Implementation and applications of Cluster Editing

Christofer Chavanne
christofer@chavanne.se

Dan Engqvist
engqvisd@student.chalmers.se

Daniel Forsberg
danfor@student.chalmers.se

David Gullmarsvik
gullet@dtek.chalmers.se

Gustav Lindqvist
gustavl@student.chalmers.se

Anders Moberg
moberg@student.chalmers.se

Max Sikström
maxs@student.chalmers.se

June 4, 2009

Abstract

The goal of Cluster Editing is to make the fewest changes to a graph by adding or removing edges such that the resulting graph is composed of isolated cliques. In this report we describe the implementation of three different Cluster Editing algorithms, with search trees of sizes $O(3^k)$, $O(2.62^k)$ and $O(2^k)$. The parameter k denotes the maximum allowed total cost of the edit steps needed to transform the input graph to a graph composed of isolated cliques. The implementation is evaluated with three different data sets. We show that the theoretical complexity of the algorithms conforms well to the running times of the implementations. Finally we propose a few possible extensions to the work presented in this report.

Contents

Glossary

Circle A set of connected vertices where each vertex has edges to exactly two other vertices.

Clique A subset of vertices in a graph where each vertex has an edge to every other vertex in the subset. In the simplest case a clique consists of just one vertex.

Cluster Graph A graph consisting of a union of cliques, where a vertex can only be a member of exactly one clique.

Conflict Triple A set of three vertices connected by exactly two edges. A conflict triple can never be a part of a cluster graph.

Cost Adding or removing an edge is associated with a cost equivalent to the absolute value of the weight of the edge.

Depth First Search(DFS) A tree traversal strategy that expands a vertex's child vertices before its neighbor vertices. I.e. the algorithm follows each branch to its full depth before backtracking.

Edge A non-negative weighted connection between two vertices in a graph where the weight is a measurement on the relation between the vertices.

Graph A set of vertices and a set of edges. A graph is commonly used as a tool to model relations between objects.

Min-cut An algorithm used for dividing a graph into two isolated components using minimum edit cost.

Incremental DFS A tree traversal strategy identical to DFS, except that the search depth is limited by a variable i that is incremented after each iteration.

Isolated Clique A clique where no vertex has edges to vertices outside the clique.

Non-edge A vertex pair that is not connected by an edge. An alternative interpretation is an edge with negative weight.

Path A set of k vertices connected by exactly $k - 1$ edges such that each vertex has at most two neighboring vertices.

Vertex A unit in a graph that represents an object. Synonymous to node.

Weak Clique A clique where at least one edge is a zero-edge.

Zero-edge An edge with weight zero.

1 Introduction

Clustering is the task of partitioning a set of objects into disjoint subsets such that each cluster only contains objects that are considered similar. In graph theory, a cluster graph is a graph where all vertices are partitioned into isolated cliques, with all vertices in a clique having edges to all other vertices within the clique.

In the Cluster Editing Problem, the objective is to transform an input graph into a cluster graph using the minimum possible number of edits, where an edit is defined as the operation of adding or removing an edge.

The Cluster Editing Problem can be expanded to an integer-weighted Cluster Editing Problem. By assigning each edge and non-edge an integer value representing the cost of editing that edge, the problem is instead to transform the graph into a cluster graph using minimum total edit cost.

1.1 History of Cluster Editing

Although the roots of Cluster Editing can be traced back to the late eighties as mentioned by Guo[1], it is a rather new problem. In 1999 Ben-Dor et al. [2] brought attention to Cluster Editing in computational biology as a tool for understanding which genes are in the same functional group. The work of Ben-Dor et al. turned out to be an important breakthrough, spawning alot of related studies.

The Cluster Editing Problem also has applications in machine learning as illustrated by Bansal et al. [3]. In 2004 Bansal et al. explored Cluster Editing as a way to solve the Document Clustering Problem, where the objective is to arrange documents in groups according to a function that describes if the documents are similar. The problem of automatically sorting large quantities of documents based on some predefined criteria becomes more and more relevant as the quantity of available information grows ever larger. Thus it is important to have powerful tools for handling this task effectively.

In 2005 Guo et al. described an algorithm [4] of complexity $O(1.92^k + n^3)$, where k is the parameter denoting the maximum total allowed editing cost of arriving at a solution and n is the number of vertices in the input graph. This algorithm is extremely complex and has according to Böcker[5] never been implemented.

1.2 Purpose of the Project

The purpose of the project is to implement Cluster Editing algorithms in C and evaluate their performance on a few data sets. The project focus lies on the discovery made by Böcker et al. in 2008 that the Cluster Editing Problem can be solved by a theoretically faster and more straightforward algorithm compared to earlier efforts in the field [5].

To evaluate and compare the efficiency of the different algorithms, an algorithm with a search tree of size $O(3^k)$ as described by Niedermeier [6] will be implemented along with the algorithms with search trees of size $O(2.62^k)$, $O(2^k)$ and if possible $O(1.82^k)$ as described by Böcker et al.[5]. In the context of Cluster Editing a search tree is a tree where each path from the root to the leaves represent a distinct possible solution. From here on the algorithms will be referred to by the sizes of their respective search trees.

1.3 Data Set Presentation

The Cluster Editing Algorithm is applied on three different data sets. A data set with fabricated pseudo-random data is used for testing purposes, while two real world data sets are used to demonstrate applications of Cluster Editing.

The first data set is what is referred to as Visual Object Feature Clustering data set throughout this report. Visual Object Feature Clustering is an application that uses a Computer Vision library to analyze a video feed or an image and arranges objects with similar characteristics into clusters. In this report an application where coins of different denominations placed in an area recorded by a web cam will be explored. The Computer Vision library makes it possible to distinguish between coins by radius and color. The collected data is used to construct a graph where each coin is represented as a vertex and the edges describe the similarity in size and color between two coins. Since the coin area is limited by the placement and resolution of the web cam, the number of coins, and thus the size of the graph, will be small.

The second data set is an instance of the Document Clustering Problem. In the Document Clustering application described in this report a set of documents are used to create a graph, where vertices represent documents and edges represent similarity relations between the documents.

Document Clustering methods can be used to group text documents or text content from web pages by subject. This feature is interesting in e. g, web search engines where a query can result in a large number of pages belonging to different categories.

2 Method

The implementation described in this paper can be broken down into three different parts: The Cluster Editing algorithms, the data set generators and the data structure.

The purpose of a data set generator is to create a graph representation of problem-specific information. The data structure consist of the graph representation and the basic operation interface required by the Cluster Editing algorithms. A Cluster Editing algorithm operates on the data set using the functionality of the data structure, transforming the generated graph into a cluster graph.

