restricted model.

On the other hand, we will see in the next chapter that optimal solutions can be computed efficiently in the scenario where hotlinks may only end in leaves.

Chapter 3

Optimal Assignment of Hotlinks Pointing to Leaves

In this chapter we consider a model where the solution space of feasible hotlink assignments is limited by an additional constraint. In many scenarios where hotlinks are actually used, they only point directly to leaves, i.e. to the destination pages of users. Examples include product recommendations of online stores, or suggestions of frequently used functions in large menus of computer applications. The supposable reason for such a restriction is that users would possibly be confused to end up on another navigation page or sub-category after following a hotlink.

It turns out that under this restriction optimal solutions can be computed in polynomial time. This is even possible when K is a function, i.e. the maximum number of outgoing hotlinks is specified for each node individually.

This chapter is organized as follows: In Section 3.1 we formally define the model. Our polynomial time algorithm is presented in Section 3.2. Section 3.3 concludes.

3.1 Problem Definition

Let $T = (V, E, \omega)$ be a weighted tree rooted at r. Let $L \subseteq V$ be the set of leaves inside T. The weight function $\omega : L \to \mathbb{R}_0^+$ assigns a non-negative weight to each leaf. Furthermore, we are given a function $K : V \to \mathbb{N}_0$ specifying the maximum number of hotlinks that are allowed to start from each node. We assume that $K(v) \leq |L|$ for any $v \in V$. Problem instances in this chapter are given by a pair (T, K).

A hotlink assignment $A \subset V \times L$ is feasible, if $|\{(u, v) \in A\}| \leq K(u)$ for each $u \in V$, and if it is feasible according to Definition 1.1. Note that

the impossibility of crossing hotlinks like shown in Figure 1.1 already follows from the restriction concerning hotchildren.

The objective function is the path length as given in Section 1.2.4. As here we are interested in the optimal solution, we could as well consider the gain.

3.2 A Polynomial Time Algorithm

We develop a dynamic programming algorithm that is able to compute optimal hotlink assignments in polynomial time.

Definition 3.1 *Given a problem instance (T, K) and a node $v \in T$, we denote by K_{+v}/K_{-v} the function obtained from K by increasing/decreasing the image of v by 1.*

Assume that there is an efficient method PullUp for transforming an optimal assignment A for (T, K) into an optimal assignment A^+ for (T, K_{+r}). Then Algorithm 1 computes an optimal solution for any problem instance (T, K), because assignment A^0 is clearly optimal for (T, K'), where K' is obtained from K by setting the image of r to 0.

Algorithm 1: L-Opt (main idea)

Input: (T, K), a problem instance, where T is rooted at r
Output: A, a hotlink assignment for (T, K)

```
1 if T has a depth of less than 2 then
2     A = ∅
3 else
4     A^0 = ⋃_{v∈ch(r)} L-Opt(T_v)
5     for i = 1, ..., K(r) do
6         A^i = PullUp(T, A^{i-1})
7     A = A^{K(r)}
```

The remainder of this section is devoted to the development of the transformation method PullUp. We start by proving two basic lemmas.

Lemma 3.1 *Let A be an optimal hotlink assignment for some problem instance. If $(v_1, l_1), (v_2, l_2) \in A$, $v_2 \in \mathrm{desc}(v_1)$, and $l_1 \in \mathrm{desc}(v_2)$, then $\omega(l_1) \geq \omega(l_2)$.*

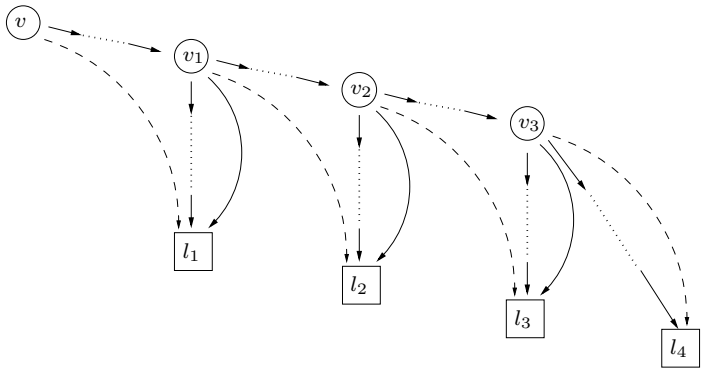


Figure 3.1: When applying the transformation sequence (l_1, l_2, l_3, l_4), the solid hotlinks are replaced with the dashed ones. Dotted lines represent tree paths.

Proof. Assume $\omega(l_1) < \omega(l_2)$. By swapping the hotchildren of v_1 and v_2, we can improve the path length of A by $\text{dist}(v_1, v_2)(\omega(l_2) - \omega(l_1))$, which is strictly positive. □

Lemma 3.2 *Let A be optimal for (T, K) and let $(v, l) \in A$. Then $A \setminus \{(v, l)\}$ is optimal for $(T \setminus \{l\}, K_{-v})$.*

Proof. If any A' was better than $A \setminus \{(v, l)\}$ for $(T \setminus \{l\}, K_{-v})$, then $A' \cup \{(v, l)\}$ would be better than A. As A is optimal, this is not possible. □

Definition 3.2 *Let A be a hotlink assignment for (T, K). A transformation sequence for A into K_{+v} is a sequence of leaves $(l_1, \ldots, l_k), k \geq 0$, where l_i has a hotparent $v_i \in \text{anc}(l_{i+1})$ in A for $1 \leq i \leq k-1$.*

Such a sequence represents the modification of A where all hotlinks pointing to $l_1, \ldots, l_k$ are replaced with (v, l_1) and (v_i, l_{i+1}) for $1 \leq i \leq k-1$. In case of $k = 0$, we speak of the empty transformation sequence, which represents the identity.

The transformation sequence is optimal if the resulting hotlink assignment is optimal for (T, K_{+v}).

Figure 3.1 shows an example of the modification that can be described by a transformation sequence.

Lemma 3.3 *Let A be an optimal hotlink assignment for (T, K), and let $v \in T$. There exists an optimal transformation sequence for A into K_{+v}.*

Proof. We prove the lemma by induction over the number of leaves. If T has only one leaf l_1, an optimal assignment for (T, K_{+v}) can be obtained by the transformation sequence (l_1) or the empty sequence.

Assume that the lemma holds for trees with $n-1$ leaves and less. Let T have n leaves and let A^+ be an optimal assignment for (T, K_{+v}). If v has less than $K(v)+1$ hotchildren in A^+, then it is clear that A is also optimal for (T, K_{+v}).

Otherwise, there is a hotlink $(v, l_1) \in A^+$ with $(v, l_1) \notin A$. The leaf l_1 becomes the first component of our transformation sequence. Let $T' = T \setminus \{l_1\}$. We distinguish between two cases.

Case 1: l_1 has no hotparent in A. Then A is also optimal for (T', K). Thus, A is at least as good as $A^+ \setminus \{(v, l_1)\}$ for (T', K). Let A' be the result of applying the transformation sequence (l_1) to A, i.e. $A' = A \cup \{(v, l_1)\}$. As also $A^+ = (A^+ \setminus \{(v, l_1)\}) \cup \{(v, l_1)\}$, A' achieves at most the path length of A^+ for (T, K_{+v}) and is therefore optimal.

Case 2: $(v_1, l_1) \in A$. Lemma 3.2 implies that $A \setminus \{(v_1, l_1)\}$ is optimal for (T', K_{-v_1}). From the induction hypothesis follows that there is a finite transformation sequence $(l_2, \ldots, l_k)$ for $A \setminus \{v_1, l_1\}$ into K, such that the resulting hotlink assignment A' is optimal for (T', K). Thus, A' is at least as good as $A^+ \setminus \{(v, l_1)\}$ for (T', K). The assignment $A'' = A' \cup \{(v, l_1)\}$ can be considered as the result of applying the transformation sequence $(l_1, \ldots, l_k)$ to A. As $A^+ = (A^+ \setminus \{(v, l_1)\}) \cup \{(v, l_1)\}$, A'' is at least as good as A^+ for (T, K_{+v}) and therefore optimal. □

Definition 3.3 *Let $(l_1, \ldots, l_k)$ be a transformation sequence for A into K_{+v}. Let $v_0 = v$ and let v_i be the hotparent of l_i for $1 \leq i \leq k-1$. We say that the sequence is ordered, if $v_i \in \mathrm{desc}(v_{i-1})$ for $1 \leq i \leq k-1$, and, in case of l_k having a hotparent v_k in A, $v_k \in \mathrm{desc}(v_{k-1})$.*

The transformation sequence depicted in Figure 3.1 is ordered.

Lemma 3.4 *Let A be an optimal hotlink assignment for (T, K). There is an optimal ordered transformation sequence for A into K_{+r}.*

Proof. We show that any optimal transformation sequence for A into K_{+r} can be modified such that it is ordered and still optimal. We do this by induction over the length of the sequence. For the basic case, the empty sequence, there is nothing to show. For the induction step it suffices to show that from any optimal sequence of length 1 or greater that is not ordered, one component can be removed without increasing the path length of the resulting assignment.

Let $(l_1, \ldots, l_k)$ be an optimal transformation sequence for A into K_{+r}, and, for $1 \leq i \leq k$, let v_i be the hotparent of l_i in A (note that v_k possibly does not exist). If v_1 violates the specification of ordered transformation sequences, i.e. is no descendant of r, then $v_1 = r$ and we can remove l_1 from the sequence without affecting the resulting assignment.

Otherwise, let A^+ be the hotlink assignment obtained from A by applying $(l_1, \ldots, l_k)$. Assume that, for some $m \geq 1$, l_{m+1} is the first component of the sequence with $v_{m+1} \notin \mathrm{desc}(v_m)$. As we know that $v_m \in \mathrm{desc}(v_{m-1})$ and A^+ is optimal, Lemma 3.1 implies $\omega(l_m) \geq \omega(l_{m+1})$. As v_{m+1} is no descendant of v_m and $l_{m+1} \in \mathrm{desc}(v_m) \cap \mathrm{desc}(v_{m+1})$, either $v_{m+1} = v_m$ or $v_{m+1} \in \mathrm{anc}(v_m)$ holds. In the former case, l_{m+1} can be removed from the sequence without affecting the resulting assignment. In the latter case, because of the optimality of A and Lemma 3.1, $\omega(l_m) \leq \omega(l_{m+1})$, i.e. $\omega(l_m) = \omega(l_{m+1})$. Then we can remove l_m from the sequence, increasing the path length of the resulting assignment by

$$\begin{aligned}
&\big((\mathrm{dist}(v_{m-1}, l_m) - 1)\,\omega(l_m) + (\mathrm{dist}(v_m, l_{m+1}) - 1)\,\omega(l_{m+1})\big) \\
-&\big((\mathrm{dist}(v_{m-1}, l_{m+1}) - 1)\,\omega(l_{m+1}) + (\mathrm{dist}(v_m, l_m) - 1)\,\omega(l_m)\big) \\
=\ &\omega(l_m)\,\big(\mathrm{dist}(v_{m-1}, v_m) - \mathrm{dist}(v_{m-1}, v_m)\big) = \ 0\ .
\end{aligned}$$

□

Theorem 2 *Given an optimal hotlink assignment for (T, K), an optimal assignment for (T, K_{+r}) can be computed in polynomial time.*

Proof. Let A be optimal for (T, K). From Lemma 3.4 follows that it suffices to find an optimal ordered transformation sequence for A into K_{+r}. We describe the algorithm for solving that task in a recursive manner. Assumed that (T, K) and A are fixed, the algorithm takes a node $v \in T$ as the input, and computes an optimal transformation sequence for $A|T_{v,A}$ into K_{+v}.

If $T_{v,A}$ contains no leaves, this is the empty sequence. Otherwise, for each leaf l in $T_{v,A}$ the algorithm computes a best ordered transformation sequence that starts with l. If l has no hotparent in A, then that sequence is (l). Else, if $(v', l) \in A$, the algorithm recursively computes an optimal transformation sequence for $A|T_{v',A}$ into $K_{+v'}$ and appends that sequence to l. For each leaf l, the algorithm calculates the benefit (decrease in path length) $s(v, l)$ caused by the corresponding best sequence and returns the sequence maximizing

that benefit. Formally,

$$s(v) = \begin{cases} \max\limits_{l \text{ is a leaf in } T_{v,A}} s(v,l) & \text{if } T_{v,A} \text{ contains a leaf} \\ 0 & \text{otherwise ,} \end{cases} \tag{3.1}$$

$$s(v,l) = \begin{cases} \big(\mathrm{dist}(l,v) - 1\big)\,\omega(l) & \text{if } l \text{ has no hotparent in } A \\ \mathrm{dist}(v',v)\,\omega(l) + s(v') & \text{if } l \text{ has a hotparent } v' \text{ in } A\ . \end{cases} \tag{3.2}$$

As (T, K) and A are fixed, a table containing all possible values of $s(v)$ and the corresponding transformation sequences has linear size. At most $|V|$ different possibilities have to be compared during the computation of any table entry, so the table can be built in quadratic time.

We formally prove the correctness of the algorithm by induction over the number of leaves in $T_{v,A}$. The base case is obvious, and for the induction step it suffices to show that Equation 3.2 is correct. The case of l having no hotparent in A is clear. Otherwise, by the induction hypothesis, an optimal ordered transformation sequence for $A|T_{v',A}$ into $K_{+v'}$ is computed by the recursive call. Let A^+ be the assignment resulting from applying that sequence to $A|T_{v',A}$. From Lemma 3.2 follows that $A^+|(T_{v',A} \setminus \{l\})$ is optimal for $(T_{v',A} \setminus \{l\}, K)$.

Let $T'_{v',A} = T_{v',A} \setminus \{l\}$. Observe that any transformation sequence for $A|T_{v,A}$ into K_{+v} that starts with l, besides adding (v,l), only affects $A|T'_{v',A}$. Thus, the corresponding resulting assignment A' can be partitioned into $A' = \{(v,l)\} \cup (A|T_{v,A} \setminus A|T_{v',A}) \cup A'|(T'_{v',A})$. As the weighted path length to the leaves in $T'_{v',A}$ is $\mathrm{dist}(v,v') + p(A'|T'_{v',A}, T'_{v',A})$, the overall path length $p(A', T_{v,A})$ is minimized when $p(A'|T'_{v',A}, T'_{v',A})$ is minimized. This is guaranteed by $A'|T'_{v',A}$ being equal to $A^+|T_{v',A} \setminus \{l\}$, which is achieved by the transformation sequence selected by the algorithm. □

We postpone a detailed pseudo-code description of PullUp to Chapter 5.1.8, where some improvements addressing the runtime will be proposed.

Corollary 3.1 *An optimal solution to any problem instance (T, K) can be computed in polynomial time.*

Proof. Let $T = (V, E)$. Straightforward observation shows the correctness of Algorithm 1. During its execution, the number of recursive calls of L-Opt is at most $|V|$, and there are at most $\sum_{v \in V} K(v) \leq |V|^2$ calls of PullUp. The latter procedure has quadratic runtime, so the overall runtime of L-Opt is $O(|V|^4)$. □

3.3 Remarks

In this chapter we have seen that the model of hotlinks pointing only to leaves is computationally tractable. We will see in Chapter 5 that L-Opt can be implemented such that on typical tree instances it is one of the fastest hotlink assignment methods. Moreover, the solutions are comparable to the best assignments of hotlinks that can point to arbitrary nodes.

We have assumed that only the leaves of a tree have non-zero weights. Without that assumption, we believe that the restriction of hotchildren to leaves is not sensible. Alternatively, one could assume that users only follow hotlinks that lead directly to their destination node. That scenario can easily be simulated by our model, namely, by relocating the weight of each internal node v to an additional leaf sibling of v.

Chapter 4

Approximation Algorithms

In this chapter we readdress the original Hotlink Assignment Problem, where hotlinks can start and end anywhere in the tree. From the hardness result in Chapter 2 follows that optimal solutions cannot be computed in reasonable worst case time unless P=NP.

