MultiComp: A Multi-Purpose Framework for Similarity Matching

Master Thesis

Steven Van Damme
<vsteven@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Alfonso Murolo, Christoph Zimmerli

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zurich

8th November 2015
Abstract

Previous work on comparison of tree-based structures has provided many different approaches based on the tree edit distance. However, these approaches either gave up result quality for increased speed or gave up universal applicability for better results in a specific use case. We thus see a need for a general framework for comparison and focus on increased result quality for a broad range of applications. One major problem that these tree-based methods have is that don’t take into account where the differences occur in the trees, be it at the level of structure elements or content elements. We propose an approach using the tree edit distance that computes the structural similarity between trees by taking into account the location of the differences between both trees. The approach uses a non-static cost model that adapts the costs and a decay factor to the use case the tree edit distance is applied to. We present a framework called MultiComp that implements the approach and two evaluations, one that judges the quality of the results by comparing the results of our approach with existing approaches for the use case of detecting data records on web pages and one that shows the flexibility of the framework for multiple use cases. The results have shown a higher quality when using our approach based on the decay factor and the flexibility of the framework is proven by showing an application that uses MultiComp to find differences in branches of a versioning system.
# Contents

1 Introduction ................................................. 1

2 Background ................................................... 5
   2.1 Tree Edit Distance ........................................... 5
      2.1.1 Tree Mapping ........................................... 7
      2.1.2 Variation of Tree Edit Distances ......................... 9
         2.1.2.1 Alignment Distance ................................. 9
         2.1.2.2 Isolated-Subtree Distance ......................... 9
         2.1.2.3 Top-Down Distance ................................. 10
         2.1.2.4 Bottom-Up Distance ................................. 11
   2.2 General Approaches ....................................... 12
   2.3 Web Approaches ........................................... 15
      2.3.1 XML Approaches ..................................... 15
      2.3.2 HTML Approaches .................................... 17
         2.3.2.1 Tree Matching .................................... 17
         2.3.2.2 Restricted Top-Down Distance .................... 20
   2.4 Summary ................................................. 22

3 Approach ..................................................... 25
   3.1 Performance ............................................... 26
   3.2 Distance & Mapping ........................................ 26
   3.3 Tree Edit Operations ...................................... 28
      3.3.1 Standard Operations ................................ 28
      3.3.2 Extended Operations ................................ 28
   3.4 Restrictions .............................................. 29
   3.5 Cost Model ............................................... 31
      3.5.1 Decay Factor ........................................ 31
Over the past years, the web has become the largest collection of structured data that is widely available. Consequently a lot of research has been done in the area of structural analysis of data structures based on the web. Computing the structural similarity between data records is often done with approaches using the tree edit distance. Thus the available data needs to be easily transformed into trees so that these tree-based approaches can be used efficiently. Thanks to its way of representing data, the web is the ideal platform for such approaches. Data across the web, be it in web applications or on web sites, can be accessed in form of XML or HTML documents. Both XML and HTML enforce various restrictions and their documents need to adhere to a very specific structure. For these reasons the data is represented in a highly structured fashion. On top of that their representation is tree-based, which makes them perfect candidates for tree-based approaches.

With regards to the web, many past and present researchers [2, 5, 6, 10, 12, 20, 23] have used approaches based on the tree edit distance to compare trees. Their goals are in most cases the detection or extraction of data records or templates. An example of the kind of data that is targeted, would be the detection of news articles in [12]. However, the approaches are often tweaked in such a way that they are good for a specific problem but unsuitable for others. Furthermore, a lot of researchers value the performance of their approach higher than the quality of results. It is often the case that an approximation of the tree edit distance is used that achieves reasonably well results but gives a huge boost in performance. Therefore, we aim to develop a multi-purpose framework capable of handling a wide range of problems that focuses mainly on the quality of the results rather than performance.

Another application where these approaches could be applied is the topic of comparing branches of different trees. Especially interesting is the comparison of branches of tree based versioning systems. One example of a such a system can be found in Archipelago 2.0, an application developed as part of the IdeaGarden project. In Archipelago 2.0 users can handle

\[http://idea-garden.org/\]
different projects and are presented with a workspace that they can freely populate with data. The data is represented by cells and can be organised by using groups. The hierarchy of the data can easily be represented as a tree. Archipelago 2.0 also offers the possibility of creating branches of a project, which enables a user to explore different organisations of the data in parallel without interference from other users. On top of that users can take snapshots of their branches. A snapshot represents the current state of the branch of the project the user is working on. It is in fact a structural representation of all the data in that branch of the project. For this application, the problem that needs to be tackled is then the comparison of the states, in the form of snapshots, of different branches of a project to find commonalities and differences.

One major problem that the current approaches are dealing with, when comparing two trees, is that it is hard to identify data records of the same entity. An entity can have many different attributes, which might vary from one data record to the other, despite both data records representing the same entity. To better illustrate this, consider the following example. Assume we are looking for data records representing the entity of an item in an online shop. The entity has three attributes ordered into two groups. First a group containing a link to the item’s page and a description of the item and second a group containing images representing the item. The entity’s structure is depicted in Figure 1.1. In our data set we now find two data records representing the above entity. Their HTML tree can be seen in Figures 1.2a and 1.2b.

![Figure 1.1: Data entity representing an item in an online shop.](image)

It is easy to see that there are slight differences between both trees. The first data record shown in Figure 1.2a has a larger description and therefore three p tags are needed to display it, while the second data record shown in Figure 1.2b only needs one. Furthermore there is only one image of the first data record while the second data record has two. Even though these are only small changes in the content of the data records, a tree edit distance algorithm will need 4 operations to transform one tree into the other. In other words 4 nodes of a tree
need to be modified to transform it into the other. This represents 40% of the nodes in the tree, which is a large amount and might prevent the algorithm from classifying both data records as the same entity. The problem at hand is that all edit operations have the same weight regardless of the location at which they are executed. This means that changes in the higher levels of the tree, which means changes in the structure, have the same influence on the tree edit distance as changes in the lower levels of the tree, representing changes in the content. As we are doing a structural comparison of two trees, in the special case of trees on the web, this is not acceptable, however none of the existing approaches offer a solution for this problem.

The goals of this thesis are to develop a multi-purpose framework for similarity matching based on the tree edit distance. The framework should also focus on producing the best results possible rather than performance. Therefore, we present a new approach for estimating the similarity between trees using the tree edit distance. Our new approach is based on the approach presented in [9] but differs from the existing approaches by taking into account the location of the changes in the trees. We introduce a new concept, the decay factor, that gives less weight to the edit operations in the lower levels of the tree and more weight to the edit operations in the higher levels of the tree. Our hypothesis is that with the decay factor we can increase the influence of changes in the structure and lower the influence of changes in the content. The structure of the trees becomes more important and is the main focus of the similarity measure. Our approach consists of two steps. First a pre-processing step, where we analyse the trees and compute the best mapping based on the costs of the tree edit operations. In a second step we then compute the similarity between the two trees.

In this thesis we provide details on how to compute the costs of the different edit operations based on our decay factor as well as the computation of the tree edit distance. We also present how our MultiComp framework can be used in two specific cases, the first being the search for data records on web pages similar to DeepDesign [8] and second the comparison of branches in Archipelago 2.0. An evaluation for both cases is also presented.

We continue this thesis with a review of related work in Chapter 2. Then we present our approach in Chapter 3 followed by the architecture and implementation details of our framework.
in Chapter 4. In Chapter 5 we provide an evaluation of our approach based on two existing application, namely DeepDesign and Archipelago 2.0. Finally a conclusion and future work are given in Chapter 6.
This chapter surveys previous work on the topic of tree similarity measures. A common method to establish the similarity between two trees is the tree edit distance. Computing the tree edit distance is a long-standing problem and is known to be very difficult to solve. Any tree edit distance measure is built around three components, in particular a set of edit operations, a mapping and a cost model. The set of edit operations defines what type of tree edit operations can be used. The mapping defines the sequence of edit operations needed to transform one tree into the other. Finally the cost model assigns a cost to each edit operation. In the end the tree edit distance is equal to the cost of the minimal cost mapping between two trees. The sections below describe the tree edit distance in more detail, especially the effect of the different components. It follows an analysis of general approaches solving the tree edit distance problem as well as domain specific approaches for XML and HTML trees. Finally a summary will be given of all the interesting and useful elements of the different approaches for our use case.

### 2.1 Tree Edit Distance

The concept behind the general tree edit distance was first introduced by Tai [14] as a solution to the tree-to-tree correction problem. The tree-to-tree correction problem describes the problem of determining the distance between two labeled ordered trees. Tai argues that the tree-to-tree correction problems comes from the string-to-string correction problem, which determines the distance between two strings by the minimum cost sequence of edit operations needed to transform one string into the other. Since the tree-to-tree correction problem can be seen as a high-dimensional generalization of the string-to-string correction problem, the tree edit distance is defined by the minimum cost sequence of tree edit operations needed to transform one tree into the other. As stated before the approach is based on labeled ordered trees, which means that the tree has a single root node, each node needs to be labeled and the
children of each node oblige a certain ordering. This helps the generalization from strings to trees, since a string can be represented as a tree of depth 2 with a root having for each character of the string a child node with its label set to the character and respecting the order of the characters in the string.

Similar to the string edit operations there are three tree edit operations available for the tree edit distance. These are: insert, delete or relabel a node. Figure 2.1 illustrates an example for all three operations. The first tree shows the initial tree consisting of four nodes. The next one shows the new updated tree after a new node with label 'x' has been inserted. Note that in this case the new node has been inserted as a first-level child and that the older nodes 'b' and 'c' have been pushed down to be children of the new node x'. In general it is possible to insert a node at any level of the tree and any number of nodes at that level can become children of the newly inserted node. The third tree shows the state of the tree after deleting a node. In this case we simply deleted a leaf node. If a non-leaf node is deleted then all its children are pushed up and attached to the parent of the deleted node. Finally the last tree shows the final state of the initial tree after a relabel operation has been executed. The node b' has had its label changed to y'. This is the simplest of the three operations as it does not alter the structure of the tree at all.

![Figure 2.1: Sequence of the insert, delete and relabel node edit operations.](image)

With the operations presented above, the tree edit distance is effectively the minimal cost sequence of insert, delete and relabel operations to transform one tree into the other, which can be expressed with the formula 2.1.
Definition 1 (Tree Edit Distance)

\[ d(T_1, T_2) = |S| \cdot p + |I| \cdot q + |D| \cdot r \]  

(2.1)

where \( S \) is the set of relables, \( I \) the set of insertions, \( D \) the set of deletions, \( p \) the cost of a relabel, \( q \) the cost of an insertion and \( r \) the cost of a deletion.

2.1.1 Tree Mapping

Even with formula 2.1 computing the tree edit distance remains a difficult problem. Especially finding the minimal cost sequence of edit operations and describing how the sequence of operations effectively transforms one tree into the other. The problem is that there is an exponential search space of sequences of edit operations, as mentioned by [3, 18], and a straightforward recursive approach without any optimisation will therefore have an exponential runtime complexity. Tai presented the first non-exponential approach where he used the concept of tree mapping. This mapping establishes a one-to-one correspondence between the nodes of two trees. The mapping also preserves the order of siblings and ancestors. Tai presented his mapping with Definition 2.

Definition 2 (Tai Mapping)

Given \( M \) the mapping between the trees \( T_1 \) and \( T_2 \) and \( i \) and \( j \) indexes of the nodes of \( T_1 \) and \( T_2 \) respectively, then \( M \) is a Tai Mapping if:

1. \( 1 \leq i \leq ||T_1|| \) and \( 1 \leq j \leq ||T_2|| \)

2. For any pair of \((i_1,j_1)\) and \((i_2,j_2)\) in \( M \):
   
   (a) \( i_1 = i_2 \) iff \( j_1 = j_2 \) (one-to-one condition)
   
   (b) \( T_1[i_1] \) is to the left of \( T_1[i_2] \) iff \( T_2[j_1] \) is to the left of \( T_2[j_2] \) (sibling order preservation condition)
   
   (c) \( T_1[i_1] \) is an ancestor of \( T_1[i_2] \) iff \( T_2[j_1] \) is an ancestor of \( T_2[j_2] \) (ancestor order preservation condition)

Since the mapping has to be a one-to-one mapping, the restriction 2. (a) is needed to avoid that one node is used multiple times to built a map. In other words, it prevents the case where there are two integer pairs \((i,j)\) with the same value of \( i \) but a different value of \( j \). To be able to preserve the order of siblings and the order of ancestors, the restrictions 2. (b) and 2. (c) are needed. They restrict the creation of a one-to-one map between two nodes if the order of their siblings or their parent are different. An example of a valid and invalid mapping can be seen in Figures 2.2 and 2.3.

After the mapping has been established the sequence of edit operations to transform one tree into the other is easily described. For each pair \((i,j)\), if the label of node \( i \) is different from the label of node \( j \), then it needs to be relabeled. Each node \( i \), that is not used in a one-to-one map, needs to be deleted from the tree \( T_1 \). Similarly each node \( j \), that is not used in a one-to-one map, has to be inserted in \( T_1 \). The combination of these operations then transforms tree \( T_1 \) into tree \( T_2 \).
2.1. TREE EDIT DISTANCE

Figure 2.2: A valid Tai Mapping between T1 and T2.

Figure 2.3: An invalid Tai Mapping between T1 and T2 because of the mapping between both nodes with label e.

With the help of the mapping describing the minimal cost sequence of operations, the problem of computing the tree edit distance can be reduced from finding a minimal cost sequence of tree edit operations to a search for a minimal cost mapping, which was proven by Tai in [14].
2.1.2 Variation of Tree Edit Distances

The distance measure presented in Section 2.1 can be seen as the general tree edit distance. Over the years slight variations of this distance measure have been introduced. The main reasons are to either better fit a certain domain of problems or to reduce the complexity of the computation. Of course these benefits don’t come for free and something needs to be sacrificed. In the case of adapting to a specific domain, one has to expect to be less proficient in regard with problems of different domains. As for decreasing the complexity of the solutions, in this case this usually involves having solutions with a lesser quality. The specialisation and the decrease of complexity are due to further restrictions on the mapping, thus allowing only specific combinations and reducing the search space. Therefore, the variations are split into groups, each one having a specific mapping component. There are four main groups that stand out and that have been presented in [19] and [21]. All of them use the same set of tree edit operations as the general tree edit distance, however they apply different restrictions on nodes with a modified tree mapping.

2.1.2.1 Alignment Distance

The first variation is the alignment distance. The alignment distance uses a special type of mapping which is called an alignment. An alignment between the trees $T_1$ and $T_2$ can be defined as a mapping where first the least possible number of nodes labeled lambda are inserted into $T_1$ and $T_2$ so that the now modified trees $T_1$ and $T_2$ are isomorphic and then a mapping line is drawn between each node in $T_1$ and its corresponding node in $T_2$. The alignment distance is then equal to the cost of a least-cost alignment mapping. The advantage of the alignment distance is that instead of trying to edit $T_1$ to be equal to $T_2$, it looks for an alignment instead, inserting extra nodes if needed. This is a simpler problem as there are less possibilities for alignments than for edit sequences. An example of an alignment can be seen in Figure 2.4.