2.1 The Cluster Editing Algorithms

The optimality goal of the Cluster Editing algorithm is achieved by performing the cheapest possible set of changes to a graph by adding or removing edges such that the resulting graph is composed of isolated cliques. It can easily be observed that the number of possible ways a graph can be edited grows exponentially in the size of the input. The problem is proven to be in the class of NP-complete problems by Shamir et al. [7] for which it is assumed not to exist an efficient way of solving an arbitrary problem instance. Changes to the graph are referred to as edit steps. Since each edge has a weight (a negative weight in the case of a non-edge), the edit cost of an edge equals the absolute value of its weight. Edges with zero weight may not be part of the input graph but can appear later if the algorithm uses merging of vertices. The zero weight means that the edge can be considered present in the graph but there is no cost associated with removing it. The algorithm examines if the necessary modifications can be made with an accumulated edit cost less than or equal to the parameter k.

In the majority of the algorithms presented, a merging technique is used to perform multiple edit steps. Merging two vertices u and v is the operation of replacing u and v by one vertex u' with edge weights $weight(u', x) = weight(u, x) + weight(v, x)$ for all vertices $x \notin \{u, v\}$. If the two edges (u, x) and (v, x) have weights where one is positive while the other is negative, the parameter k is immediately reduced by $min(|weight(u, x)|, |weight(v, x)|)$. In the case of $weight(u, x) = -weight(v, x)$, k is reduced by $min(|weight(u, x)|, |weight(v, x)|) - \frac{1}{2}$ and when the created zero-edge is later resolved, k is reduced by $\frac{1}{2}$. This bookkeeping trick guarantees that k is always reduced in every edit step, since it ensures that resolving a zero-edge is associated with a cost, analogous to the cost of resolving an edge or a non-edge. The argument for reducing k immediately is that any conflict triple containing this edge must be resolved and when resolving the conflict, at least this minimum cost must be paid. In an optimal solution, choosing to branch on the other edge will cause the remaining cost to be paid. Edges can also be set to forbidden, which means to turn an edge into a non-edge and at the same time disallow any further editing of the edge.

The basic Cluster Editing algorithm proceeds by searching for any triple (a set of three distinct vertices) in the graph where a conflict exist. A conflict triple is a set of three vertices connected by exactly two edges. Conflict triples can never be a part of a cluster graph so in order to reach a final state of isolated cliques, every such conflict has to be resolved. As described later, there are several ways of resolving a conflict.

In the $O(3^k)$ algorithm, a conflict triple can be resolved in three different ways, either by removing one of the two edges or by adding the missing edge, thus bounding the size of the search tree by 3^k. In the $O(2.62^k)$ algorithm, the strategy is to find a conflict triple and arbitrarily branch on one of the two edges, either by merging its vertices or removing the edge.

The refined algorithms $O(2^k)$ and $O(1.82^k)$ use branching vectors to determine which edge to branch on. The branching vector is composed of two elements. The first element is the cost of removing the edge and the second is the cost of merging the two vertices that the edge connects. The decision is based on the branching rules of the algorithm. The rules differ for the two algorithms and are more detailed in the $O(1.82^k)$ algorithm. A branching rule is a definition of how an algorithm should proceed to solve a problem by processing two or more subproblems.

To determine an upper bound of the size of the search tree given a branching vector, consider the following: Let T_k denote the number of leaves in a search tree with the parameter k, and let the search tree branch in r different ways. The number of leaves in T_k equals the sum of the number of leaves for each branch. With d_j being how much k is reduced in branch $j \in \{1, 2, ..., r\}$, the following equation describes the number of leaves.

$$T_k = T_{k-d_1} + T_{k-d_2} + ... + T_{k-d_r} \tag{1}$$

Let z denote the factor by which the number of leaves of the tree increases when k increases by one, and with $T_0 = 1$, thus $T_k = z^k$, it follows from (1) that

$$z^k = z^{k-d_1} + z^{k-d_2} + ... + z^{k-d_r} \tag{2}$$

for any $k \geq max(d_1, d_2, ..., d_r)$, and especially for $d = max(d_1, d_2, ..., d_r)$, giving what is referred to as the characteristic polynomial.

$$z^d = z^{d-d_1} + z^{d-d_2} + ... + z^{d-d_r} \tag{3}$$

The first observation to be made is that $\forall j \in \{1, 2, ..., r\} : d_j > 0$ must hold for a positive solution to (3) to exist. Let $f(z) = z^d$ and $g(z) = z^{d-d_1} + z^{d-d_2} + ... + z^{d-d_r}$. Observe that $g(0) \geq 1$, $f(0) = 0$, $\forall z > 0 : f'(z) > 0$, and that $\deg f > \deg g$ thus

$$\exists z_1 > 1 : f(z_1) = g(z_1) \tag{4}$$

Suppose that z_1 is the smallest positive root to (3) and that another positive root z_2 exists, and let $\alpha = \frac{z_2}{z_1} > 1$.

$$\alpha^d\, f(z_1) - f(z_2) = \alpha^d\, z_1^d - \alpha^d\, z_1^d = 0 \tag{5}$$

Substitute $f(z_1)$ by $g(z_1)$ and $f(z_2)$ by $g(z_2)$ in (5) to get

$$\begin{gathered} \alpha^d\, g(z_1) - g(z_2) = \\ \alpha^d z_1^{d-d_1} - \alpha^{d-d_1} z_1^{d-d_1} + \alpha^d z_1^{d-d_2} - \alpha^{d-d_2} z_1^{d-d_1} + ... + \alpha^d z_1^{d-d_r} - \alpha^{d-d_r} z_1^{d-d_1 r} = \\ (\alpha^d - \alpha^{d-d_1}) z_1^{d-d_1} + (\alpha^d - \alpha^{d-d_2}) z_1^{d-d_2} + (\alpha^d - \alpha^{d-d_r}) z_1^{d-d_r} = 0 \end{gathered} \tag{6}$$

But

$$\forall j \in \{1, 2, ..., r\} : (z_1^{d-d_j} > 0) \wedge (\alpha^d - \alpha^{d-d_j} > 0) \Rightarrow \alpha^d\, g(z_1) - g(z_2) > 0 \tag{7}$$

which is a contradiction, thus (3) must have a unique positive root.

The positive root of the characteristic polynomial derived from the branching vector is called the branching number. The highest possible branching number b for any branching vector the algorithm uses, bounds the size of the search tree by $O(b^k)$.

While the $O(3^k)$ and $O(2^k)$ algorithms always terminate in a state with a union of isolated cliques, the $O(1.82^k)$ algorithm may end up in a state that cannot effectively be solved by further applying the algorithm. In such a state the recursion stops and one of three polynomial-time algorithms are applied to solve the remaining subgraph, which is described further in section 2.1.5.

The implementation of the Cluster Editing algorithms described in this report continue to traverse the search tree after finding a result, but limits the depth to the currently known lowest cost, which makes the implementation capable of finding all possible optimal solutions. The implementation does however discard all but one best solution in its current state.

2.1.1 Kernelization

To improve the running time of the algorithms, a method called kernelization is applied, as described by Böcker et al. [8]. Kernelization is a technique that reduces the size of the problem kernel in polynomial time such that the problem kernel is bound in size by a function of the parameter k rather than the size of the input. The problem kernel is defined as a reduced instance of the original problem [6]. In the kernelization method described by Böcker [8], the number of vertices in the problem kernel are $O(k^2)$ after kernelization has been applied.

