Optimizing Hierarchical Classification Through Tree Evolution[*]

Xiaoguang Qi Brian D. Davison
Department of Computer Science & Engineering
Lehigh University
Bethlehem, PA 18015 USA
{xiq204,davison}@cse.lehigh.edu

Abstract

Hierarchical classification has been shown to have superior performance than flat classification. It is typically performed on hierarchies created by and for humans rather than for classification performance. As a result, classification based on such hierarchies often yields suboptimal results. In this paper, we propose a novel genetic algorithm-based method on hierarchy adaptation for improved classification. Our approach customizes the typical GA to optimize classification hierarchies. In several text classification tasks, our approach produced hierarchies that significantly improved upon the accuracy of the original hierarchy as well as hierarchies generated by state-of-the-art methods.

1 Introduction

Classification can be performed based on a flat set of categories, or on categories organized as a hierarchy. In flat classification, a single classifier learns to classify instances into one of the target categories. In hierarchical classification, a separate classifier is trained for each non-leaf node in the hierarchy. During training, each classifier is trained to categorize the instances that belong to any of the descendants of the current node into its direct subcategories. When deployed, an instance is first classified by the root classifier, and then passed to one of the first level categories with the highest probability. This process is repeated iteratively from top to bottom, invoking one classifier at each level, until reaching a leaf node.

Previous work has shown that hierarchical classification has performance superior to that of flat classification (e.g., [4, 10, 1]). By organizing categories into a hierarchical structure and training a classifier for each non-leaf node, each classifier can focus on a smaller set of subcategories, and thus reduce the confusion from sibling branches.

[*]Technical Report LU-CSE-11-002, Dept. of Computer Science and Engineering, Lehigh University, Bethlehem, PA, 18015. A summary of this report is published as a poster paper [18].

Hierarchical classification is typically performed utilizing human-defined hierarchies. Since such hierarchies reflect a human view of the domain, they are easy for people to understand and utilize. However, these hierarchies are usually created without consideration for automated classification. As a result, hierarchical classification based on such hierarchies is unlikely to yield optimal performance.

In this paper, we propose a new classification method based on genetic algorithms to create hierarchies better suited for automatic classification. In our approach, each hierarchy is considered to be an individual. Starting from a group of randomly generated seed hierarchies, genetic operators are randomly applied to each hierarchy to slightly reorganize the categories. The newly generated hierarchies are evaluated and a subset that are better fitted for classification are kept, eliminating hierarchies with poor classification performance from the population. This process is repeated until no significant progress can be made. In our experiments on several text classification tasks, our algorithm significantly improved classification accuracy compared to the original hierarchy and also outperformed state-of-the-art adaptation approaches. Compared with previous work, our approach is different in at least two aspects:

- After each iteration, we keep a comparatively large number of best performing hierarchies rather than only keep the best one and discard the rest; we will show later that the best hierarchy does not always come from an adaptation of the previous best hierarchy.
- Unlike previous approaches that gradually adapt a hierarchy by making a slight change at each step, the crossover operators we customize for hierarchy evolution allow a new hierarchy to inherit characteristics from both parents, so that it is significantly different from either parent. This will take previous approaches many more iterations to achieve, or perhaps unachievable at all because of the use of a greedy search strategy.

Our contributions include:

- a new, better-performing approach to improved classification by hierarchy adaption; and,
- adaptation of genetic operators on hierarchies and an analysis of their utility.

The rest of this paper is organized as follows. Related work is reviewed in Section 2. We motivate and introduce our approach in Section 3, report the experimental setup and results in Section 4, and conclude in Section 5.

2 Related Work

Here, we review existing work on hierarchical classification and hierarchy adaption.

2.1 Hierarchical classification

In this subsection, we focus on work on hierarchical classification without changing the hierarchical structure. Kiritchenko [7] provided a detailed review of hierarchical text categorization.

Using classification tasks on web pages, Dumais and Chen [4] demonstrated that hierarchical classification is more efficient and accurate than flat classification. One major challenge in general for classification tasks is data sparsity, in which a category has too few labeled instances for a classifier to learn a reasonable model. This problem is prominent in hierarchical classification at lower levels. For such nodes, McCallum et al. [12] proposed to use information from parent nodes to smooth the estimated parameters. A similar idea is used in "Hierarchical Mixture Model" [26] proposed by Toutanova et al., where a generative model incorporates the term probabilities from all parent classes into the current class. Wibowo and Williams [27] suggested that an instance should be assigned to a higher level category when a lower level classifier is uncertain.

The hierarchical classification approaches mentioned above share a common characteristic: they were posed as meta-classifiers built on top of base classifiers. Since information about the hierarchy is handled by the meta-classifier, the base classifier is not aware of the hierarchical structure. Cai and Hofmann [2] proposed a new approach called hierarchical support vector machines in which the hierarchical structure information is incorporated into the loss function of base classifier (in this case, SVM). This approach can also be applied to a general, multi-label classification.

Scalability and effectiveness of hierarchical classifiers on large-scale hierarchies is always a critical issue on real-world applications. Liu et al. [10] studied this problem both analytically and empirically. They found that although hierarchical classification is better for SVM classifiers compared with flat classification, it decreases classification performance when using k-Nearest Neighbor and naive Bayes classifiers.

Utilizing additional features that are specific to a particular domain can potentially improve classification performance. Complementary features that are only available in hierarchical classification scenarios are also useful as shown by Bennett and Nguyen [1] using two methods called "refinement" and "refined experts", respectively. "Refinement" enhances the training process by performing cross-validation on the training set and using the predicted labels to filter training data so that it better matches the distribution of test data. In "refined experts", an augmented document representation is generated by including the predicted labels from lower level categories. In their experiments, both methods outperform the typical hierarchical SVM, while "refined experts" yields a better performance.

With regard to the evaluation of hierarchical classification, Sun and Lim [22] proposed measuring the performance of hierarchical classification by the *degree* of misclassification, as opposed to measuring the correctness, considering distance measures between the classifier-assigned class and the true class.

2.2 Hierarchy adaptation

A variety of approaches have been proposed for hierarchy generation or adaptation. Some of them aim to better assist human browsing (e.g., [13, 8, 19]). Some are proposed and evaluated for general purposes, rather than accurate classification (e.g., [20, 6, 16]). Here, we only focus on the methods that are solely or partially designed for a better automatic classification. These methods can be categorized into two subcategories: generative approaches and adaptation approaches.

Based on a set of predefined leaf categories and associated documents, a generative approach generates a hierarchy using clustering algorithms according to certain similarity measures. A method using linear discriminant projection to generate hierarchies was proposed by Li et al [9]. In this approach, all documents within the hierarchy are first projected onto a lower dimensional space. Then, the leaf categories are clustered using hierarchical agglomerative clustering to generate the hierarchy. Instead of building the hierarchy bottom-up, Punera et al. [17] proposed a hierarchy generation method using top-down clustering.