2.1.2.2 Isolated-Subtree Distance

Next is the isolated-subtree distance. The mapping behind the isolated-subtree distance is similar to the mapping of Definition 2. It has only been modified to be a structure preserving mapping. The mapping is structure preserving because two disjoint subtrees of $T_1$ must be mapped to two disjoint subtrees of $T_2$. This leads to the new restriction that for any three pairs $(i_1,j_1), (i_2,j_2)$ and $(i_3,j_3)$ in the mapping, the least common ancestor of nodes $i_1$ and $i_2$ is an ancestor of node $i_3$ if and only if the least common ancestor of nodes $j_1$ and $j_2$ is an ancestor of node $j_3$. As for the other mappings, a minimal cost isolated-subtree mapping determines the isolated-subtree distance between two trees. The advantage of the isolated-subtree distance is that with the extra restrictions the number of possible mappings is reduced, making it easier to find the optimal mapping. On top of that the mapping is structure preserving which is beneficial when trying to estimate the structural similarity between two trees. Figure 2.5 shows an example of a valid isolated-subtree mapping.
2.1.2.3 Top-Down Distance

Third is the top-down distance. As the name already suggests, the top-down distance is used for approaches traversing the tree from the top and is defined by the minimal cost top-down mapping. The top-down mapping is again an extension of the general mapping of
Definition 2. The additional restrictions that the mapping needs to consider is that for any pair \((i,j)\), where nodes \(i\) and \(j\) are not the root, then the pair \((\text{par}(i),\text{par}(j))\) is also part of the mapping, with \(\text{par}(x)\) representing the parent node of node \(x\). One effect of the top-down distance is that due to its mapping the differences in the higher level nodes of the tree have a higher impact on the overall cost. This is helpful for the structural similarity analysis in case the higher level nodes represent the structure of a tree. A valid top-down mapping for the trees \(T_1\) and \(T_2\) can be seen in Figure 2.6.

![Figure 2.6: A top-down mapping between \(T_1\) and \(T_2\).](image)

2.1.2.4 Bottom-Up Distance

Finally, there is the bottom-up distance. Opposed to the top-down distance, the bottom-up distance gives a higher influence to the lower level nodes. For a mapping to be considered to be a bottom-up mapping, additionally to the restrictions of Definition 2, for any pair \((i,j)\) in the mapping each child node of node \(i\) has to be mapped to its respective child node of node \(j\). As for all the other variations the bottom-up distance equals the minimal cost bottom-up mapping. The effect of the bottom-up mapping is opposite to the one from the top-down mapping. In this case the differences in the lower levels of the tree have a higher impact on the overall cost. Figure 2.7 illustrates a valid bottom-up mapping.

The hierarchy of and relationships between the different variations of the distance measures can be seen in Figure 2.8. It is interesting to note that top-down and bottom-up mappings overlap so that a mapping can be both a top-down and bottom-up mapping depending on the tree structure.
2.2 General Approaches

The tree edit distance and its variations can be applied to a wide area of problems. A lot of problems that involve structured data benefit from the use of these distance measures. This section will present approaches that use the different concepts of tree edit distances presented in Section 2.1. The order in which they are presented will be the same as the order in which the different mappings have been presented.

As mentioned earlier Tai was the first to present the general tree edit distance in [14]. He
presented the tree edit distance as a solution to the tree-to-tree correction problem. However, he did not only present the concept behind the tree edit distance but also an algorithm using the tree edit distance to solve the problem at hand. The tree edit operations used by his algorithm are the ones depicted in Figure 2.1. His mapping, also known as Tai mapping, is defined by Definition 2. Tai’s algorithm consists of the following three different stages. First a pre-processing step where the cost of the different elements in the tree are computed. During the next step the minimal cost mapping is established. This is done with a recursive algorithm. Finally, in the third step the tree edit distance is computed based on the minimal mapping from the second step. The algorithm uses a pre-order traversal of the trees during these three steps. Tai argues that his method is well suited for several applications, namely measuring the similarity between trees, automatic error recovery and correction for programming languages as well as finding the largest common substructure of two trees. Tai’s algorithm has a runtime complexity of $O(n^3 \times m^3)$ with $n$ and $m$ being the sizes of tree $T_1$ and $T_2$ respectively.

Zhang and Shasha [24] presented another algorithm based on Tai’s solution that can be considered as the state of the art algorithm for tree edit distance problems. The reason is that this solution is mentioned by most of the newer algorithms as its inspiration and starting point. Zhang and Shasha wanted to present a simple algorithm to the tree edit distance problem that would be computationally less expensive than Tai’s algorithm. The operations and the mapping used by the algorithm are the same as for Tai’s algorithm. However, there are a few differences between both algorithms. First off, Zhang and Shasha use a post-order traversal method, which leads to computing the minimal cost mappings of descendants of a node before computing it for the node itself. This allows them to reuse the results of subproblems that have already been computed. Furthermore, the concept of keyroots is introduced. Keyroots are defined as the root of a tree plus all nodes that have a left sibling. This gives Zhang and Shasha the opportunity to reduce the amount of computations needed by considering the forest distance between nodes instead of the tree distance during intermediate steps. The proof behind the concept can be found in [24]. Together these new concepts result in the creation of a two step algorithm. First a pre-processing step to compute the left-most leaf descendant as well as the keyroots for each node in both trees. The second step is then used to compute the tree edit distance. Zhang and Shasha use a dynamic programming approach for this step as the problem can be split into multiple easier subproblems. It is also mentioned that their algorithm can easily be parallelized and can be generalized to solve approximate tree matching problems. The runtime complexity of this algorithm is $O(n \times m \times \min(depth(T_1), leaves(T_1)) \times \min(depth(T_2), leaves(T_2))$, where $leaves(T)$ is the number of leaves of tree $T$.

A solution using the alignment distance has been presented in [4]. The algorithm is proposed as an alternative to the tree edit distance to measure the similarity between trees. One specific problem domain where they want to improve the results is the domain of RNA ¹ secondary structure comparison. They argue that the alignment distance can be computed faster and yields better results for this specific type of trees. Since the alignment distance is used it is obvious that the alignment mapping is used as well. The edit operations can’t really be considered in this case due to the nature of the mapping. Instead of inserting, removing or relabeling existing nodes, the mapping introduces new nodes to make both trees isomorphic. The algorithm also uses a post-order traversal and consists of two steps. A first step is needed

¹ urlhttps://en.wikipedia.org/wiki/RNA
to align the different subforests of both trees to each other. The second step then recursively computes the alignment distance between the two trees. There is of course a clear difference between the concepts of tree edit distance and alignment distance. While the former finds the largest common subtree, the latter yields the smallest common supertree. However, to measure the similarity between trees it is clear that both concepts are related. The runtime complexity of this approach is \(O(n \times m \times (\text{degree}(T_1) + \text{degree}(T_2))^2)\) in the worst case and \(O(n \times m)\) for trees with bounded degrees.

Tanaka and Tanaka were the first to present an algorithm based on the isolated-subtree distance. They present their ideas in [15] and argue that their solution yields better results for measuring the similarity between trees where the structure of those trees is important. This is why they need a new mapping model, which extends the Tai mapping to be a structure preserving model, which is described above as the isolated-subtree mapping. The tree edit operations are not altered compared to those Tai presented. Computing the isolated-subtree distance between two trees is done with a dynamic programming approach. An extra level of abstraction is used with which the problem is reduced to finding the shortest path in a diagram. Details behind the concept of the diagram can be seen in [15]. Tanaka and Tanaka even present a very refined algorithm for the special case where each node can have at most one child. Overall their approach can be used in numerous areas, such as pattern recognition or syntactic tree comparison and classification. This approach has runtime complexity \(O(n \times m \times \min(\text{leaves}(T_1), \text{leaves}(T_2)))\).

The first top-down distance has been presented by Selkow [13]. This distance can be used as a solution for the tree-to-tree correction problem. The tree edit operations related to this distance are again the same as the ones used by Tai. However, there are two major restrictions imposed on these operations. First, inserting a node can only be executed if the node is inserted as a leaf. It is forbidden to insert a node as an intermediate node in the tree. Second, deleting a node is only possible if the node is a leaf. This means that if in a subtree a node has to be deleted that is not a leaf node, all the descendants of that node need to be deleted first before that specific node can be deleted. These restrictions can also be expressed with the top-down mapping presented above. The reasoning behind the restrictions is to reduce the search space of mappings resulting in a faster solution. But it also means that differences in the higher levels of the trees will have a higher impact on the overall cost and thus on the distance. Selkow's algorithm is a two step algorithm. At first a pre-processing step is needed to compute the cost of relabeling each pair of labels, the cost of deleting each node of the source tree and the cost of inserting each node of the destination tree. The second step then recursively establishes the top-down distance between both trees. The overall runtime complexity is \(O(n \times m)\).

Valiente [19] has presented an efficient algorithm computing the similarity between trees using the bottom-up distance. Valiente argues that its greatest benefit lies in the low computational complexity of the algorithm. The bottom-up distance is based on the bottom-up mapping that has been presented in Section 2.1.2.4 and uses the standard insert, delete and relabel node operations. The algorithm can be split into three steps. First, a pre-processing step that computes a compacted acyclic graph representation of the disjoint union of the two trees. The correspondence between nodes of both trees and nodes of the graph is returned and used in the second step to extract a mapping from one tree to the other. Finally, the last step then uses that mapping to compute the bottom-up distance. Details on how the graph is
constructed and how the mapping is extracted from it can be found in [19]. Valiente mentions that his solution is especially useful as a quick filter for searching in a metric space of trees. The runtime complexity of the approach is $O(n + m)$.

2.3  Web Approaches

The format in which data on the web can be found is often either XML or HTML. Both formats come with a clear structure and XML or HTML documents represent their data as ordered labeled trees. This makes them suitable to be analysed by algorithms based on the tree edit distance and its variations. Below a few variations of approaches will be presented that specifically concentrate on XML and HTML based trees. The main goal of the approaches is to detect and/or extract data records or templates from the web.

2.3.1 XML Approaches

In the special case of XML trees computing the similarity between two of those trees might require some tweaking of the existing approaches. Those little tweaks should improve the quality of results and create new approaches that are specific to the domain of XML trees [9, 11, 16, 17]. Two of these approaches are closely related to our use case and are presented below.

Nierman and Jagadish [9] have presented one solution taking into account the structure and features of XML trees. The distance, used as similarity measure, can be classified as a top-down distance. The three basic operations insert, delete and relabel node are again available. Furthermore, two operations are added, namely insert tree and delete tree. These two extra operations should help to better handle the XML tree structure. A graphical illustration of the two new operations can be seen in Figure 2.9. The first tree represents the initial state of the tree, the second the new state of the tree after inserting a subtree and the last tree shows the final state of the tree after deleting a subtree.

Since all of the edit operations have unit costs Nierman and Jagadish define so called allowable sequences. These are essentially sequences of operations that are allowed to be executed. If there were no restrictions at all, then with the insert tree and delete tree operations having only unit cost, it would be possible to transform any tree into another with just two operations by first deleting the whole source tree and then inserting the whole destination. Thus resulting in a minimal cost sequence of operations with value 2, regardless of the number of differences in both trees. The exact definition of an allowable sequence can be found at Definition 3.

Definition 3 (Allowable Sequence)

A sequence of edit operations is allowable if it satisfies the following two conditions:

1. A tree $P$ may be inserted if and only if $P$ already occurs in the source tree $A$. A tree $P$ may be deleted only if $P$ occurs in the destination tree $B$.

2. A tree that has been inserted via the insert subtree operation may not subsequently have additional nodes inserted. A tree that has been deleted via the delete subtree operations may not previously have had nodes deleted.
The first condition limits the use of the insert and delete tree operations to the case when the subtree is shared between source and destination tree. This is needed to solve problems when XML documents have optional elements. The second condition provides an efficient way to compute the costs of inserting and deleting subtrees. Nierman’s and Jagadish’s algorithm requires two steps. First a pre-processing step, where the costs for each node in each tree are computed in a bottom-up fashion. The tree distance is then computed in the second step. The tree is traversed in post-order and using a dynamic programming style the tree distance is computed. Finally it is shown that the algorithm performs well for establishing the similarity between XML documents on both real and synthetic data sets. The overall runtime complexity of the approach is $O(n \times m)$.

Another approach that has been shown to do well with XML trees has been presented by Pawlik in [11]. He argues that the currently available solutions are not satisfactory and that those solutions either often have to deal with the worst case or only work efficiently for very specific tree shapes. He presents a solution that is more robust and can handle many tree shapes reasonably well. For this reason his algorithm is named RTED, the robust tree edit distance algorithm. The concept behind the robustness of the algorithm lies in the way the trees are traversed. For a rooted labeled ordered tree there are always three paths that can be chosen to go through. These paths are the left path, right path and heavy path, as can be seen in Figure 2.10.

Zhang and Shasha [24] always pick the left path, while Klein [7] always picks the heavy
path. However, depending on the tree structure this might lead to the computation of extra subproblems which would not have been necessary if the optimal path was chosen in the beginning. Therefore, as a first step of his algorithm Pawlik computes the best strategy to traverse the tree, the so called LRH strategy. Details of how the LRH strategy is computed are omitted at this point, but can be found in [11] in the form of a baseline algorithm as well as an efficient algorithm. The distance between two trees is then computed with the help of the LRH strategy that was found. Pawlik developed a general tree edit distance algorithm, called GTED, to do so. The algorithm is based on the previous work, for example Zhang and Shasha’s work [24], and uses the standard edit operations. As mentioned before, as input for the GTED algorithm the optimal LRH strategy is needed as well as both trees that are to be compared. The algorithm then recursively computes the tree edit distance between both trees based on the given LRH strategy. Pawlik also proves with experiments on real and synthetic data sets that his solution is the most efficient over a wide range of tree shapes. RTED’s runtime complexity is $O(\max(n, m)^2)$.

2.3.2 HTML Approaches

This section presents a few examples of tree edit distance approaches for HTML trees. They will be presented in two groups, first the approaches based on tree matching and second the approaches using a restricted top-down distance.

2.3.2.1 Tree Matching

The approaches presented in [2, 5, 6, 22, 23] are all based on tree matching, a different similarity measure using an approximation of the tree edit distance. They all use the Simple-TreeMatching (STM) algorithm, given by Algorithm 1, that has been presented by Yang [22] and somewhat modify it to accommodate their specific needs.

Yang has presented STM as a solution for identifying the differences between two programs. The concept behind the STM approach is a different one than the approaches presented above. Instead of finding the operations needed to transform one tree into the other, STM rather
Algorithm 1 SimpleTreeMatching(T1, T2)

1: if string(i) = path(j) then return (0).
2: \( m := \) the number of first-level subtrees of \( T_1 \).
3: \( n := \) the number of first-level subtrees of \( T_2 \).
4: Initialisation. \( M \left[ i, 0 \right] := 0 \) for \( i = 0, \ldots, m. \) \( M \left[ 0, j \right] := 0 \) for \( j = 0, \ldots, n. \)
5: for \( i := 1 \) to \( m \) do
6: for \( j := 1 \) to \( n \) do
7: \( M \left[ i, j \right] := \max(M \left[ i, j - 1 \right], M \left[ i - 1, j \right], M \left[ i - 1, j - 1 \right] + W \left[ i, j \right]) \)
8: where \( W \left[ i, j \right] = \text{SimpleTreeMatching}(T_1_i, T_2_j) \)
9: where \( T_1_i \) and \( T_2_j \) are the \( i \)th and \( j \)th first-level subtrees of \( T_1 \) and \( T_2 \) respectively
10: return \((M \left[ m, n \right] + 1)\)

looks for the number of matching elements between the two trees. This means that STM is much faster than a standard tree edit distance approach as it does not need to compute the edit operations. However the increased performance comes at the cost of the quality of the result. The example in Figure 2.11 will help to demonstrate how STM works.

![Figure 2.11: Example trees for the STM algorithm.](image-url)
see that the labels are the same, thus increasing the score by one and go on to the first-level children. At this point both subtrees containing only the node $a$ are matched and increase the score again by one. The subtrees rooted in $b$ and $x$ get classified as different since the labels don’t match and the algorithm stops, hence the score of 2. However, even though it is obvious that out of the five nodes in each tree four of them match, the STM approach is unable to detect that. In contrast the tree edit distance approach would need only a single operation to transform one tree into the other, namely relabeling node $b$ to $x$. Even though STM has clear weaknesses, it is still popular because of its low runtime complexity of $O(n \cdot m)$.