The kernelization described by Böcker et al. is quite simple and works as follows. For each pair of vertices the minimum induced cost of forbidding the edge between the vertices and for merging the vertices is calculated. More formally the minimum induced cost for forbidding an edge (u, v) is the sum of $min(weight(u, x), weight(v, x))$ for all vertices x that is a common neighbor of u and v. The minimum induced cost for merging two vertices u and v is the sum of $min(|weight(u, x)|, |weight(v, x)|)$ for all vertices x that have an edge to exactly one of u or v.

If there exist two vertices with an induced cost of forbidding or merging that is greater than the parameter k, the vertices are forbidden or merged accordingly and the parameter k is reduced by the cost of merging or forbidding. When forbidding or merging, the induced cost of other pairs of vertices can be affected as well. The induced costs for the pairs affected are updated and if there is no induced cost larger than k the iteration is stopped. When the iteration has stopped the problem kernel is bounded to have at most $O(k^2)$ vertices [8].

2.1.2 $O(3^k)$ Cluster Editing Algorithm

The $O(3^k)$ algorithm finds a conflict triple in the input graph and resolves it by branching on either inserting the missing edge or deleting either one of the two edges that are present. The cost for deleting or inserting an edge will be added to the accumulated editing cost for the branch. The algorithm will continue to branch until all conflict triples have been resolved or there is no branch where the accumulated editing cost is less than or equal to k. This is the Cluster Editing algorithm with a search tree of size $O(3^k)$ as described by Niedermeier [6]. Pseudo code for the algorithm can be found in appendix A.1.

In the simplest case this algorithm will have the complexity $O(n^3\,3^k)$, where the n^3 term corresponds to the number of triples. The naive approach on finding a conflict triple is to list all triples and for every triple check if it is a conflict triple. This can be done by iterating through every unique combination of three vertices in $O(n^3)$ time. The 3^k term corresponds to the the worst case size of the search tree in which all edit costs for edges and non-edges are 1 and -1 respectively. The algorithm recurses to the depth k with three separate calls on each depth, resulting in a search tree of size $O(3^k)$. By using kernelization before applying the algorithm a time complexity of $O(k^6\,3^k + n^3 \log n)$ can be achieved. The k^6 term is due to the fact that the kernelization guarantees that the size of the modified graph will be $O(k^2)$. In each step of the recursion, a conflict triple has to be found in the modified graph, which is done in $O((k^2)^3)$ time, giving the combined complexity of $O(k^6)$. The search tree is still of size $O(3^k)$ in the worst case so that factor remains unchanged. Finally, the kernelization takes $n^3 \log n$ time.

To further reduce the complexity a technique called interleaving [9] can be used. The general idea is to use kernelization while traversing a search tree. When applying interleaving, the polynomial factor in front of the exponential is reduced to a constant factor giving an overall complexity of $O(3^k + n^3 \log n)$.

2.1.3 $O(2.62^k)$ Cluster Editing Algorithm

In the $O(2.62^k)$ Cluster Editing algorithm the strategy is to find a conflict triple, and arbitrarily select one of the two edges in the conflict triple to branch on. The branching can occur in two ways, either the edge will be forbidden or the vertices of the edge will be merged. In either way the edit cost for the operation will be added to the accumulated total edit cost. The algorithm will continue to branch until either all conflict triples have been resolved or no branch with an edit cost of at most k remains. The complexity analysis is omitted as it has been done by Bcker et. al. [5]. Pseudo code for the algorithm can be found in appendix A.2.

The analysis of the $(2.62^k$ algorithm is simular to the $O(3^k)$ one, mainly differing in the size of the search tree. When deciding which edge to branch on, the worst case gives a branching vector of $(1, \frac{1}{2})$ occurring when resolving a zero-edge, which gives a search tree of size $O(2.62^k)$. This in turn means that the simplest case will have the complexity $O(n^3\,2.62^k)$. When using kernelization combined with interleaving the complexity becomes $O(2.62^k + n^3 \log n)$.

2.1.4 $O(2^k)$ Cluster Editing Algorithm

In the $O(2^k)$ Cluster Editing algorithm, the strategy is to find the edge in the graph with the lowest branching number. The algorithm will then branch on this edge, which is done by either following the branch where the selected edge has been deleted or the branch where the vertices of the selected edge have been merged and in both cases the edit cost for the deletion or merge will be added to the accumulated total edit cost for the whole graph. The algorithm will continue to branch using this approach until either all branching numbers are infinite or there is no branch that resolves the input graph with an accumulated edit cost of at most k. If all the branching numbers are infinite a solution has been found. This is because in a cluster graph, all pairs of vertices will either have zero cost for merging the vertices if they are in the same cluster or zero cost for fordidding the edge between the vertices if they are in different clusters. The costs for forbidding and merging two vertices are the components of the branching vector, if one of these two are zero, the branching vector gives a infinite branching number. What distinguishes the $O(2^k)$ algorithm from the $O(2.62^k)$ algorithm is that the former examines the whole graph, while the latter only considers the first observed conflict triple. This is the Cluster Editing algorithm with a search tree of size $O(2^k)$ as described by Böcker et al. [5]. Pseudo code for the algorithm can be found in appendix A.3.

In the $O(2^k)$ algorithm, the worst case branching vector is $(1, 1)$ which generates a search tree of size at most $O(2^k)$. Hence, the algorithm will have the complexity $O(n^3\,2^k)$ in the simplest case. Using kernelization and interleaving the complexity is reduced to $O(2^k + n^3 \log n)$.

2.1.5 $O(1.82^k)$ Cluster Editing Algorithm

Unfortunately the $O(1.82^k)$ algorithm was not completely implemented due to problems discussed in section 4.3. The algorithm will be explained here for the sake of completeness. The $O(1.82^k)$ algorithm is the currently fastest known algorithm for solving the Cluster Editing Problem. It has its origin in the $O(2^k)$ algorithm but use a larger set of more specific branching rules. While the $O(2^k)$ algorithm only branches on the edge with the lowest branching number, the $O(1.82^k)$ algorithm applies five more complex branching rules. There is however no guarantee that only applying the five rules is sufficient to give a union of isolated cliques as result. The algorithm is interrupted if neither of the branching rules can be applied to the remaining isolated subgraphs, which at this point must be either paths, circles, weak cliques, cliques minus an edge, or graphs of size at most four. In the case of a weak clique, the clustering is completed. In the case of a path or circle the cost of resolving the remaining graph is found using dynamic programming. In the case of a clique minus an edge, the cost is determined by the minimum cost of adding the missing edge or the minimum cost of splitting the almost-clique in two, with the vertices of the missing edge on either side of the splitting. The minimum splitting cost is found using a min-cut algorithm. Graphs consisting of four vertices are solved by means of brute force.