Unlike generative approaches, adaptation approaches need an existing hierarchy to start. Such initial hierarchies are usually built by humans, but could also be those built by automated methods. The high level idea shared among this category is to make changes to the existing hierarchy such that classification performed on the adapted hierarchy is more accurate than the original. Peng and Choi [15] proposed an efficient method to classify a web page into a topical hierarchy and automatically expand the hierarchy as new documents are added. In order to classify search results into a large hierarchy accurately and efficiently, a "deep classification" approach is proposed by pruning the hierarchy into a smaller one before classification is performed [28, 29]. Another adaptation approach called "hierarchy adaptation algorithm" [24] is proposed by Tang et al., in which each node in the hierarchy is checked iteratively, and slight modifications are then made locally to particular nodes. This approach can also be used to model dynamic change of taxonomies [23]. Nitta [14] extended this approach to make it more efficient on large-scale hierarchies. Based on an observation that unnecessarily deep hierarchies usually do not perform well, Malik [11] proposed a method to "flatten" a hierarchy by promoting low level categories up to the k-th level and removing the internal nodes.

In summary, approaches in both categories aim to produce hierarchies that are better for classification. The adaptation approaches usually require a reasonable initial hierarchy. In this paper, we will propose a hierarchy adaptation method that does not rely on a human built hierarchy, and performs better than existing approaches.

3 Approach

3.1 Motivation

As we introduced previously, hierarchical classification can often perform better than flat classification. The main reason is that, by classifying objects first into high level categories, and then iteratively into finer-grained subcategories, the classifier at each branching point should have an easier task than classifying into all categories at once. However, this is not always true. Consider the example in Figure 1, where class $A \cup B$, class C, and class $D \cup E$ can be easily separated, while separating class A from class B, class D from class E is comparatively difficult. If a classifier first separates $A \cup B$, C, $D \cup E$ from each other, then separates A from B, D from E, it should be easier than classifying all the five classes at once. However, if based on a suboptimal hierarchy, it first tries to separate $A \cup D$ from $B \cup C \cup E$, the increased difficulty may significantly reduce the quality of classification. From this example, we can see

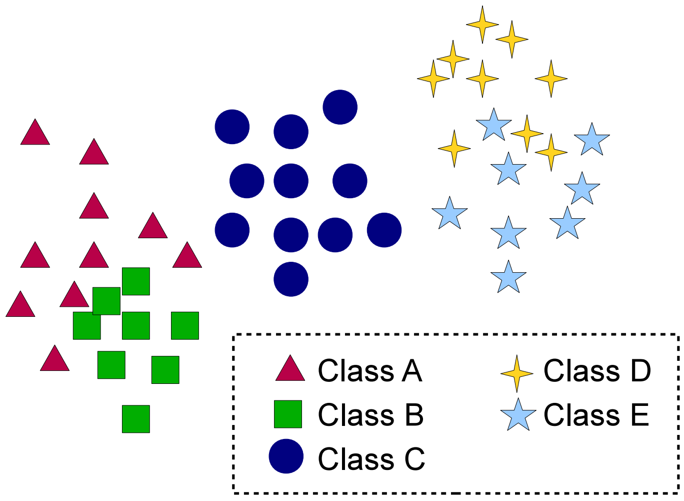

Figure 1: An imaginary five-class classification problem.

that although hierarchical classification often performs better than flat classification, it depends on the choice of hierarchy.

In order to further motivate this work using real-world data, we randomly selected 7 leaf categories containing 227 documents from the LSHTC dataset (see Section 4.1 for details about LSHTC dataset), exhaustively generated all the possible hierarchies based on the selected categories, and tested the classification performance for every hierarchy. In total, 39,208 hierarchies were generated, out of which 48.2% perform worse than flat classification in terms of accuracy. The distribution of accuracy is shown in Figure 2. The top 0.03% of all the hierarchies can achieve 100% accuracy, with the next 0.03% at 31.6% accuracy. Around 51.7% of the hierarchies perform as well as flat classification with an accuracy of 27.1%. The rest perform worse than flat classification. Some even classify all instances incorrectly. This further verifies our intuition that improving hierarchical classification needs a well-chosen hierarchy. In the following, we will describe how to adapt genetic algorithms to search for a better hierarchy.

3.2 Overview

A genetic algorithm (GA) is a search/optimization method that resembles the evolution process of organisms in nature. Like any other search method, it searches through the solution space of a problem, looking for an optimal solution. In our problem, we consider each possible hierarchy to be a solution (or an individual in the GA, specifically). As illustrated previously, our solution space is often too large to perform an exhaustive search except for very small datasets.

A typical GA usually starts with an initial population of individuals, then iteratively repeats the following search procedure until the stopping criterion is satisfied. In each iteration, a subset of individuals is selected for reproduction by applying mutation and

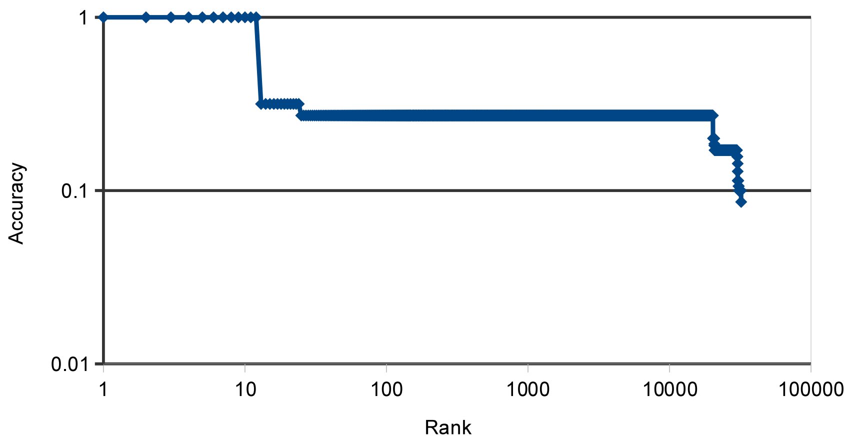

Figure 2: Distribution of accuracies of all possible hierarchies based on the seven randomly selected leaf categories.

crossover operations on them. Then the fitness of each new individual is evaluated, and low-fitness individuals are dropped, leaving a better fitted population at the start of next iteration. In our approach, we will leave the high level procedure of GAs as described above unchanged, while adapting representation method and reproduction operators to make them fit the hierarchical classification problem. We chose a GA instead of other generic search methods for at least two reasons. First, it intrinsically supports large populations rather than greedy, single path optimization. Second, we can adapt its reproduction operators to allow significant changes to the solutions without changing the high level search procedure. In an analysis of experimental results in Section 4, we will show that the above properties are essential to the performance improvement.