Therefore, we address the task of developing polynomial time algorithms having at least constant approximation ratios. We present two constant factor approximations and one PTAS, and thus substantially improve upon the best approximation factors previously known for both the path length and the gain.

This chapter is organized as follows: In Section 4.1, some additional notation is introduced. The analyses of our algorithms are based on an analytical tool set consisting of three operations for hotlink modification. These operations are given in Section 4.2. The approximation algorithms are then presented in the following three sections. Section 4.6 concludes the chapter.

4.1 Notation

We introduce a number of notational concepts that facilitate the description and analysis of the approximation algorithms.

In this chapter, unless otherwise stated, we assume that the maximum number of outgoing hotlinks is some constant K. Furthermore, we consider the model where any tree node can carry a non-zero weight.

The *cumulated weight* of a node u is obtained by summing up the weights of all nodes v where u is located on the greedy user's path from r to v. Formally,

$$W^A(u) = \sum_{v \in T_{u,A}} \omega(v) \ . \tag{4.1}$$

The cumulated weights have a natural interpretation when ω constitutes an access probability distribution. Regarding the hotlink assignment A, $W^A(u)$ is the probability with which node u is traversed. Note that $W^A(u)$ indeed depends on A. We abbreviate $W^\emptyset$ with W.

Now, the gain of a hotlink assignment A can be reformulated as the sum over the path shortening contributions of its hotlinks, i.e.

$$g(A) = \sum_{(u,v)\in A} W^A(v)(\text{dist}^\emptyset(u,v) - 1) \ . \tag{4.2}$$

The gain of a single hotlink $(u,v) \in A$ is defined as $g^A(u,v) = (\text{dist}((u,v) - 1) \cdot W^A(v)$. Intuitively, the gain of a hotlink is the number of bypassed nodes times the total weight of the nodes affected by the hotlink. We abbreviate $g^\emptyset(u,v)$ by $g(u,v)$.

Lemma 4.1 *Let $A = A_1 \dot\cup A_2$ be a partition of a feasible hotlink assignment. Then $g(A) \leq g(A_1) + g(A_2)$.*

Proof. As $A_1 \subseteq A$, $T_{u,A}$ is completely contained in T_{u,A_1} for any $u \in T$. Therefore, $W^A(u) \leq W^{A_1}(u)$ due to Equation 4.1. Analogously, $W^A(u) \leq W^{A_2}(u)$.

According to Definition 1.1, each node in T has only one hotparent in A. This implies that the sets of nodes that are reached by hotlinks from A_1 and A_2 are disjunctive, and

$$\begin{aligned}
g(A) &= \sum_{(u,v)\in A} W^A(v)(\text{dist}(u,v) - 1) \\
&= \sum_{(u,v)\in A_1} W^A(v)(\text{dist}(u,v) - 1) + \sum_{(u,v)\in A_2} W^A(v)(\text{dist}(u,v) - 1) \\
&\leq \sum_{(u,v)\in A_1} W^{A_1}(v)(\text{dist}(u,v) - 1) + \sum_{(u,v)\in A_2} W^{A_2}(v)(\text{dist}(u,v) - 1) \\
&= g(A_1) + g(A_2) \ .
\end{aligned}$$

□

4.2 Basic Operations

In this section we introduce an analytical tool set consisting of three basic operations for the modification of hotlink assignments. Note that none of

these operations is actually applied by any algorithm given in the thesis. We require them however for the performance analyses in this chapter.

Typically, we show that an optimal hotlink assignment $\textsc{Opt}(T)$ can be transformed into the assignment $\textsc{Alg}(T)$ computed by the algorithm Alg. The transformation can be expressed in terms of those basic operations, making it easier to prove that the solution quality deteriorates only by a certain factor.

In the following, we assume T and A to be fixed. The first operation aims to get rid of all hotlinks starting in a certain node: After applying $\textsc{PushDown}(u)$ to A, the node u is guaranteed to have no hotchild.

$\textsc{PushDown}(u)$

```
1 if u has a hotchild in A then
2     U ← {v ∈ ch(u) | ∃(u, v') ∈ A : v' ∈ {v} ∪ desc(v)}
3     forall v ∈ U do
4         PushDown(v)
5         forall (u, v') ∈ A with v' ∈ {v} ∪ desc(v) do
6             replace (u, v') with (v, v')
7     delete any self-loop from A
```

Lemma 4.2 *Operation* $\textsc{PushDown}(u)$ *causes a decrease in gain of at most* $W^A(u)$*. If* $K = 1$*, then* $\textsc{PushDown}(u)$ *causes a decrease in gain of at most* $W^{A\setminus\{(u,v')\}}(v)$*, where* v' *is the hotchild of* u *in* A *and* $v \in \mathrm{ch}(u) \cap \mathrm{anc}(v')$*.*

Proof. We prove the lemma by induction over the depth of the tree. For trees of depth 1 or less there is nothing to show. For the induction step, let us assume that T has a depth of n, and let A' be the hotlink assignment that results from the application of $\textsc{PushDown}(u)$ to A. If A assigns no hotchild to u then $A' = A$, and no decrease in gain occurs.

Otherwise, let U be the set computed in Line 2 of the operation, and let $v_1, \ldots, v_{K'}$ be the hotchildren of u in A.

By the induction hypothesis, Line 4 causes a total decrease in gain of at most

$$\sum_{v \in U} W^A(v) \leq W^A(u) - \sum_{j=1}^{K'} W^A(v_j) \ .$$

Due to Line 6, the gain decreases by

$$\sum_{j=1}^{K'} W^A(v_j) \ .$$

Summing up both kinds of losses gives a maximum total decrease in gain of $W^A(u)$.

We now consider the case of $K = 1$. By the induction hypothesis, Line 4 decreases the gain by at most $W^A(v)$. Line 6 costs $W^A(v')$, so the total loss is not more than $W^{A\setminus\{(u,v')\}}(v)$. □

A convenient property of the clairvoyant user model is that hotlinks can arbitrarily be added to an existing assignment without decreasing the gain. This property does not directly carry over to the greedy user model, as here the feasibility constraint could be violated.

However, with the help of FreeInsert, it is possible to insert a new hotlink (u, v) into an assignment whenever $v \in T_{u,A}$. The operation makes a number of further adjustments such that both the feasibility and the gain is preserved.

FreeInsert(u, v)

Precondition: $v \in T_{u,A}$

1 $(u, u_1, \ldots, u_n, v) \leftarrow$ the path from u to v
2 $u_{f(1)}, \ldots, u_{f(m)} \leftarrow$ nodes having a hotchild in T_v, where f is *decreasing*
3 **for** *i = 1,...,m* **do**
4 PushDown(v)
5 **foreach** *hotchild* v' *of* $u_{f(i)}$ *with* $v' \in T_v$ **do**
6 replace hotlink $(u_{f(i)}, v')$ with (v, v')
7 insert hotlink (u, v)

Lemma 4.3 *No decrease in gain occurs when applying* FreeInsert.

Proof. We adopt the definitions from Line 1 and 2 of FreeInsert. For $1 \le i \le m$, let

$$w_i = \sum_{(u_{f(i)},v') \in A, v' \in T_v} W^A(v')$$

be the sum of the cumulated weights of $u_{f(i)}$'s hotchildren in T_v. Let $P = \{u, u_1, \ldots, u_n, v\}$, let b be the number of nodes in P that a bypassed by some hotlink that both starts and ends in P. For $1 \le i \le m$, let b_i be the number of such nodes that are descendants of $u_{f(i)}$.

After the ith iteration of outer loop, the cumulated weight of v is

$$W^A(v) + \sum_{j=1}^{i} w_j \ .$$

Thus, in the ith iteration we loose a maximum amount of gain of

$$\big(W^A(v) + \sum_{j=1}^{i-1} w_j\big) + (n - f(i) + 1 - b_i)w_i \ .$$

The first summand accounts for Line 4, while the second summand accounts for Line 5 and 6. Summing up for $1 \leq i \leq m$ gives a total loss of at most

$$\begin{aligned} &\sum_{i=1}^{m}\Big(W^A(v) + \sum_{j=1}^{i-1} w_j + \big(n - (f(i) - 1) - b_i\big)w_i\Big) \\ &= mW^A(v) + \sum_{i=1}^{m}(m-i)\cdot w_i + \sum_{i=1}^{m}\Big(n - \big(f(i) - 1\big) - b_i\Big)w_i \ . \end{aligned}$$

Because of the feasibility condition, $m \leq n - b$. Regarding A, $(m - i)$ is the number of nodes in $P \cap \mathrm{anc}(v_{f(i)})$ having hotchildren in T_v, and $(b - b_i)$ is the number of nodes bypassed by hotlinks from $\big(P \cap \mathrm{anc}(v_{f(i)})\big) \times \big(P \cap \mathrm{anc}(v_{f(i)})\big)$. So, $|P \cap \mathrm{anc}(v_{f(i)})| = f(i) - 1 \geq (m - i) + (b - b_i)$. Thus, the loss term can be bounded above by

$$(n - b)\big(W^A(v) + \sum_{i=1}^{m} w_i\big) \ ,$$

which is exactly the increase in gain achieved by Line 7 of FreeInsert. □

The third and last operation splits a hotlink (u, v) into two hotlinks (u, v') and (v', v). Like the other operations, some further adjustments are made to the assignment in order to preserve feasibility and bound the gain loss.

Split(u, v', v)

Precondition: $(u, v) \in A, v' \in \mathrm{desc}(u) \cap \mathrm{anc}(v)$

1 FreeInsert(u, v')
2 PushDown(v')
3 replace hotlink (u, v) with (v', v)

Lemma 4.4 *Let A' be the hotlink assignment obtained from A by the application of* Split(u, v', v). *Then $g(A) - g(A') \leq W^{A'}(v')$.*

operation	precondition	effect	maximum cost
PUSHDOWN(u)	-	u has no hotchild	K-HLA: $W^A(u)$ 1-HLA: $W^{A'}(v)$
FREEINSERT(u, v)	$v \in T_{u,A}$	$(u, v) \in A'$	0
SPLIT(u, v', v)	$(u, v) \in A$ $v' \in \text{desc}(u) \cap \text{anc}(v)$	$(u, v) \notin A'$ $(u, v') \in A'$ $(v', v) \in A'$	$W^{A'}(v')$

Table 4.1: Overview of the operations introduced in Section 4.2. A and A' represent the assignment before and after applying the operation.

Proof. Let A^1 be the HLA that results from applying FREEINSERT(u, v'), and let A^2 be the assignment obtained after applying PUSHDOWN(v'). From Lemma 4.2 and 4.3 follows that $g(A) \leq g(A^1) \leq g(A^2) + W^{A^1}(v') = g(A') + W^{A^2}(v) + W^{A^2}(v') = g(A') + W^{A'}(v')$. □

It is straightforward to observe that PUSHDOWN, FREEINSERT, and SPLIT preserve the feasibility of A, and the resulting hotlink assignment is still a K-HLA. Table 4.1 summarizes the characteristics of the operations.

4.3 A Natural Greedy Strategy

In this section we analyze a natural greedy strategy GREEDY for the K-Hotlink Assignment Problem.

Greedy-like hotlink assignment methods have been proposed before. In [CKK+01], different greedy algorithms are compared experimentally. One of those algorithms is also studied experimentally in [KKM02]. A greedy algorithm for the clairvoyant user model has been proved in [MP03] to be a 2-approximation in terms of the gain.

The description of a greedy strategy called *greedyBFS* in [GKL+07] corresponds to the version we consider in Algorithm 2. Nevertheless, our version is more general as it is applicable for the K-Hotlink Assignment Problem.

Line 5 of Algorithm 2 corresponds to solving the *Bookmark Assignment Problem*, where K' hotlinks starting in the root have to be chosen so as to maximize the gain. An optimal set of bookmarks can be computed in polynomial time (cf. Chapter 1.4.1).

4.3.1 Upper Bound

Theorem 3 *GR holds an approximation ratio of 2 in terms of the gain.*

Algorithm 2: GREEDY

Input: $T = (V, E, \omega)$, a weighted tree rooted at r
Output: A, a hotlink assignment for T

```
1 if T has a depth of less than 2 then
2     A = ∅
3 else
4     K' ← min{K, |V| − 1}
5     V' ← arg max_{U⊂V,|U|=K'} g({(r, v') | v' ∈ U})
6     A ← {(r, v') | v' ∈ V'}
7     foreach T' ∈ {T_v \ V' | v ∈ ch(r)} ∪ {T_{v'} \ (V' \ {v'}) | v' ∈ V'} do
8         A ← A ∪ GREEDY(T')
```

Proof. We prove the claim by induction over the tree depth. For trees of depth 1 or less there is nothing to show. Assume that the theorem has already been proven for trees of depth less than n and let T have a depth of n. Let A^* be optimal for T and let A^{GREEDY} be the assignment computed by GREEDY. Let further V^* be the set of r's hotchildren in A^*. We obtain A^1 from A^* by deleting all hotlinks starting in r. Let V' be the set of hotchildren that are chosen at Line 5 of Algorithm 2. We obtain A^2 from A^1 by applying FREEINSERT(r, v) for all $v \in V'$. Consider the partition $R = \{T_v \setminus V' \mid v \in \text{ch}(r)\} \cup \{T_{v'} \setminus (V' \setminus \{v'\}) \mid v' \in V'\}$ of T. We obtain A^{GREEDY} from A^2 by deleting all hotlinks from $A^2|T'$ and applying GREEDY to T' for each $T' \in R$. Applying Lemma 4.1, 4.3, and the greedy criterion, we can upper bound the gain of A^* by

$$\begin{aligned} g(A^*) &\leq g(\{(r,v) \mid v \in V^*\}) + g(A^1) \leq g(\{(r,v) \mid v \in V^*\}) + g(A^2) \\ &= g(\{(r,v) \mid v \in V^*\}) + g(\{(r,v') \mid v' \in V'\}) + \sum_{T' \in R} g(A^2|T', T') \\ &\leq 2g(\{(r,v) \mid v \in V'\}) + 2\sum_{T' \in R} g(A^{\text{GREEDY}}|T', T') = 2g(A^{\text{GREEDY}}) \,. \end{aligned}$$

□

4.3.2 Lower Bounds

The instance depicted in Figure 4.1 shows that, concerning the gain, the approximation ratio of GREEDY is not better than 2. With $K = 1$, the optimal assignment achieves a gain of $2n-1$, while the gain of GREEDY is n.

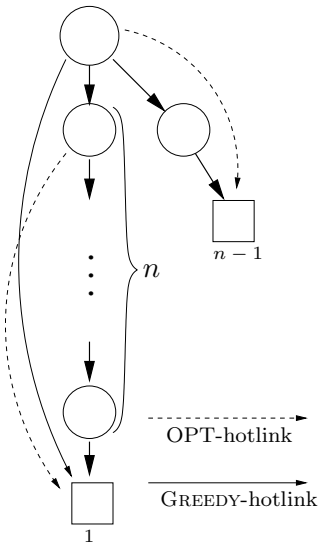


Figure 4.1: GREEDY holds no better approximation ratio than 2.

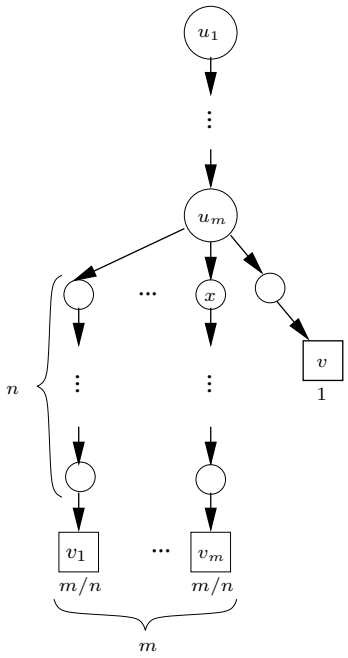


Figure 4.2: GREEDY holds no constant approximation ratio in terms of the path length.