These cases can all be resolved in polynomial time, and are applied only once per subgraph, thus it will not affect the exponential part of the complexity of the Cluster Editing algorithm. This is the Cluster Editing algorithm with a search tree of size $O(1.82^k)$. The proof of the worst case search tree size of the algorithm will be omitted in this report, however it has been described by Böcker et al. [5].

In the worst case the $O(1.82^k)$ algorithm has search tree of size $O(1.82^k)$, which means that the algorithm will have complexity $O(n^4\,1.82^k)$, since the complexity for finding an edge to branch on has the worst case complexity $O(n^4)$. With kernelization and interleaving the algorithm will have the complexity $O(1.82^k + n^3 \log n)$.

2.1.6 Search Tree Traversal

Since Cluster Editing algorithms potentially deal with large graphs, leading to a large number of computations, it is advantageous if the implementation of the algorithm is scalable, to allow it to perform well on both small and large problems. A way of making the algorithm scalable is to allow it to be executed on several processors or computers in parallel. The implementation described in this report uses a scheduler which manages jobs from an algorithm according to a strategy. In this case the algorithm is an arbitrary Cluster Editing algorithm and the strategy is a incremental Depth-First Search tree traversal. A job in the scheduler is a vertex in the search tree, including the handling of editing cost and dispatching of new sub branches to the scheduler. The benefit gained is that there could potentially be several computers acquiring jobs from a central scheduler and adding jobs to the same central scheduler. It is also possible to change the strategy of the scheduler to an optional tree traversal algorithm other than incremental Depth-First Search. Having a scheduler does not imply that the computer program described in this paper can be run on several computers in parallel, only that the necessary preparations have been implemented to allow extensions of the program for parallelization of the Cluster Editing algorithms.

2.2 Data Structure

The data structure represents a graph consisting of vertices and edges, and provides a set of operations to modify the graph. Vertices are represented by integers denoting the index of the vertex. Edges are represented by an index and an integer, where the edge index is a unique value determined by the two vertices it connects, and the integer is the edge weight.

Some algorithms calculate branching numbers out of branching vectors, and therefore need to apply bookkeeping explicitly. Therefore support for costs with resolution of halves is needed, which is implemented using fixed point values, stored as integers.

Table 1: The names of the operations in the data structure, their description and their time complexity.

Operation	Description	Complexity
getWeight	Get the weight of an edge	$O(1)$
setWeight	Set the weight of an edge	$O(1)$
setAllCosts	Set the weight of all edges	$O(n^2)$
merge	Merge two nodes	$O(n)$
setForbidden	Set an edge to forbidden	$O(1)$
setPermanent	Set an edge to Permanent	$O(1)$

Coupled with the data structure, a change set handler is used to save storage space and maintain information about the graph. As described in sections 2.1.2, 2.1.3, 2.1.4 and 2.1.5, all of the different algorithms using the data structure will branch into several different possible solutions with different graph states. Consequently, some technique to keep track of the local graph state for the different branches is needed. Making a local copy of the current graph for each branch would be an extremely inefficient use of memory due to the potentially large number of possible branches and sub branches with near identical graphs. A more efficient solution is to use change sets. In a change set, a master graph state is maintained and the different branches only keep track of a set of changes that can be applied to the master graph state transforming it into the local graph state that is relevant to that specific branch. Furthermore the change sets used are reversible. A reversible change set allows for having the master graph state represent a local graph state of some branch, where it is possible to transform it to represent the local graph state of another branch through a set of changes and then transform it back to the original branch by applying the set of changes in reverse. I.e, the reversible change sets facilitate making changes to the master graph state between different local graph states without losing any information and while still maintaining just one copy of the master graph state.

2.3 Data Set Generation

In this project there are three data set generators, Document Clustering, Visual Object Feature Clustering, and one generating pseudo-random graphs with certain properties. In the Document Clustering and Visual Object Feature Clustering data sets the edges represent similarities in word occurrences between documents and the difference in size and color between coins respectively.

2.3.1 Random Data Set

A pseudo-random data set generator is used to produce test data. The generated data set has a defined number of vertices, cliques, and a well known maximum value of k, making it appropriate for testing purposes.

The generator initially creates a cluster graph with a given number of vertices, and a given number of cliques, where all cliques have equal size. Values of the edges are given a random number between 1 and a defined upper limit. Non-edges are generated as edges, but assigned a negative value.

Noise is added by inverting a number of random edges. To create a well defined total cost, the value is set to half the upper limit, making the total cost of converting the graph to the original cluster graph, half the upper limit multiplied by the number of changed edges.

2.3.2 Visual Object Feature Clustering Data Set

Visual Object Feature Clustering is the process of grouping objects in one or several images into clusters with similar objects. The objective is to group coins of different denominations into clusters of coins with the same denomination, based on their radius and color in a web cam stream. The coins are placed in an area that is captured by a web cam stream. The web cam stream is analyzed using an open source Computer Vision library, openCV [10], identifying circular objects along with their attributes. The difference in radius and the difference in colors between the coins are scaled and added to generate a similarity value. The similarity value is subtracted from a threshold value to generate the weight of the edge between the coins. An edge or non-edge is thereby present between each pair of vertices and thus there is a graph representation of the original image. This graph is used as the input graph for the Cluster Editing. The output from the algorithm will be a cluster graph where all coins that are similar are present in exactly one clique of the cluster graph and that way it can be determined how many coins of each denomination is present in the image.

2.3.3 Document Clustering Data Set

The Document Clustering algorithm creates an n-dimensional vector space from the set of all words in all documents combined, with each word representing one dimension. High frequency words with little or no relevance called stopwords, are filtered out and do not constitute a base for calculating the relations.

Each document is represented as a vector in the created vector space, with the coordinate for each dimension corresponding to the number of occurrences of that word in the document. As a value of similarity, the cosine of the angle between the vectors is used and is calculated by taking the scalar product of the normalized vectors. The Cluster Editing algorithm works with integer valued edge costs, while the similarity values are real numbers in range $[0, 1]$. Therefore the values are scaled, truncated and have an offset subtracted before the Cluster Editing algorithm is applied. The offset is referred to as the similarity threshold.

The set of data selected as input for the proposed Document Clustering application consist of articles extracted from Wikipedia. The advantages of using Wikipedia is that the articles are already classified by subjects. The assumption is that it is likely that articles in a certain category will use similar language which will cause the Document Clustering application to group such articles in the same cluster. The articles are extracted from an XML dump [11] of all articles on Wikipedia and therefore contain layout tags used to present the articles on Wikipedia.org. The layout tags have been added to the stopword list to avoid causing interference in the comparisons. The stopword list can be found in appendix B. A total of 20 categories containing more than 30 article pages per category were selected from Wikipedia by using the random article feature on the Wikipedia front page. For each category the 30 initial articles were selected. Further, the 30 articles were constrained to contain more than 250 words to avoid articles that contain very little information. To get an equal number of articles in every category, categories containing less than 20 articles were removed and finally each remaining category was truncated until it contained exactly 20 articles (articles are removed in a last in, first out order). In this case 17 categories containing 20 articles each remained. The categories are listed in appendix C. On this set of articles, a few different tests were conducted as described in section 3.2. Note that the algorithms does not require that the same number of articles are selected from each category. However 20 articles were chosen from each category to make the data uniform.