One approach based on STM has been presented in [2]. The presented work shows a new technique to help the extraction of data from web pages. It is argued that data extraction from web pages is often done with wrappers, using logic-based methods. However in many cases the robustness of the wrappers is very important, as the structure between pages might slightly change, which should not cause the wrapper to fail. Therefore an automatic wrapper adaptation by three edit distance matching is presented. This automatic wrapper uses the STM algorithm with a small modification. The 10th line in Algorithm 1 is replaced by the code snippet of Algorithm 2.

**Algorithm 2** ClusteredTreeMatching($T_1$, $T_2$)

1:  Change line 10 with the following code.
2:  if $m > 0$ AND $n > 0$ then
3:      return $M[m,n] + 1/\max(t(T_1), t(T_2))$
4:  else
5:      return $M[m,n] + 1/\max(t(T_1), t(T_2))$
6:  where $t(T)$ returns the number of siblings of $T$

With this change the approach is now called ClusteredTreeMatching (CTM). In CTM, the value for a match between two nodes is changed from the static value of 1, to a weighted value of 1 divided by the greater number of siblings of the two compared nodes. This is introduced as a reduction of the impact of missing and added nodes. Furthermore it is argued that contrary to STM, CTM returns immediately a similarity in the form of a percentage value, while the maximum matching value needs subsequent operations to establish a similarity value. A short evaluation shows that the newly presented CTM outperforms the classic STM. The runtime complexity of CTM is the same as for STM.

In [6] an approach is presented using STM as a skeleton. This approach is called HTML Tree Matching (HTM) and as the name suggests it has been developed for handling HTML trees. The main goal is to use HTM as a method to extract information from web pages. HTM tries to solve the issue of recognizing similar patterns in web pages with which previous work has struggled. It is argued that the reason is that in previous work it is assumed that each node of the tree has the same value and accordingly each node is assigned the same weight. This is changed in HTM by computing different values and weights for each node in the tree. The weight of a node is defined as the information value of the node divided by the sum of information values of the sibling nodes. The information value of a node can be defined as a value to show how important the node is. The formula for the information value differs depending on the type of node. For an image node, the information value is computed based on the displayed size of the image, while the information value of a text node is influenced
2.3. WEB APPROACHES

by the byte size and font size of the text. Furthermore the information value for a parent node is equal to the sum of information values of its children. The actual value assigned to a node is defined by Definition 4.

Definition 4 (Information Value)
\[ V(n) = [S(n) + (V(parent(n)) - 1)] \times W(n) \]

where \( S(n) \) is the number of siblings of \( n \), \( V(parent(n)) \) is the information value of the parent of node \( n \) and \( W(n) \) is the weight of \( n \).

The values are inserted into the algorithm by replacing the 10th line of Algorithm 1 with the code in Algorithm 3. This leads to a two step approach, where in a first step the weights and values for the nodes are computed and in a second step the similarity between two trees is established with the help of HTM. Two formulas to better grasp the similarity between two trees are also given, one based on STM and one on HTM. These formulas are then used in a small evaluation to show that HTM outperforms STM in regard of quality of results. The performance has the same asymptotic runtime complexity.

Algorithm 3 HTMLTreeMatching(T1, T2)

1: Change line 10 with the following code.
2: \textbf{return} \( M[m,n] + \text{AVG}(V(m), V(n)) \)
3: where \( \text{AVG}(m,n) = (m + n)/2 \)

2.3.2.2 Restricted Top-Down Distance

Reis [12] presents a different type of approach. The approach is based on the top-down distance and thus uses a top-down mapping. However, to better match their problem of automatically extracting news from web pages, a new version of top-down mapping is defined, the \textit{restricted top-down mapping}. In addition to the insert and delete operations, as in the normal top-down mapping, the relabel operation is also restricted to leaf nodes only. Figure 2.12 illustrates an example of a restricted top-down mapping.

To find the restricted top-down distance between two trees, the minimal cost restricted top-down mapping has to be found. The new algorithm, called RTDM, first needs to compute the set of all identical subtrees. This will be needed at a later point as a shortcut to accelerate the computation. Once all identical subtrees have been found, the minimal mapping can be computed with the RTDM algorithm, presented by Algorithm 4.

It is interesting to note that the algorithm can easily be adapted to use the non-restricted top-down distance. To do so, one simply needs to alter lines 16 - 21. Moreover, all the costs used in the algorithm are given as functions of the nodes. This means that there are no static costs but in fact the costs are very flexible. Depending on the specification of the problem the costs can be adapted to better suit them. In the case of automatically extracting web news, Reis argues that the problem can be split into two tasks, first the crawling of web pages and second the extraction of web news. While the first task is not really of interest to us, the second is all the more. According to Reis the task is divided into four steps, page clustering, extraction pattern generation, data matching and data labeling. In three of those four steps, the
Algorithm 4 RTDM(T1, T2, $\epsilon$: threshold)

1: $m :=$ the number of first-level subtrees of $T1$.
2: $n :=$ the number of first-level subtrees of $T2$.
3: $M[i, 0] := 0$ for $i = 0, ..., m$.
4: $M[0, j] := 0$ for $j = 0, ..., n$.
5: for $i := 1$ to $m$ do
6:     for $j : 1$ to $n$ do
7:         $C_i \leftarrow$ descendents($T1[i]$)
8:         $C_j \leftarrow$ descendents($T2[j]$)
9:         $d \leftarrow M[i - 1, j] + \sum_{k \in C_i}^{T1[k]} delete(T1[k])$
10:        $d \leftarrow M[i, j - 1] + \sum_{k \in C_j}^{T2[k]} insert(T2[k])$
11:        if $M[i - 1, j - 1] > \epsilon$ then
12:           $s \leftarrow \infty$
13:        else if $T1[i]$ and $T2[j]$ are identical subtrees then
14:           $s \leftarrow 0$
15:        else
16:           if $T1[i]$ is a leaf then
17:              $s \leftarrow replace(T1[i], T2[j])$
18:              $s \leftarrow s + \sum_{k \in C_j}^{T2[k]} insert(T2[k])$
19:           else if $T2[j]$ is a leaf then
20:              $s \leftarrow replace(T1[i], T2[j])$
21:              $s \leftarrow s + \sum_{k \in C_i}^{T1[k]} delete(T1[k])$
22:           else
23:              $s \leftarrow RTDM(T1[i], T2[j], \epsilon)\)\n24:        end if
25:        $M[i, j] \leftarrow \min(d, i, s)$
26:     end for
27: end for
28: return $(M[m, n])$
RTDM algorithm is used each time in a different way. In the page clustering step, the RTDM algorithm is used as the distance measure in a classic hierarchical clustering algorithm. While generating extraction patterns, RTDM is used to implement the composition operation of two patterns. Finally during the matching step, the RTDM algorithm is used to match different patterns and uses a special cost model specific for this problem. Details of the different use cases can be found in [12]. Reis also presents a small evaluation, where the RTDM algorithm is used to extract news articles from Brazilian news web pages, and argues that his approach yields very good results. The worst case runtime complexity of RTDM is $O(n, m)$.

Two more approaches [10, 20] extending the RTDM algorithm have been presented. The authors of [10] give a new approach using RTDM but having a better performance than the original in the worst case. Moreover [20] presents an adaptation to the given algorithm to detect and extract web page templates instead of data records. Since both approaches don’t relate closely to our problem details are omitted at this point.

2.4 Summary

Finding the right approach for our problem is not an easy task. However, there is much we can learn from existing work to help us built our approach. As have been presented in Section 2.1, there are several types of distance measures, each of them linked to a specific mapping. While all the different distance measures have their benefits, it appears that one specific distance is often used in the case of handling web-related data. This distance is the top-down distance. The main advantage of the top-down distance comes from the top-down mapping, which gives a higher importance to nodes in the higher levels of the tree. In the case of XML or HTML trees, these nodes represent the structure of the tree and since we want to
find structurally similar trees, this is beneficial to us. The tree edit operations don’t tend to vary that much from approach to approach. Nonetheless, several approaches present, as an addition to the tree edit operations presented in Figure 2.1, new extended operations. These extended operations are used to better handle the specific nature of the XML and HTML trees. Especially the case of having optional elements or repeating structures can be exploited to increase the performance of the approach, as has been shown in [9]. Furthermore, some researchers impose additional restrictions on the edit operations. Some of them are used to further specialize the approach [12], while others exist to help the approach work but don’t limit its use cases [9]. Therefore, we should be careful when adding additional restrictions, to ensure that our approach remains useful for multiple purposes. A clear weakness of many approaches is their static cost model. Each operation for each node has the same unit cost 1. While this makes sense in the case of general trees where each node has the same importance, it is clearly not the case with XML and HTML trees. There are definitely nodes in those trees that have more or less influence in the structure of the tree. The leaf nodes will always contain the content that is to be represented, while the higher level nodes generate the skeleton of the structure of the tree. Accordingly an adaptive cost model, similar to the ones presented in [2, 6], is highly favoured. Even though performance is not our main goal, it is clear from previous work that solving the problem of the tree edit distance is hard and simply using a recursive approach will take a long time. An easily made optimization would be to create an approach using the dynamic programming method. This would allow us to split the computation of the tree edit distance between two trees into many easier subproblems and their results can then be used to solve the main problem.

In the next chapter, we will present our new approach taking into account everything we learned from previous work and combining the best attributes of the different approaches into one.
The goals of this thesis are to build a framework for similarity measures based on a new version of the tree edit distance. The framework should be usable for multiple purposes and handle a wide range of problems. On top of that the results should be of high quality even if that means that performance needs to be sacrificed. Our goal is to reach these high quality results by having a flexible cost model based on our new concept, the decay factor. This allows us to give different weights to differences in the structure and in the content of the tree. In our case differences in the structure should have a higher impact on the overall distance than differences in the content. In this chapter we describe our newly developed tree edit distance. We will first give a general overview of the different features and benefits it provides. Then we will explain each feature in detail and finally put everything together and present the final complete approach.

During the previously discussed related work, many different approaches have been presented. These approaches present many different and interesting ideas that mostly try to adapt a tree edit distance algorithm for the structural analysis of trees on the web. Most results are quite promising but leave room for improvement. Thanks to our research, we are able to specify a list of features or specifications that our new algorithm should have, to hopefully achieve better results. These are summarized in the following list:

1. Dynamic programming for increased performance
2. Top-down edit distance and mapping
3. Basic and extended tree edit operations
4. Restrictions on the tree edit operations
5. Flexible cost model
All these features have shown positive effects on the quality of results their approaches have achieved. They are also a collection of improvements from several approaches, therefore we expect that their combination yields even better results, than their individual application. While many approaches are used as an inspiration for our new approach, there is one approach that our algorithm is mainly based on, namely [9]. This specific approach covers already the first four features on its own, which is why we have decided to use it as the skeleton for our approach.

It follows one section for each desired feature. The section will cover the feature itself, its advantages for the approach as well as how exactly the approach needed to be adapted to incorporate the new feature.

### 3.1 Performance

Even though performance is not our main concern, while developing our new approach some optimizations are still welcome. Making the use of our approach more enjoyable and faster does benefit both the end users as well as ourselves. One performance optimization occurring over and over again in previous work, like [9] that we base our approach on, is using a dynamic programming approach to compute the tree edit distance. A dynamic programming schema is perfect for our problem at hand. The idea behind a dynamic programming schema is to split up an initial hard problem into smaller easier subproblems. In our case of comparing tree structures, it is obvious that the initial hard problem is comparing two complicated trees. With the help of dynamic programming, we don’t need to handle that problem directly but instead can start by comparing the subtrees of both trees. These subproblems are already easier to compute because the trees are less complex. We can further split the subproblems into even smaller and at some point trivial problems. Finally we can combine the results of the different subproblems to obtain the solution for the initial hard problem. Moreover, since the tree edit distance between two trees needs to be computed recursively, as shown in previous work, one method to do so is to use a dynamic programming approach with memorization. This method, as opposed to classic recursive approaches, prevents the approach to compute the result of a subproblem twice and instead memorizes and reuses the computed result. This will in some cases greatly reduce the number of subproblems that need to be computed and hence increase performance. Therefore we think dynamic programming is a reasonable and straightforward optimization and opted to put dynamic programming as the first wanted feature for our approach.

### 3.2 Distance & Mapping

The second desired feature is a top-down distance measure. There are numerous variations of tree edit distances, as seen in Section 2.1, however the top-down distance has been shown to perform best for the structural analysis of trees on the web. This becomes very clear by the fact that most of the web-based approaches, seen in Section 2.3, all work with some variation of a top-down distance. All of those approaches have had a fair share of success. The reason of success for this type of distance with web based trees comes from the underlying top-down mapping, presented in Section 2.1.2.3. The main benefit of this mapping is that
differences in the higher level nodes of the trees have a higher impact on the final distance. In the case of XML and HTML trees, these nodes often represent the structural elements of the tree, while the lower level nodes, especially the leaf nodes, represent and contain the content of the tree. Thus the top-down mapping gives structural differences between trees more importance, which is perfect since we want to establish a structural similarity between them. As an example, consider the two different mappings for two trees in Figures 3.1a and 3.1b.

![Diagram](image)

(a) Example of a Tai mapping.  
(b) Example of a top-down mapping.

Figure 3.1: Examples of different mappings.

In Figure 3.1a, a Tai mapping of the trees is displayed. All linked nodes are unchanged. Any node that is not linked in the source tree is deleted and any node not linked in the destination tree is inserted. Assuming a static cost model with unitary costs for all operations, the tree edit distance between both trees is 1. The most interesting cases for us occur in the subtree rooted at node x. Since there is no counterpart in the source tree, node x is not linked, but the remaining nodes are linked to their counterparts. Thus resulting in a single insertion operation that is needed to transform one subtree into the other and resulting in a tree edit distance of 1 even though the structure is clearly different. Now with the top-down mapping, represented in Figure 3.1b, considering the same subtree the mapping is clearly different. Because of the difference in structure and the third rule of top-down mappings it is not possible to simply establish a link between all the remaining nodes. Since node x has no counterpart, it can’t be linked to another node and due to the third rule neither can its children nodes. This results in a sequence of operations needed to transform one subtree into the other, namely deleting all of node b’s children, inserting node x itself and finally inserting all of the missing nodes again. Hence resulting in a tree edit distance of 5 for the subtrees and 5 for the complete trees. The increased value of the tree edit distance indicates a higher structural difference between both trees using the top-down distance compared to the standard tree edit distance, which in the end is exactly what we would expect from our approach since there is a clear difference in structure.
3.3 Tree Edit Operations

An essential part of any tree edit distance are the tree edit operations. As already mentioned before, any tree edit distance tries to find the minimal cost sequence of tree edit operations to transform one tree into another. These tree edit operations then clearly describe how the transformation is done. Therefore, tree edit operations need to be clearly defined and their costs established beforehand. Our cost model will be presented in Section 3.5. First, we define our tree edit operations, which can be divided into two groups, the standard operations and the extended operations. Both groups are presented in detail below.

3.3.1 Standard Operations

The operations that we consider as standard operations are the insert, delete and relabel node operations. These operations have already been presented in Section 2.1 and are shown in Figure 2.1. As seen in Chapter 2 all approaches use those three operations. This is obvious because those insert and delete are the most trivial operations once could use to transform one tree into any other tree, and relabel is a very convenient one to have. In the case of general trees, this is good enough to estimate a structural similarity between them, however we consider this is not the case with XML and HTML trees. In those specific cases, things like optional elements or repeated structures occur on a regular basis. These cases make it hard for a tree edit distance approach using only the standard edit operations to give a good similarity estimation. Consider the two trees in Figure 3.2.

Both trees consist of a root node colored in blue and a set of subtrees attached to that root node colored in green. All of these subtrees have the exact same structure and represent different data records of the same entity. The tree edit distance with a standard approach will be very high between those trees, due to the fact that there are just so many more nodes in the second tree. However, if considering only the structure of both tree one could expect the structural similarity measure to be high, since structure wise both trees have a root and a certain number of structurally identical subtrees. With only the standard edit operations there are unfortunately no ways to grasp this similarity. That’s why we introduce two extended edit operations, the insert and delete subtree operations, that will help us reduce the tree edit distance between two trees as shown above and therefore result in a higher structural similarity measure. Of course it makes no sense to say that both trees in Figure 3.2 are identical structure wise, but they should be considered close. A more detailed explanation of the extended operations can be found in the following Section 3.3.2.