The instance illustrated in Figure 4.2 shows that GREEDY holds no constant approximation ratio in terms of the path length. For $n \gg m$ and $K = 1$, GREEDY assigns the hotlinks $\{(u_i, v_i) | 1 \leq i \leq m\}$. The path length is

$$m + 1 + \sum_{i=1}^{m} i \cdot \frac{m}{n} = m + 1 + \frac{O(m^3)}{2n} \ .$$

A better assignment is $\{(u_1, v)\} \cup \{(u_i, v_{i-1} | 2 \leq i \leq m\} \cup \{(x, v_m)\}$, achieving a path length of

$$1 + \sum_{i=2}^{m+1} i\frac{m}{n} = 1 - \frac{m}{n} + \frac{O(m^3)}{2n} \ .$$

For $n \gg m$ the ratio between GREEDY and the better assignment is m.

4.4 An Approximation Scheme for the Gain

In this section we present an approximation scheme in terms of the gain for the K-Hotlink Assignment Problem. In [MP07], Matichin and Peleg show that restricting hotlinks to a length of 2 at most halves the possible gain. They give an algorithm that computes an optimal length 2 hotlink assignment and, therefore, is a 2-approximation. From an abstract point of view, our approach can be interpreted as a generalization of theirs. However, both our proof technique and our algorithmic idea are completely different.

4.4.1 Length Restricted Hotlink Assignment

Let the *length* of a hotlink (u, v) be defined as $\text{dist}(u, v)$. We prove that at most a fraction of $\frac{1}{h}$ of the possible gain must be sacrificed when restricting hotlinks to a maximum length of h.

Lemma 4.5 *For any tree T and integer $h > 1$, there is a hotlink assignment A^h for T with $\text{dist}(u, v) \leq h$ for each $(u, v) \in A^h$ and $\frac{h}{h-1} g(A^h) \geq g(A^*)$, where A^* is an optimal HLA for T.*

Proof. We prove the claim by induction over the tree depth. For trees having a depth of h and less, there is nothing to show. For a fixed $n > h$, assume that T has a depth of n and that the lemma has already been proven for trees of depth less than n. Let $V_1 = \{v \in V \mid (r, v) \in A^*, \text{dist}(r, v) > h\}$, $V_2 = \{v \in V \mid (r, v) \in A^*, \text{dist}(r, v) \leq h\}$, and let $U = \{v' \in V \mid \text{dist}(r, v') = h, \text{desc}(v') \cap V_1 \neq \emptyset\}$.

For each $v' \in U$ we apply $\textsc{Split}(r, v', v)$ for some $v \in V_1 \cap \mathrm{desc}(u)$. From the definition of Split follows that this causes v' to have only one hotchild. So for any $v'' \in \mathrm{desc}(v') \cap (V_1 \setminus \{v\})$, we can replace (r, v'') with (v', v'') and still have a K-HLA. We denote the resulting hotlink assignment as A'.

Let $V' = U \cup V_2$ and consider the partition $R = \left(\bigcup_{v \in \mathrm{ch}(r)} T_v \setminus V'\right) \cup \left(\bigcup_{v \in V'} T_v \setminus (V' \setminus \{v\})\right)$. We obtain A^h from A' by, for each $T' \in R$, replacing $A'|T'$ with an optimal length h hotlink assignment $A^h|T'$. Clearly, A^h is a length h assignment. Its gain is bounded below by the following inequation:

$$
\begin{aligned}
& g(A^*) \leq \sum_{v' \in U} W(v') + g(A') \\
= & \sum_{v' \in U} W(v') + g(\{(r, v) \mid v \in V'\}) + \sum_{T' \in R} g(A'|T', T') \\
\leq & \frac{1}{h-1} g(\{(r, v') \mid v' \in U\}) + g(\{(r, v) \mid v \in V'\}) + \sum_{T' \in R} \frac{h}{h-1} g(A^h|T', T') \\
\leq & \frac{h}{h-1} g(\{(r, v) \mid v \in V'\}) + \frac{h}{h-1} \sum_{T' \in R} g(A^h|T', T') \;=\; \frac{h}{h-1} g(A^h) \; .
\end{aligned}
$$

□

4.4.2 Algorithm Lpath

Now we show how to efficiently compute optimal length-restricted hotlink assignments. The PATH algorithm presented in [PLdS04a] computes an optimal hotlink assignment A that satisfies $\mathrm{dist}^A(r, u) \leq h$ for any node u. We give a modified version Lpath which computes an optimal K-HLA A under the restriction that $\mathrm{dist}^{A \setminus \{(u,v)\}}(u, v) \leq h$ for any $(u, v) \in A$. Since the latter restriction is weaker than demanding $\mathrm{dist}^{\emptyset}(u, v) \leq h$, the HLA computed by Lpath is at least as good as any K-assignment of hotlinks of maximum length h. Thus, Lemma 4.5 implies an approximation ratio of $\frac{h}{h-1}$.

We begin by fixing some arbitrary order "$\succ$" among the nodes of T. The *successor sibling* $\mathrm{succ}(u)$ of a node u is the sibling v of u with $u \succ v$ and $v \succ v'$ for any other sibling v' with $u \succ v'$. The *first child* $\mathrm{fc}(u)$ of a node u is the unique child v of u having no sibling $v' \succ v$.

Here we employ a slightly different view of the gain. In Equation 4.2, the gain of a single hotlink $(u, v) \in A$ depends on A in the sense that $W^A(u)$ might be less than $W(u)$. In the description of Lpath below, we consider the equivalent formula

$$
g(A) = \sum_{(u,v) \in A} W(v)(\mathrm{dist}^{A \setminus \{(u,v)\}}(u, v) - 1) \; , \tag{4.3}
$$

so here the gain of a single hotlink $(u, v) \in A$ depends on A in the sense that other hotlinks might already shorten the path between u and v. Remark that these other hotlinks must be contained in $\mathrm{desc}(u) \cap \mathrm{anc}(v)$.

The main algorithmic idea is to determine the concrete hotchildren of the nodes as late as possible. Assume that we are given a subtree T' together with a number of hotlinks that start outside that subtree and have already been decided to end somewhere in T'. One of these hotlinks could end in the root r' of T'. For each of the others, we only decide whether or not they point into the subtree rooted at the first child of r'.

Formally, subproblems are given by a triple (a, b, v). The first component $a = (a_1, \ldots, a_n, a_{n+1}) \in \{0, \ldots, K\}^{n+1}$ is a $(K+1)$-ary vector, $b \in \{0, 1\}$, and $v \neq r$ is a node in T. Such a triple represents the problem of computing an optimal assignment for the tree $T^{n,v}$, which is defined as follows: Let $r' = \mathrm{par}(v)$, and let $T^v = T_{r'} \setminus \{v' \in \mathrm{ch}(r') \mid v' \succ v\}$ be the maximum subtree of T such that v is the first child of the root of T'. $T^{n,v}$ is obtained by appending T^v to a path $q_1 \to \ldots \to q_n$ of length n via the edge (q_n, r'). The vector a represents a number of restrictions concerning hotlink assignments for $T^{n,v}$. No path node q_i is allowed to be a hotchild, and at most a_i hotlinks are allowed to start in q_i. Furthermore, r' is allowed to have at most a_{n+1} hotchildren and may have a hotparent only if $b = 1$. The original problem is given by $((K), 0, \mathrm{fc}(r))$.

We proceed describing how LPATH computes an optimal HLA for (a, b, v) (cf. Figure 4.3). The algorithm considers two main possibilities. The first possibility only exists if $b = 1$.

Case I: There is a hotlink that ends in r'. The hotparent of r' is some node q_i with $a_i \geq 1$. Due to the impossibility of crossing hotlinks, there can be no reasonable hotlink starting in any q_j with $j > i$. In terms of Equation 4.3, the distance between q_j and v is reduced by $(\mathrm{dist}(q_i, r') - 1)$. Consequently, the path nodes $q_{i+1}, \ldots, q_n$ can be deleted. The gain of an optimal assignment for (a, b, v) is calculated by

$$\tilde{g}_{\mathrm{I}}(a, b, v) = \max_{a_i \geq 1} \{(n - i) \cdot W(r') + \tilde{g}((a_1, \ldots, a_{i-1}, a_i - 1, a_{n+1}), 0, v)\} \ .$$

Case II: No hotlink ends in r'. The hotchildren of r' and any q_i must be either in T_v or $T^v \setminus \{v\} = T^{\mathrm{succ}(v)}$. The algorithm has to distinguish between four sub-cases.

Case IIa: The node v is a leaf and has no sibling in T^v. Then the best assignment consists of only one hotlink, and

$$\tilde{g}_{\mathrm{II}}(a, b, v) = \begin{cases} (n - \min\{i \mid a_i \geq 0\} + 1)\omega(v) & \exists i : a_i \geq 1 \\ 0 & \text{otherwise} \end{cases} \ .$$

Case IIb: The node v is an internal node and has no sibling in T^v. Then

$$\tilde{g}_{\mathrm{II}}(a,b,v) = \tilde{g}((a_1,\ldots,a_{n+1},K),1,\mathrm{fc}(v)) \ .$$

Case IIc: The node v is a leaf and has a sibling in T^v. For $1 \le i \le n+1$, let a^{i-} be the vector obtained from a by decrementing the ith component. Then

$$\begin{aligned}\tilde{g}_{\mathrm{II}}(a,b,v) = \max(\ &\{\tilde{g}(a,0,\mathrm{succ}(v))\}\cup \\ &\{(n-i+1)W(v)+\tilde{g}\big((a^{i-},0,\mathrm{succ}(v))\big)\} \mid a_i \ge 1\}\ ) \ .\end{aligned}$$

Case IId: The node v is an internal node and has a sibling in T^v. We use a vector $c \in \{0,\ldots,K\}^{n+1}$ to describe the distribution of hotchildren among the two subtrees T_v and $T^{\mathrm{succ}(v)}$. The set C contains all feasible vectors for the subproblem, i.e. $C = \{c \in \{0,\ldots,K\}^{n+1} \mid c_i \le a_i \text{ for } 1 \le i \le n+1\}$. The gain of an optimal assignment is calculated by

$$\begin{aligned}\tilde{g}_{\mathrm{II}}(a,b,v) = \max_{c\in C}\{&\tilde{g}((c_1,\ldots,c_{n+1},K),1,\mathrm{fc}(v)) + \\ &\tilde{g}((a_i-c_1,\ldots,a_{n+1}-c_{n+1}),0,\mathrm{succ}(v))\} \ .\end{aligned}$$

As we want to bound the relative length of hotlinks to h, we cut off the first $n-h$ components of a whenever $n > h$. This constitutes the only difference between PATH and LPATH, as PATH would set the gain to $-\infty$ in case of $n > h$.

Observe that in Case IIa and IIc a length $h+1$ hotlink leading to v is possibly assigned. This could be prevented by explicitly excluding q_1 from the set of possible hotparents for v. However, by allowing q_1 to be selected as the hotparent of v, the solution quality can only be improved.

The main formula used by LPATH for calculating the gain of an optimal HLA for the subproblem (a,b,v) is

$$\tilde{g}(a,b,v) = \begin{cases} \tilde{g}((a_{n-h+1},\ldots,a_{n+1}),b,v) & \text{if } n > h \\ \tilde{g}_{\mathrm{II}}(a,b,v) & \text{if } b = 0 \\ \max\{\tilde{g}_{\mathrm{I}}(a,b,v),\tilde{g}_{\mathrm{II}}(a,b,v)\} & \text{otherwise .} \end{cases} \tag{4.4}$$

We postpone a pseudo-code description of LPATH to Chapter 5, where some practical improvements will be discussed.

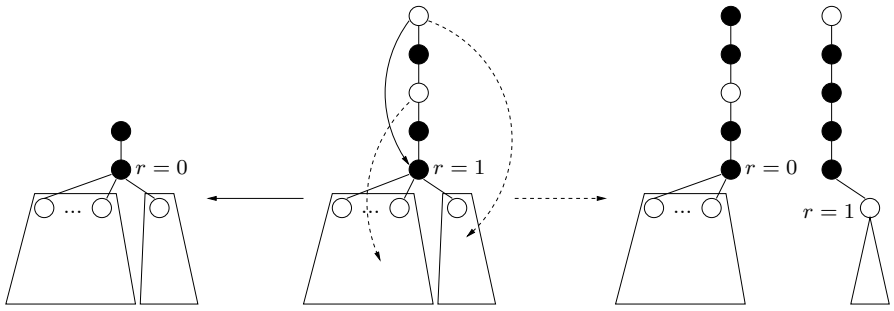


Figure 4.3: Computation scheme of LPATH for $K = 1$. Only white path nodes are allowed to have a hotchild (i.e. $a_i = 1$ for the corresponding component of a). From the subproblem depicted in the middle, the set of dashed hotlinks leads to the subproblems on the right, while the solid hotlink would lead to the subproblem on the left.

4.4.3 Resource Requirements

LPATH employs dynamic programming, maintaining a table which stores $\tilde{g}(a, b, v)$ and the corresponding selection of hotlinks for each configuration of (a, b, v). Thus, the memory requirements are $O(|V|(K+1)^h)$.

For the runtime analysis we have to sum up the number of possibilities that are compared for each subproblem. For Case II, the most comparisons are clearly performed in Case IId, and this number dominates the Case I comparisons. So in our calculation we neglect Case I and assume that Case IId applies to each node in V. For each $v \in V$, we have the runtime

$$\sum_{a \in \{0,\ldots,K\}^{h+1}} \prod_{i=1}^{h+1} (a_i + 1) = \sum_{a_1=1}^{K+1} a_1 \sum_{a_2=1}^{K+1} a_2 \ldots \sum_{a_{h+1}=1}^{K+1} a_{h+1}$$
$$= \left(\frac{(K+2)(K+1)}{2} \right)^{h+1} .$$

Therefore, the total runtime is $O(|V|(\frac{1}{2}(K+2)(K+1))^h)$. For $K = 1$, the runtime is $O(|V|3^h)$.

Theorem 4 *For any $\epsilon > 0$,* LPATH *computes a $(1+\epsilon)$-approximation in time $O(|V|(\frac{1}{2}(K+2)(K+1))^{\frac{1}{\epsilon}})$ and space $O(|V|(K+1)^{\frac{1}{\epsilon}})$.*

Proof. For a given $\epsilon > 0$, choose h such that $\frac{h}{h-1} \leq 1 + \epsilon < \frac{h-1}{h-2}$. According to Lemma 4.5, the left inequality implies that LPATH with parameter h guarantees an approximation ratio of $(1 + \epsilon)$.

For the runtime and memory analysis we assume $(1+\epsilon) = \frac{h-1}{h-2}$, as a smaller ϵ only implies weaker runtime and memory demands. Thus, $h = 2+\frac{1}{\epsilon}$ and the runtime of LPATH is in $O(|V|(\frac{1}{2}(K+1)(K+2))^{2+\frac{1}{\epsilon}}) = O(|V|(\frac{1}{2}(K+1)(K+2))^{\frac{1}{\epsilon}})$, while the algorithm uses $O(|V|K^{\frac{1}{\epsilon}})$ space. □

4.4.4 Lower Bounds

Assignments of hotlinks having a maximum length of h cannot generally achieve more than $(h-1)/h$ of the optimal assignment's gain, and they do not hold a constant approximation ratio in term of the path length. These propositions are easy to observe considering a path of sufficient length.

However, as LPATH considers all hotlink assignments satisfying a weaker restriction, it is not clear if the ratio of $h/(h-1)$ is a tight bound for the gain of that algorithm. But considering a very long path it is still possible to argue that LPATH is no constant factor approximation in terms of the path length.

Proposition 4.1 LPATH *is no constant factor approximation algorithm in terms of the path length.*

Proof. Let $T = v_1 \rightarrow \ldots \rightarrow v_m \rightarrow l$ with $\omega(l) = 1$ and $\omega(v_i) = 0$ for $1 \leq i \leq m$ be our counter-example. The optimal hotlink assignment for T achieves a path length of 1. In order to have an approximation factor of c, the greedy user's path from v_1 to l must contain at most c hotlinks. So all we have to show is that the length of any hotlink assigned by LPATH is bounded above.