2.4 Programming Language and Development Methodology

The programming language chosen for the project is C, mainly because the project group is familiar with the language. Three other programming languages were discussed, C++, Java and Python. The advantages of these languages over C would have been an extra layer of abstraction because of their object oriented nature and in the case of Python, possibly less development time. However, the project group felt that because of the relatively small scope of the project these advantages would not have been significant.

The development method was an incremental approach with early prototyping. The first prototype was the $O(3^k)$ Cluster Editing algorithm with a minimal data structure using an adjacency matrix and local copies of the whole data structure to keep track of the graph state. Later prototypes used the more advanced Cluster Editing algorithms and progressively more refined data structures as well as change sets instead of local copies. The incremental approach with early prototyping was exercised because the project group saw a need to change the underlying data structure as more insight was obtained about the algorithms during the course of the project. Thus, the aim was to have one part of the project group developing a prototype as soon as possible while another sub-group simultaneously started evaluating the data structure and developing the next prototype.

3 Result

Three Weighted Cluster Editing algorithms have been implemented, with search trees of size $O(3^k)$, $O(2.62^k)$ and $O(2^k)$ as described by Niedermeier [6] and Böcker et al. [5]. As a component in the algorithms, the kernelization rules described by Böcker et al. [8] have been implemented as well. Additionally two distinct applications that uses Cluster Editing have been developed, Document Clustering and Visual Object Feature Clustering. In this section the results of the tests that were conducted will be presented.

3.1 Comparison of the Algorithms

To get a firm grasp of how the Cluster Editing algorithms compare to each other in practice, a number of running time comparisons were made. Cluster graphs with 50 and 200 vertices were randomly generated and distorted by adding or removing a number of edges, making the resulting graphs non cluster graphs. Each time measurement is the average of three separate execution times of the same Cluster Editing algorithm on a graph with the same number of vertices and the same number of added or removed edges. The distortion on the input graphs ranged from 5 to 60 added or removed edges. Running times were measured in CPU time on an Intel Core2 Duo 1.6 Ghz with 3.2 GB RAM running Gentoo Linux 2.6.26.

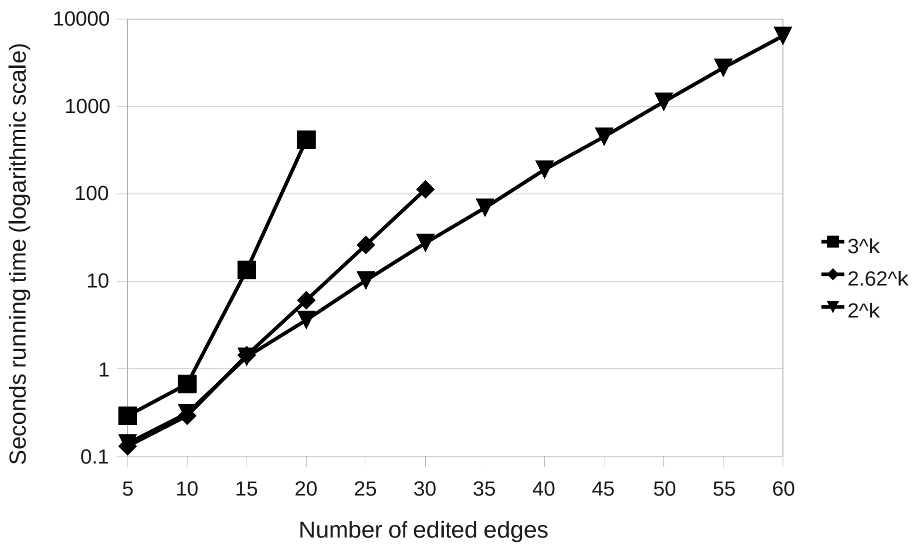

Figure 1: Running time comparison for the three Cluster Editing algorithms with interleaving. The graphs used in this test series have 50 vertices. Each graph in the test series is a modified cluster graph, with a number of noise edges, i.e. a number of edges that has been removed or added to the original cluster graph.

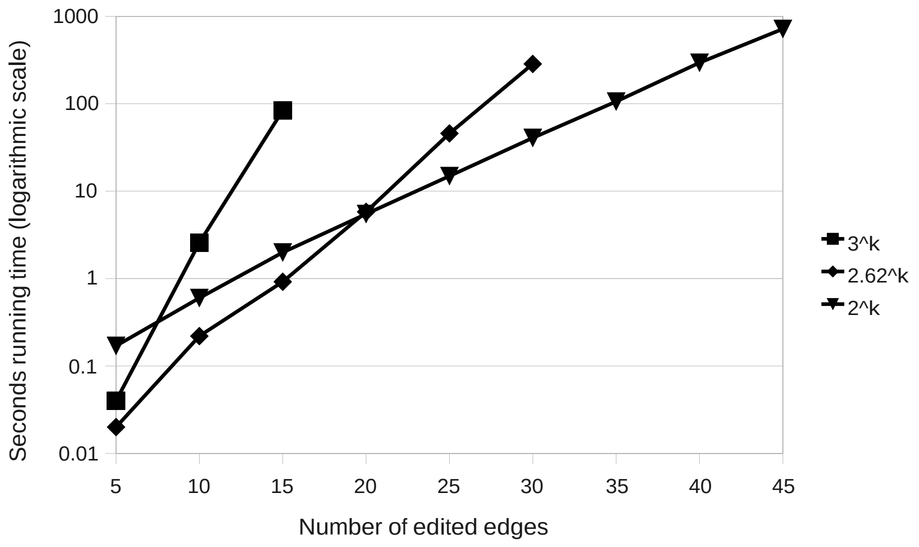

Figure 2: Running time comparison for the three Cluster Editing algorithms without interleaving. The graphs used in this test series have 50 vertices. Each graph in the test series is a modified cluster graph, with a number of noise edges, i.e. a number of edges that has been removed or added to the original cluster graph.

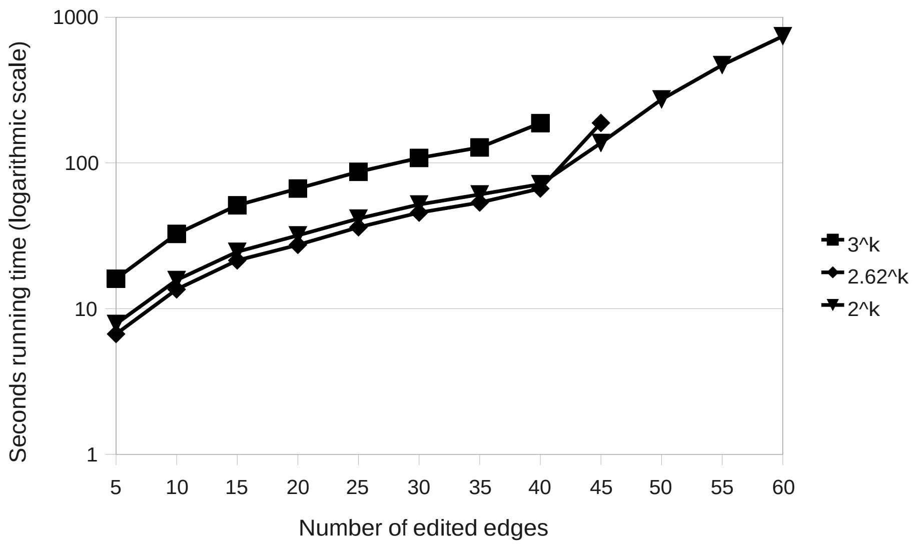

Figure 3: Running time comparison for the three Cluster Editing algorithms with interleaving. The graphs used in this test series have 200 vertices. Each graph in the test series is a modified cluster graph, with a number of noise edges, i.e. a number of edges that has been removed or added to the original cluster graph.