3.3 Hierarchy representation

We start by describing how to represent a hierarchy using a string. In a GA, reproduction among the population of a given generation produces the next generation. For easier reproduction operations and duplicate detection, we need to design a serialized representation for hierarchies. In our work, each hierarchy is represented as a sequence of numeric IDs and parentheses, in which each ID corresponds to a leaf category, and each pair of parentheses represents a more generic category consisting of the categories in between them. Multiple levels in the hierarchy are reflected using nested parentheses.

More formally, we represent a hierarchy using the following rules:

1. Each leaf node is represented by its numeric id;
2. Each non-leaf node is represented by a list of all its children nodes enclosed in a

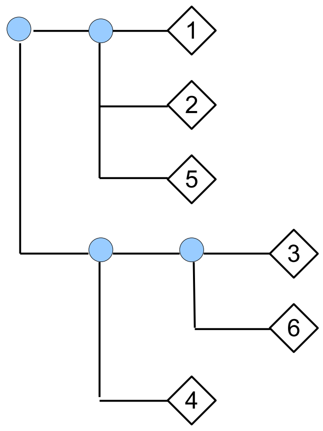

Figure 3: A small hierarchy example.

pair of parentheses;

3. The hierarchy is represented recursively using Rule 1 and 2.

4. The outermost pair of parentheses is omitted.

Figure 3 illustrates a small example, which will be represented as (1 2 5) ((3 6) 4).

3.4 Representation string canonicalization

The hierarchy representation method above serializes a hierarchical tree into a sequence of tokens. However, different representations may correspond to the same hierarchy. Using the example in Figure 3, (1 2 5) ((3 6) 4) and (2 1 5) ((6 3) 4) define the same hierarchy. Since we limit the size of the population, detecting duplicate hierarchies not only saves fitness evaluations for already evaluated hierarchies, but also encourages variety in the population, which is an important factor for performance improvement as we will show later. Therefore, we need a mechanism to normalize the representations so that duplicates are easily detected. We call this process canonicalization. Two steps of canonicalization are used in our work: trimming and sibling order canonicalization.

Trimming. We define a trivial node as a node with only one child. Trivial nodes are not useful in hierarchical classification. We use the following rule to trim the hierarchy and eliminate trivial nodes: $((S_t)) \Longrightarrow (S_t)$, where S_t is the representation of a subtree t.

Sibling order canonicalization. As described earlier, each non-leaf node is represented by a list of all its children nodes enclosed in a pair of parentheses. In this list, the order of the nodes is not important, i.e., if the only difference of two representation strings is the ordering of sibling nodes, they should be considered the same.

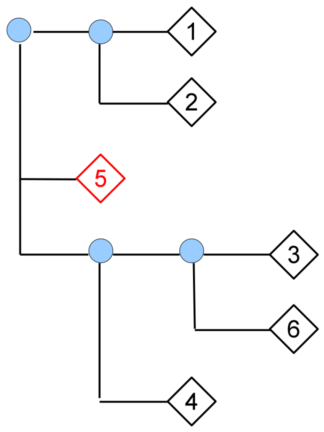

Figure 4: An example of promotion mutation: promoting Node 5 in Fig. 3.

For any node n_i in the hierarchy and the subtree t_i rooted at n_i, we define the function $smId(n_i)$ as the smallest ID in t_i. More formally, for any node n_i, $smId(n_i)$ is recursively defined as:

$$smId(n_i) = \begin{cases} n_i & \text{if } n_i \text{ is a leaf} \\ \min_{k:n_k \subset n_i} smId(n_k) & \text{otherwise} \end{cases} \tag{1}$$

where n_k is a child of n_i. In the representation, n_i and its siblings are sorted according to their $smId(\cdot)$ value. For example, (2 1 5) ((3 6) 4) and (2 1 5) (4 (3 6)) will both be canonicalized into (1 2 5) ((3 6) 4).

3.5 Seed generation

In GA, an initial population needs to be generated before the iterative evolution process is simulated. In our approach, we use a random algorithm to generate each hierarchy in the initial population. The pseudocode is shown in Algorithm 1. The algorithm first sequentially initializes a sequence using integers from 0 to $n-1$, where n is the number of leaf categories in the hierarchy. Then, using the Fisher Yates shuffling algorithm [5], the integers representing each leaf category are randomly shuffled such that each integer has an equal probability (i.e., $1/n$ in this case) appearing at any position in the sequence. After that, k pairs of parentheses are inserted into the sequence at random positions, where k is a random number between 0 and $n-1$.

3.6 Reproduction

3.6.1 Mutation

Mutation is the genetic operator in which an individual is slightly changed to maintain the diversity of the population. In a typical GA, this is performed by switching a random bit in the chromosome string. However, this operation is not as straightforward

Algorithm 1 Algorithm to generate a random hierarchy with n leaf nodes.

```
for $i := 0$ to $n - 1$ do
    $sequence[i] \leftarrow i$
end for

{randomly shuffle the sequence using Fisher Yates shuffling process}
for $i := n - 1$ downto 0 do
    randomly choose an integer $r$ from the range $[0..i]$, inclusive
    swap ($sequence[i]$, $sequence[r]$)
end for

randomly choose an integer $k$ from the range $[0..n - 1]$, inclusive
{$k$ will be the number of pairs of parentheses to be inserted}

{insert parentheses}
for $i := 0$ to $k - 1$ do
    randomly choose two unequal integers $pos1$ and $pos2$ from the range
    $[0..sequence.length]$, inclusive
    if $pos1 > pos2$ then
        swap ($pos1$, $pos2$)
    end if
    insert a left parenthesis at $pos1$ in $sequence$
    insert a right parenthesis at $pos2$ in $sequence$
end for
```

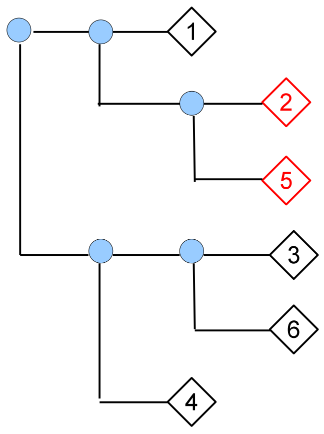

Figure 5: An example of grouping mutation: grouping Node 2 and 5 in Fig. 3.

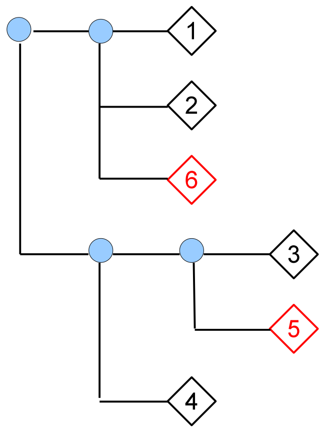

Figure 6: An example of switching mutation: switching Node 5 and 6 in Fig. 3.

in the hierarchical setting. We design three mutation methods that are suitable for hierarchy evolution: promotion, grouping, and switching.