Throughout the proof we drop the employment of dynamic programming in LPATH, i.e. we assume that no solutions to subproblems are stored and reused. This does not influence the computed solutions, but it makes the analysis more intuitive.

In the original description of the PATH algorithm in [PLdS04a], subproblems are not only determined by (a, b, v), but also by a path $q = q_1, \ldots, q_n$. That path contains the nodes the selected hotchildren are actually assigned to. While this notation makes the algorithm more intuitive, it is not necessary to carry q along, because the solution to the subproblem only depends on (a, b, v). Nevertheless, we assume in this proof that q is carried along. This does not alter the computed solution, but it makes it easier to identify the nodes which are represented by the components of a.

Due to the structure of T, LPATH can only choose between Case I and Case IIb. In both of these cases only one further subproblem has to be considered. This means that the computation of an assignment can be regarded

as a linear sequence of subproblems. During such a sequence, after a node has left q (because it has been cut off from q or because it has been bypassed by a hotlink), it will never enter q again. Moreover, as LPATH never assigns a hotlink that ends in a component of q, that path constitutes the greedy user path from q_1 to q_n in the assignment computed by LPATH.

Let (u, v) be a hotlink assigned by LPATH. From the point of time when u enters q until the time when (u, v) is assigned, u must remain in q. Between these two points of time, every node between u and v is entering q. The fact that the length of q never exceeds h implies that the greedy user path from u to any node between u and v has a length of at most h in the final assignment.

The unification of the user paths from u to the nodes in $\mathrm{desc}(u) \cap \mathrm{anc}(v)$ results in a tree which contains all nodes bypassed by (u, v). The degree of that tree is at most $K + 1$, and its depth is at most h, so the number of bypassed nodes is bounded above by $O((K + 1)^h)$. □

4.5 A 2-Approximation for the Path Length

In this section we develop the first constant factor approximation in terms of the path length for the 1-Hotlink Assignment Problem. We give a lower bound $p_{\min}$ for the path length of any hotlink assignment. Then we show that $p_{\min}$ is an upper bound for the detriment induced by restricting our attention to hotlink assignments that satisfy the *centipede property* (see Definition 4.2). Finally, we give a polynomial time algorithm for computing optimal centipede assignments.

Throughout this section, we denote the set of leaves of T by L.

4.5.1 A Lower Bound for the Path Length

Definition 4.1 *The function $p_{\min}$ is defined as*

$$p_{\min}(T) = \sum_{v \in V \setminus \{r\}} W(v) - \sum_{v \in V \setminus L} \max_{v' \in \mathrm{ch}(v)} W(v') \ .$$

Intuitively, $p_{\min}$ is the sum over the cumulated weights of all nodes having a heavier sibling.

Lemma 4.6 *$p_{\min}(T)$ is a lower bound for the path length that any 1-hotlink assignment can achieve.*

Proof. Let A be a hotlink assignment for T. By applying PushDown to every internal node of T in BFS-order, due to Lemma 4.2 we increase the path length by at most $\sum_{v \in V \setminus L} \max_{v' \in \mathrm{ch}(v)} W(v')$. We obtain the empty assignment, which has a path length of $\sum_{v \in V \setminus \{r\}} W(v)$. Thus, the path length of A is at least $p_{\min}(T)$. □

4.5.2 Centipede Hotlink Assignments and Trees

Definition 4.2 *A hotlink assignment A satisfies the centipede property, if and only if*

$$(u, v) \in A, u' \in \mathrm{desc}(u) \cap \mathrm{anc}(v) \Rightarrow u' \succ u'' \ \forall \text{ siblings } u'' \text{ of } u' \ ,$$

where "$\succ$" is some fixed total order of the children of a node with $W(u') > W(u'') \Rightarrow u' \succ u''$.

Intuitively, in a centipede hotlink assignment, no node having a heavier sibling is bypassed by a hotlink. Ties are broken by fixing a suitable order "$\succ$". Note that in this section "$\succ$" has to satisfy stronger constraints than in Section 4.4.

Lemma 4.7 *For any weighted tree T there exists a hotlink assignment A^c satisfying the centipede property with $p(A^c) \leq 2 \cdot p(A)$ for any assignment A.*

Proof. We show how to transform any hotlink assignment A into a centipede assignment A^c while increasing the path length by at most the factor 2. Fix some order "$\succ$" satisfying the property demanded in Definition 4.2. We show the claim by induction over the tree depth. For trees of depth 1 or less, there is nothing to show. Assume that T has a depth of n and assume that the lemma has already been shown for trees having a depth of less than n. Let r be the root of T and let u be the first child of r with respect to "$\succ$". Furthermore, let v be the node closest to r that is bypassed by the hotlink starting in r and is not the first child of its parent. If v is a sibling of u, then $W(v) \leq W(u)$, i.e. $W(v) \leq \frac{1}{2} \sum_{v' \in \mathrm{ch}(r)} W(v')$. Otherwise, $W(v) \leq \frac{1}{2} W(u) \leq \frac{1}{2} \sum_{v' \in \mathrm{ch}(r)} W(v')$ as well. We obtain the assignment A_1 by applying Split(r, v, v'') to $A \ni (r, v'')$. Consider the partition $R = \bigcup_{v' \in \mathrm{ch}(r)} T_{v', A_1} \cup T_v$ of T. We obtain A^c from A_1 by replacing $A_1|T'$ with a centipede assignment $A^c|T'$ under exploitation of the induction hypothesis for each $T' \in R$. The

path length of A^c is bounded above by the following inequality:

$$\begin{aligned} p(A) &\geq p(A_1) - W(v) \\ &= \sum_{v' \in \mathrm{ch}(r)} W(v') + \sum_{T' \in R} p(A_1|T', T') - \frac{1}{2} \sum_{v' \in \mathrm{ch}(r)} W(v') \\ &\geq \frac{1}{2} \sum_{v' \in \mathrm{ch}(r)} W(v') + \sum_{T' \in R} \frac{1}{2} p(A^c|T', T') = \frac{1}{2} p(A^c). \end{aligned}$$

□

It remains to show that an optimal HLA satisfying the centipede property can be computed in polynomial time. Assume that we want to compute a centipede assignment for a tree T. For any node u having a sibling $u' \succ u$, there can be no hotlink from an ancestor of u to a descendant of u. This fact has two implications. The first is that, when computing the hotchildren of u's ancestors, we can consider the tree T' obtained from T by deleting u's descendants and transforming u into a leaf of weight $W(u)$. The second implication is that the partial assignment $A|T_u$ can be computed independently from $T \setminus \{u\}$. So the subtrees T' and T_u can be considered separately.

By applying this observation to every node u having a sibling $u' \succ u$ (cf. Figure 4.4), we split the tree into the set of *heavy centipedes* $C_1, \ldots, C_n$, where C_i is a *centipede tree* for $1 \leq i \leq n$.

Algorithm 3: SplitIntoCentipedes

Input: T, a weighted tree rooted at r
Output: $\mathcal{C}$, a collection of centipede trees

```
1 C ← ∅
2 u ← r
3 while u is an internal node do
4     foreach non-heaviest child v of u do
5         C ← C ∪ SplitIntoCentipedes(T_v)
6         replace T_v with a leaf of weight W(v) in T
7     u ← the heaviest child of u
8 C ← C ∪ {T}
```

Definition 4.3 *A centipede tree is a tree whose internal nodes have at most one non-leaf child. Let h be the depth and r be the root of a centipede tree.*

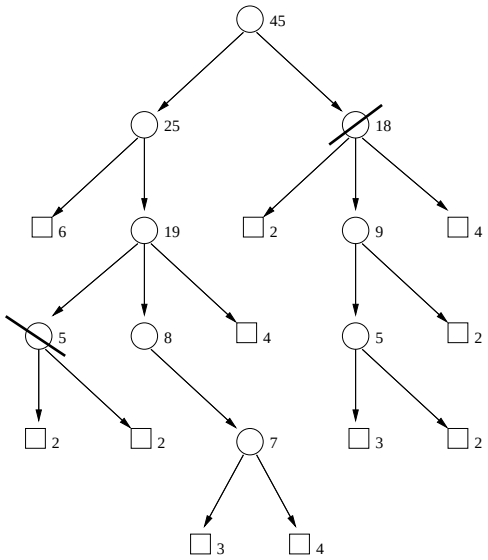


Figure 4.4: A tree is split into the set of heavy centipedes.

Then

$$\mathrm{lev}(v) = \begin{cases} h - \mathrm{dist}(v,r) & \text{if } v \text{ is an internal node,} \\ h - \mathrm{dist}(v,r) + 1 & \text{if } v \text{ is a leaf.} \end{cases}$$

The definition of lev implies that in a non-trivial centipede tree each level consists of exactly one internal node and the leaf children of that node.

In the remainder of the section we show that for the heavy centipedes $C_1, \ldots, C_n$ it is possible to efficiently compute optimal hotlink assignments $A_1, \ldots, A_n$. Then the union $\bigcup_{1 \le i \le n} A_i$ of these assignments is an optimal centipede HLA for T and, due to Lemma 4.7, holds an approximation ratio of 2.

4.5.3 Optimal Assignments for Centipede Trees

We prove two structural properties of optimal hotlink assignments for centipede trees and subsequently formulate a dynamic programming algorithm that takes advantage of these properties.

Lemma 4.8 *For any centipede tree C rooted at r there is an optimal hotlink assignment A satisfying the following property:*
$(r,l) \in A, l$ is a leaf $\Rightarrow \mathrm{lev}(l) \le \mathrm{lev}(l')$ for all leaves l' with $\omega(l') \ge \omega(l)$.

Proof. Let $A \ni (r,l)$ be an optimal hotlink assignment for C and let l be a leaf. Assume that there are leaves l' with $\omega(l') \ge \omega(l)$ and $\mathrm{lev}(l') < \mathrm{lev}(l)$.

Among those leaves, we consider one l' that is on the lowest possible level. It suffices to show that the assignment A can be transformed into a HLA containing (r, l') or (r, v) for some internal node v without decreasing the gain.

If the greedy user path length from r to l' in A is not shorter than $\text{dist}^{A\setminus\{(r,l)\}}(r, l)$, then (r, l) can be replaced with (r, l') without decreasing the gain. Otherwise, there must be a hotlink (v', v) with $v' \neq r$ that bypasses the parent of l. If there is more than one such hotlink, we consider the one where v is on the lowest possible level. So v is on the greedy user path from r to l' and $W^A(v) \geq W(l') \geq W(l)$. Thus, we can swap the hotchildren of r and v' without decreasing the gain. □

Motivated by the preceding lemma, we now extend "$\succ$" (see Definition 4.2) to be some total order of all leaves of a centipede tree satisfying

$$\Big(\omega(l) > \omega(l')\Big) \vee \Big(\omega(l) = \omega(l') \wedge \text{lev}(l) < \text{lev}(l')\Big) \Rightarrow l \succ l' \ .$$

The relation will serve as a priority with which leaves will be chosen to be the hotchild of the root. This policy is justified by the following lemma.

Lemma 4.9 *For any centipede tree C rooted at r there is an optimal hotlink assignment A satisfying the following property: If the hotchild of r is a leaf l, then $l \succ l'$ for any leaf l' with $\text{lev}(l') \leq h - 2$, where h is the depth of C.*

Proof. Let A be an optimal hotlink assignment for C satisfying the property specified in Lemma 4.8. It suffices to show that, if $(r, l) \in A$ and l is a leaf, then there is no leaf l' with $\omega(l') > \omega(l)$ and $h - 2 \geq \text{lev}(l') \geq \text{lev}(l)$. Throughout the proof we will assume that there is such a leaf l'. By an extensive case analysis we show that this contradicts the optimality of A. Let u be the internal node at level 2, and let v be the hotchild of u in A.

Case I: $\text{lev}(v) > \text{lev}(l')$. Let v' be the parent of l'. Observe that $\text{dist}^A(r, v') \geq 3$. The following modification improves A: (1) Insert (r, l'). This increases the gain by at least $2\omega(l')$. (2) If there are hotlinks bypassing v', let (u_1, v_1) be the one with the source u_1 being on the highest level. Apply $\textsc{Split}(u_1, v', v_1)$ and obtain the assignment A_2. This costs a maximum amount of gain of $W^{A_2}(v')$. (3) Apply $\textsc{PushDown}(v')$. This also costs at most $W^{A_2}(v')$. (4) Insert (r, v'). This increases the gain by at least $2W^{A_2}(v')$. In the resulting assignment, r has three hotchildren and no hotlink is leaving v'. (5) Replace (r, l) by (v', l), decreasing the gain by $\omega(l)$. (6) Remove the hotlink (r, l'), which costs $\omega(l')$. We obtain a 1-hotlink assignment and have experienced a total increase in gain of at least $\omega(l') - \omega(l)$, which is strictly positive.

Case II: $\text{lev}(v) = \text{lev}(l')$. If v is a leaf and $\omega(v) < \omega(l')$, then the assignment is improved by interchanging the hotparents of v and l'. If v is a leaf and $\omega(v) \geq \omega(l')$, we improve A by interchanging the hotparents of l and v.

Otherwise, v is the parent of l'. Then we consider the following modification: (1) Insert (r, l'), causing the gain to increase by $2\omega(l')$. Obtain the assignment A_1. (2) Apply PushDown(v), which decreases the gain by at most $W^{A_1}(v)$. (3) Replace (u, v) with (r, v) to compensate for the loss that has occurred the preceding step. (4) Remove (r, l'). This decreases the gain by $\omega(l')$. (5) Replace (r, l) with (v, l), decreasing the gain by $\omega(l)$. We obtain a 1-hotlink assignment and have experienced a total increase in gain of at least $\omega(l') - \omega(l)$, which is strictly positive.

Case III: v is an internal node with $\text{lev}(l') > \text{lev}(v) \geq \text{lev}(l)$. The following modification improves A: (1) Insert (u, l'). This increases the gain by at least $\omega(l')$. (2) Apply PushDown(v). The gain is decreased by at most $W^A(v)$. (3) Replace (u, v) with (r, v). This increases the gain by exactly $W^A(v)$. (4) Replace (r, l) with (v, l). The gain is decreased by $\omega(l)$. We obtain a 1-hotlink assignment and have experienced a total increase in gain of at least $\omega(l') - \omega(l)$, which is strictly positive.

Case IV: v is a leaf with $\text{lev}(l') > \text{lev}(v) \geq \text{lev}(l)$. Let v' be the parent of v. Consider the following modification: (1) Replace (u, v) by (r, v), increasing the gain by $\omega(v)$. (2) Apply FreeInsert(u, v'), obtain the assignment A_2. (3) Apply PushDown(v'), decreasing the gain by at most $W^{A_2}(v')$. (4) Replace (u, v') with (r, v'). This increases the gain by exactly $W^{A_2}(v')$. In the resulting assignment, r has the three hotchildren v', v and l, while no hotlink leaves u or v'. (5) Remove (r, v) from the assignment, decreasing the gain by $\omega(v)$. (6) Replace (r, l) with (v', l). This decreases the gain by $\omega(l)$. (7) Finally, insert (u, l'), which increases the gain by at least $\omega(l')$. The whole modification improves the gain by an amount of at least $\omega(l') - \omega(l)$, and the resulting HLA is a 1-hotlink assignment.

Case V: $\text{lev}(v) < \text{lev}(l)$. Let v' be the parent of l. Our modification works as follows: (1) Apply Split(u, v', v) to obtain the assignment A_1. The maximum decrease in gain is $W^{A_1}(v')$. (2) Replace (u, v') with (r, v'). This causes the gain to increase by $W^{A_1}(v')$. (3) Remove the hotlink (r, l), decreasing the gain by $\omega(l)$. (4) Insert (u, l'), causing a benefit of at least $\omega(l')$. Again, we obtain a 1-HLA and we have achieved a strictly positive total surplus of at least $\omega(l') - \omega(l)$. □

Theorem 5 *An optimal hotlink assignment for centipede trees can be computed in polynomial time.*

Proof. Let C be a centipede tree of depth h. For any leaf l and $h \geq x >$

$y \geq 1$, the tree $C[x, y, l]$ is defined as the maximum subtree of C satisfying the following properties:

(a) It contains only nodes v with $x \geq \text{lev}(v) \geq y$.