3.3.2 Extended Operations

As explained earlier there is a need for extended operations to better handle specific cases like optional elements and repeated structures in XML and HTML trees. Nierman and Jagadish [9] describe two additional operations for these events, the insert and delete subtree operations. These operations have been presented in Section 2.3.1 and a visualisation is given by Figure 2.9. The extended operations allow us to have a much higher similarity measure between trees that are different but have similar structures, like the trees in Figure 3.2. The extended operations also handle another problem specific to XML and HTML trees, that
was introduced by choosing a top-down distance. On web pages, it happens quite often that different elements are put into a wrapper element so that they obey to some styling rules. However, this wrapper introduces a new high level node which will be considered as part of the structure. Nonetheless these wrappers are usually invisible and only used to apply the desired styling to the element. If we consider the tree edit distance between two trees with the standard edit operations only and stumble upon two identical subtrees with the exception of a wrapper element around one and not the other. Then we would have to first delete the subtree one node at a time, just to delete the wrapper element and then insert the subtree again one node at a time. Depending on the size of the subtree the tree edit distance would be huge and the two trees would never be considered as similar. With the two extended operations however, this special case can be solved with a sequence of three simple operations. First we delete the subtree, then we delete the wrapper element and finally we reinsert the whole subtree. This will result in a tree edit distance of 3, assuming unit costs for each operation and thus both trees are estimated to be much closer to each other.

3.4 Restrictions

Our previously presented tree edit operations have to obey to a set of rules. These rules are mainly restrictions of when, where and how the edit operations can be used. We have
two types of restrictions, restrictions imposed by our mapping and restrictions we introduce ourselves. By opting for the top-down distance and thus a top-down mapping approach, we force restrictions, presented in Section 2.1.2.3, upon the standard edit operations, insert and delete node. The set of restrictions, we introduce ourselves apply to the extended operations, insert and delete subtree. These have been described in Section 2.3.1 with the Definition 3.

While the allowable sequence definition intuitively solves the problem of repeated structures, there is still the problem of optional elements. However, the allowable sequence definition also handles this problem. When saying that a subtree that is to be inserted or deleted is shared between the source and destination tree, it means that the subtree is contained in the source tree for the insert operation and similarly the subtree is contained in the destination tree for the delete operation. This notion of contained in is defined by Definition 5.

**Definition 5 (Contained In)**

A pattern tree $P$ is contained in tree $T$ if:

1. All nodes of $P$ are present in $T$
2. All nodes of $P$ have the same parent/child edge relationships in $T$
3. The order of siblings in $P$ is the same as in $T$

In other words, the definition forces that all nodes are present in both trees and that their hierarchy and order is respected. However, this makes it possible to have additional nodes in $T$ between sibling nodes of the subtree in $T$ mapped to $P$ as long as the order between siblings that occur in $P$ and in $T$ stays the same. An example to illustrate the contained in concept can be seen in Figure 3.3.

Consider the pattern tree $P$ and the trees $A$, $B$ and $C$. The pattern tree $P$ can exactly be found in tree $A$ and therefore is considered to be contained in $A$. In the case of tree $B$, tree $B$ and $P$ are identical with the exception of the extra node $z$ in $B$. However, node $z$ is just an additional sibling among the first-level children of the root and does not compromise the hierarchy or the order of the common nodes between $B$ and $P$. Thus $P$ is contained in $B$. Finally for tree $C$, while all nodes are present the parent/child relationship is not the same for tree $C$ and for tree $P$, thus $P$ is not contained in $C$. This example clearly proves that the allowable sequence definition also handles the problem of optional elements. Therefore we can now consider that both extended operations together with our restrictions are suitable to analyse structural similarities between trees, especially XML and HTML trees.
3.5 Cost Model

Having established a way to describe the transformation from one tree into another with a sequence of edit operations, the only missing piece needed to be able to compute the edit distance between two trees is a cost model. This cost model should define the cost associated to each operation. In general each operation will be assigned the same basic cost, however it is also possible to give each individual operation a different cost. Most existing approaches work with a static cost model, but [2, 6] have shown that having a flexible cost model can be beneficial. Accordingly Section 3.5.1 presents our decay factor, which is used to generate a flexible cost model.

3.5.1 Decay Factor

When using a static cost model each operation gets the same cost assigned and any differences regardless of where they appear in the tree are assigned that cost. This is of course a huge limitation. As we mentioned before to improve the results of a structural analysis, we think it is beneficial to handle differences in the tree differently depending where they appear, in the structure or the content. Therefore, we introduce a new flexible cost model based on the decay factor. With this flexible model we want to better handle the problem of giving differences in the trees weights based on their location. Principally we want to reduce the importance
of differences in the content and focus more on differences in the structure when comparing two trees. Accordingly our decay factor will be applied to the cost of each operation based on their location. This decay factor is the weight we give to the operation and is called that way because the cost decays the lower you go in height. Of course the decay factor needs to be flexible too and might change depending on the type of trees we are analyzing. Yet before we can apply a weight to an operation, we need to figure out how to do so. We opt for the solution where we assign each operation a fixed cost and multiply that cost with the decay factor. The fixed cost can be seen as the static cost model that we use as skeleton for the flexible cost model. We allow each operation to have an individual cost for each of the standard edit operations and for the extended operations a cost influenced by its counterpart in the standard operations. In other words, the cost of the insert subtree operations and delete subtree operation are related to the cost of the insert node respectively delete node operations. In [9] the cost of the extended operations is identical to cost of the standard operations. We introduce a slightly different cost computation for the extended operations. Consider the two trees in Figure 3.4.

![Tree A and Tree B](image)

Figure 3.4: Example for the extended operations cost model.

With Definition 5, it is clear that tree A is contained in tree B even though tree B has a few extra elements. The cost model of [9] allows to insert or delete tree A if tree B is shared between source and destination tree for the same cost as a single node insert or delete operation. However, we think that the differences should also be accounted for. Even though we allow additional nodes in our contained in definition, as long as they don’t interfere with the hierarchy and order of the nodes in the pattern tree, such as the two green subtrees in tree
B, at some point if there are too many additional nodes, the difference between the two trees becomes too large to not take it into consideration. Therefore, the cost of the insert and delete subtree operation will be defined by Definition 6.

**Definition 6 (Extended Operations Costs)**

For a pattern tree $P$ and a tree $T$, if $P$ is contained in $T$ then the cost of the insert subtree operation is equal to the cost of inserting a single node and the number of differences between $P$ and its corresponding pattern in $T$.

Similarly, the cost of the delete subtree operation is equal to the cost of deleting a single node and the number of differences between $P$ and its corresponding pattern in $T$.

This leads to having the same cost as in [9] if the pattern tree is exactly found in the tree. In any other case, a small additional cost based on the number of extra elements is added to take into account the differences between both trees. So for the trees in Figure 3.4, our approach will add an additional factor of 4 to the cost of a single node insert or delete operation for the insert and delete subtree operations.

Now that the skeleton has been established, we can focus on the decay factor. There are several ways how a decay factor could be introduced. Be it a linear decay, an exponential decay or a fixed decay factor once the content levels are reached. At this point we will present the decay factor we used for the analysis of web pages. Note that this is just an example and that any type of decay factor could be inserted into our approach. When analysing web pages we have chosen to use a linear decay factor. The reason is that with linear decay we can slowly but steadily introduce and increase the decay factor. We think that in the case of web pages the exponential decay is just too strong and that the fixed decay factor does not give us enough flexibility. Our decay factor can be described by a linearly decreasing function. For HTML trees it makes sense to use this kind of decay function since the content is located in the leaf nodes and it is the cost of operations to handle differences in the content that we want to decay. To better understand our decay factor, let us introduce the notions of leaf distance, strong decay level and no decay level with the Definitions 7, 8 and 9.

**Definition 7 (Leaf Distance)**

The leaf distance of a node is the distance of that node to the furthest leaf node in the same subtree.

**Definition 8 (Strong Decay Level (SDL))**

The strong decay level is a specified value that marks the level in the tree at which pure content elements are to be expected. All nodes with a leaf distance less or equal to the strong decay level have the costs of their edit operations decay to 0.

**Definition 9 (No Decay Level (NDL))**

The no decay level is a specified value that marks the level in the tree at which pure structure elements are to be expected. All nodes with a leaf distance greater or equal to the strong decay level pay the full cost for any edit operation.

Our decay factor starts decaying at the NDL and linearly decreases until it reaches the SDL where it reduces all costs to the minimum 0. Based on the Definitions 7, 8 and 9 we can establish a decay factor function, defined by Definition 10. Figure 3.5 shows the progress of the decay factor function based on the leaf distance.
Definition 10 (Decay Factor Function)

Given a node \( n \), its leaf distance \( l \), SDL and NDL:

\[
decay\_factor(n) = \begin{cases} 
1 & \text{if } l > NDL \\
0 & \text{if } l \leq SDL \\
1 - \frac{NDL-l+1}{NDL-SDL+1} & \text{else}
\end{cases} 
\]  

(3.1)

For simplicity, the value of the decay function will always be rounded to the first digit after the decimal point. \(^1\)

![Graph of the progress of the decay factor.](image)

By fixing the limit level with the NDL we can mark the nodes in the tree that might be content and differences in those nodes should be less influential on the similarity measure. On top of that the SDL indicates the nodes that are considered pure content and have nothing to do with the structure of the tree and therefore should not influence the similarity measure at all. Note that these two values need to be given before computing the decay factor. We consider it wise to analyse the trees that are to be compared and fix both levels based on those trees. For web pages, we use the following method to derive the no decay level and strong decay level. We consider pure content elements and pure structure elements. As pure content elements we define elements with the tags:

- A
- H1
- H2
- H3
- H4
- H5
- H6
- IMG
- P
- SPAN

Pure structure elements need to have one of the following tags:

- DIV
- LI
- OL
- TABLE
- TD
- TH
- TR
- UL

\(^1\)The \( \geq \) sign has been corrected to a \( \leq \) sign in the second condition of this definition. This is a correction compared to the first version of October 20th.
For all elements in the HTML tree with a content tag, we compute the leaf distance and average them together, which will be set as the SDL. The elements with structure tags are used to compute the NDL. However not all of them are wanted, we limit ourselves to those that we estimate to be part of the structure of data records on the web page. We have evaluated and observed multiple web pages, presented in detail in Section 5.1.1, and estimate that the average subtree structure representing a data record has a leaf distance from the root between 3 and 8. The root of an HTML tree, representing a whole web page, has on average a leaf distance of 20. Therefore, many elements with higher leaf distances are structural elements of the whole page but not of different data records and for our specific use case those elements do not need to be considered. Unfortunately there are no statistics available for this kind of data, so our values are based on our observations. Nonetheless, these values can be configured depending on the user’s need, as they are simple parameters of our approach. Taking into account only the limited amount of elements with structure tags, we compute the NDL similarly as the SDL with the average leaf distance. For the rare event where the SDL is greater or equal to the NDL, we set $SDL = NDL - 1$. This occurs only for small trees containing almost only content and having very little structural elements. An example of a tree with the leaf distance and the decay factor can be seen in Figure 3.6. For the example NDL is set to 3 and SDL is set to 1.

![Figure 3.6: An example the leaf distances and decay factors of a tree.](image)

### 3.6 Classification

The previous sections present our new approach and the way how we introduce each desired features. All together the different features give us a way to compute the tree edit distance between two trees. The remaining question is, what to do with that value? Is it that intuitive that anybody being presented two trees and their tree edit distance knows how similar they are? We don’t think that it is straightforward. We agree that the tree edit distance gives some notion of similarity. However we think it is necessary to show how our tree edit distance can be used to give a clear understanding of similarity.

Let’s consider again the case of analysing web pages. We are looking for data records of similar structure. Therefore, we need a sample tree that represents the structure of such a
data record and then look for similar trees in the web page. It is obvious that we use our approach to compute the tree edit distance between our sample tree and all subtrees in the web page. Once we have all the tree edit distances, we still need a way to specify which trees are similar and which not. In other words we need a way to classify the trees based on their tree edit distance. There are multiple ways to do this, be it with a fixed value for any two trees or a value depending on the set of trees that we are analysing. We think that it makes no sense to have fixed value, let’s say 3, as a limit to classify that any two trees with a tree edit distance of less than 3 are similar and else not. This classifying value should be influenced by the type of trees that are analysed. In our case it makes sense to consider the structure of the web page, containing all the trees that are analysed, as well as the sample tree itself. Parameters such as SDL, NDL or the sample tree size can all make sense to be used while fixing the classification threshold. With these ideas in mind, we have decided that we need a classification threshold function based on some of these parameters as well as sample data. Therefore, we have analysed multiple web pages and applied our approach to detect data records. More details about the set of pages can be found in Section 5.1.1. Since we decide what sample tree to use for each page, we can establish a ground truth containing all the data records that should be returned. Note that since there is no existing approach for this problem yet, that returns all wanted data records in all cases with a 100% accuracy, this ground truth is manually generated and therefore only an approximation in some cases. The two graphs in Figure 3.7 present our gathered data for two example web pages. The x-axis displays the subtree id and the y-axis the tree edit distance between the subtree and the sample tree. Marked with yellow dots are subtree in our ground truth that should be returned and orange are all the other subtrees.

Figure 3.7a shows that our ground truth approximation is rather good, all yellow dots have a very low tree edit distance value and are clearly separated from the other dots. Figure 3.7b shows some problems that we have encountered with our ground truth. Most of the yellow dots have a low tree edit distance, which is good, however there are a few outliers and also a few orange dots are mixed in between the yellow ones. After further observations, we have figured out that the yellow outlier dots display similar content than the others, however its structure is different and should therefore not be returned. In contrast, the orange dots represent subtrees that either show something completely different content-wise but have a very similar structure to our sample tree or are similarly structured data records that also have similar content but that we missed. This shows the weakness of our ground truth estimation, we can only make the assumption that data records that look similarly in the rendered web page have a similar structure. To further increase accuracy we could also have a detailed look at the HTML tree. More details about the exact procedure of our ground truth estimation can be found in Section 5.1.1.

However, based on our established ground truth we can still give a good approximation for the classification threshold. The idea is to find a function, using parameters such as the sample tree size, that fits our gathered data. We have tried to manually generate such a function to fit our data and have found reasonable success. The function is given by Equation 3.2 and uses as parameters $STS$ the sample tree size and $DN$ the number of decayed nodes.

$$\text{threshold}(STS, DN) = \frac{|\log 6DN - \sqrt{DN}|}{2} - \frac{DN}{STS}$$ (3.2)
Nonetheless, it makes more sense to have such a function generated from a tool that specialises in generating functions to fit certain data. The tool we have used is Eureqa \(^2\). Using the tool is simple, we just need to input our data, specify our parameters, as well as mathematical

\(^2\text{http://www.nutonian.com/products/eureqa/}\)
3.7 Summary

Combining everything together that has been presented in this chapter so far, results in our new approach. The process diagram of our approach can be seen in Figure 3.8.

First we need to initialise our approach. This is done by providing the two trees that are to be compared and a set of parameters to our approach. The complete list of parameters will be discussed in detail in Chapter 4. The next step after the initialisation is the pre-processing step. This step computes the optimal cost to insert or delete each subtree in both the source and destination tree. Again, for details about the pre-processing step refer to Chapter 4. The third step is the main step of the approach. During this step the actual tree edit distance is computed. This is done with a dynamic programming algorithm given by Algorithm 5. It is a
standard dynamic programming algorithm that finds the minimal cost sequence of operations to transform one tree into the other. The costs that have previously been calculated serve as guidance as to which operation should be chosen. At the end the algorithm returns the tree edit distance between the two trees. The last step on the process diagram is an optional post-processing step. It is only required if an immediate classification is desired. Since at this point in the process the tree edit distance between the two trees has been computed. One simply needs to go over the computed tree edit distance and check if it matches the classification criteria. If this is the case those trees get classified as similar and if not then those trees are classified as not similar. The runtime complexity without any special pre- and post-processing is the same as the algorithm from [9], in particular $O(n \times m)$.