The *promotion* operator randomly selects a node n (which can be either a leaf node or a non-leaf node), and promote n as a child of its grandparent, i.e., n becomes a sibling of its parent node. For example, promoting Node 5 in Figure 3 generates the hierarchy in Figure 4. As a special case, promoting the root node results in no change in the hierarchy. If node n has only one sibling m, then promoting n is equivalent to promoting m, and also equivalent to promoting both m and n while removing their parent.

The *grouping* operator randomly selects two sibling nodes and groups them together. If a non-leaf node n has k children, $C = \{c_i | i = 1..k\}$, where $k \geq 2$. We randomly select two nodes c_x and c_y from c_1 through c_k, and remove them from C.

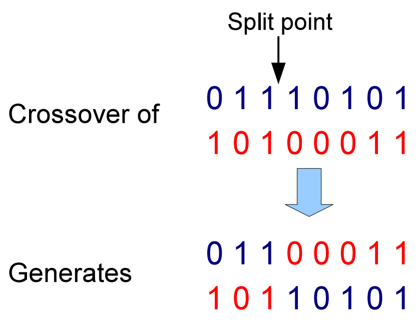

Figure 7: An example of crossover in a generic GA setting.

Then we add a new node c_z into C so that c_z becomes a child node of n. Finally, we make c_x and c_y children of c_z. For example, grouping Node 3 and Node 5 in Figure 3 generates the hierarchy in Figure 5.

The *switching* operator randomly selects two nodes m and n (and the subtrees rooted at those locations) in the whole hierarchy, and switches their positions. For example, switching Node 5 and Node 6 in Figure 3 generates the hierarchy in Figure 6. In the above examples, all mutation operations happened at leaf nodes. This is only for the purpose of easy illustration. The three mutation operators introduced here can promote, group, or switch non-leaf nodes.

3.6.2 Crossover

Crossover is the genetic operator in which two individuals (parents) are combined to generate new individuals (children) so that each child will inherit some characteristics from each parent. In a typical GA, crossover is performed by swapping segments of the chromosome string between the parents (illustrated in Figure 7). In the hierarchical setting, however, directly swapping parts of the hierarchy representation will generate invalid hierarchies. As shown in the example in Figure 8, the resulting hierarchy representations have missing/duplicate leaf nodes and unmatched pairs of parentheses. Therefore, we need crossover methods customized for hierarchy evolution.

We used two types of methods: swap crossover and structural crossover. The two parents are noted as h_{p1} and h_{p2}, the children h_{c1} and h_{c2}. In *swap crossover*, a child h_{c1} is generated using the following steps. First a split point p is randomly chosen in h_{p1}. We note the part starting from the beginning of the representation string to the split point p as h'_{p1}. We remove the segment after p from h_{p1} and only consider h'_{p1}. Then right parentheses are added at the end to balance with the existing left parentheses. Suppose S is the set of leaf nodes that appear in h'_{p1}; we go through h_{p2} and remove all the nodes n if $n \in S$. This removal transforms h_{p2} into h'_{p2}. Finally, h'_{p1} and h'_{p2} are concatenated to form h_{c1}. The other child h_{c2} is generated by switching h_{p1} and h_{p2} before applying the above procedure. The above operation guarantees that the generated children h_{c1} and h_{c2} are valid hierarchies while each inherits certain characteristics from their parents h_{p1} and h_{p2}. Figure 9 illustrates the process of swap

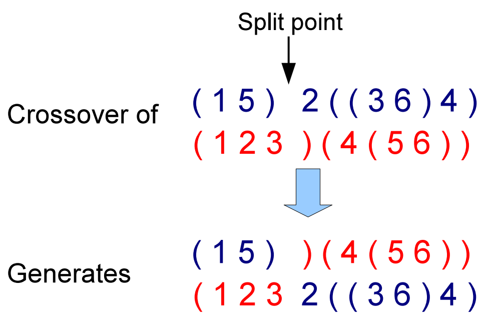

Figure 8: Applying the generic crossover operator on hierarchies may generate invalid offsprings.

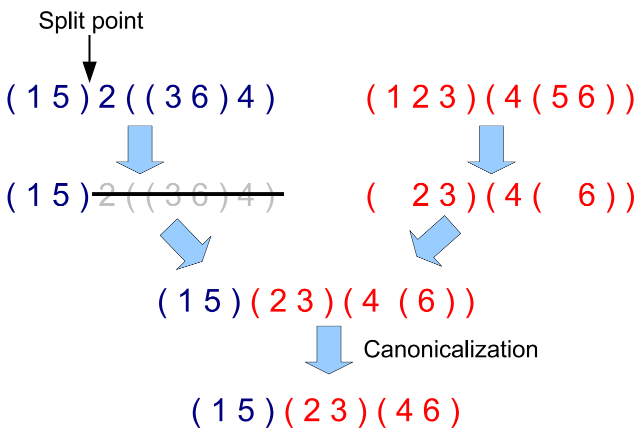

Figure 9: An example of swap crossover.

crossover.

A hierarchy can be seen as an integration of two independent characteristics: the tree structure and the placement of leaf nodes. At a high level, *structural crossover* aims to "mix and match" these two factors. A child hierarchy inherits the structural information from one parent, and placement of leaf nodes from the other. In our implementation, h_{c1} is generated using the following method. First, every leaf node in h_{p1}'s representation is replaced with a blank space. Then these blank spaces are filled with the leaf nodes in h_{p2} using the order that they appear in h_{p2}. h_{c2} is generated by switching h_{p1} and h_{p2}. Figure 10 illustrates the process of structural crossover. Although the above reproduction operators guarantee the validity of the generated child hierarchies, they may generate non-canonical representations. Therefore, canonicalization is needed after reproduction.

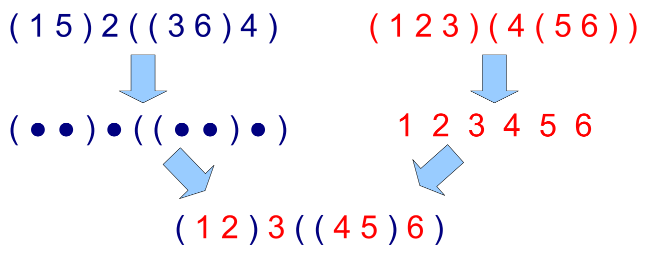

Figure 10: An example of structural crossover.