(b) It contains no leaves on level x.

(c) It contains no leaf $l' \succeq l$.

To describe the original tree we consider $C[h, 1, l_D]$, where l_D is a dummy leaf heavier than any leaf in C. From an optimal algorithm's point of view there is no difference between C and $C[h, 1, l_D]$, because hotlinks to nodes on level h would never be assigned.

For $1 \leq i \leq h$, let v_i denote the internal node at level i. When computing an optimal hotlink assignment for $C[x, y, l]$, there are two main possibilities for choosing the hotchild of that subtree's root v_x.

The first possibility is that the hotchild of v_x is a leaf. Lemma 4.9 implies that this leaf is either the first leaf $l'_{\max}$ with respect to "$\succ$" among all nodes on a level between $x-2$ and y, or it is on level $x-1$. In the latter case, for reasons of optimality, we can assume that $l_{\max}$ is the first among all leaves of its level. Lemma 4.9 further implies that $l_{\max} \succ l'$ for any leaf l' on any other level. Therefore, $l_{\max}$ is the first leaf in the whole subtree $C[x, y, l]$, see Figure 4.5 for an example. Note that in practice it will often be the case that $l'_{\max} = l_{\max}$. Under the assumption that the hotchild of v_x is a leaf, the gain of an optimal assignment for $C[x, y, l]$ is calculated as

$$G_L(x, y, l) = \max\{g(v_x, l'_{\max}) + \tilde{g}(C[x-1, y, l'_{\max}]) \\ g(v_x, l_{\max}) + \tilde{g}(C[x-1, y, l_{\max}])\} \ .$$

The second possibility is that the hotchild of v_x is an internal node. In that case the gain of an optimal assignment is

$$G_N(x, y, l) = \max_{y \leq i < x} \{ \ g((v_x, v_i), C[x, y, l]) + \\ \tilde{g}(C[x-1, i+1, l]) + \tilde{g}(C[i, y, l])\} \ .$$

The main formula for the gain of an optimal HLA for $C[x, y, l]$ is

$$\tilde{g}(C[x, y, l]) = \begin{cases} \max\{G_L(x, y, l), G_N(x, y, l)\} & \text{if } x - y \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Let n be the number of nodes in C. There are $O(n^3)$ different configurations of (x, y, l). Thus by dynamic programming we can compute an optimal hotlink assignment for C with space requirements $O(n^3)$. For each configuration of (x, y, l) we have to compare $O(n)$ different possibilities and thus the overall runtime is $O(n^4)$. □

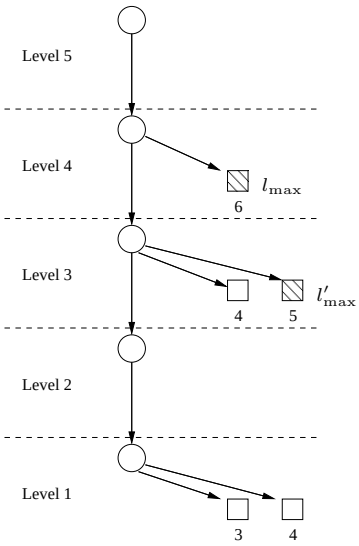


Figure 4.5: One of the centipede trees resulting from the application of SplitIntoCentipedes to the tree in Figure 4.4

Corollary 4.1 *There is a polynomial time 2-approximation algorithm in terms of the path length for the* 1*-hotlink assignment problem.*

Proof. The claim follows directly from Lemma 4.7 and Theorem 5. □

4.5.4 Lower Bound

By restricting HLAs to those satisfying the centipede property we can loose an unbounded percentage of the maximum gain. Consider a tree rooted at r, where $\mathrm{ch}(r) = \{u_1, u_2\}$, $ch(u_1) = l_0$ and $ch(u_2) = \{l_1, \ldots, l_n\}$. The weight function is defined such that $\omega(l_0) = 1$, $\omega(l_i) = (1+\epsilon)/n$ for $i = 1, \ldots, n$, and the internal nodes' weights are zero. The ratio between the gain of the best HLA and the gain achieved by the best centipede HLA converges to ∞ for large n and small ϵ.

4.5.5 Generalization to $K > 1$?

The results of this section are not directly generalizable to the model where nodes can have up to K hotchildren. The lower bound $p_{\min}$ would have to be generalized such that it is the sum over the weights of all nodes, except for the K heaviest children of each node. So for our 2-approximation, the tree would have to be decomposed into a generalized form of centipedes, where

Algorithm 4: C-Opt

Static: C, a centipede tree; $\mathcal{T}$, the dynamic programming table
Input: $[x, y, l]$, defining a subtree of C
Output: A, a hotlink assignment for $C[x, y, l]$

```
1 if T contains solution for [x, y, l] then
2     A ← T[x, y, l]
3 else
4     recursively compute the hotlink assignment A achieving the gain
      specified in Equation 4.5
```

Algorithm 5: Centipede

Input: T, a weighted tree
Output: A, a hotlink assignment for T

```
1 C ← SplitIntoCentipedes(T)
2 A ← ∅
3 foreach C ∈ C do
4     initialize the static variables in C-Opt according to C
5     A ← A ∪ C-Opt([x, y, l]), where C[x, y, l] = C
```

each internal node can have up to K non-leaf children. Computing optimal hotlink assignments for such trees seems to be much more difficult than it is for standard centipede trees.

4.6 Remarks

In this chapter we have presented three approximation algorithms for the Hotlink Assignment Problem. Each of these algorithms guarantees an at least constant approximation factor either for the path length or for the gain. We have also seen that each algorithm only approximates one of the performance measures up to a constant ratio.

We remark that none of the analyses can be generalized to the model considered in Chapter 3, where the maximum number of hotchildren is arbitrarily determined by a function $K : V \to \mathbb{N}_0$. Even when $K : V \to \{0, 1\}$, the operations PushDown, FreeInsert, and Split cannot be applied. The approximability of that more general problem remains open.

Chapter 5

An Experimental Study

In the preceding two chapters we have proposed a number of polynomial time hotlink assignment algorithms, and we have proven worst case bounds for the quality of their solutions. In this chapter we focus on their response to inputs that occur in practice.

We report on an extensive experimental study performed using a test bed of real-world and realistic random instances. The real-world instances represent the structure of large university web sites containing up to several hundred thousands of pages. The synthetic instances have been generated by a new random construction method based on a technique proposed by Barabási and Albert [BA99]. We evaluate solution quality and runtime of the algorithms, and draw conclusions about which strategies are recommendable in practice.

Besides the hotlink assignment strategies we have presented in Chapter 3 and 4, our experimental setup also includes two methods recently proposed by Douïeb and Langerman [DL05, DL06]. Those algorithms are tailored to compute hotlink assignments whose path length is close to the entropy (cf. Section 1.4.2 and 1.4.5). Moreover, we propose a new heuristic method that has turned out to compute near-optimal solutions to all test instances in our experiments.

Not all algorithms evaluated in this chapter are designed for the K-Hotlink Assignment Problem. Therefore, we mainly concentrate on the model where only one hotlink is allowed to start in each node. Recall that this problem version is already NP-hard (cf. Chapter 2). For the same reason, our test bed consists of instances where only the leaves carry non-zero weights.

As optimal solutions can be computed in time exponential in the tree depth [GKL$^+$07], and this depth is typically not too large, we are in the lucky situation of being able to compute optimal hotlink assignments for a

large subset of our test instances. Therefore, one main focus of our study is on approximation ratios that are achieved in practice.

This chapter is organized as follows: In Section 5.1 we give a description of all algorithms included in our study, except for those that have already been described in the preceding chapters. We also discuss implementation issues here. In Section 5.2 we specify our experimental setup, including the test bed and the performance measures. Section 5.3 reports and interprets the results of our study, and Section 5.4 concludes the chapter.

5.1 Algorithms

In this section we recapitulate all algorithms that are are included in the experimental study. Wherever our implementation is not straightforward, we also discuss that issue.

5.1.1 Notation

Before beginning to specify the algorithms, we introduce some additional notation simplifying their description.

The relation "$\succ$" is defined, like in Definition 4.2, as a total order among siblings such that $\omega(u) > \omega(u') \Rightarrow u \succ u'$ for any pair u, u' of siblings. Ties are broken arbitrarily. The direct successor sibling of u with respect to that order is denoted by $\mathrm{succ}(u)$. The *first child* $\mathrm{fc}(u)$ is the unique child of u having no predecessor.

Next, we define the *heavy path* of a node u (cf. [DL05]). If u is a leaf, then its heavy path only consists of u. Otherwise, it is the path obtained by appending the heavy path of $\mathrm{fc}(u)$ to u.

The term *top-down method* term has been introduced in [DL06] and denotes hotlink assignment algorithms that assign a hotlink (r, v) and then recursively apply themselves to T_v and all $T_u \setminus \{v\}, u \in \mathrm{ch}(r)$. Thus, any top-down method is fully characterized by the choice of v.

5.1.2 Greedy

Greedy is specified in Algorithm 2, Chapter 4.3. Observe that it can also be formulated as a top-down method. It has exhibited the best performance among the approximation algorithms studied experimentally so far [CKK+01, CKK+03, PLdS04b, GKL+07].

5.1.3 H/Ph

The H/Ph-strategy is also a top-down method. Let h be the node whose weight is closest to $\omega(r)/2$. If $\omega(h) > \alpha\omega(r)$, then h is chosen as the hotchild of r. Otherwise the parent p_h of h becomes r's hotchild. The threshold α is given as the solution of $(\frac{\alpha}{1-\alpha})^{2(1-\alpha)} = \alpha$, i.e. $\alpha \approx 0.2965$.

The H/Ph-algorithm has been proposed in [DL06], where it is proved to guarantee a path length of at most $1.141H(\omega) + 1$. Thus it is asymptotically a $(1.141 \log(\Delta + 1))$-approximation in terms of the path length, where Δ is the outdegree of the tree (cf. Section 1.4.2).

In [DL06] the authors propose an implementation of H/Ph that runs in worst case time $O(n \log n)$. They observe that it suffices to traverse the root's heavy path when looking for h. As this path can have a length of $O(n)$ in the worst case, an involved tree representation is employed for finding h in time $O(\log n)$. We do not adopt the tree representation in our implementation, because in typical tree instances, including our test set, the depth is rather small.

5.1.4 PMin

Assume that there was an oracle telling the exact minimum path length $p_{\text{opt}}(T)$ a hotlink assignment can achieve for a given tree T. Then an optimal assignment could be easily computed by a top-down method that chooses the hotchild v of r such that

$$p_{\text{opt}}(T_v) + \sum_{u \in \text{ch}(r)} p_{\text{opt}}(T_u \setminus \{v\})$$

is minimized. Due to Chapter 2, such an oracle needs superpolynomial answer time unless P=NP.

Our algorithmic idea is to guess p_{opt} by some fast estimation method. After experimenting with a number of such methods, it turned out that the usage of $p_{\min}$ from Definition 4.1 produces excellent results and is very robust. The results are even improved if we employ

$$p'_{\min}(T) = p_{\min} + \max_{v \in \text{ch}(r)} W(v) \ .$$

For an efficient implementation, instead of explicitly calculating $p'_{\min}$ we evaluate the *improvement* to $p'_{\min}(T)$ caused by the hotlink (r, v). For determining that improvement it suffices to traverse the path between v and r.

5.1.5 HeavyPath

HeavyPath has been proposed in [DL05]. It works as follows: First split the tree into the set of heavy paths. This can be done in linear time by recursively computing the set of heavy paths of the subtrees rooted at the children of r, joining these sets, and appending r to the path containing the first child of r. Then interpret each of these paths as a list of weighted elements. The weight $\mathcal{W}(u)$ of each such element u is $W(u) - W(\text{fc}(u))$. If u is a leaf, then $\mathcal{W}(u) = \omega(u)$.

A hotlink assignment for each such list $u_1, \ldots, u_n$ is computed as follows: Assign a hotlink (u_1, u_i) such that $\sum_{1 \le j < i} \mathcal{W}(u_j)$ and $\sum_{i < j \le n} \mathcal{W}(u_j)$ are both at most $\frac{1}{2} \sum_{1 \le j \le n} \mathcal{W}(u_j)$, and recursively apply the algorithm to the sublists $u_2, \ldots, u_{i-1}$ and $u_i, \ldots, u_n$. Using exponential search for determining u_i, the hotlinks can be assigned in time $O(n)$.

Therefore, the heavy path strategy has the lowest worst case runtime among all strategies evaluated in this chapter. It guarantees a maximum path length of $3H(\omega)$, i.e. it is a $(3\log(\Delta + 1))$-approximation in terms of the path length.

5.1.6 Lpath

The approximation scheme Lpath has been presented in Section 4.4.2. Recall that for a given value of h it computes a hotlink assignment that is at least as good as the best length h assignment. In our experiments, we also use Lpath as an optimal algorithm by setting h to the tree's depth.

We discuss a number of implementation issues using the terminology of Section 4.4.2. Note that there the order "$\succ$" has been arbitrary, so the definition introduced at the beginning of this chapter constitutes a possible implementation.

The dynamic programming approach for the algorithm has been adopted from the PATH algorithm of Pessoa, Laber, and Souza [PLdS04b]. The only difference to Lpath is that, whenever a component a of a subproblem (a, b, v) represents a path longer than h, PATH gives up.

In [PLdS04b] Pessoa et. al. have proposed two improvements to their algorithm. The first is to increase the number of considered hotlink assignments by always cutting the first component off from a until $a_1 = 1$. That improvement is already included in the original definition of Lpath, as the latter strategy always cuts the first components off from q and a when they become too long.

The second improvement is based on the observation, that, in the optimal solution to any subproblem (a, b, v), the total number of hotlinks bypassing

v is limited by the number of leaves in T^v. To our knowledge, a proof of that claim has not yet been published, neither in [PLdS04b] nor in the journal version of the paper [GKL+07]. Therefore, for reasons of completeness, we give a proof at this point.

Lemma 5.1 *For any subproblem (a, b, v) with T^v containing m leaves, there is an optimal solution where at most m hotlinks are bypassing v.*

Proof. For each leaf $l \in T^v$ there is at most one hotlink on the greedy user path to l that is bypassing v. Therefore, there are at most m hotlinks that bypass v and shorten the path to a leaf in T^v. Any other hotlink bypassing v is obsolete and can be removed without increasing the path length. □

Note that this version of the lemma only holds in the model where only the leaves carry weights. If also internal nodes have weights, then the number of hotlink bypassing v is bounded by the total number of nodes in T^v.

Let $\hat{a} = \sum_{1 \le i \le |a|} a_i$ and let m be the number of leaves in T^v. We adopt the second improvement by inverting the $\hat{a} - m$ components of a having the highest possible indices from 1 to 0 whenever $\hat{a} > m$.

We have observed in our experiments that the memory requirements of LPATH are by far more critical than the runtime is. For all tree instances the algorithm either terminated in reasonable time (2-3 minutes or less), or the memory requirements exceeded our hardware limit. So the purpose of our main improvement is to reduce space consumption.

The total number of subproblems considered by LPATH is $\Theta(|V|2^h)$. Fortunately, we do not need to store all of them simultaneously. For any fixed node v, let S_v be the set of solutions for all possible values of (a, b, v). Then $S_{\mathrm{succ}(v)}$ and $S_{\mathrm{fc}(v)}$ suffice to compute any element of S_v. Furthermore, $S_{\mathrm{succ}(v)}$ and $S_{\mathrm{fc}(v)}$ are not required for any further computation, so these sets can be removed from memory after computing S_v.

Let $v_1, \ldots, v_n$ be the unique postorder sequence of $V - \{r\}$ where each node appears before its successor with respect to "$\succ$". We successively compute $S_{v_1}, \ldots, S_{v_n}$ and delete any set as soon as it is not required anymore. It is easy to observe that with this policy any solution of a subproblem is guaranteed to be available when it is required. The solution to the original problem is contained in S_{v_n}. The following lemma bounds the number of solution sets that are simultaneously stored.