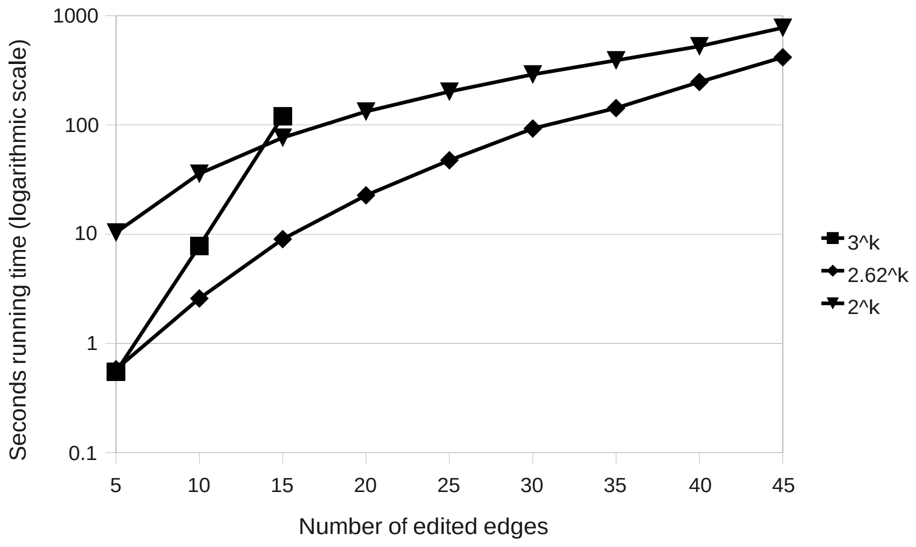

Figure 4: Running time comparison for the three Cluster Editing algorithms without interleaving. The graphs used in this test series have 200 vertices. Each graph in the test series is a modified cluster graph, with a number of noise edges, i.e. a number of edges that has been removed or added to the original cluster graph.

As can be seen in figure 1, 2, 3 and 4 the graphs for the $O(3^k)$ algorithm generally have the highest running time, except for very low numbers of noise edges in the non-interleaved cases. The $O(2^k)$ algorithm is slower than the other two algorithms for low numbers of noise edges. This is explained by the fact that the $O(2^k)$ algorithm calculates branching numbers for all edges after each edit step. These extra calculations will have a large impact on the overall running time when the number of edit steps is small, as is generally the case when the number of noise edges is small. It should also be noted that the running time grows exponentially with the parameter k for all three algorithms.

For lower numbers of noise edges, the algorithms with interleaving is slower, as seen in figure 1 and 3 compared to 2 and 4. The explanation for this is that in the cases where the number of noise edges are low, the time for interleaving has a greater impact on the overall running time.

3.2 Document Clustering

The goal of the Document Clustering tests is to get an idea of the usefulness of Document Clustering coupled with Cluster Editing for categorizing documents.

Three sets of tests were conducted on the Document Clustering data set. In the first test set, 20 articles from the same category were used as input. Cluster Editing was then applied on the graph generated from these articles, and the result was examined to see if there were any clusterings within the single category. Such clusterings would imply that there are articles that can be considered similar within the category. Absence of such clusterings could have several explanations, e. g. that the texts are not similar according to the vector representation of documents, that a better similarity threshold and scale for the Document Clustering algorithm need to be found or that the method used is not useful for clustering documents according to their similarity.

In the second and third test sets, articles from two and three categories respectively were clustered, and the results were examined to see if the clusters contained articles exclusively from a distinct category, which would imply that articles from different categories are clearly separated. In the second test set, 20 articles from each of two chosen categories are used as input articles, for a total of 40 input articles in each test. In the third test set, the input consists of 15 articles from each of three chosen categories for a total of 45 input articles in each test. Since no running time comparisons were done, the $O(2^k)$ Cluster Editing algorithm was used in all tests.

The results from the first test set can be seen in appendix D.1. It is worth noticing that 12 out of 17 categories have at least one large cluster. In this case, a cluster is considered large if it contains at least one third of the set of objects. No category resulted in clusters all containing only one article. Furthermore, most of the results contain several clusters of two or more articles.

The results from the second and third test sets can be seen in appendix D.2 and D.3 respectively. Note that there are consistently very few clusters containing articles from more than one category.

3.3 Visual Object Feature Clustering

A Visual Object Feature Clustering application was implemented, although primarily for demonstration purpose. Only minor tests were conducted on this application.

The implementation used a function available in OpenCV to detect arcs as part of objects in the image but resulted in detecting non-existing circles. Shadows interfered with the border of the coins and therefore resulted in detecting the radius incorrectly.

The Visual Object Feature Clustering implementation did manage to generate graphs near cluster graphs. In many cases, 20 or more coins resulted in a low cost and the implementation generally managed to correctly cluster different types of coins.

4 Discussion

In addition to implementing and comparing the Cluster Editing algorithms, the purpose of this project was to examine the suitability of certain problem instances for Cluster Editing. In this section, the suitability of the Document Clustering data set and the Visual Object Feature Clustering data set will be discussed, followed by an accounting of the parts of the $O(1.82^k)$ implementation that were not completed. Finally a few extensions to the implementations described in this report will be suggested.

4.1 Usefulness of the Document Clustering Data Set

Considering the results from the Document Clustering tests presented in section 3.2, assuming that articles in the same category on Wikipedia share similarities, these results seem to indicate that a Cluster Editing algorithm coupled with a Document Clustering data set generator will find similar documents and group them together. The fact that documents from different categories very rarely end up in the same cluster also seem to indicate that it is possible to classify two documents in the same cluster as also belonging to the same category, given a good similarity threshold for the documents.

However, the focus of this project was not on finding an optimized similarity function for comparing documents or to choose the input texts in a way that there is a guarantee that there will be groups of documents that are similar. Because of this, the graphs generated by the Document Clustering generator is not necessarily close to being cluster graphs. Therefore the cost of editing the generated graph into cluster graphs can be large. Unfortunately this implicates that the implementation described in this report is unsuitable in the general case for clustering a large number of input documents.