3.7 Fitness function

In each iteration of the GA, the individuals (i.e., hierarchies) in the new generation need to be evaluated. We define the fitness function $fit(h)$ of a hierarchy h simply as the classification accuracy on h. Given a set of training data D_{train}, validation data $D_{validation}$, and the base hierarchical classifier CL,

$$fit(h) = accuracy(CL(D_{train}), D_{validation}) \quad (2)$$

where accuracy is defined as the ratio of the correctly classified documents out of all the documents being classified.

3.8 Stopping criterion

A GA needs a stopping criterion to terminate the iterative evolution process. In our algorithm, we keep a watch list of top N_{watch} best hierarchies. If the performance of the top hierarchies do not change between two consecutive iterations, the algorithm stops and outputs the top hierarchies. In the following experiments, we set N_{watch} to 5.

4 Experiments

In this section, we test our hierarchy evolution algorithm using real-world data, and compare its performance with previous methods.

4.1 Experimental setup

In order to test our algorithm, we used three public datasets. The first two datasets are from the first Large Scale Hierarchical Text Classification (LSHTC) challenge[1] held in 2009. We selected a toy dataset from Task 1 with 36 leaf categories and 333 documents, which will be referred to as LSHTC-a. We also used the dry-run dataset from Task 1, which has 1,139 leaf categories and 8,181 documents. We will refer to this dataset as LSHTC-b. Both datasets are partitioned into three subsets: a training set used to train

[1]http://lshtc.iit.demokritos.gr/node/1

Category	Num. of Documents
course	930
department	182
faculty	1,124
other	3,764
project	504
staff	137
student	1,641

Table 1: Categories and class distribution in WebKB dataset.

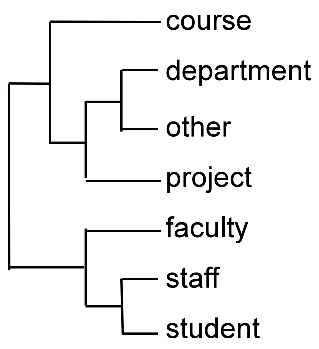

Figure 11: The hierarchy automatically generated by Linear Projection (redrawn based on the experimental result in the LP paper [9]).

classifiers during the training process, a validation set used to estimate the fitness score of each generated hierarchy, and a test set to evaluate the final output hierarchy. The third dataset is WebKB[2], containing 7 leaf categories and 8,282 documents. The documents in WebKB dataset are web pages crawled from the following four universities: Cornell (867 web pages), Texas (827 web pages), Washington (1,205 web pages), and Wisconsin (1,263 web pages), plus 4,120 web pages from other universities. These web pages are manually categorized into one of the categories listed in Table 1. On the split of training and test data, the provider of the dataset suggests "training on three of the universities plus the misc collection, and testing on the pages from a fourth, held-out university". According to this, we performed four-fold cross-validation on WebKB with a minor adaptation of the split method. Each fold trains on data from two of the universities plus the "misc" collection (web pages from other universities), validates on a third university, and tests on a fourth university. LibSVM [3] is used as the base classifier to implement the standard hierarchical SVM. We used all the default settings in LibSVM, including the radial basis kernel function as it yields better performance than linear kernel according to our experiments. In our algorithm, we set the population size to 100 on LSHTC-a and WebKB, 500 on LSHTC-b.

[2]http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

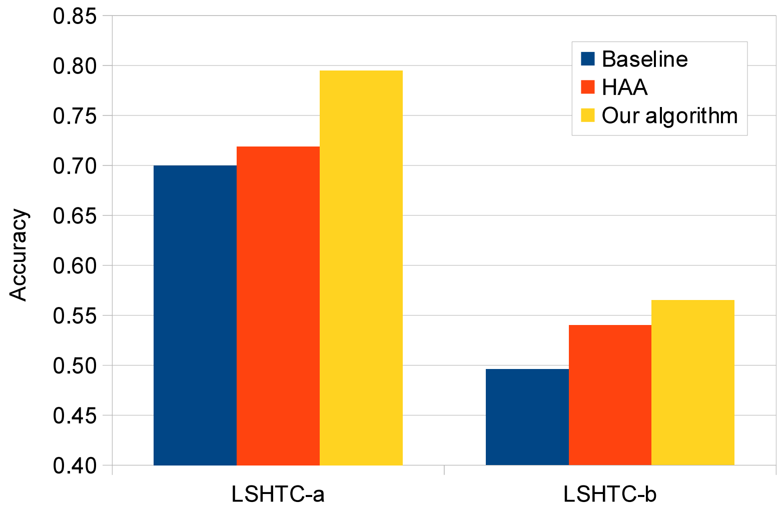

Figure 12: Accuracy on LSHTC datasets compared across different methods.

We compared our approach (Evolution) with two existing state-of-the-art approaches: a hierarchy adaptation approach called Hierarchy Adjusting Algorithm (HAA) [24], and a hierarchy generation approach called Linear Projection (LP) [9]. We implemented HAA according to the algorithm outlined in Figures 12 and 13 of [24]. All three search methods were implemented. In our implementation of HAA, we set the stopping criterion to 0.001. That is, when the improvement between two consecutive iterations is less than 0.001, the algorithm terminates. We also compared our algorithm with Linear Projection on the WebKB dataset. Instead of re-implementing the LP algorithm, we directly used the automatically generated hierarchy on WebKB reported in the Linear Projection paper, and performed the four-fold cross-validation based on that hierarchy (shown in Figure 11).

4.2 Experimental results

On the small LSHTC-a dataset, a flat classification has an accuracy of 68.6%. The hierarchical classification using the original, human-built hierarchy performs slightly better at 70% accuracy. HAA converged after two iterations with the accuracy improved to 71.9%. Since the initial population in our Hierarchy Evolution algorithm is generated randomly, we ran our algorithm three times using different random seeds. The averaged accuracy is 79.5%. On the LSHTC-b dataset, it took 50.3 iterations for our algorithm to converge (averaged across 6 runs), and 18 iterations for HAA. HAA improved upon the baseline's accuracy of 49.6% to 54.0% (an improvement of 8.9%), while our algorithm's final output hierarchy has a 56.5% accuracy averaged across 6

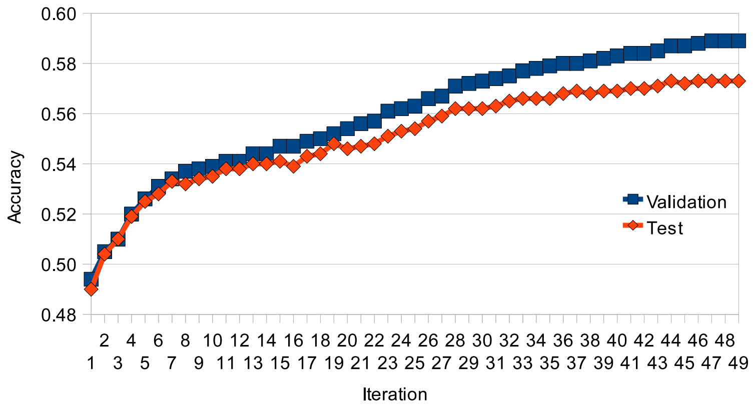

Figure 13: Best accuracy at each iteration.