Lemma 5.2 *Let d be the depth of the tree. At any time during the execution of* LPATH*, for $1 \le x \le d$, there is at most one node v with $\mathrm{dist}(r, v) = x$ whose solution set S_v is currently stored and not currently computed.*

Proof. Let v and v' be two nodes that have the same distance to the root and let v' occur after v in the postorder sequence. Let $S_{v'}$ be currently stored and completely computed. If v and v' are siblings, then S_v has been used for the computation of $S_{v''}$, where v'' is either v' or another sibling between v and v', so S_v has already been deleted. If v and v' are not siblings, then the parent u of v occurs in the postorder sequence before v'. In that case S_u has already been computed, which implies that S_v has been used and thus deleted. □

As only one solution set is computed at a time, it follows directly from Lemma 5.2 that at most $d+1$ solution sets are simultaneously stored and thus the memory requirements of our implementation are $O(d2^h)$. The improvement is significant in practice, as the depth is typically small compared to the number of nodes (cf. Table 5.1).

Algorithm 6 outlines our implementation of LPATH. Remark that the pseudo-code description applies to the model considered in this chapter, where each node can have at most one hotchild and only leaves carry non-zero weights.

Algorithm 6: LPATH

Input: T, a weighted tree rooted at r; h, an integer
Output: A, a hotlink assignment for T

```
1  v_1, ..., v_n ← postorder sequence of T − {r} with
   v_i ≺ v_j ∧ par(v_i) = par(v_j) ⇒ i < j
2  for v = v_1, ..., v_n do
3      h' ← min{dist(r, v), h}
4      m ← the number of leaves in T^v
5      foreach a ∈ {0,1}^{h'+1}, b ∈ {0,1} do
6          a' ← a
7          â' ← Σ_{1≤i≤h'+1} a'_i
8          if â' > m then
9              invert the â' − m components of a' having the highest
               possible indices from 1 to 0
10         compute solution for (a', b, v) according to Equation 4.4,
           cutting the first component of any a'' ∈ {0,1}^{h+2} that might
           occur in a subproblem before looking up the solution
11         store solution at table position (a, b, v)
12     delete any solution for (·, ·, succ(v)) and (·, ·, fc(v)) from the table
```

5.1.7 Centipede

The Centipede algorithm is completely specified in Section 4.5. We have implemented it as outlined in Algorithm 3, 4, and 5.

5.1.8 L-Opt

Algorithm L-Opt is the main result of Chapter 3. It computes an optimal hotlink assignment under the restriction that hotlinks only end in leaves.

Implementing Algorithm 1 in a straightforward manner would result in $\sum_{v \in T} K(v)$ calls of the method PullUp, which is described in the proof of Theorem 2. This would mean that the dynamic programming table for the values of $s(v)$ (Equation 3.2) has to be rebuilt $\sum_{v \in V} K(v)$ times from scratch.

Instead, we maintain one table throughout the execution of the algorithm, updating it each time PullUp is executed.

Recall the definitions given in Section 3.2. Let A be optimal for (T, K) and let $(l_1, \ldots, l_n)$ be an optimal ordered transformation sequence for A into K_{+r}. Let v_n be the hotparent of l_n in A, or, if l_n has no hotparent in A, let v_n be the parent of l_n. Any hotparent of a component l_i of the sequence is an ancestor of v_n.

Consider any node v. Recall that $s(v)$ represents the path length reduction achieved by an optimal ordered transformation sequence for $A|T_{v,A}$ into K_{+v}, and this sequence only depends on $T_{v,A}$ and $A|T_{v,A}$. So after applying the transformation sequence $(l_1, \ldots, l_n)$, the value of $s(v)$ has to be updated only if a component l_i from the sequence is contained in $T_{v,A}$. This is the case only if $v \in \text{anc}(v_n)$.

So, for correctly updating the table, it suffices to consider the table entries corresponding to the path from v_n to r. The following lemma further reduces that set.

Lemma 5.3 *When updating a value of $s(v)$, that value never increases.*

Proof. In this proof, the assignment A in Equation 3.1 and 3.2 will not always be clear from the context, so we will write $s(v, A)$ and $s(v, l, A)$.

We prove the lemma by induction over the depth of T_v. The basic case is trivial. Let A^+ be the assignment obtained from A by applying the transformation sequence. If the lemma does not hold, then $s(v, l', A^+) > s(v, l', A)$ for the leaf l' maximizing Equation 3.2 with respect to A^+. This is only possible if l' has a hotparent v' in A^+. If l' has the same hotparent in A, then $s(v')$ must have increased, which is not possible due to the induction hypothesis. Therefore, l' is contained in the transformation sequence, which means that

v' has a hotchild $l \neq l'$ in A which is the predecessor of l' in the sequence. As the sequence is ordered, Lemma 3.1 implies $\omega(l) \geq \omega(l')$. Thus, $s(v, l, A) = \text{dist}(v', v)\omega(l) + s(v', A) \geq \text{dist}(v', v)\omega(l') + s(v', A^+) = s(v, l', A^+)$. As l' maximizes $s(v, l', A^+)$, we have $s(v, A) \geq s(v, A^+)$, a contradiction. □

When updating a value of $s(v)$, we first look up the leaf l maximizing Equation 3.2 from the table. If $s(v, l)$ has not decreased, neither has $s(v)$. Since due to Lemma 5.3 that value also never increases, there is no need to update then.

Our implementation of L-Opt is described in Algorithm 7. The recursive calls are performed in Line 4. The dynamic programming table $\mathcal{T}$ is built during the recursive execution of the algorithm, so Line 9 is a simple table lookup. After the transformation sequence has been applied to A, the dynamic programming table is updated in Line 16-20. Note that here, for any descendant v' of the current u_i, the updated value of $s(v')$ is already available in $\mathcal{T}$. The main computational effort of the algorithm is required for executing Line 20, as here the complete subtree $T_{u_i,a}$ has to be scanned.

5.2 Experimental Setup

5.2.1 Real Instances

Our set of real instances consists of 104 trees. 84 of them represent the structure of Brazilian and U.S. university sites and have been made available by the authors of [GKL+07]. Access patterns measured over the time horizon of one week are available for one of those instances, namely `puc-rio.br`. We have extended the set by 20 instances representing web sites of German universities. All trees have been extracted from the corresponding sites using breadth-first search, which implies that, for any page v in the original structure, a shortest path from the home page to v is the unique path from r to v in the resulting tree.

We note that the German university instances, unlike the others, partially have a large depth up to 179. These kinds of trees are especially hard to handle by optimal algorithms. Typically, very deep subtrees correspond to online tutorials or image series that do not have an index page. Table 5.1 shows the main characteristics of our test set.

As only the access frequency distribution of one tree instance is available to us, we have assigned weights to the leaves of the others randomly. We did so using Zipf distribution, i.e., the ith heaviest leaf, which is chosen uniformly at random, is assigned a weight of $\frac{1}{iH_m}$, where H_m is the mth Harmonic number and m is the number of leaves. Zipf distribution is considered as

Algorithm 7: L-Opt (implementation)

Input: (T, K), a problem instance
Output: A, a hotlink assignment for (T, K)
Static: $\mathcal{T}$, the dynamic programming table, storing a value $s(v)$ and the leaf l maximizing $s(v, l)$ for each $v \in T$

```
1  A ← ∅
2  r ← the root of T
3  foreach v ∈ ch(r) do
4      A ← A ∪ L-Opt(T_v, K)
5  repeat K(r) times
6      transformation sequence t ← ()
7      v ← r
8      while T_{v,A} contains a leaf do
9          append the leaf l maximizing Equation 3.1 to t
10         if l has a hotparent v' in A then
11             v ← v'
12         else
13             break loop
14     apply transformation sequence t to A
15     u_1, ..., u_n ← path from r to the last component of t
16     for i = n, ..., 1 do
17         l ← the leaf stored in T as the leaf maximizing s(u_i, l)
18         if s(u_i) < s(u_i, l) then
19             find leaf l' maximizing s(u_i, l') in subtree T_{u_i,A}
20             store l' together with the new value of s(u_i) in T
```

Table 5.1: Characteristics of the set of real tree instances

	min	max	average
size	32	512484	32652
depth	179	3	16

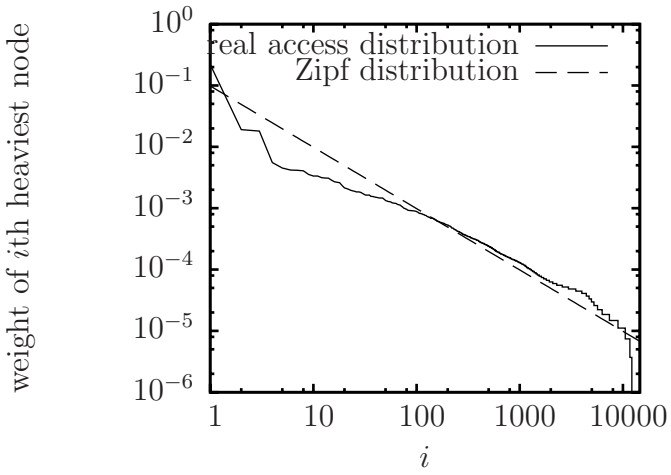


Figure 5.1: Real and synthetic access frequency distribution

the typical access distribution, see e.g. [Pit99]. The same method has also been employed in [CKK+01, PLdS04b, GKL+07]. Figure 5.1 confirms the validity of that approach by comparing the access frequency distribution of `puc-rio.br` with Zipf distribution.

5.2.2 Synthetic Instances

Making reliable statements about the algorithms' behavior for different input sizes requires a large number of test instances. The available set of real trees it not sufficient for that purpose. In [PLdS04b], Pessoa et. al. increase the number of instances by considering each subtree of minimum depth 3 that is rooted at a node in one of their original trees. This approach causes strong dependencies in the data set and is therefore problematic in our opinion.

Instead, we randomly generate a large number of synthetic trees. Czyzowicz et. al. [CKK+03] have observed that the outdegree Δ of internal nodes follows a power-law, i.e. $p[\Delta = i] \sim i^{-k}$. They have chosen the exponent $k = 2.72$ from literature about the graph structure of the World Wide Web. Their proposed algorithm maintains a FIFO-queue where all new nodes are stored. In each iteration step a node is removed from that queue and is

assigned i new nodes as its children, where i is chosen at random according to the abovementioned power-law distribution. The algorithm terminates as soon as the desired number of nodes has been generated.

In our opinion there are two problems with that method. The first is that the leaves of any possible output tree are basically all on the same level, which is a consequence of the breadth-first and top-down manner of the construction. The second problem is that, according to our given data sets, the chosen value of k does not seem to be adequate. Maximum-likelihood-estimation of that parameter (see e.g. [GMY04]) based on our real instances resulted in $k \approx 1.78$. This value is also problematic as for $k \leq 2$ the expectation of the number of children is infinity, which is unrealistic for trees having a limited number of nodes.

We have therefore developed a new algorithm that is based on a model of Barabási and Albert [BA99]. In their model a graph, initially containing a small number of m_0 nodes, is built by adding a new node v_i adjacent to m existing nodes in each iteration step $i = 1, \ldots, n$. The probability for any node v to be chosen as one of v_i's neighbors is proportional to one plus the number of nodes already adjacent to v. The authors show that for large n the number of neighbors of a node converges against a power-law distribution. In our case $m_0 = m = 1$, so that the resulting graph is a tree. Unlike the algorithm of Czyzowicz et. al., our construction generates trees with leaves at all levels.

The data set generated for our experiments consists of trees having sizes $1000, 2000, \ldots, 100000$. For each size ten instances have been generated, so their total number is 1000. Like for the real web trees, the weights of the leaves have been generated using Zipf distribution.

In these synthetic trees the distribution of the nodes' outdegree is similar to the typical distribution in real instances, as Figure 5.2 shows. Figure 5.3 displays the depth of the generated trees. It turns out that the depth tends to grow only very slowly with the size.

5.2.3 Test Environment

We have run all algorithms described in Section 5.1 on both the real and the synthetic tree instances. In case of LPATH we have applied each configuration of $h = 2, \ldots, 15$. Computations that required more than one hour or exceeded a RAM limit of 500MB have been aborted.

We have measured the path length of the resulting hotlink assignments as well as the runtime of the algorithms. Based on the path length we have calculated a number of additional values. As defined in Section 1.2.4, the gain $g(A)$ of an assignment A is $p(\emptyset) - p(A)$. The main focus of our study lies on

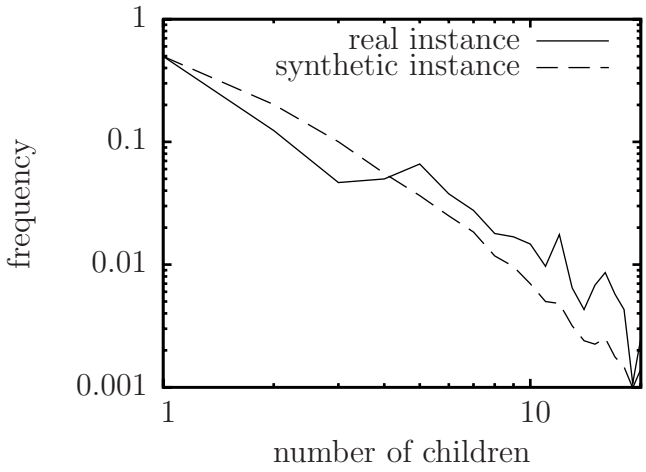


Figure 5.2: Distribution of the number of children in real and synthetic instance

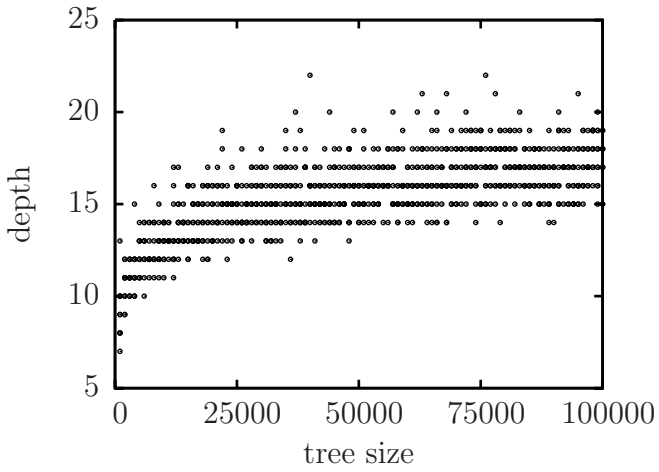


Figure 5.3: Relation between size and depth in the synthetic trees

the approximation ratios that occur in practice. Therefore, for all instances where an optimal solution OPT could be computed by our implementation of LPATH, we have calculated $p(A)/p(\text{OPT})$ as well as $g(\text{OPT})/g(A)$.

The algorithms and the test environment have been programmed in Java. The experiments have been conducted on an Intel Pentium 4 at 2,53 GHz with 1 GB of RAM, Fedora Linux 5 Version 2.6.15-1.2054_FC5, and Java HotSpot Client Virtual Machine (build 1.5.0_10-b03). The heap size limit of the Virtual Machine has been adjusted to 500MB.

5.3 Results

Our implementation of LPATH returns optimal solutions for all but 16 of the real instances. This improves upon the implementation of an optimal solution proposed in [PLdS04b], since we have been able to solve some of the instances that had remained open in that work. However, in the journal version of that paper [GKL+07] those instances have also been solved. The authors of the latter report about two hard instances that require 488MB and >1GB of RAM for being solved by the dynamic programming algorithm. Our experience with these two instances is nearly the same.

Our remaining unsolved instances have a depth of between 14 and 179 and a size of between 74042 and 512484. Due to their smaller depth (cf. Figure 5.3), optimal solutions for all but one synthetic instances could be computed within our resource bounds.