Algorithm 5 TED(T1, T2)

1: $m :=$ the number of first-level subtrees of $T_1$.
2: $n :=$ the number of first-level subtrees of $T_2$.
3: $D[0,0] := Cost_{relab}$(root of $T_1$, root of $T_2$)
4: for $i := 1$ to $m$ do
5: \hspace{1em} $D[i,0] := D[i-1,0] + Cost_{delete}(T_1_i)$
6: for $j := 1$ to $n$ do
7: \hspace{1em} $D[0,j] := D[0,j-1] + Cost_{insert}(T_2_j)$
8: for $i := 1$ to $m$ do
9: \hspace{1em} for $j := 1$ to $n$ do
10: \hspace{2em} $D[i,j] := \min(D[i-1,j-1] + TED(T_1_i,T_2_j), D[i,j-1] + Cost_{insert}(T_2_j), D[i-1,j] + Cost_{delete}(T_1_i))$
11: \hspace{1em} return $(D[m,n])$

It follows a detailed explanation of the architecture and implementation as well as an evaluation of the framework using the here presented approach.
This chapter concentrates on the architecture and implementation of the framework built around our new approach. We will give a detailed explanation of the different modules of the framework and how the whole working process works. We also give more insights to the initialisation of our framework and all the different parameters that can be adjusted depending on the use case. But first we present our tree representation, in which any two trees that we want to compared need to be represented in.

4.1 Tree Representation

We have chosen to introduce our own tree representation together with our framework. There are several reasons why we do this. First for simplicity, having a fixed tree representation means that we know exactly what properties and attributes of the trees are available. Consider for example the case of the label of a node. In some tree representations it might be accessible under the attribute called label in others maybe under the attribute called name. For each tree depending on its representation, we would need to specify how to access any property or attribute for our framework. This cumbersome task can be avoided by introducing our own tree representation. Another benefit of representing any tree with our tree representation is that obsolete data that is contained in the original representation of the tree can be discarded. Similarly by reconstructing the input trees in our tree representation, we have the possibility to add additional information to each node or tree that is needed by our approach. This is especially useful for extra information such as the leaf distance or depth of a node and allows us to compute these values only once and store them with their respective nodes. Furthermore we consider our tree representation to be very simple and extendible in the case some user handles a very specific type of problems and additional functions or attributes are needed, then it is very straightforward how to extend our tree interface to adapt to those specific problems. This will be discussed in more detail in Section 4.4.
Our tree representation is composed of two entities. First a tree entity and second a node entity. The tree entity is used to represent a tree and can be seen in Figure 4.1a, while the node entity represents a node of a tree and is shown in Figure 4.1b. The attributes and their corresponding types can be found in Table 4.1 for the tree entity and in Table 4.2 for the node entity. A few of the attributes might need an additional explanation but most of them should be straightforward. These would be the `labelList` attribute of the tree entity which is the list of all the labels that at least one node in the tree have and for the node entity the `containedIn` and `size` attributes. The first will be explained in detail in Section 4.3 and the second is the size of the subtree rooted in that specific node.

![Diagram](image)

(a) Tree entity.

![Diagram](image)

(b) Node entity.

Figure 4.1: Internal Tree Interface.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>labelList</td>
<td>list[String]</td>
</tr>
<tr>
<td>postOrder</td>
<td>list[Node]</td>
</tr>
</tbody>
</table>

Table 4.1: Attributes of the tree entity.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>list[Node]</td>
</tr>
<tr>
<td>containedIn</td>
<td>list[Node]</td>
</tr>
<tr>
<td>cost</td>
<td>Float</td>
</tr>
<tr>
<td>depth</td>
<td>Integer</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
</tr>
<tr>
<td>leafDistance</td>
<td>Integer</td>
</tr>
<tr>
<td>parent</td>
<td>Node</td>
</tr>
<tr>
<td>root</td>
<td>Boolean</td>
</tr>
<tr>
<td>size</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Table 4.2: Attributes of the node entity.

4.2 Tree Interface

We also provide a tree interface that allows the user to transform a tree regardless of its representation into our tree representation. The user simply needs to specify the tree to be transformed, the name of the attribute where the label is stored and the name of the attribute or method to access the children. Additionally the interface provides numerous functions, such as getters and setters for the different attributes and often recurring queries, for example retrieving a certain subtree or checking the containedIn list. As an example consider and HTML tree. In HTML trees the label of the node actually called tag and can be accessed under the tagName attribute. The children can be accessed by the children method. So to transform an HTML tree into our tree representation we simply provide the tree and specify the label and children attributes with tagName and children respectively.

4.3 Architecture

The goal was to have a lightweight but powerful and flexible framework that can easily be inserted into many different applications. It is also important that other users can easily understand the concepts and the workflow behind the framework so that they can more easily use it and extend it if needed. Therefore, we have chosen to split up our framework into four different modules that are each responsible for a certain step in the process of the approach shown in Figure 3.8. Those modules are the initialisation module, pre-processing module, tree edit distance module and post-processing module. The diagram of the architecture seen in Figure 4.2 is therefore strongly related to the process diagram of Figure 3.8.

There is one module assigned to each step of the process. The initialisation module is paired with the initialisation step of our approach. It contains all sorts of methods and procedure that handle the user’s input and configure every parameter of the framework for the computation of the tree edit distance later. Table 4.3 shows all the different parameters that can be configured. Note that it is possible to configure each parameter manually by providing additional user input. However, for the problem of detecting data records on web pages, we also provide a method that automatically adjusts the parameters based on a
few user inputs. Whether a parameter needs to be specified manually or is generated automatically is also specified in Table 4.3. To initialise a parameter manually, the user simply needs to set the public field of that parameter in the framework, `MultiComp.NDL = 4;`. In case the user wants to use the initialisation method for the detection of data records, this can be done as follows `MultiComp.initialise("children", ["div"], ["a","p"], "tagName", HTML_tree, 8, 3, "Linear", 1, 1 );`. The last step of the initialisation step is to specify the source and destination tree. During the initialisation step the given trees are also transformed into our tree representation.

Once that all parameters are configured and the source and destination tree are set, the functions of the pre-processing module are used to execute the pre-processing step. This step is done automatically and essentially applies our cost model to both trees. This is done by computing the `containedIn` lists for all subtrees followed by the optimal cost for each subtree. We need the `containedIn` lists to compute the optimal cost, because for each subtree we need to compare if its cheaper to use a combination of single node operations or a subtree operation and we only know whether we can use the subtree operations if we have the `containedIn` lists. The algorithm behind the computation of the `containedIn` lists uses a bottom-up approach and is given by Algorithm 6. We start from the leaf nodes and step by step go up to the root. This helps us reduce the number of computations we need to do because if the children of a subtree are not contained in any other tree than that subtree won’t be contained in those tree neither.
### Table 4.3: Parameters for the initialisation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Initialisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>childrenMethod</td>
<td>String</td>
<td>Manual</td>
</tr>
<tr>
<td>containerLabels</td>
<td>list[String]</td>
<td>Manual</td>
</tr>
<tr>
<td>contentLabels</td>
<td>list[String]</td>
<td>Manual</td>
</tr>
<tr>
<td>labelMethod</td>
<td>String</td>
<td>Manual</td>
</tr>
<tr>
<td>page</td>
<td>Tree</td>
<td>Manual</td>
</tr>
<tr>
<td>noDecayMax</td>
<td>Integer</td>
<td>Manual</td>
</tr>
<tr>
<td>noDecayMin</td>
<td>Integer</td>
<td>Manual</td>
</tr>
<tr>
<td>decayType</td>
<td>String</td>
<td>Manual</td>
</tr>
<tr>
<td>relabelCost</td>
<td>Float</td>
<td>Manual</td>
</tr>
<tr>
<td>insertAndDeleteCost</td>
<td>Float</td>
<td>Manual</td>
</tr>
<tr>
<td>NDL</td>
<td>Integer</td>
<td>Manual or Automatic</td>
</tr>
<tr>
<td>SDL</td>
<td>Integer</td>
<td>Manual or Automatic</td>
</tr>
<tr>
<td>decayFactor</td>
<td>Function</td>
<td>Manual or Automatic</td>
</tr>
<tr>
<td>threshold</td>
<td>Float</td>
<td>Manual or Automatic</td>
</tr>
</tbody>
</table>

Algorithm 6 findContainedIn(T1, T2)

1: if \( T1 = \text{leaf node} \) then
2: \( \text{containedIn} = \text{list of all nodes with the same label} \)
3: \( \text{else} \)
4: \( \text{containedIn} = \text{list of all subtrees of T2 for which containedIn}(T1, \text{subtree of T2}) = \true \) by Definition 5

Algorithm 7 presents the algorithm to compute the optimal costs of a subtree. The calculation of the costs is also done in a bottom-up fashion. For each node, we calculate the cost of inserting or deleting that node and add the already calculated cost of all its children. Additionally if its containedIn list is not empty we compute the insert and delete cost of the subtree rooted in that node with the extended operations. The lowest of the two costs is chosen.

Algorithm 7 findOptimalCost(N)

1: \( \text{standard cost} = \text{MultiComp.insertAndDeleteCost} + \sum_{k \in \text{first-level subtrees of } N} k.\text{cost} \)
2: if \( N.\text{containedIn} \) is not an empty list then
3: \( \text{extended cost} = \text{MultiComp.insertAndDeleteCost} + \# \text{ of differences between } N \text{ and best match in } N.\text{containedIn} \)
4: if \( \text{standard cost} > \text{extended cost} \) then
5: \( \text{return extended cost} \)
6: \( \text{return standard cost} \)

After the computation of the optimal costs the tree edit distance module is initiated. The tree edit distance is computed based on the algorithm presented in Section 3.7. In case some more processing is needed the post-processing module can be initiated to do so. A detailed example will be given in Section 4.4. It is worth noticing that all of the modules together offer
4.4 Extensibility

Our framework mainly offers the basic functionalities to compute the tree edit distance between two trees as seen in Section 4.3. However, extending the framework is very simple. The user essentially only needs to extend the functionalities of the different modules. There are mainly two modules that can be extended the pre-processing and post-processing module. These two modules should allow anybody to specialise the framework in such a way to handle any type of problem related to the tree edit distance. Let’s consider the example of detecting data records in web pages. The extra pre-processing that needs to be done is to analyse the web page to find the best possible values for NDL and SDL, as well as analyse the sample tree to set the classification threshold. This is done by two simple functions and is included in the initialisation method we provide. The post-processing that needs to be done is also straightforward. Based on the computed tree edit distances we must classify which subtrees are matched and which not. Algorithm 8 presents the function needed to do the post-processing.

Algorithm 8  post_processing()
1: match = []
2: for all Subtrees in MultiComp.page do
3:  MultiComp.initialiseTrees(subtree,sample tree) ▷ Initialisation
4:  MultiComp.containedIn()       ▷ Pre-processing step 1
5:  MultiComp.costs()                ▷ Pre-processing step 2
6:  distance = MultiComp.ted()     ▷ TED
7:  if distance < MultiComp.threshold then
8:     match.push(subtree)
9:  return match

It is also worth noticing that the tree representation can be changed to include any wanted attribute for either the tree or node entity. The user just needs to specify the corresponding function that computes the value for that attribute.

4.5 Implementation

As for the implementation of our framework, since we focus on web related problems we have chosen to use web related technologies. To keep the concept of a lightweight framework, the framework is written in JavaScript, the standard client side scripting language for the web. It can be inserted into any existing web page or web application without the need of a backend server. We also think that the framework will mainly be used as part of an application so we keep any external dependencies to the minimum. Therefore, we use only plain JavaScript to implement our framework and don’t force the use of any other existing frameworks or
plugins. Of course our framework is developed in such a way that it still works with existing frameworks and causes no problems if used in combination. This is achieved by using our own namespace and defining any function in that namespace to avoid naming collisions.
Evaluation

With the evaluation of the framework, we mainly want to investigate two different things. First, the quality of the results our framework produces in comparison to existing solutions and second the fitness for a multitude of purposes of our framework. Therefore, in a first step, we consider a similar scenario as in [8] for the DeepDesign tool. That scenario consists of detecting data records on web pages. In Section 5.1, we will provide a comparison of our decayed tree edit distance, a non-decayed tree edit distance and the distance measure PQ-Gram [1], because it is used in the DeepDesign tool. To show the flexibility of our framework to adapt to multiple use cases, Section 5.2 presents the use of our framework together with Archipelago 2.0, where the goal is to compare different snapshots and show differences between them.

5.1 DeepDesign

For each web page, we select one sample data record, representing a certain data entity, and look for similarly structured data records. In other words, our goal is to find all the data records representing the same data entity since we assume that they will all have a similar structure. For each web page a ground truth will need to be established, so that we can compute the precision and recall values to judge the quality of the results. Since we can’t judge the quality of our results solely based on results generated by our decayed tree edit distance, we also need to compare it to different distance measures. The main novelty of our approach is the decay factor, which is why we are going to compare our decayed tree edit distance against a non-decayed version. The non-decayed tree edit distance has all the features mentioned in Chapter 3, such as the extended tree edit operations, except for the decayed cost model, which is replaced with a static cost model. This comparison will clearly show if the decay factor has any effect at all and if yes whether it is beneficial or not. A comparison between the decayed tree edit distance and an existing tree edit distance would...
also be beneficial as we would have a comparison to a completely different approach that has not been developed by us. Considering that our use case for the evaluation is taken from [8], it makes sense to make the comparison against the distance measure used in the DeepDesign tool of that paper, which is PQ-Gram. PQ-Gram is a tree similarity measure that uses an approximation of the tree edit distance. As DeepDesign uses the Javascript implementation of pq-gram, called jq-gram ¹, we are also going to use that implementation.

Having fixed our three edit distances, the decayed tree edit distance (D-TED), normal tree edit distance(N-TED) and PQ-Gram(PQ-TED), we need to specify how exactly our evaluation process works. The idea behind our evaluation can be described as a multi-step process. The different steps are summarized in following list.

1. Select a web page
2. Select a sample tree
3. Compute TEDs
4. Classify TEDs
5. Establish ground truth
6. Judge quality

The first step of the process is obvious, we just need to select a web page where we want to detect some data records. For the second step, we need to specify a sample tree that is structurally similar to the data records we want to retrieve. To specify a sample tree, our framework gives us the opportunity to specify the sample tree with a set of CSS rules. In other words, we can identify a wanted data record on the web page and give the exact CSS rules needed to retrieve that data record. This data record will then become the sample tree. Of course we can also give the framework a random sample tree as long as it obeys to our tree interface. Once the sample tree is selected, we compute the distance between the sample tree and all the trees in the web pages with our three distance measures, D-TED, N-TED and PQ-TED. The following step will then analyse the obtained results and classify each tree as similar or nt similar to the sample tree. This classification is done with a classification function, the details of how we obtained that function are presented later in Section 5.1.1. Before we can judge the quality of the results, we need to establish the ground truth first. This means, we need to establish which trees in the web page are similar to the sample tree and which are not. As mentioned in Section 3.6, there is no existing approach to do this, which is why we need to do it manually. The way we do it, is that we look at the CSS class of the sample tree and assume that all trees with that class have a similar structure. This is a reasonable assumption but of course there might always be an exception, which is why our ground truth stays an approximation. However, we can increase the accuracy of our ground truth by also checking the HTML when establishing it and verifying if each tree with the same class actually has a similar structure. To guarantee a high accuracy of our ground truth we did exactly that. Now that our ground truth is established we can conclude the evaluation process by judging the quality of our results. With our ground truth and classified trees, we

¹https://github.com/hoonto/jqgram
can compute the precision and recall of the approach and thus have a metric to compare the
different approaches.