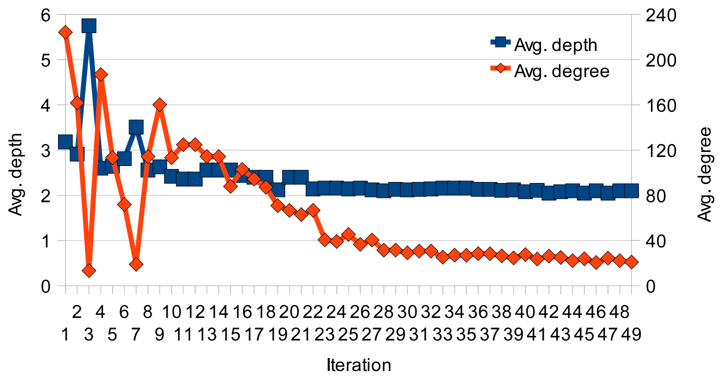

Figure 14: Average depth and degree of the best-performing hierarchy at each iteration.

runs (an improvement over the baseline of 13.9%). The results are shown in Figure 12. To be certain of a fair comparison, we let HAA continue running for three more iterations after convergence, but did not observe any additional improvement. Two-tailed t-tests show that our algorithm outperforms other approaches statistically significant on both LSHTC datasets ($p\ value \leq 0.02$).

When running our algorithm on LSHTC-b, at each iteration, we extracted the best hierarchy in terms of its classification accuracy on the validation set. For comparison, we plotted the per-iteration best hierarchy's accuracy on the validation and test set for one of the six trials in Figure 13. The validation accuracy increases monotonically until convergence. Although the test accuracy fluctuates a little, it maintains an increasing trend in general. Figure 14 shows the average depth and degree of the best-performing hierarchy at each iteration.

On the WebKB dataset, we compared our algorithm with flat classification, HAA,

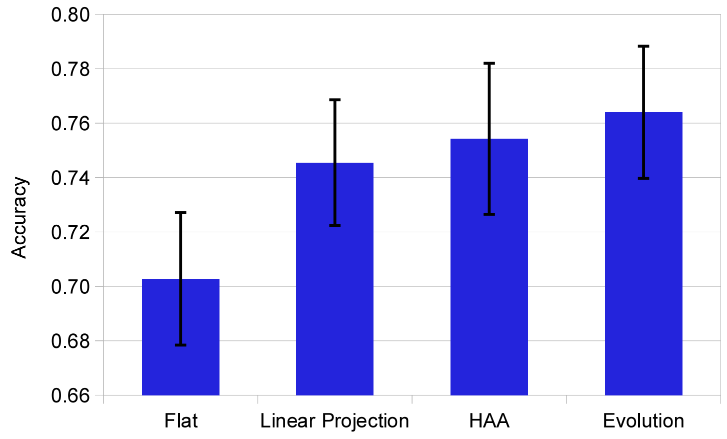

Figure 15: Accuracy on WebKB dataset compared across different methods.

Methods	**Flat**	**LP**	**HAA**	**Evolution**
Fold 1	0.687	0.734	0.742	0.749
Fold 2	0.694	0.721	0.721	0.738
Fold 3	0.691	0.774	0.780	0.788
Fold 4	0.739	0.753	0.774	0.781
Average	0.703	0.746	0.754	0.764
STDEV	0.024	0.023	0.028	0.024

Table 2: Accuracy of each fold on WebKB compared across different methods.

and Linear Projection. Based on the seven leaf categories, a flat classification has an accuracy of 70.3% averaged across the four folds. Linear Projection and HAA improve the accuracy to 74.6% and 75.4%, respectively. Our algorithm further improves to 76.4%, a 21% reduction in error rate compared with flat classification. Figure 15 shows the average performance and standard deviation for each method. The variance is mainly caused by the difference of data across folds. From Table 2, we can see that our approach consistently performs better than other methods on all folds. Two-tailed t-tests showed that our algorithm significantly outperforms all other algorithms being compared, with p-values under 0.03 for all tests. Figure 16 shows the best hierarchy generated by our algorithm on the first fold of WebKB cross-validation.

4.3 Experiment analysis

As we pointed out previously, our approach differs from existing hierarchy adaptation approaches from at least two aspects:

1. genetic operators that allow more significant changes in a hierarchy; and,

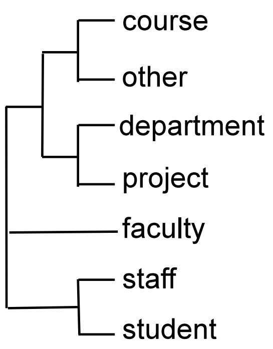

Figure 16: The hierarchy automatically generated by our Hierarchy Evolution Algorithm.

2. a larger population size to maintain population variety.

Now we analyze quantitatively whether these differences make our approach outperform existing methods. The following analysis is performed on the LSHTC-b dataset.

In order to evaluate the effectiveness of the genetic operators, we calculate the improvement that each type of operator brings to a hierarchy. Figure 17 shows the average improvement in terms of accuracy. On average, all the five operators have a negative impact on the accuracy. For example, the "swap crossover" even decreases accuracy by 0.018 when averaged across all evolution operations. That is, on average, every time a "swap crossover" is applied on a hierarchy, the newly generated hierarchy has an accuracy lowered by 0.018. Fortunately, the genetic algorithm only keeps the best hierarchies in the population, and discards the rest. Therefore, the degradation is counter-balanced by such a selection process, making an overall increasing trend (as we showed previously in experimental results). We also calculated the standard deviation of the improvement, as well as minimal and maximal improvement. The error bars in Figure 17 show the minimal and maximal improvement. The standard deviation of the operators are 0.0004, 0.0005, 0.0009, 0.0100, 0.0028, respectively. This indicates that the mutation operators only have slight impact on the accuracy while changes made by crossover are more significant. In the best case, "swap crossover" improved accuracy by 0.026, "structural crossover" improved 0.029, while all the mutation operators can only improve no more than 0.002 at their best. These statistics match our intuition that more significant changes can potentially bring better improvements than local modifications.