5.3.1 Solution Quality

Table 5.2 gives an overview of the approximation ratios the algorithms have achieved on the real and random instances[1], and compares these ratios to the theoretical upper bounds. For H/PH and HEAVYPATH no bound in terms of the gain has been given yet. However, trees having a depth of 2 to which no hotlink is assigned at all by these strategies are easy to construct, so their ratios in terms of the gain do not exist. As we can see in the table, this scenario even occurs in practice. Also L-OPT, used as a heuristic in the model where any node can be a hotchild, guarantees no constant approximation ratio. A counterexample is a long path whose last node has a large number of leaf children of equal weight.

Although the results achieved for the real and random trees are not identical in terms of absolute values, the relative performance ranking of the

[1]Of course only those instances are considered in the table where an optimal hotlink assignment is available.

	approximation ratios in terms of the gain						
	worst	real instances			random trees		puc-
	case	max	average	deviation	avg.	dev.	rio.br
Greedy	2	1.351	1.029	0.066	1.046	0.034	1.283
PMin	?	1.094	1.008	0.018	1.010	0.010	1.000
H/Ph	∞	∞	∞	∞	1.772	0.265	2.966
HeavyPath	∞	∞	∞	∞	1.731	0.272	2.977
Centipede	∞	2.041	1.138	0.193	1.084	0.026	1.064
L-Opt	∞	2.523	1.222	0.263	1.101	0.077	1.558
Lpath, $h = 2$	2	1.530	1.128	0.136	1.205	0.056	1.113
Lpath, $h = 3$	1.5	1.096	1.013	0.023	1.040	0.022	1.000
Lpath, $h = 4$	4/3	1.047	1.001	0.006	1.005	0.006	1.000
	approximation ratios in terms of the path length						
	worst	real instances			random trees		puc-
	case	max	average	deviation	avg.	dev.	rio.br
Greedy	∞	1.062	1.009	0.013	1.025	0.018	1.051
PMin	?	1.052	1.004	0.009	1.006	0.006	1.000
H/Ph	$O(\log \Delta)$	1.460	1.167	0.078	1.241	0.034	1.154
HeavyPath	$O(\log \Delta)$	1.488	1.195	0.077	1.233	0.040	1.155
Centipede	2	1.148	1.036	0.022	1.043	0.010	1.014
L-Opt	∞	1.547	1.096	0.122	1.052	0.040	1.084
Lpath, $h = 2$	∞	1.372	1.069	0.090	1.100	0.035	1.024
Lpath, $h = 3$	∞	1.090	1.010	0.019	1.023	0.014	1.000
Lpath, $h = 4$	∞	1.030	1.001	0.004	1.003	0.004	1.000

Table 5.2: Theoretical worst case approximation ratios and experimentally observed ratios

algorithms is consistent. The deviation among the real instances is significantly larger. We suppose that this is due to the higher variability of the tree depth.

Concerning the path length, H/Ph and HeavyPath achieve approximation factors less than 1.5 on all instances. However the ratios of all other algorithms are better for both ratios. Besides the PTAS, the best results are achieved by PMin, which outperforms its competitors in terms of approximation factors. It is only beaten by Lpath for $h \geq 4$. Like observed in previous experiments on hotlink assignment, Greedy also achieves comparatively good results. It is by far better than Lpath for $h = 2$, which has the same worst case approximation ratio as Greedy and simulates the behavior of the 2-approximation given in [MP07]. Centipede, being a 2-approximation in terms of the path length, computes solutions of higher quality than H/Ph and HeavyPath, but is not that excellent as Greedy or even PMin.

The rows corresponding to L-Opt show that one sacrifices not too much by restricting hotlinks to end in leaves, even though there are no constant worst case bounds for its approximation ratios. In practice, the algorithm achieves ratios between those of Centipede and HeavyPath. The deviation of the solution quality, however, is comparably high.

The results with respect to `puc-rio.br` can be found in the rightmost column of Table 5.2. They confirm all observations described in this paragraph.

Our experiments based on the random instances allow for some more detailed findings. The average gain and the standard deviation for varying tree size is visualized in Figure 5.4. The figure shows exemplarily for the gain of Greedy that the algorithms' solution quality is basically independent from the tree size. The picture looks similar for all algorithms and types of ratios.

The relative performance ranking observed from Table 5.2 is confirmed by the histograms in Figure 5.5, 5.6, and 5.7. For almost all instances, PMin is less than 5% away from the optimal solution's path length (Figure 5.5). The ratio of Greedy is better than 1.1 on most instances. The same holds for Centipede, but the majority of its solutions are clearly worse than the solutions computed by Greedy. The quality of L-Opt's solutions is distributed among a comparably wide range, being sometimes nearly optimal and somtimes far-off from its competitors. The ratios of algorithms H/Ph and HeavyPath are between 1.15 and 1.35. They are not included in the histograms for reasons of readability.

All these phenomena can also be observed for the gain (Figure 5.6), although all ratios are typically higher here. The difference between PMin

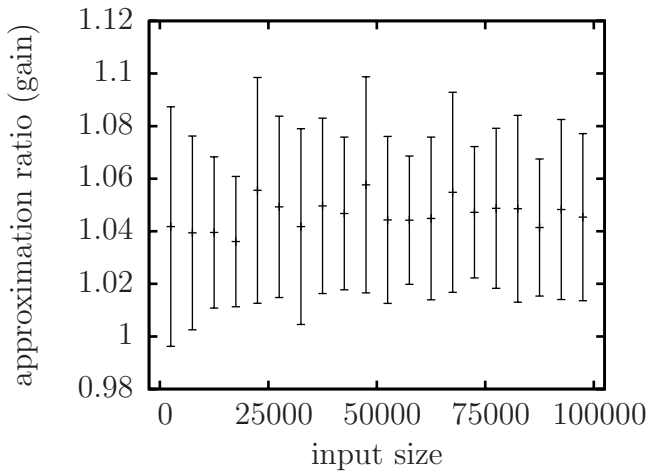


Figure 5.4: Ratio (gain) of GREEDY and PMIN for different tree sizes

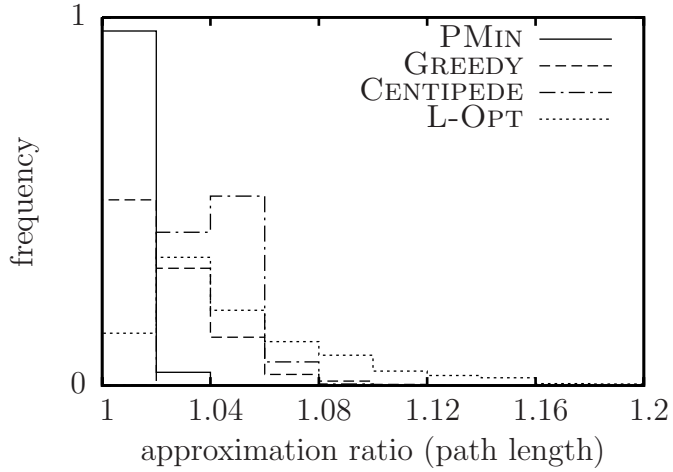


Figure 5.5: Histograms of the approximation ratio in terms of the path length

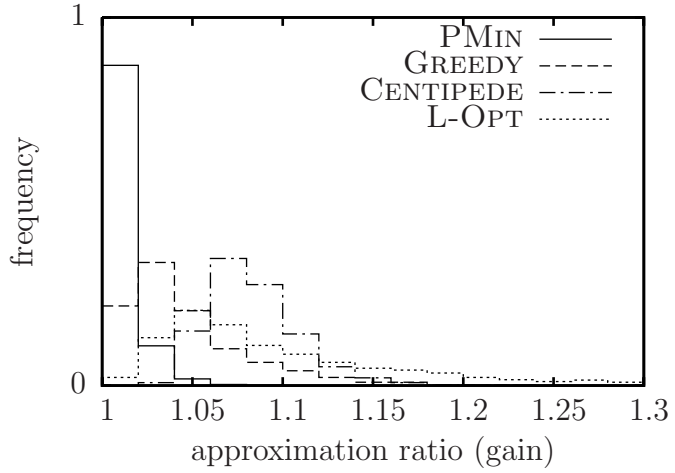


Figure 5.6: Histograms of the approximation ratio in terms of the gain

and Greedy is even more pronounced, and the ratios for the gain of H/Ph and HeavyPath are mainly distributed between 1.3 and 2.2.

Figure 5.7 visualizes the behavior of Lpath for different values of h. Only for $h > 3$ Lpath performs better than PMin. In Figure 5.8 the influence of parameter h is visualized. On each of the four selected instances of real trees, the approximation ratio converges to 1 much faster than in theory. Typically, the speed of convergence is lower for trees having a greater depth d, but there are exceptions. Recall that for $h \geq d$ a ratio of 1 is guaranteed.

5.3.2 Runtime

The runtime of Lpath, as expected, grows exponentially with h, up to some characteristic value (Figure 5.9). For h greater than that value the runtime remains constant. The characteristic value depends on the tree instance. It is typically slightly smaller than the tree's depth, when there are only few possibilities for length h hotlinks.

The runtimes for processing the synthetic instances by the other algorithms are depicted in Figure 5.10, 5.11, and 5.12. Apparently, for each algorithm the runtime values form a point cloud that is quite compact, i.e. the runtimes are reliable. The only exception is the Greedy, having many outliers above it typical runtime. Note that the order of magnitude differs among the runtime scales in the figures. Centipede is by far the slowest algorithm, and it is also the only method whose runtime is clearly superlinear in practice. It is followed by algorithms PMin, H/Ph, and Greedy.

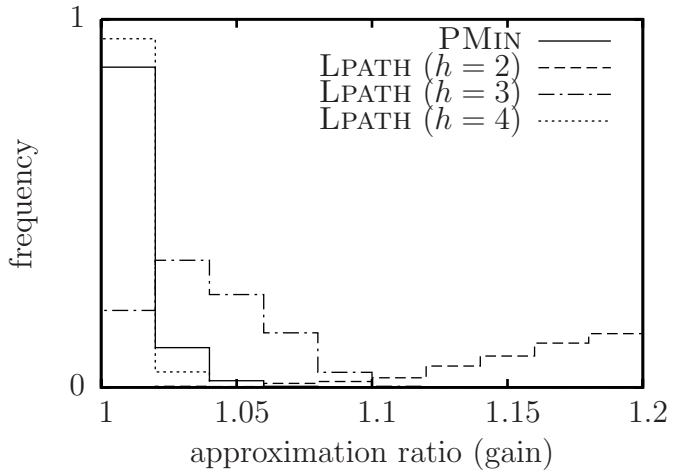


Figure 5.7: Histograms of the approximation ratio in terms of the gain for Lpath.

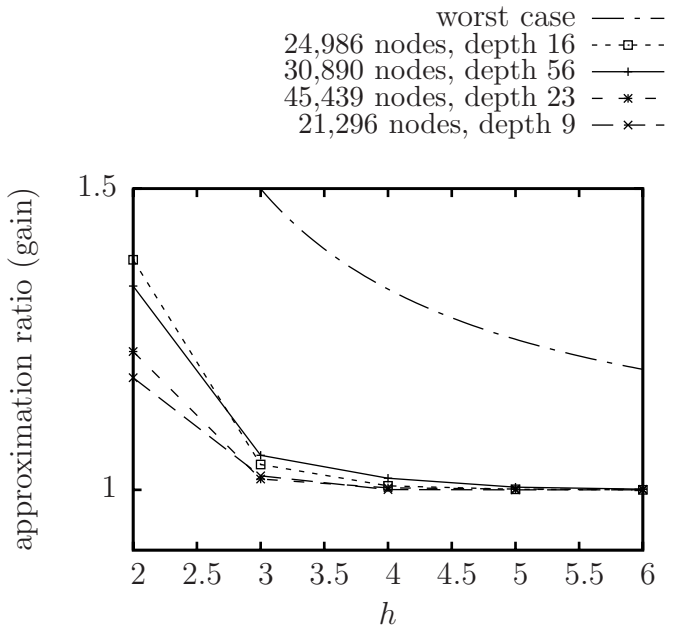


Figure 5.8: Approximation ratio resulted from running Lpath on tree instances for different values of h

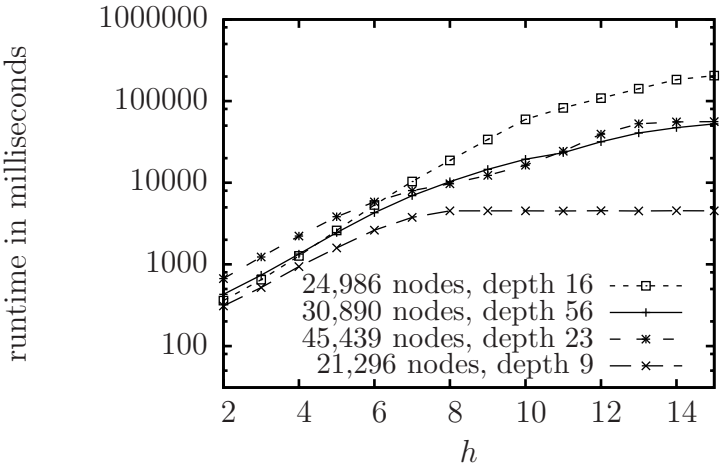


Figure 5.9: Runtime resulted from running LPATH on tree instances for different values of h

Surprisingly, our implementation of L-OPT is much faster than all top-down algorithms. The fastest method is HEAVYPATH. This also is somewhat surprising, as from its description it seems more complicated than, for example, H/PH. On the other hand, HEAVYPATH is the only method having linear worst case runtime.

5.3.3 Investigation of L-Opt

In Chapter 3 we have shown that L-OPT is optimal in the model where hotlinks may only point to leaves, and where the maximum number of outgoing hotlinks is specified by a function $K : V \to \mathbb{N}_0$. In the preceding sections of this chapter we have demonstrated for $K = 1$ that in practice, despite $O(n^3)$ worst case running time, an efficient implementation of that algorithm is faster than most other hotlink assignment methods.

For investigating the behavior of L-OPT for larger numbers of outgoing hotlinks, we consider constant functions $K = 1, \ldots, 15$. Figures 5.13 and 5.14 exemplarily show the behavior of runtime and path length on four selected instances. Apparently, the runtime grows linearly with K. This is what one would expect from the worst case of $O(n^2 \sum_{v \in V} K(v))$.

Figure 5.14 visualizes the benefit of allowing more hotlinks to start in the nodes. The curves show that the path length decreases only logarithmically with K. As a larger number of hotlinks not only increases the runtime of the assignment algorithm, but also reduces the clearness of web pages, one can conclude that only small values of K are advisable in practice.