4.2 Usefulness of the Visual Object Feature Clustering Data Set

As opposed to the Document Clustering, the Visual Object Feature Clustering did manage to generate input graphs near cluster graphs, as mentioned in 3.3. This means that Visual Object Feature Clustering has the characteristics needed for efficiently being solved by Cluster Editing. Although the clustering technique is satisfactory, the problem of finding a good similarity threshold remains. Improvements could be made on the instruments and environment during the captures, e.g. using a background that reduces reflection of light along with proper lighting.

4.3 Incomplete Implementations

The implementation of the $O(1.82^k)$ Cluster Editing algorithm described in section 2.1.5 was never completed during the course of this project. The final stage of the algorithm is to solve the remainder of a graph when no edge meeting any of the branching rules can be found. The algorithms that solve the remainder of the graph all have in common that they require the graph to be a connected component. This means that the graph has to be split into separate subgraphs of connected vertices any time a group of vertices is discovered such that no edge connect this group with the rest of the graph. To be able to split a graph while traversing the search tree there has to be some way of keeping track of which subgraphs belong to the same vertex in the search tree. Unfortunately, the scheduler described in section 2.1.6 cannot store information on which subgraphs belong together without major changes. A module for identifying and splitting a graph into subgraphs has been implemented but is not used due to these constraints.

A scheduler is still desired, since it means that the algorithm could potentially be processed in parallel. However, if a scheduler is implemented with the $O(1.82^k)$ algorithm in mind, it has to be able to handle groups of subgraphs belonging to the same graph. If the project was granted more time, this is one of the first improvements that would have been implemented.

4.4 Possible Extensions of the Project

This project was carried out during a limited period of time and there are several ways to continue the work presented. The most obvious extension is to complete the $O(1.82^k)$ algorithm that was never finished. The flaws that lead to an incomplete $O(1.82^k)$ implementation were discussed in section 4.3.

More data reduction rules could be implemented and compared, e.g. those described by Böcker et al. [12].

Additionally, the application could be further evolved, possibly by a more thorough analysis of Document Clustering coupled with Cluster Editing, or by evolving Visual Object Feature Clustering into a real world application for practical use.

The Visual Object Feature Clustering application could be compared to a similar application using Classic Clustering. Another extension of Visual Object Feature Clustering could be to make it more generic, for example by using a better object identification method, and comparing more properties of the objects.

5 Conclusion

As shown in section 3.1 the theoretical complexity of the Cluster Editing algorithms translate well to practice, but a few exceptions exist. For input graphs where a very small number of edits is needed, the theoretically slower algorithms outperform the faster $O(2^k)$ algorithm unless interleaving is used. This is due to the overhead from computing branching numbers in the $O(2^k)$. As described in section 3.1 kernelization and interleaving will impose a significant impact on reducing the running time when the number of required edits grows.

The Document Clustering implementation described in this report was not suited for handling larger amount of documents. This was due to the fact that the input graphs were far from being close to cluster graphs, resulting in a large number of edit steps, as described in section 4.1.

As the Visual Object Feature Clustering implementation was mainly used for demonstration purposes, and as no extensive testing was performed, conclusions about the suitability of using Cluster Editing for solving this type of problem can not be drawn. However, the application of Visual Object Feature Clustering seems to be a promising subject for further studies, as discussed in section 4.2.

References

[1] Guo J, “A more effective linear kernelization for cluster editing.” *LNCS*, vol. 4614, pp. 36–47, 2007.

[2] Ben-Dor A, Shamir R, and Yakhini Z, “Clustering gene expression patterns.” *Journal of Computational Biology*, vol. 6, pp. 281–297, 1999.

[3] Bansal N, Blum A, and Chawla S, “Correlation clustering.” *Machine Learning 56*, pp. 89–113, 2004.

[4] Gramm J, Guo J, Hüffner F, and Niedermeier R, “Automated generation of search tree algorithms for hard graph modification problems.” *Algorithmica 39(4)*, pp. 321–347, 2004.

[5] Böcker S, Briesemeister S, Bui Q. B. A, and Truss A, “Going weighted: Parameterized algorithms for cluster editing.” *LNCS*, vol. 5165, pp. 1–12, 2008.

[6] Niedermeier R and Guo J. (2006) Fixed-parameter algorithms, lecture slides. Accessed 2009-05-01. [Online]. Available: http://theinf1.informatik.uni-jena.de/teaching/niedermeier-guo-fpt.pdf

[7] Shamir R, Roded R. S, and Tsur D, “Cluster graph modification problems.” *Discrete Applied Mathematics*, vol. 144, pp. 173–182, 2004.

[8] Böcker S, Briesemeister S, Bui Q, and Truss A, “A fixed-parameter approach for weighted cluster editing.” *APBC*, vol. 5, pp. 211–220, 2008.

[9] Niedermeier R and Rossmanith P, “A general method to speed up fixed-parameter-tractable algorithms.” *Information Processing Letters*, vol. 73, pp. 125–129, 2000.

[10] Sourceforge . (2009) opencvlibrary — sourceforge. Accessed 2009-05-01. [Online]. Available: http://sourceforge.net/projects/opencvlibrary/

[11] Wikipedia . (2009) “articles, templates, image descriptions, and primary meta-pages.”. Accessed 2009-05-11. [Online]. Available: http://download.wikimedia.org/enwiki/20090306/

[12] Böcker S, Briesemeister S, and Klau G. W, “Exact algorithms for cluster editing: Evaluation and experiments.” *LNCS*, vol. 5038, pp. 289–302, 2008.

A Pseudo Code

All algorithms take the same input and give the same result:

Input: The algorithm takes a graph G and a positive integer k, specifying the maximum allowed cost for editing edges in G.

Output: All cluster graphs with at most k edit cost from G.

A.1 $O(3^k)$ Cluster Editing Algorithm

```
name: clusterEdit3k ($G$, $k$)
If $k \geq 0$
    Find a conflict triple ($u$, $v$, $w$) in $G$ if exists
        clusterEdit3k (($G \setminus (u, v)$), ($k - |\text{edgeWeight}(u, v)|$))
        clusterEdit3k (($G \setminus (v, w)$), ($k - |\text{edgeWeight}(u, w)|$))
        clusterEdit3k (($G \cup (u, w)$), ($k - |\text{edgeWeight}(v, w)|$))
    Otherwise
        $G$ is a cluster graph
```

name: edgeWeight (u, v)
return the weight of the edge or non-edge (u, v).

A.2 $O(2.62^k)$ Cluster Editing Algorithm

```
name: clusterEdit262k ($G$, $k$)
If $k \geq 0$
    find an edge ($u$, $v$) in a conflict triple in $G$ if exists
        clusterEdit262k (($G \setminus (u, v)$), ($k - |\text{edgeWeight}(u, v)|$))
        clusterEdit262k ((merge($G$,$u$, $v$)), ($k -$ mergeCost($u$,$v$)))
    otherwise
        $G$ is a cluster graph
```

name: edgeWeight (u, v)
return the weight of the edge or non-edge (u, v).

name: merge(G, u, v)
returns an updated copy of the graph G where the vertex pair (u, v) has been merged and stored in u.

name: mergeCost(u, v)
returns the cost of merging the vertex pair (u, v).