Another method to examine the utility of different genetic operators is to use only a subset of the operators in our algorithm and check the accuracy of the final output hierarchy. The result of this analysis on LSHTC-a is shown in Figure 18. Using all operators yields a 80% accuracy as the algorithm converges after 7 iterations. Using both crossover operators without any mutation yields the same accuracy with a slightly slower convergence speed (8 iterations). Using structural crossover only, we can still discover a hierarchy with the same accuracy at 80%, but at a cost of two more additional iterations (10 iterations until convergence). Using swap crossover only, the algorithm

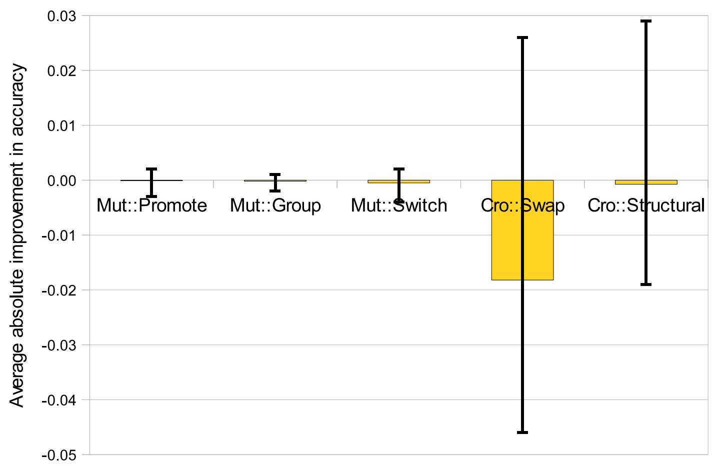

Figure 17: Improvement compared across different genetic operators.

converges after 9 iterations with an accuracy of 77%. If we take out both crossover operators and only use mutation operators, the performance of the final output hierarchy is significantly decreased down to 73%.

Unlike previous approaches that only carry the best hierarchy into the next iteration, we keep hundreds of hierarchies in the population. This raises a reasonable concern about our approach: is the large population necessary? To answer this question, we first identified the top 5 hierarchies at each iteration, then back-traced their parents from the previous iteration, and finally found out the ranks for their parents. The results are shown in Figure 19. A point (x,y) means that if we were to keep only top x hierarchies in the previous generation, then only a percentage of y hierarchies that were in the top 5 of the next generation still exist. In other words, the rest of top hierarchies (i.e., 1-y) will no longer exist because their parents were removed from the previous generation. We can see that we can still generate a significant portion of top hierarchies by keeping 300 hierarchies in each iteration. From another point of view, if we were to keep only 50 hierarchies, we would have lost approximately 81% of the top 5 hierarchies at each iteration. Although we could probably shrink the population size by 20% without significant degradation in accuracy, this supports the idea that the comparatively large population is necessary. In order to further verify this conjecture, we performed the experiments again based on various population sizes from 100 to 500, and compared the test accuracy across the final output hierarchies. Figure 20 shows that increasing the population size from 100 to 400 gives a significant improvement in terms of accuracy on the final output hierarchy, while no additional benefit is perceived beyond the size of 400.

An additional experiment with respect to the usefulness of a large population size is performed on LSHTC-a dataset. In this experiment, instead of running HAA from a single initial hierarchy (i.e., the original, human created hierarchy), we ran HAA 100 times, with each trial starting from a randomly generated hierarchy. We controlled the

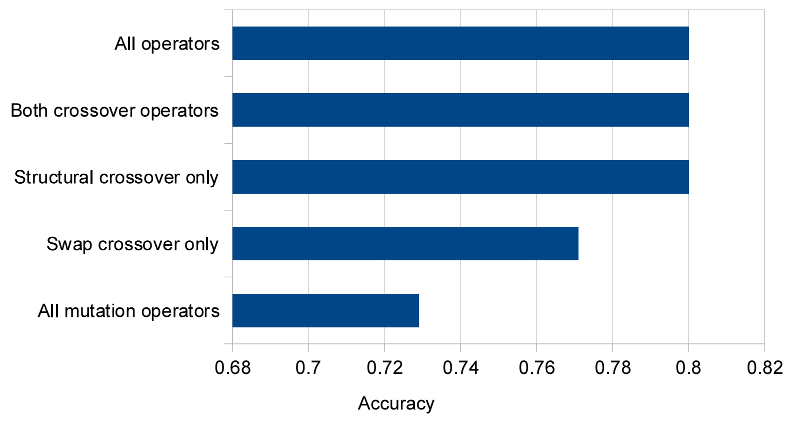

Figure 18: Accuracy comparison using different subsets of genetic operators.

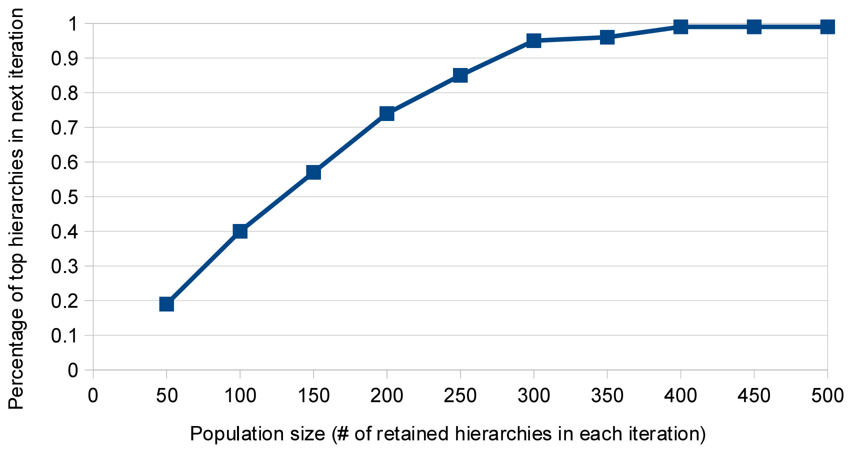

Figure 19: Top hierarchies that can be generated when a smaller population size is used.

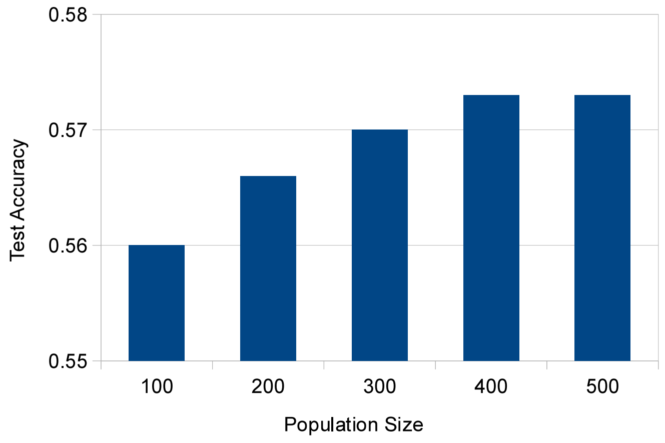

Figure 20: Classification accuracy on the output best hierarchy when varying population size.

random hierarchy generation process so that the 100 initial hierarchies are exactly the same as the initial population previously generated in the first trial of our Hierarchy Evolution algorithm. The purpose of this experiment is twofold: to test how much a large population contributes to finding a better solution, and to provide a fair comparison between HAA and our approach. Among the 100 trials, the best hierarchy found by HAA has a 74.3% accuracy on the test set (with an average accuracy of 69% across all trials). In comparison, the first trial of our algorithm found a best hierarchy with 80% accuracy. The result indicates that, although the contribution from the large population is clearly visible, it is the combination of the two factors that make our algorithm better than previous methods: a large population and the new genetic operators.