As mentioned in Section 3.6, we need a classification function for the evaluation. But there
isn’t one available that we could use. We therefore separate our evaluation into two phases, a
training phase and a test phase. During the training phase, we will use a training set of web
pages that will help us generate a classification function. In the second phase, we will then
use a test set and use the classification function we obtained during the training phase to test
the different distance measures. Details of the training phase a presented in Section 5.1.1 and
the test phase is presented in Section 5.1.2.

5.1.1 Training Phase

The main goal of the training phase is to generate a classification function to be used during
the test phase to judge the quality of the results. However, depending on the use case there
may be different ways to judge the quality of the results based on the precision and recall
values. Therefore, it would be desirable to have multiple classification functions, one for
each way of judging. We think that there are mainly three goals of quality that an approach
can achieve, namely having maximum precision, maximum recall or optimal precision and
recall, given by Equation 5.1.

\[
\text{optimal}(P, R) = \text{Max}(P + R)
\]  

This is why for each distance measure, we establish three different classification functions
during the training phase. The training set that we want to use to build these classification
functions has been put together by us. It contains 20 web pages, mostly pages that are among
the top 20 Alexa\footnote{http://www.alexa.com/topsites} sites. There are a few exceptions among them due to various reasons, for
example Google is represented multiple times in the top 20 for different regions and it makes
no sense to have two identical pages with exception of the language in our training data set.
The complete list of our training set can be seen in the summary table Table 5.2 at the end
of this section. These 20 pages will be used as input to generate the classification functions.
For each page, we need to select a sample tree. Therefore, we have analysed all the pages
and carefully selected one sample tree for each web page. We made sure that our training set
contains sample trees of different sizes, since we don’t want our classification function to be
specific to only small or big sample trees. We want a classification function that works on a
wide range of web pages, without having any restrictions. Each sample tree represents a data
entity which is represented multiple times, so that we can see how many of them we actually
retrieve. Two of the pages in our training set are Reddit and Twitter. For Reddit, we selected a
post as a sample tree, as for Twitter, we chose a tweet as a sample tree. Both data records are
marked with a red box in Figure 5.1a and Figure 5.1b respectively. The CSS class to identify
the sample for each page in the training set can be seen in Table 5.2.

We then visualise the data to find the boundaries for the classification threshold. An example
visualisation of the data for each distance measure for Twitter can be seen in Figure 5.2. The
x-axis contains the ids of the trees of the web page in post-order and the y-axis shows the
distance between the sample tree and the candidate tree. The yellow dots represent trees that
5.1. DEEPDESIGN

Figure 5.1: Examples of samples trees for Reddit and Twitter. 

are hits in our ground truth and orange dots represent trees that are misses in our ground truth. We can see that there is a separation into two different clusters, one mainly consisting of hits and one mainly consisting of misses. This means that we can easily identify the boundaries between which our classification threshold should be. Since we have to establish three different classification functions, we need to define three sets of boundaries to best fit the goals. We search these sets of boundaries for each page in the training set. Then we use a tool called Eureqa, as we mentioned in Section 3.6, to find the classification functions for each distance measure. It specialises in finding functions that fit the data it is fed. As also mentioned earlier, several parameters could influence the classification functions. At this point we have chosen to use the following three parameters: the sample tree size (STS), the number of decayed nodes in the sample tree (DN) and the cost that is decayed in the sample tree (DC). They are all influenced by the sample tree as well as the whole page, since the sample tree is part of it. These three parameters will be input values for the classification function of the D-TED. Since there is no decay in both N-TED and PQ-TED, it makes no sense to use DN and DC and therefore only STS is used. The different values for the parameters STS, DN and DC for each page can also be seen in Table 5.2. All the classification functions generated during the training phase can be seen in Table 5.1.
Distance Measure | Quality Measure | Classification Function
--- | --- | ---
D-TED | Max P | \( \sin(-2.12 \times STS) + 0.03 \times STS \)
D-TED | Max R | \( 0.55 + 0.22 \times DC - 0.25 \times DN - 0.06 \times \tan(0.06 \times STS) \)
D-TED | Optimal | \( 0.39 + 0.03 \times STS \times \cos(-1.04 \times STS \times DN) - 0.05 \times STS - 0.04 \times \cos(-57.12 \times STS) \)
N-TED | Max P | \( \sin(5.72 + STS) + \tan(0.11 \times STS - 0.73) - 3.57 \)
N-TED | Max R | \( 4.05 \times \cos(1.59 \times STS + \tan(-0.02 \times STS) - 0.41 \times STS) \)
N-TED | Optimal | \( 0.17 \times STS \times \sin(3.68 + 3.11 \times STS) - 1.79 - 0.17 \times STS \)
PQ-TED | Max P | \( \frac{6.52 \times \sin(STS) + \tan(STS - 0.77) - STS}{14.74 + STS} \)
PQ-TED | Max R | \( - 0.71 - 0.15 \times \sin(4.04 + 59 \times STS) \)
PQ-TED | Optimal | \( 63.18 + 19.18 \times \cos(3.38 + STS) + \tan(5.52 + STS) \)

Table 5.1: Classification functions for the different distance measures and quality measures.

<table>
<thead>
<tr>
<th>Page</th>
<th>CSS Class</th>
<th>Id</th>
<th>STS</th>
<th>DN</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>20min</td>
<td>clusterLeft</td>
<td>1</td>
<td>33</td>
<td>30</td>
<td>24.3</td>
</tr>
<tr>
<td>Amazon</td>
<td>feed-carousel</td>
<td>0</td>
<td>84</td>
<td>82</td>
<td>66.5</td>
</tr>
<tr>
<td>Ask</td>
<td>partial-category-questions-item</td>
<td>0</td>
<td>19</td>
<td>17</td>
<td>15.7</td>
</tr>
<tr>
<td>Baidu</td>
<td>result c-container</td>
<td>0</td>
<td>22</td>
<td>22</td>
<td>20.1</td>
</tr>
<tr>
<td>Ebay</td>
<td>big-hero</td>
<td>0</td>
<td>44</td>
<td>41</td>
<td>34.3</td>
</tr>
<tr>
<td>Facebook</td>
<td>_4-u2 mbm _5jmm _5pat _5v3q _4-u8</td>
<td>0</td>
<td>84</td>
<td>65</td>
<td>57.8</td>
</tr>
<tr>
<td>Google</td>
<td>g</td>
<td>1</td>
<td>19</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>Instagram</td>
<td>-ex-PRIVATE-PostsGrid_row</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>14.4</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>card</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>10.8</td>
</tr>
<tr>
<td>MSN</td>
<td>secondary</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>6.4</td>
</tr>
<tr>
<td>NY Times</td>
<td>split-3-layout</td>
<td>1</td>
<td>58</td>
<td>48</td>
<td>34.2</td>
</tr>
<tr>
<td>Pinterest</td>
<td>item</td>
<td>0</td>
<td>43</td>
<td>39</td>
<td>33.2</td>
</tr>
<tr>
<td>QQ</td>
<td>contentMod</td>
<td>8</td>
<td>58</td>
<td>57</td>
<td>48.6</td>
</tr>
<tr>
<td>Reddit</td>
<td>thing link</td>
<td>11</td>
<td>30</td>
<td>30</td>
<td>26.4</td>
</tr>
<tr>
<td>Taobao</td>
<td>channel-item</td>
<td>0</td>
<td>32</td>
<td>30</td>
<td>25.7</td>
</tr>
<tr>
<td>Tumblr</td>
<td>not_mine is_original post brick</td>
<td>0</td>
<td>79</td>
<td>72</td>
<td>60.2</td>
</tr>
<tr>
<td>Twitter</td>
<td>original-tweet</td>
<td>1</td>
<td>99</td>
<td>94</td>
<td>80.3</td>
</tr>
<tr>
<td>Weibo</td>
<td>WB_cardwrap WB_feed_type S_bg2 WB_feed_vipcover</td>
<td>0</td>
<td>55</td>
<td>48</td>
<td>40.7</td>
</tr>
<tr>
<td>Yahoo</td>
<td>content voh-parent cf</td>
<td>0</td>
<td>26</td>
<td>25</td>
<td>20.3</td>
</tr>
<tr>
<td>Youtube</td>
<td>yt-shelf-grid-item yt-ui-x-shelfslider-item</td>
<td>0</td>
<td>44</td>
<td>37</td>
<td>32.2</td>
</tr>
</tbody>
</table>

Table 5.2: Training set data.
Figure 5.2: Visualisations of the data for each distance measure for Twitter.
5.1.2 Test Phase

We have built a test set of 10 web pages to evaluate the three different edit distances using our classification functions. The information about the pages, their sample tree and parameters is summarized in Table 5.3.

<table>
<thead>
<tr>
<th>Page</th>
<th>CSS Class</th>
<th>Id</th>
<th>STS</th>
<th>DN</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alibaba</td>
<td>f-icon m-item</td>
<td>0</td>
<td>97</td>
<td>87</td>
<td>81.4</td>
</tr>
<tr>
<td>CNET</td>
<td>riverPost</td>
<td>0</td>
<td>24</td>
<td>24</td>
<td>20.4</td>
</tr>
<tr>
<td>CNN</td>
<td>cn_column</td>
<td>0</td>
<td>13</td>
<td>11</td>
<td>9.1</td>
</tr>
<tr>
<td>CNN Sports</td>
<td>column</td>
<td>0</td>
<td>36</td>
<td>30</td>
<td>23.3</td>
</tr>
<tr>
<td>DBLP</td>
<td>entry editor</td>
<td>0</td>
<td>88</td>
<td>82</td>
<td>71.9</td>
</tr>
<tr>
<td>ETHZ</td>
<td>newListBox</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>7.4</td>
</tr>
<tr>
<td>Eurosport</td>
<td>rounded-content-container</td>
<td>0</td>
<td>32</td>
<td>26</td>
<td>23</td>
</tr>
<tr>
<td>IMDb</td>
<td>list_item</td>
<td>0</td>
<td>66</td>
<td>61</td>
<td>54.2</td>
</tr>
<tr>
<td>PC Mag</td>
<td>moderate-story</td>
<td>0</td>
<td>11</td>
<td>8</td>
<td>6.4</td>
</tr>
<tr>
<td>Stackoverflow</td>
<td>question-summary</td>
<td>0</td>
<td>24</td>
<td>23</td>
<td>17.6</td>
</tr>
</tbody>
</table>

Table 5.3: Test set data.

During the test phase, we follow the evaluation process of Section 5.1 for each page in the test set. This means for the first and second step, we select a page and pick a sample tree for it. We pick a sample tree with the same strategy as during the training phase. We want again multiple sample tree sizes and we want the data entity that is represented by the sample tree to be contained in the page multiple times. An example of a sample tree is a movie review for the IMDb page marked in Figure 5.3 by the red box.

The following step is then to compute the distance between the sample tree and all the trees in the page. For the D-TED and N-TED distance measures we use our MultiComp framework and for the PQ-TED we use the jq-gram implementation. The data we collect during this step is then analysed and for each combination of distance measure and classification function we classify each tree either as hit or miss. Next on the evaluation process is the computation of the ground truth. Once that is done we can visualise the data and add the classification threshold so that all dots below the threshold are similar trees and all dots above the threshold are not similar. As an example the visualisation of the data from the D-TED and the optimal classification function for the IMDb page can be seen in Figure 5.4. The results are of better quality the more yellow dots and the fewer orange dots are below the classification threshold, represented by the red line.

The last step is to compute the precision and recall for each combination of distance measures and classification functions. Finally, the average precision and recall for each combination can be computed. The results are presented in Section 5.1.3.

5.1.3 Results

Table 5.4 summarizes the precision and recall values averaged over the set of pages for each combination of distance measure and classification function. The color schema for the preci-
Figure 5.3: Sample data record for the IMDb page.

Figure 5.4: Visualisation of the data for the D-TED and the optimal classification function of the IMDb page.

The precision and recall columns indicates the quality of the results. Green is good, yellow is average and red is bad. We can see that the goal of the classification function is not always reached, for example with the D-TED and the maximal recall goal we actually achieve a worse recall value than with its alternatives. However, this is ok since the goal of our evaluation was to see how our D-TED performs compared to other distance measures. An explanation for
the failure of certain classification functions could be related to Eureqa. Eureqa uses a randomized algorithm to generate the functions, which in a few cases might lead to unexpected results. When comparing both the D-TED and N-TED distance, we see a clear improvement of results which means that the introduction of the decay factor has a beneficial effect on the distance measure. The precision improves by 23.2% and the recall improves by 16.3% on average over the three classification functions. Similarly there is also a major improvement when comparing the D-TED with the PQ-TED. In this case the precision improves by 29.3% and the recall by 15.7%. This shows that our new approach can compete and even achieve better results than this existing approach and thus it would be an improvement to use the D-TED instead of the PQ-TED for this specific use case.

<table>
<thead>
<tr>
<th>Distance Measure</th>
<th>Quality Measure</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-TED</td>
<td>Max P</td>
<td>98.3%</td>
<td>84.3%</td>
</tr>
<tr>
<td>D-TED</td>
<td>Max R</td>
<td>93.5%</td>
<td>83.4%</td>
</tr>
<tr>
<td>D-TED</td>
<td>Optimal</td>
<td>98.4%</td>
<td>91.0%</td>
</tr>
<tr>
<td>N-TED</td>
<td>Max P</td>
<td>86.8%</td>
<td>65.4%</td>
</tr>
<tr>
<td>N-TED</td>
<td>Max R</td>
<td>66.4%</td>
<td>89.5%</td>
</tr>
<tr>
<td>N-TED</td>
<td>Optimal</td>
<td>82.3%</td>
<td>67.5%</td>
</tr>
<tr>
<td>PQ-TED</td>
<td>Max P</td>
<td>76.5%</td>
<td>73.3%</td>
</tr>
<tr>
<td>PQ-TED</td>
<td>Max R</td>
<td>80.7%</td>
<td>72.2%</td>
</tr>
<tr>
<td>PQ-TED</td>
<td>Optimal</td>
<td>67.3%</td>
<td>78.1%</td>
</tr>
</tbody>
</table>

Table 5.4: Precision and recall values for the different distance measures and quality measures.

For each quality goal, be it maximum precision, maximum recall or optimal precision and recall, there is one best combination of distance measure and classification function. That combination is our D-TED with the optimal classification function. This combination outperforms any other combination in both precision and recall. This means that regardless of the quality goal the D-TED with the optimal classification function should be used. It is also worth noticing that any combination with our D-TED performs very well and nearly outclasses any combination with another distance measure regardless of the quality goal. This becomes clear when considering the average precision and recall values over the three classification function for each distance measure. The average values can be found in Table 5.5. We also think that there is one best possible combination of tree edit distance and classification function for each distance measure. For our distance measure the combination of D-TED and optimal classification function is best. Both the N-TED and PQ-TED perform best with their maximum recall classification function.
5.2 Archipelago 2.0

As mentioned in Chapter 1, another interesting use case of the tree edit distance is the comparison of branches of a versioning system. An example of such a system is Archipelago 2.0. In Archipelago 2.0, projects can be created from which different branches can emerge in case multiple people want to work on the same project without disturbing each other for example. However, being able to quickly see the differences between branches is a desirable feature that is not yet available. The data in Archipelago 2.0 has a very specific structure, it is always contained in a cell and the cells can be organized thanks to the groups. This concept of cells and groups can easily be transformed into a tree representation, as seen in Figure 5.5.