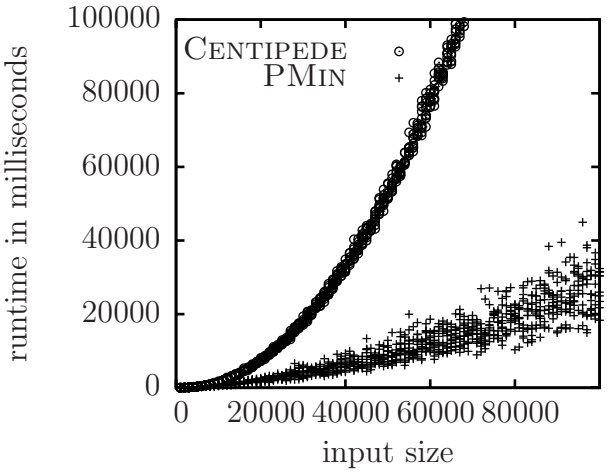


Figure 5.10: Runtime of CENTIPEDE and PMIN on random instances

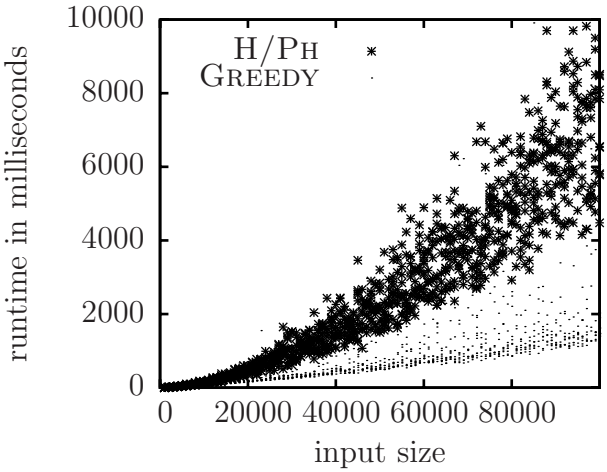


Figure 5.11: Runtime of H/PH and GREEDY on random instances

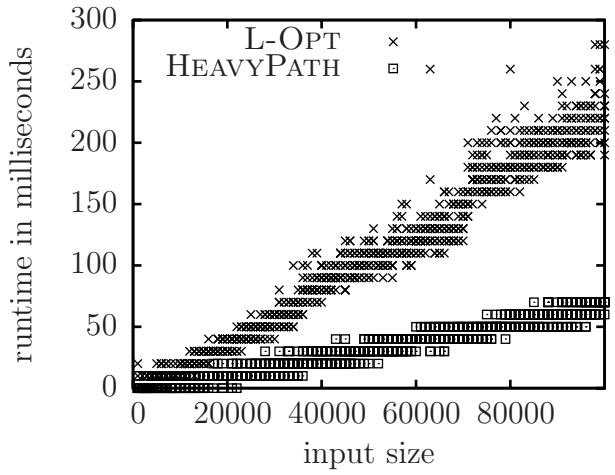


Figure 5.12: Runtime of HeavyPath and L-Opt on random instances

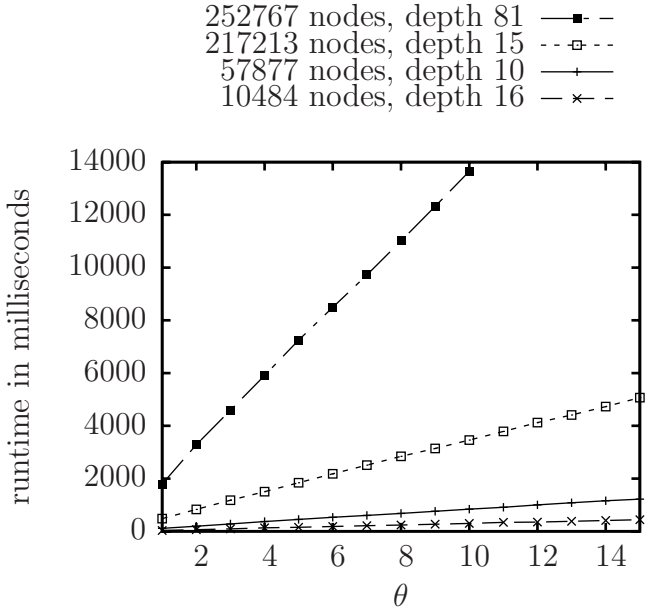


Figure 5.13: Runtime of L-Opt for varying values of K

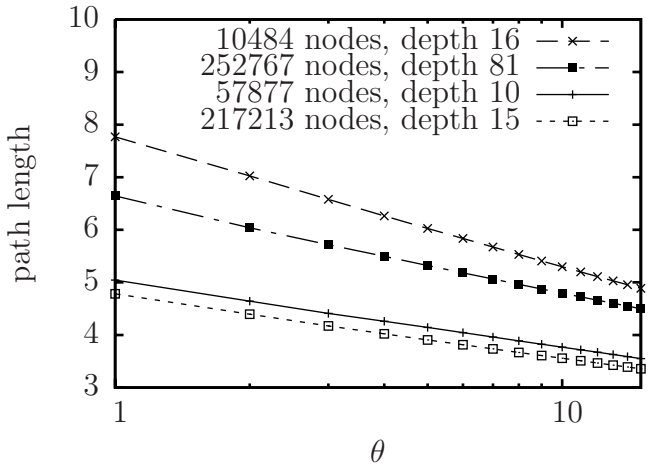


Figure 5.14: Path length achieved by L-Opt for varying values of K

5.4 Summary and Conclusion

We summarize the most important findings of our study.

Algorithm PMin computes better solutions than all approximation algorithms with known performance guarantees except for the PTAS. It is easy to implement, but the running time is quite high compared to other strategies.

The Greedy strategy also exhibits a very good performance. It is slightly worse than PMin in practice, but the guaranteed approximation ratio of 2 concerning the gain is what makes this algorithm a good choice. It is also easy to implement and runs faster than PMin.

Although the path length seems to be the more natural optimization term, strategies tailored to approximate it are not the first choice in practice. The Centipede algorithm has a very high running time. As the quality of its solutions is only moderate, this algorithm is only advisable if for some reason one wants a guaranteed approximation factor of 2 for the path length.

Algorithm H/Ph and HeavyPath exhibit the worst performance among all algorithms studied in this work. Anyhow, HeavyPath is extremely fast, making it interesting for quickly computing hotlink assignments for large trees. L-Opt is also one of the fastest algorithms and, in terms of solution quality, it is significantly better than HeavyPath.

For small values of h, Lpath is outperformed by Greedy and PMin. So this PTAS is only recommendable in practice for $h > 3$.

Chapter 6

Conclusion and Outlook

In this thesis we have established a number of new theoretical and practical results regarding the Hotlink Assignment Problem.

In Chapter 2, we have proven that it is NP-complete to decide whether there is a 1-hotlink assignment for a given tree that achieves a given gain value. In retrospect, this has been the most challenging part of our theoretical work. We remark that the leaves of the tree instances we reduce to carry weights that are exponentially large, i.e., it is still open whether there exists a pseudopolynomial optimal algorithm for the problem. In contrast, the Hotlink Assignment Problem for DAGs is known to be strongly NP-hard [BKK+00] in the clairvoyant user model.

In Chapter 3, we have given a polynomial time optimal algorithm L-Opt for the case that hotlinks may only end in leaves. The experiments described in Chapter 5 have shown that, regarding solution quality, this is typically not a severe restriction. Furthermore, a careful implementation of L-Opt is faster than most approximation algorithms for the general model. So, in applications where hotlinks pointing to internal nodes might be confusing for the users, L-Opt is certainly the algorithm of choice.

An alternative approach would be to consider the leaf-restricted model as a weighted bipartite matching problem. One issue for future research is to investigate if this approach yields an algorithm improving upon the worst-case and/or practical runtime of L-Opt.

We have presented approximation algorithms for the path length and the gain in Chapter 4. More precisely, we have proven the existence of a PTAS in terms of the gain, and we have given a 2-approximation in terms of the path length. Therefore, some gaps concerning the approximability remain. In particular, it is not known yet if there is a constant factor approximation in terms of the path length for the K-Hotlink Assignment Problem.

Unlike previous papers [FKW01, KKS04, DL05, DL06], we have not em-

ployed the entropy bound (cf. Chapter 1.4.2) in our analyses. For any fixed value of Δ, there are simple instances of Δ-ary trees where it is not possible to achieve a path length less than $H(\mathbf{p})$. As the entropy bound for Δ-ary trees is $H(\mathbf{p})/\log(\Delta+1)$, it is impossible to prove a constant approximation factor by comparing some global upper bound on the path length achieved by an algorithm to the entropy bound. The experiments from [CKK+03] and Chapter 5 demonstrate that algorithms tailored to approximate the entropy are dominated by other simple hotlink assignment methods also in practice.

Instead, our analyses are based on a tool set of three new hotlink modification operations that might be similarly useful in future work. Note however that these operations require each node to have the same maximum number of outgoing hotlinks. This is a reason why there is no approximation algorithm yet known for the model where the number of hotchildren is specified by an arbitrary function $K : V \to \mathbb{N}_0$.

The path length and the gain of a hotlink assignment for a tree T always sum up to $p(\emptyset, T)$. Therefore, the path length of an assignment is strictly decreasing in its gain. This observation enables us to easily combine different approximation methods. For example, an algorithm that chooses the best HLA between Greedy(T) and Centipede(T) is a 2-approximation in terms of both performance measures.

The latter idea can be carried over to practice. In Chapter 5, we have demonstrated that, regarding solution quality, the heuristic PMin outperforms nearly all algorithms with provable worst case approximation ratios. So, for example, by choosing the best assignment between those computed by PMin, Greedy, and Centipede, one would obtain a solution which has the quality of PMin and is guaranteed to be a 2-approximation in terms of both the gain and the path length.

Although large web sites typically have a hierarchical structure, it is reasonable to ask whether the results for trees can be extended to general graphs. Unfortunately, the question has to be answered negatively regarding the greedy user model. As mentioned in the introduction, the greedy model is not even well-defined for graphs. In our opinion, there is no straightforward way to model user behavior here. We outline three main possibilities.

The first possibility is to resort to the clairvoyant user model. Here, computing an optimal hotlink assignment is strongly NP-hard [BKK+00], and the natural greedy heuristic yields an approximation factor of 2 in terms of the gain [MP03]. The main drawback of the clairvoyant model is that it is rather unrealistic with respect to users of web graphs. There could however be applications where shortest paths are indeed taken. Examples are computer networks, road networks, or electric circuits.

The second way to model user behavior is to assume that users visiting

a node v all take the same unique path from the root to v. The union of all these paths is not necessarily a tree, but it is questionable if the non-tree case ever occurs in practice.

The third possibility emerges from the second one by considering user behavior as nondeterministic. For any target node v, the user chooses between a number of alternative paths from the root to v. Here the gain of a hotlink (u, v) is the product of the probability with which a user travels from u to v and the expected length of the corresponding path. This seems to be the most realistic model, but it is also the most complicated one. Furthermore, the web server would have too keep track of the probabilities for a very large (possibly exponential) number of alternative paths, which is not practicable. A possible simplification is to assume that the event of a user clicking a hyperlink (u, v) is independent from the path the user has taken to reach node u. In that case it suffices that the web server logs the access frequencies of the pages and the frequencies with which the hyperlinks are clicked. The downside is that now the objective function is rather complicated. The input data corresponds to a Markov chain, and a system of linear equations has to be solved in order to determine the gain of even a single hotlink.

A different generalization which also makes sense when the input graph is a tree is to assume that users take hotlinks only with a certain probability γ. We believe that this extra parameter adds more realism to the greedy model. Note that virtually any hotlink (u, v) of an assignment has a positive probability of being taken unless $\gamma = 0$ or $\gamma = 1$, so a feasibility restriction like Definition 1.1 is not reasonable here. Optimal solutions can be computed by an adapted version of the algorithm proposed in [PLdS04a, GKMP03], whose runtime is exponential only in the tree depth. To investigate whether the known approximation algorithms can also be generalized that way is another task to be addressed in future work.

Bibliography

[Abr63] Norman Abramson. *Information Theory and Coding*. McGraw-Hill, 1963.

[BA99] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[BKK+00] Prosenjit Bose, Evangelos Kranakis, Danny Krizanc, Miguel V. Martin, Jurek Czyzowicz, Andrzej Pelc, and Leszek Gasieniec. Strategies for hotlink assignments. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC)*, pages 23–34, 2000.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[CKK+01] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel V. Martin. Evaluation of hotlink assignment heuristics for improving web access. In *Proceedings of the 2nd International Conference on Internet Computing (ICOMP)*, pages 793–799, 2001.

[CKK+03] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel V. Martin. Enhancing hyperlink structure for improving web performance. *Journal of Web Engineering*, 1(2):93–127, 2003.

[CKPM02] Jurek Czyzowicz, Evangelos Kranakis, Andrzej Pelc, and Miguel V Martin. Optimal assignment of bookmarks to web pages. Unpublished manuscript, 2002.

[DL05] Karim Douïeb and Stefan Langerman. Dynamic hotlinks. In *Proceedings of 9th Workshop on Algorithms and Data Structures (WADS)*, pages 182–194, 2005.

[DL06] Karim Douïeb and Stefan Langerman. Near-entropy hotlink assignments. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 292–303, 2006.

[FKW01] Sven Fuhrmann, Sven Oliver Krumke, and Hans-Christoph Wirth. Multiple hotlink assignment. In *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 189–200, 2001.

[GKL+07] Ori Gerstel, Shay Kutten, Eduardo S. Laber, Rachel Matichin, David Peleg, Artur A. Pessoa, and Criston Souza. Reducing human interactions in web directory searches. *ACM Transactions on Information Systems*, 25(4):20, 2007.

[GKMP03] Ornan O. Gerstel, Shay Kutten, Rachel Matichin, and David Peleg. Hotlink enhancement algorithms for web directories. In *Proceedings of the 14th International Symposium on Algorithms and Computation (ISAAC)*, pages 68–77, 2003.

[GMY04] Melvyn L. Goldstein, Sidney A. Morris, and Gary G. Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B - Condensed Matter and Complex Systems*, 41(2):255–258, 2004.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Jac07] Tobias Jacobs. Constant factor approximations for the hotlink assignment problem. In *Proceedings of the 10th Workshop on Algorithms and Data Structures (WADS)*, pages 188–200, 2007.

[Jac08a] Tobias Jacobs. An experimental study of recent hotlink assignment algorithms. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics(ALENEX/ANALCO)*, pages 142–151, 2008.

[Jac08b] Tobias Jacobs. On the complexity of optimal hotlink assignment. In *Proceedings of the 15th European Symphosium on Algorithms (ESA)*, pages 540–552, 2008.

[JLH+00] Xiaohua H. Jia, Dengxin Y. Li, XXiaodong D. Hu, Hao J. Huang, and Ding Z. Du. Optimal placement of proxies of replicated web

servers in the internet. In *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE)*, pages 55–61, 2000.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Applications*, 00:85–103, 1972.

[KKM02] Evangelos Kranakis, Danny Krizanc, and Miguel V. Martin. The hotlink optimizer. In *Proceedings of the 3rd International Conference on Internet Computing (ICOMP)*, pages 87–94, 2002.

[KKM03] Evangelos Kranakis, Danny Krizanc, and Miguel V. Martin. Optimizing web server's data transfer with hotlinks. In *Proceedings of the IADIS International Conference WWW/Internet*, pages 341–346, 2003.

[KKS04] Evangelos Kranakis, Danny Krizanc, and Sunil Shende. Approximate hotlink assignment. *Information Processing Letters*, 90(3):121–128, 2004.

[KLM02] Danny Krizanc, Stefan Langerman, and Pat Morin. Asymmetric communication protocols via hotlink assignments. In *Theory of Computing Systems*, pages 33–40, 2002.

[KRS00] Paddy Krishnan, Danny Raz, and Yuval Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, 2000.

[LDGS98] Bo Li, Xin Deng, Mordecai J. Golin, and Kazem Sohraby. On the optimal placement of web proxies in the Internet: The linear topology. In *Proceedings of the IFIP TC-6 8th International Conference on High Performance Networking (HPN)*, pages 485–495, 1998.

[LS04] Keqiu Li and Hong Shen. Optimal placement of web proxies for tree networks. In *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE)*, pages 479–486, 2004.

[MP03] Rachel Matichin and David Peleg. Approximation algorithm for hotlink assignments in web directories. In *Proceedings of 8th Workshop on Algorithms and Data Structures (WADS)*, pages 271–280, 2003.

[MP07] Rachel Matichin and David Peleg. Approximation algorithm for hotlink assignment in the greedy model. *Theoretical Compututer Science*, 383(1):102–110, 2007.

[PE97] Mike Perkowitz and Oren Etzioni. Adaptive web sites: an AI challenge. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 16–23, 1997.

[PE00] Mike Perkowitz and Oren Etzioni. Towards adaptive web sites: conceptual framework and case study. *Artificial Intelligence*, 118(1-2):245–275, 2000.

[Pit99] James E. Pitkow. Summary of www characterizations. *World Wide Web*, 2(1-2):3–13, 1999.

[PLdS04a] Artur A. Pessoa, Eduardo S. Laber, and Críston de Souza. Efficient algorithms for the hotlink assignment problem: The worst case search. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 778–792, 2004.

[PLdS04b] Artur A. Pessoa, Eduardo S. Laber, and Críston de Souza. Efficient implementation of hotlink assignment algorithm for web sites. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALCO)*, pages 79–87, 2004.

List of Figures

List of Tables

List of Algorithms