5 Discussion and Conclusion

Compared with previous approaches, our method explores a much larger space searching for better hierarchies. Although this enables us to find better solutions, it also brings significant cost. At each iteration, our approach evaluates thousands of hierarchies. Fortunately, the evaluation can be easily parallelized. Using a Condor [25] distributed computing platform running on around 100 nodes shared with multiple users, each iteration on the LSHTC-b dataset can be finished within approximately 2.5 hours in real elapsed time. For the additional improvement in classification performance, we consider this extra one-time cost worthwhile. The evaluation cost can be reduced using less expensive, approximate fitness evaluations. For example, smaller training and validation sets, and fast classifiers can be used in the evaluation process. Furthermore, since some generated hierarchies share common subtrees, trained models on such subtrees can be reused.

In this paper, we proposed a hierarchy adaptation approach by using the standard mechanisms of a genetic algorithm along with special genetic operators customized for hierarchies. Unlike previous approaches which only keep the best hierarchy in the search process and modify the hierarchy locally at each step, our approach maintains population variety by allowing simultaneous evolution of a much larger population, and enables significant changes during evolution. Experiments on multiple classification tasks showed that the proposed algorithm can significantly improve automatic classification, outperforming existing state-of-the-art approaches. Our analysis showed that the variety in population and customized reproduction operators are important to improvement in classification performance.

One drawback of our approach is that the genetic operators select mutation points and split points purely at random. Smarter operators may select such points based on heuristic rules so that they are more likely to generate better hierarchies.

The choice of genetic operators is quite arbitrary and primitive. Although those operators are shown to be effective through our experiments, they are probably not the best or the only effective operators for the problem. Are there other operators that can work effectively on the problem? Are there better operators? Besides those discussed in this paper, what other properties are shared among good operators? A comprehensive operator study is needed to answer these questions.

In our experiments, the parameters of the GA were arbitrarily assigned, and remained constant over the search process. Genetic algorithms with adaptive parameters (adaptive GA) [21] may bring further improvement. In addition, if the solutions are confined to binary trees, it may change the speed of fitness evaluation and the rate of convergence. Therefore, additional modifications to our algorithm might be necessary.

Acknowledgments

We thank Dr. Hector Munoz-Avila for discussions that inspired the use of genetic algorithms, and Liangjie Hong and Na Dai for helpful discussions. This material is based upon work supported by the National Science Foundation under Grant Number IIS-0545875.

References

[1] P. N. Bennett and N. Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–18. ACM, 2009.

[2] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, pages 78–87, New York, NY, USA, 2004. ACM.

[3] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[4] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, New York, NY, 2000. ACM Press.

[5] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. Oliver & Boyd, London, 1938.

[6] E. Glover, D. M. Pennock, S. Lawrence, and R. Krovetz. Inferring hierarchical descriptions. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 507–514. ACM, 2002.

[7] S. Kiritchenko. *Hierarchical text categorization and its application to bioinformatics*. PhD thesis, University of Ottawa, 2005.

[8] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of the 13th International Conference on World Wide Web*, pages 658–665, 2004.

[9] T. Li, S. Zhu, and M. Ogihara. Hierarchical document classification using automatically generated hierarchy. *Journal of Intelligent Information Systems*, 29:211–230, October 2007.

[10] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations Newsletter*, 7(1):36–43, 2005.

[11] H. Malik. Improving hierarchical svms by hierarchy flattening and lazy classication. In *Proceedings of Large-Scale Hierarchical Classification Workshop*, 2010.

[12] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 359–367. ACM, 1998.

[13] A. Möller, J. Dörre, P. Gerstl, and R. Seiffert. The TaxGen framework: Automating the generation of a taxonomy for a large document collection. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*, volume 2, page 2034, 1999.

[14] K. Nitta. Improving taxonomies for large-scale hierarchical classifiers of web documents. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 1649–1652. ACM, 2010.

[15] X. Peng and B. Choi. Automatic web page classification in a dynamic and hierarchical way. In *Proceedings of the IEEE International Conference on Data Mining*, pages 386–393, Washington, DC, 2002. IEEE Computer Society.

[16] S. P. Ponzetto and M. Strube. Deriving a large scale taxonomy from wikipedia. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 1440–1445. AAAI Press, 2007.

[17] K. Punera, S. Rajan, and J. Ghosh. Automatically learning document taxonomies for hierarchical classification. In *Special interest tracks and posters of the 14th International Conference on World Wide Web*, pages 1010–1011. ACM, 2005.

[18] X. Qi and B. D. Davison. Hierarchy evolution for improved classification. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 2193–2196, October 2011.

[19] X. Qi, D. Yin, Z. Xue, and B. D. Davison. Choosing your own adventure: automatic taxonomy generation to permit many paths. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 1853–1856, October 2010.

[20] M. Sanderson and B. Croft. Deriving concept hierarchies from text. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 206–213, 1999.

[21] M. Srinivas and L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656 –667, April 1994.

[22] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *Proceedings of IEEE International Conference on Data Mining*, pages 521–528, Washington, DC, November 2001. IEEE Computer Society.

[23] L. Tang, H. Liu, J. Zhang, N. Agarwal, and J. J. Salerno. Topic taxonomy adaptation for group profiling. *ACM Transactions on Knowledge Discovery from Data*, 1:1:1–1:28, February 2008.

[24] L. Tang, J. Zhang, and H. Liu. Acclimatizing taxonomic semantics for hierarchical content classification. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 384–393. ACM, 2006.

[25] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17:323–356, February 2005.

[26] K. Toutanova, F. Chen, K. Popat, and T. Hofmann. Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 105–113, New York, NY, USA, 2001. ACM.

[27] W. Wibowo and H. E. Williams. Strategies for minimising errors in hierarchical web categorisation. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 525–531, New York, NY, 2002. ACM Press.

[28] D. Xing, G.-R. Xue, Q. Yang, and Y. Yu. Deep classifier: automatically categorizing search results into large-scale hierarchies. In *Proceedings of the International Conference on Web Search and Data Mining*, pages 139–148. ACM, 2008.

[29] G.-R. Xue, D. Xing, Q. Yang, and Y. Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 619–626. ACM, 2008.