Since a branch in Archipelago 2.0 is nothing else than a container in which the current content of the project, in other words the data, is stored, a branch is nothing else than a collection of trees. Comparing two branches in Archipelago 2.0 is thus a comparison of trees, which can be handled by our framework. To be able to compare two branches to each other, we need

<table>
<thead>
<tr>
<th>Distance Measure</th>
<th>Average Precision</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-TED</td>
<td>96.7%</td>
<td>86.2%</td>
</tr>
<tr>
<td>N-TED</td>
<td>78.5%</td>
<td>74.1%</td>
</tr>
<tr>
<td>PQ-TED</td>
<td>74.8%</td>
<td>74.5%</td>
</tr>
</tbody>
</table>

Table 5.5: Average precision and recall values over the quality measures for the different distance measures.
to be able to save and extract their current state, this is done with the snapshotting feature of Archipelago 2.0. So the problem of comparing two branches essentially becomes the problem of comparing two snapshots.

We consider the scenario where resources are collected first and put into Archipelago 2.0 as cells. Later different groupings of those cells are explored in parallel over multiple branches. Our goal is to visualise the commonalities and differences between branches with the same content but a different grouping. For simplicity, we will use cells that contain integers for this evaluation but the content of the cells does not matter as long as it is the same in both snapshots. We will present the approach we use to solve this problem and argue why we think its is well suited for our goal. Sections 5.2.1 and 5.2.2 each present different concepts needed to reach the final approach presented in Section 5.2.3. The results are then summarized in Section 5.2.4.

5.2.1 Best Match

Due to the introduced restriction, the two snapshots that are to be compared contain exactly the same cells. This means that we can focus on how these cells are grouped. Hence finding differences and commonalities between two snapshots will be strongly related to the groups. Since the differences and commonalities are related, we only need to look for one of the two. We decide to look for commonalities of the groups in the two snapshots. For each group we want to find its most similar one in the other snapshot. We call the most similar one its best match. We define a group’s best match with Definition 11.

Definition 11 (Best Match)
Given group \( G \) and snapshot \( S \):
Any group \( g_i \) of snapshot \( S \) is best match of \( G \) if

\[
g_i \text{ and } G \text{ have at least one common cell} \tag{5.2}
\]

and

\[
ted(g_i, G) = \min(ted(g_i, G)) \text{ for all } g_i \text{ in } S \tag{5.3}
\]

Since our focus lies on how the cells are grouped, with the above definition we enforce the comparison of groups with at least one common cell wherever possible. Furthermore, we also adapt the cost model used by our framework for the decayed tree edit distance. Any operations related to groups have unitary cost, while all operations related to cells have cost 5. This makes differences among the cells much more expensive so that groups with a lot of different cells have a very high tree edit distance. An example can be seen in Figure 5.6.

Both groups B and C have at least one common cell with group A, but there are clear differences between B and C. In this case we argue that the best match for group A would be group B because all the cells match and we essentially just have a different grouping of the cells. With a classic cost model the tree edit distance between group A and B would be 4 and 2 between group A and C. Hence group C would get matched to group A which is not what we want. Thanks to our new cost model, while the tree edit distance between group A and B stays the same, it increases to 10 for group A and C and group A and B get matched together.
Always mapping each group only with its best match might also cause problems. What if a group is the best match for multiple groups? If we only allow for 1-to-1 mappings all the other groups will either be left without any group that matches them or are mapped with another group that is less similar to them. Consider the example in Figure 5.7.

Figure 5.6: Example of what we consider to be the best match.

Figure 5.7: Example of 1-to-n mapping.
For both groups B and C, group A is their best match. In case we map group A with group B, group C is left without any possible match and part of the commonalities between both snapshots is lost. However if we allow a group to be the best match of multiple groups, we can capture all commonalities. It is hence important to allow the following three different types of mappings, 1-to-n, 1-to-1 and 1-to-0. In a 1-to-n mapping, one group is the best match for several groups, in a 1-to-1 mapping the groups are each other’s best match and a 1-to-0 mapping means that there was no possible match for the group according to Definition 11. Examples of a 1-to-n, 1-to-1 and 1-to-0 can be seen in Figure 5.8. The examples of the 1-to-n and 1-to-1 mapping are obvious, the 1-to-0 mapping might need some further clarification. The cells 6 and 7 are not grouped in the left snapshot but are grouped in the right snapshot. This group can only be part of a 1-to-0 mapping because no other group has a common cell and cells that are contained in it, namely 6 and 7, are not part of a group in the left snapshot but stand alone. This makes it impossible to match the group with cells 6 and 7 to any other group.

Figure 5.8: Example of the different types of mappings.

An algorithm to find the best match of each group in both snapshots that are to be compared is given by Algorithm 9. Our algorithm can be divided into two main steps, a filtering and grouping of candidates for the best match and the computation of the best match.

During the filtering and grouping step multiple things are done. In the beginning for each group of a snapshot we have all the groups of the opposing snapshot as candidates for best match. We want to filter them by applying Definition 11. Condition 5.2 is met after all groups with no common cells are removed from the set of candidates. To fulfill Condition 5.3, we sort the remaining candidates by tree edit distance. The set of candidates for best match for each group is then the set of candidate groups with the lowest tree edit distance. Additionally we add a best match annotation to each candidate group contained in the best match set. This annotation will come in handy in the second step to compute the actual best match for each
Algorithm 9 find_best_match(groups)
1: \[\text{best} = \text{empty list}\]
2: 
3: \textbf{for all} Groups \(g\) in groups \textbf{do}
4: \hspace{1em} Remove all candidates of \(g\) with no common cells
5: \hspace{1em} Sort remaining candidates by TED
6: \hspace{1em} Add best match annotation to candidates with lowest TED
7: \hspace{1em} Sort groups by TED of best candidate
8: 
9: \textbf{while} groups is not empty \textbf{do}
10: \hspace{1em} Pop group \(g\) with lowest TED
11: \hspace{1em} \textbf{if} \(g\) has multiple candidates \textbf{then}
12: \hspace{2em} top\_candidate = candidate with lowest TED
13: \hspace{2em} \textbf{if} \(g\) has multiple best match annotations \textbf{then}
14: \hspace{3em} reference\_group = g
15: \hspace{3em} best\_match = all candidates that added annotation to \(g\)
16: \hspace{3em} Add reference\_group with best\_match to result
17: \hspace{2em} \textbf{else}
18: \hspace{3em} \textbf{if} top\_candidate has multiple best match annotations \textbf{then}
19: \hspace{4em} reference\_group = top\_candidate
20: \hspace{4em} best\_match = all candidates that added annotation to top\_candidate
21: \hspace{4em} Add reference\_group with best\_match to result
22: \hspace{2em} \textbf{else}
23: \hspace{3em} reference\_group = g
24: \hspace{3em} best\_match = top\_candidate \ (1-to-1 mapping)
25: \hspace{3em} Add reference\_group with best\_match to result
26: \hspace{1em} \textbf{else if} \(g\) has one candidate \textbf{then}
27: \hspace{2em} top\_candidate = candidate with lowest TED
28: \hspace{2em} \textbf{if} top\_candidate has multiple best match annotations \textbf{then}
29: \hspace{3em} reference\_group = top\_candidate
30: \hspace{3em} best\_match = all candidates that added annotation to top\_candidate
31: \hspace{3em} Add reference\_group with best\_match to result
32: \hspace{2em} \textbf{else}
33: \hspace{3em} reference\_group = g
34: \hspace{3em} best\_match = top\_candidate
35: \hspace{3em} Add reference\_group with best\_match to result
36: \hspace{1em} \textbf{else}
37: \hspace{2em} reference\_group = g
38: \hspace{2em} No best\_match
39: \hspace{2em} Add reference\_group to result
40: 
41: \textbf{Remove} reference\_group and best\_match from groups and candidate lists
42: \textbf{Update} groups and sort groups by TED
43: \textbf{return} result
group. To conclude we put all groups of both snapshots with their respective candidates into one big list and sort them by tree edit distance. This will help us process them from the smallest tree edit distance to the highest. This step is executed on Lines 3-7.

The second step is described on lines 9-42. During this step we establish the best match for each group in both snapshots. This step consists of a big loop where in each step we take the group with the lowest overall TED and find its best match. To find the best match for each group, we need to find what kind of mapping is established between the group and its best match. Therefore, multiple cases need to be distinguished. First, we need to distinguish between the three cases where a group either has multiple candidates, exactly one candidate or none.

In case a group $g$ has multiple candidates, it can be assigned to either a 1-to-1 or 1-to-n mapping. In this case three smaller cases need to be considered. These cases are either $g$ is part of a 1-to-n mapping as the one group that is mapped to n other groups or $g$ is part of the n groups in a 1-to-n mapping or $g$ is part of a 1-to-1 mapping. The steps to handle these cases are listed on lines 11-25. First, we check whether $g$ has multiple best match annotations. If this is the case then we are in the first case where $g$ is part of a 1-to-n mapping and is the one group mapped to multiple groups. This means that we have found $g$’s best matches, in particular the candidates that annotated $g$. This case is handled by the code on lines 13-16. In case the first condition is not met and $g$ only has one best match annotation, we need to distinguish between the two remaining cases as shown on lines 18-25. We get $g$’s top candidate, which represents our best match until now. If this top candidate has multiple best match annotations then this means that $g$ is among the n groups of a 1-to-n mapping with its top candidate. Therefore, we have found $g$’s best match as well as the best match for it’s best candidate and return the top candidate and all the groups that annotated it, including $g$, as a 1-to-n mapping. In case $g$’s top candidate also has only one best match annotation, then the best match for $g$ is its top candidate in a 1-to-1 mapping. This method covers all possible mappings that can occur if a group has multiple candidates.

The second case in which a group $g$ has exactly one candidate is handled in a similar way as the last two smaller cases for a group with multiple candidates. The reason why is that $g$ can only be part of two different mappings, either among the n groups in a 1-to-n mapping or in a 1-to-1 mapping. These are the exact same options as before, which is why the code on lines 28-35 that handle groups with only one candidate is the exact same code as on lines 18-25.

Finally the last case covers all groups that have an empty candidate list. They represent groups that after the filtering of candidates with Definition 11 have no candidates left and can’t be matched to another group. These groups are returned as a 1-to-0 mapping.

After each step in the loop where a group is extracted from the list and its best match is found, we need to remove all references of this group and its best matches from the list. This is done by removing them from the list and from any candidate list of the remaining groups. Once this is done the best match annotations need to be updated and the group is sorted again by TED before the next step of the loop can be executed. After the list is empty the algorithm concludes and the best matches for each group are returned in the form of mappings.
5.2.2 Visualisation

We want to introduce a visualisation method to help us visualise the commonalities between snapshots. Therefore, we introduce different colors for different groups and cells. Each snapshot is assigned a color for its groups and cells. The source snapshot has red groups and orange cells and the destination snapshot has blue groups and lightblue cells. Groups and cells of a specific mapping that are only in one of the snapshots and not in both will take their snapshot’s color, while groups and cells common to both snapshots will be grey and white respectively. For each type of mapping we have a very specific visualisation strategy. In Figure 5.9 we show the visualisation of commonalities between the two snapshots of Figure 5.8. 1-to-n mappings have the groups in the mapping colored with respect to the snapshot they came from, while common cells are white and cells specific to one snapshot have again that snapshot’s color. 1-to-1 groups get merged to one common group in grey. The group of the 1-to-0 mapping keeps its snapshot’s color but can cover common cells between the group and cells that stand alone in the opposing snapshot. Remaining cells are assigned the color of their snapshot.

Figure 5.9: Visualisation with color schema.

5.2.3 Approach

The concepts presented in Sections 5.2.1 and 5.2.2 are used in our approach to visualise the commonalities between two snapshots. The approach uses our MultiComp framework as well as Algorithm 10 that uses Algorithm 9 to compute the best match for each group. The MultiComp framework needs to be initialised with the decay levels and decay function. We chose to use the linear decay function presented in Section 3.5.1 together with \( NDL = 3 \) and \( SDL = 11 \). As mentioned in Section 5.2.1 any operation related to groups costs 1
and any operations related to cells costs 5. Algorithm 10 is a recursive algorithm and is first called with two lists, one containing the trees of the source snapshot and the other the trees of the destination snapshot. The recursive algorithm can be divided into two steps, a computation step and a visualisation step. The computation step is straightforward and simply computes the TED between each group in the source snapshot and each group in the destination snapshot. The visualisation step handles four different cases.

First, if both the source list and destination list have groups, then the best match for each of those groups need to be computed. This is done with the method discussed in Section 5.2.1. The visualisation of the groups is then done differently for each mapping of the returned best matches. Lines 6-8 cover the case of the mapping being a 1-to-n mapping. In that case to visualise the mapping we need to add an additional recursive step that compares the n groups in the mapping to the first-level children of the 1 group they are matched to. Only after that extra step we can correctly apply the color schema presented in Section 5.2.2. Lines 9-11 cover the 1-to-1 mappings. In this case a common group between both snapshots has been found and what’s left to look for are the commonalities between the children of the common groups. Again an extra recursive step is added, but this time the first-level children of both groups are compared before the 1-to-1 mapping can be visualised. Finally lines 12-13 cover the case of the 1-to-0 mappings. All groups of a 1-to-0 mapping are exclusive to one snapshot and don’t need to be further analysed. To conclude after all the groups have been visualised, any remaining cells without a groups are visualised.

The second and third case handle a similar situation. In this case either the source list or the destination list contains no groups. This results in 1-to-0 mappings for all groups of either the source or destination list. These visualisation are straightforward and have been covered before. Again after the groups, remaining cells need to be visualised. These cases are covered by lines 15-20.

The final case occurs when both lists contain no groups and all that is left to do is visualising the cells. These steps cover all the possible cases and conclude the algorithm.
Algorithm 10 compare_trees(source_list, destination_list)

1: Compute TED between each ground in source_list and destination_list
2:
3: if Multiple source groups and multiple destination groups then
4:   best_match = find_best_matches(source and destination groups)
5:   for all m ∈ best_match do
6:     if m is a 1-to-n mapping then
7:       compare_trees(1-group first-level children, n-groups)
8:       Visualise 1-to-n mapping m
9:     else if m is a 1-to-1 mapping then
10:    compare_trees(1-group first-level children, 1-group first-level children)
11:    Visualise 1-to-1 mapping m
12:    else
13:       Visualise 1-to-0 mapping m
14:     Visualise cells
15:   else if No source groups then
16:     Visualise 1-to-0 mapping of destination groups
17:     Visualise cells
18:   else if No destination groups then
19:     Visualise 1-to-0 mapping of source groups
20:     Visualise cells
21:   else
22:     Visualise cells

5.2.4 Results

The results of our approach are promising. As Figures 5.10, 5.11 and 5.12 show, it is much easier to see the commonalities between two snapshots with our visualisation method rather than without. The merged state of both snapshots together with the color schema help us to quickly see, which cells were grouped in a similar way in both snapshots. When looking at the colored visualisation it is always possible to see only one snapshot by focusing on groups and cells that are either grey or white or in the color of that snapshot and ignoring all groups and cells in the color of the opposing snapshot. Our goal of being able to visualise the commonalities between snapshots is hereby reached. Hence we have shown how to adapt our framework for a different type of problem and prove that our framework can indeed be used for multiple purposes.
(a) Source Snapshot.  
(b) Destination Snapshot. 

c) Visualisation with color schema.

Figure 5.10: Example of the visualisation of the commonalities between the source and destination snapshot.
Figure 5.11: Example of the visualisation of the commonalities between the source and destination snapshot.
(a) Source Snapshot. 
(b) Destination Snapshot. 
(c) Visualisation with color schema.

Figure 5.12: Example of the visualisation of the commonalities between the source and destination snapshot.
In summary we have presented a new approach based on the tree edit distance to compute the structural similarity between trees. Our new approach is a combination of different features from several existing approaches and most importantly a novel feature that has not been presented yet, the decay factor. When computing the tree edit distance between trees most of the existing approaches use a static cost model, giving all the differences and nodes the same weight. In the case we have been analysing, in particular the web, this is not a good idea because certain nodes are more closely related to the structure than others. and a flexible cost model that gives us the possibility to handle differences and nodes differently is needed. This is exactly what a decay factor offers. Through the analysis of trees on the web, we have established that the content is mainly concentrated in the lower levels of the tree, thus rendering differences in those levels less important when computing the structural similarity. By introducing the decay factor, we are able to reduce the weight of differences in the lower content levels, so that differences in the higher levels, representing differences in the structure of both trees, have a greater influence and on the tree edit distance. During our evaluation we have shown that our assumption that the decay factor is beneficial to the tree edit distance is valid. We compared our decayed tree edit distance to a non-decayed tree edit distance and an existing approach called PQ-Gram. The results show a significant improvement in both precision and recall compared to the other distance measures. On average over the three classification functions the decayed tree edit distance had an increase in precision of 23.2% and 29.3% compared to the non-decayed tree edit distance and the PQ-Gram distance. As for the recall it increased by 16.3% and 15.7% compared to the respective alternatives. We think that the results are very promising and that replacing the standard tree edit distances with our new decayed tree edit distance in existing applications for the scenario of detecting data records in web pages will produce improved results. We have also shown that our approach and framework can be used for multiple purposes. While one use case of the framework was to detect data records in web pages, we have shown that it can also be used for different types of applications, such as the comparison of branches in a versioning system. The framework
offers the base functionalities to compute the tree edit distance between trees and for each specific use case only small adjustments need to be made. Any problem related to the tree edit distance should be solvable with our framework.

As for future work, there are of course some things that could be done. An extended evaluation with a larger training and test set would probably be helpful to confirm the results we found. The method to establish the ground truth could also be improved. Right now we generate the ground truth by hand and thus it is as accurate as possible but in some cases there are still some flaws. Additionally it would also be interesting to see how the framework performs both with existing applications in regard of extracting data records from web pages and maybe existing or new applications in different areas related to the tree edit distance. Especially the integration of the framework for the computation of the tree edit distance in existing approaches would be very interesting. An extended evaluation of such applications would clearly show how the new decayed tree edit distance affects them. Finally an extended evaluation with Archipelago 2.0 where different scenarios are explored would also be very interesting. The scenario we covered restricts the branches to have the same content, however this limitation could be removed. It is not straightforward how different content would be handled since our framework is only capable of detecting structural differences between trees. An additional feature to compare content elements would be required.
This chapter will present additional information and data used during the evaluation of Section 5.1.

<table>
<thead>
<tr>
<th>Page</th>
<th>URL</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>20min</td>
<td><a href="http://www.20min.ch/">http://www.20min.ch/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Amazon</td>
<td><a href="https://www.amazon.com/gp/gw/ajax/s.html">https://www.amazon.com/gp/gw/ajax/s.html</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Baidu</td>
<td><a href="http://www.baidu.com/s?ie=utf-8&amp;f=8&amp;rsv_bp=0&amp;rsv_idx=1&amp;tn=baidu&amp;wd=ethz&amp;rsv_pq=8c982445000eb918&amp;rsv_t=fa19Xq%2BISjGQ%2BFznt%2BOWRYz8vC51J5XLjm4kqhEL2Xl1vNmA9yp7%2BumIUrsv_enter=1&amp;rsv_sug3=16&amp;rsv_sug1=4&amp;rsv_sug2=0&amp;inputT=4527&amp;rsv_sug4=4528">http://www.baidu.com/s?ie=utf-8&amp;f=8&amp;rsv_bp=0&amp;rsv_idx=1&amp;tn=baidu&amp;wd=ethz&amp;rsv_pq=8c982445000eb918&amp;rsv_t=fa19Xq%2BISjGQ%2BFznt%2BOWRYz8vC51J5XLjm4kqhEL2Xl1vNmA9yp7%2BumIUrsv_enter=1&amp;rsv_sug3=16&amp;rsv_sug1=4&amp;rsv_sug2=0&amp;inputT=4527&amp;rsv_sug4=4528</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>CNET</td>
<td><a href="http://www.cnet.com/">http://www.cnet.com/</a></td>
<td>13.10.2015</td>
</tr>
<tr>
<td>DBLP</td>
<td><a href="http://dblp.uni-trier.de/db/conf/icwe/">http://dblp.uni-trier.de/db/conf/icwe/</a></td>
<td>13.10.2015</td>
</tr>
<tr>
<td>Ebay</td>
<td><a href="http://www.ebay.com/">http://www.ebay.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Platform</td>
<td>URL</td>
<td>Date</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>------------</td>
</tr>
<tr>
<td>Facebook</td>
<td><a href="https://www.facebook.com/ETHZ.CH">https://www.facebook.com/ETHZ.CH</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Google</td>
<td><a href="https://www.google.ch/?gfe_rd=cr&amp;ei=mvYUVq6RDgnC8gep5K_QDA#q=ethz">https://www.google.ch/?gfe_rd=cr&amp;ei=mvYUVq6RDgnC8gep5K_QDA#q=ethz</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Instagram</td>
<td><a href="https://instagram.com/explore/tags/ethz/">https://instagram.com/explore/tags/ethz/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>LinkedIn</td>
<td><a href="https://www.linkedin.com/home?trk=nav_responsive_tab_home">https://www.linkedin.com/home?trk=nav_responsive_tab_home</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>MSN</td>
<td><a href="http://www.msn.com/de-ch/">http://www.msn.com/de-ch/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>NY Times</td>
<td><a href="http://www.nytimes.com/">http://www.nytimes.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>PC Mag</td>
<td><a href="http://www.pcmag.com/">http://www.pcmag.com/</a></td>
<td>13.10.2015</td>
</tr>
<tr>
<td>Pinterest</td>
<td><a href="https://www.pinterest.com/categories/popular/">https://www.pinterest.com/categories/popular/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>QQ</td>
<td><a href="http://www.qq.com/">http://www.qq.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Reddit</td>
<td><a href="https://www.reddit.com/">https://www.reddit.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Stackoverflow</td>
<td><a href="http://stackoverflow.com/">http://stackoverflow.com/</a></td>
<td>13.10.2015</td>
</tr>
<tr>
<td>Taobao</td>
<td><a href="https://www.taobao.com/">https://www.taobao.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Tumblr</td>
<td><a href="https://www.tumblr.com/search/Ethz">https://www.tumblr.com/search/Ethz</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Twitter</td>
<td><a href="https://twitter.com/TwitterDev?original_referer=https%3A%2F%2F2Pdev.twitter.com%2F&amp;profile_id=2244994945&amp;related=twitterapi%2Ctwitter&amp;tw_i=631945990530985984&amp;tw_p=embeddedtimeline&amp;tw_w=507301828308922368">https://twitter.com/TwitterDev?original_referer=https%3A%2F%2F2Pdev.twitter.com%2F&amp;profile_id=2244994945&amp;related=twitterapi%2Ctwitter&amp;tw_i=631945990530985984&amp;tw_p=embeddedtimeline&amp;tw_w=507301828308922368</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Weibo</td>
<td><a href="http://www.weibo.com/">http://www.weibo.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Yahoo</td>
<td><a href="https://www.yahoo.com/">https://www.yahoo.com/</a></td>
<td>08.10.2015</td>
</tr>
<tr>
<td>Youtube</td>
<td><a href="https://www.youtube.com/">https://www.youtube.com/</a></td>
<td>08.10.2015</td>
</tr>
</tbody>
</table>

Table A.1: URL and dates of extraction of the web pages used in the training and test set.
Figure A.1: Sample data record marked with the red border for 20min.

Figure A.2: Sample data record marked with the red border for Alibaba.

Figure A.3: Sample data record marked with the red border for Amazon.

Figure A.4: Sample data record marked with the red border for Ask.
Figure A.5: Sample data record marked with the red border for Baidu.

Figure A.6: Sample data record marked with the red border for CNET.

Figure A.7: Sample data record marked with the red border for CNN.

Figure A.8: Sample data record marked with the red border for CNN Sports.
Figure A.9: Sample data record marked with the red border for DBLP.

Figure A.10: Sample data record marked with the red border for Ebay.

Figure A.11: Sample data record marked with the red border for ETHZ.

Figure A.12: Sample data record marked with the red border for Eurosport.
Figure A.13: Sample data record marked with the red border for Facebook.

Figure A.14: Sample data record marked with the red border for Google.

Figure A.15: Sample data record marked with the red border for Instagram.

Figure A.16: Sample data record marked with the red border for LinkedIn.
Figure A.17: Sample data record marked with the red border for MSN.

Figure A.18: Sample data record marked with the red border for NY Times.

Figure A.19: Sample data record marked with the red border for PC Mag.

Figure A.20: Sample data record marked with the red border for Pinterest.
Figure A.21: Sample data record marked with the red border for QQ.

Figure A.22: Sample data record marked with the red border for Stackoverflow.

Figure A.23: Sample data record marked with the red border for Taobao.

Figure A.24: Sample data record marked with the red border for Tumblr.
Figure A.25: Sample data record marked with the red border for Weibo.

Figure A.26: Sample data record marked with the red border for Yahoo.

Figure A.27: Sample data record marked with the red border for Youtube.
Figure A.28: Comparison of the visualisation of Alibaba for the best classification functions of D-TED and PQ-TED.

Figure A.29: Comparison of the visualisation of CNET for the best classification functions of D-TED and PQ-TED.
Figure A.30: Comparison of the visualisation of CNN for the best classification functions of D-TED and PQ-TED.

Figure A.31: Comparison of the visualisation of CNN Sports for the best classification functions of D-TED and PQ-TED.
Figure A.32: Comparison of the visualisation of DBLP for the best classification functions of D-TED and PQ-TED.

Figure A.33: Comparison of the visualisation of ETHZ for the best classification functions of D-TED and PQ-TED.
Figure A.34: Comparison of the visualisation of Eurosport for the best classification functions of D-TED and PQ-TED.

Figure A.35: Comparison of the visualisation of IMDb for the best classification functions of D-TED and PQ-TED.
Figure A.36: Comparison of the visualisation of PC Mag for the best classification functions of D-TED and PQ-TED.

Figure A.37: Comparison of the visualisation of Stackoverflow for the best classification functions of D-TED and PQ-TED.
List of Figures

1.1 Data entity representing an item in an online shop. .................. 2
1.2 Examples of two data records. ........................................ 3

2.1 Sequence of the insert, delete and relabel node edit operations. ...... 6
2.2 A valid Tai Mapping between T1 and T2. ............................ 8
2.3 An invalid Tai Mapping between T1 and T2 because of the mapping between both nodes with label e. ............................ 8
2.4 An alignment between T1 and T2 .................................... 10
2.5 An isolated-subtree mapping between T1 and T2. ...................... 10
2.6 A top-down mapping between T1 and T2. ............................. 11
2.7 A bottom-up mapping between T1 and T2. ............................ 12
2.8 Hierarchy between the different types of mappings. .................. 12
2.9 Sequence of the extended operations insert and delete subtree. ....... 16
2.10 Path types. Left path (blue), right path (green) and heavy path (orange) including the root. ........................................ 17
2.11 Example trees for the STM algorithm. ............................... 18
2.12 Example of a restricted top-down mapping. ........................... 22

3.1 Examples of different mappings. ...................................... 27
3.2 Two trees with similarly structured subtrees. ......................... 29
3.3 Examples of the procedure behind Definition 5 ........................ 31
3.4 Example for the extended operations cost model. ..................... 32
3.5 Graph of the progress of the decay factor. ............................ 34
3.6 An example the leaf distances and decay factors of a tree. .......... 35
3.7 Tree edit distances for the trees of two web pages. ................... 37
3.8 Process diagram. ...................................................... 38

4.1 Internal Tree Interface. ................................................. 42
4.2 Architecture diagram. ................................................. 44
5.1 Examples of samples trees for Reddit and Twitter. .............. 52
5.2 Visualisations of the data for each distance measure for Twitter. 54
5.3 Sample data record for the IMDb page. .......................... 56
5.4 Visualisation of the data for the D-TED and the optimal classification function of the IMDb page. ...................... 56
5.5 Transformation from the group/cell model to trees. ............... 58
5.6 Example of what we consider to be the best match. .............. 60
5.7 Example of 1-to-n mapping. ........................................ 60
5.8 Example of the different types of mappings. ....................... 61
5.9 Visualisation with color schema. ................................... 64
5.10 Example of the visualisation of the commonalities between the source and destination snapshot. ....................... 67
5.11 Example of the visualisation of the commonalities between the source and destination snapshot. ....................... 68
5.12 Example of the visualisation of the commonalities between the source and destination snapshot. ....................... 69
A.1 Sample data record marked with the red border for 20min. ...... 75
A.2 Sample data record marked with the red border for Alibaba. ... 75
A.3 Sample data record marked with the red border for Amazon. ... 75
A.4 Sample data record marked with the red border for Ask. ...... 75
A.5 Sample data record marked with the red border for Baidu. ...... 76
A.6 Sample data record marked with the red border for CNET. .... 76
A.7 Sample data record marked with the red border for CNN. ...... 76
A.8 Sample data record marked with the red border for CNN Sports. 76
A.9 Sample data record marked with the red border for DBLP. ...... 77
A.10 Sample data record marked with the red border for Ebay. ...... 77
A.11 Sample data record marked with the red border for ETHZ. .... 77
A.12 Sample data record marked with the red border for Eurosport. 77
A.13 Sample data record marked with the red border for Facebook. 78
A.14 Sample data record marked with the red border for Google. ... 78
A.15 Sample data record marked with the red border for Instagram. 78
A.16 Sample data record marked with the red border for LinkedIn. 78
A.17 Sample data record marked with the red border for MSN. ...... 79
A.18  Sample data record marked with the red border for NY Times. ....... 79
A.19  Sample data record marked with the red border for PC Mag. .......... 79
A.20  Sample data record marked with the red border for Pinterest. ....... 79
A.21  Sample data record marked with the red border for QQ. ............. 80
A.22  Sample data record marked with the red border for Stackoverflow. .. 80
A.23  Sample data record marked with the red border for Taobao. ......... 80
A.24  Sample data record marked with the red border for Tumblr. ......... 80
A.25  Sample data record marked with the red border for Weibo. ......... 81
A.26  Sample data record marked with the red border for Yahoo. ......... 81
A.27  Sample data record marked with the red border for Youtube. ....... 81
A.28  Comparison of the visualisation of Alibaba for the best classification functions of D-TED and PQ-TED. ........................................ 82
A.29  Comparison of the visualisation of CNET for the best classification functions of D-TED and PQ-TED. ........................................ 82
A.30  Comparison of the visualisation of CNN for the best classification functions of D-TED and PQ-TED. ........................................ 83
A.31  Comparison of the visualisation of CNN Sports for the best classification functions of D-TED and PQ-TED. ................................ 83
A.32  Comparison of the visualisation of DBLP for the best classification functions of D-TED and PQ-TED. ...................................... 84
A.33  Comparison of the visualisation of ETHZ for the best classification functions of D-TED and PQ-TED. ...................................... 84
A.34  Comparison of the visualisation of Eurosport for the best classification functions of D-TED and PQ-TED. ......................... 85
A.35  Comparison of the visualisation of IMDb for the best classification functions of D-TED and PQ-TED. ............................... 85
A.36  Comparison of the visualisation of PC Mag for the best classification functions of D-TED and PQ-TED. ............................... 86
A.37  Comparison of the visualisation of Stackoverflow for the best classification functions of D-TED and PQ-TED. ...................... 86
List of Tables

4.1 Attributes of the tree entity. ........................................... 42
4.2 Attributes of the node entity. ........................................... 43
4.3 Parameters for the initialisation. ................................. 45

5.1 Classification functions for the different distance measures and quality measures. ........................................... 53
5.2 Training set data. ......................................................... 53
5.3 Test set data. .............................................................. 55
5.4 Precision and recall values for the different distance measures and quality measures. ........................................... 57
5.5 Average precision and recall values over the quality measures for the different distance measures. ................................. 58

A.1 URL and dates of extraction of the web pages used in the training and test set. ........................................... 74
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

MultiComp: A Multi-Purpose Framework for Similarity Matching

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):
Van Damme

First name(s):
Steven

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date
Zurich, 20.10.2015

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.