A.3 $O(2^k)$ Cluster Editing Algorithm

```
name: clusterEdit2k ($G$, $k$)
If $k \geq 0$
    $branchingNumber_{min}$ := infinity
    For each edge ($u$, $v$) in $G$
        $branchingNumber_{uv}$ := getBranchingNumber($u$, $v$)
        if $branchingNumber_{uv} < branchingNumber_{min}$
            $branchingNumber_{min} := branchingNumber_{uv}$
            $(u_{min}, v_{min}) := (u, v)$
    If $branchingNumber_{min}$ is not infinity
        clusterEdit2k(($G \setminus (u_{min}, v_{min})$), ($k$ − edgeWeight($u_{min}$, $v_{min}$)))
        clusterEdit2k(merge($G$, $u_{min}$, $v_{min}$), ($k$ − mergeCost($G$, $u_{min}$, $v_{min}$)))
    Else
        $G$ is a cluster graph
```

name: edgeWeight(u, v)
returns the weight of the edge (u, v).

name: merge(G, u, v)
returns an updated copy of the graph G where the vertex pair (u, v) has been merged and stored in u.

name: mergeCost(u, v)
returns the cost of merging the vertex pair (u, v).

name: getBranchingNumber(u, v)
returns the branching number of branching vector (u, v), which is the maximum root z to the equation $z^d = z^{d-u} + z^{d-v}$ where $d = max(u, v)$.

B Stopword List

This is an alphabetical list of all the stop words that are used when processing text files with the document clustering algorithm. Items that are preceded by an asterisk (*) is in the stop word list because they are template words for wikipedia pages. Furthermore, combinations containing less than or equal to three characters are removed as stop words (not included in this list).

a about above *accessdate across after again against *align all almost alone along already also although always among *amp an and another any anybody anyone anything anywhere are area areas around as ask asked asking asks *asp at away b back backed *background backing backs be became because become becomes been before began behind being beings best better between big *book *border both but by c came can cannot *caption case cases *category *cellpadding *center certain certainly *cfm *cite *clear clearly *collapse come could d *date *dateformat did differ different differently do does done down down downed downing downs during e each early either *em *en end ended ending ends enough even evenly ever every everybody everyone everything everywhere *expand f face faces fact facts far felt few find finds *first *float for *format four from full fully further furthered furthering furthers g gave general generally get gets give given gives go going good goods got great greater greatest group grouped grouping groups *gt h had has have having he her here herself high high high higher highest him himself his how however i if *image *img important in interest interested interesting interests into is *isbn it its itself j *jpg just k keep keeps kind knew know known knows l *lang large largely *last later latest least less let lets like likely *list long longer longest *lt m made make making man many *margin may me member members men might more most mostly mr mrs much must my myself n *nbsp necessary need needed needing needs never new new newer newest next no nobody non noone not nothing now nowhere number numbers o of off often old older oldest on once one only open opened opening opens or order ordered ordering orders other others our out over p part parted parting parts *pdf per perhaps place places *png point pointed pointing points *position possible present presented presenting presents problem problems *publisher put puts *px q quite *quot r rather really *ref *relative *right room rooms s said same saw say says second seconds see seem seemed seeming seems sees several shall she should show showed showing shows side sides since small smaller smallest so some somebody someone something somewhere state states still *style such sure t take taken *text than that the their them then there therefore these they thing things think thinks this those though thought thoughts three through *thumb thus *title to toc today together too took toward turn turned turning turns two u under until up upon *url us use used uses v very w want wanted wanting wants was way ways *wb we *web well wells went were what when where whether which while who whole whose why *width will with within without work worked working works would x y *year years yet you young younger youngest your yours z

C List of Categories used in the Document Clustering Application

Dog Types
Military Technology
Dolls
Planetary Science
Water Transport
Game Shows
Food Ingredients
Coffee
Judaism
Currency
Native American Leaders
Furniture
Cities, Towns and Villages in Somalia
Lakes
Algebra
Drag Racing
Alcohol

D Document Clustering Results

D.1 Single Category Tests

Total Number of articles in each test: 20
Similarity Threshold used: -5
Scale used: 20
Cluster Editing algorithm used: $O(2^k)$

Category	# Clusters	Size of largest Cluster
Dog types	7	6
Military technology	11	5
Dolls	8	7
Planetary Science	6	12
Water Transport	14	4
Game Shows	7	12
Food Ingredients	19	2
Coffee	5	12
Judaism	11	7
Currency	6	14
Native American Leaders	15	3
Furniture	14	4
Cities, Towns and Villages in Somalia	2	19
Lakes	2	14
Algebra	7	9
Drag Racing	8	8
Alcohol	11	9

As an example: The test with texts from the *Dog Types* category:
DtXX: Text XX from the Dog types category
Total number of texts: 20
Largest cluster: 6 texts
Number of clusters: 7
Results:

Cluster 0: Dt0 Dt14 Dt3 Dt4
Cluster 1: Dt1
Cluster 2: Dt10 Dt2 Dt5 Dt7
Cluster 3: Dt11 Dt16 Dt21
Cluster 4: Dt12
Cluster 5: Dt15 Dt17 Dt18 Dt20 Dt6 Dt9
Cluster 6: Dt8

D.2 Double Category Tests

Total Number of articles in each test: 40
Similarity Threshold used: -5
Scale used: 20
Cluster Editing algorithm used: $O(2^k)$

Categories	# Clusters	size of largest cluster	# mixed Clusters
Dog types, Native American leaders	22	6	0
Dog types, Military Technology	18	6	0
Dog types, Food Ingredients	26	6	0
Dog types, Judaism	18	7	0
Military Technology, Currency	17	14	0
Military Tech., Cities in Somalia	13	19	0
Military Technology, Lakes	13	14	0
Dolls, Water Transport	22	7	0
Dolls, Native American leaders	23	7	0
Dolls, Drag Racing	16	8	0
Planetary Science, Coffee	10	12	1
Planetary Science, Alcohol	18	12	1
Water Transport, Drag Racing	22	8	0
Game Shows, Water Transport	21	12	0
Coffee, Currency	11	14	0
Coffee, Algebra	12	12	0
Judaism, Cities in Somalia	13	19	0
Currency, Native American leaders	21	14	0
Furniture, Food Ingredients	32	4	1
Furniture, Alcohol	26	7	1
Cities in Somalia, Lakes	4	19	1
Lakes, Algebra	9	14	0
Lakes, Drag Racing	10	14	0
Algebra, Alcohol	20	9	0

As an example: The test with texts from the *Planetary Science* and the *Coffee* categories:
PsXX: Text XX from the Planetary Science category
CoXX: Text XX from the Coffee category
Total number of texts: 40
Results:

Cluster 0: Ps0 Ps2 Ps6 Co1
Cluster 1: Ps1 Ps20
Cluster 2: Ps10 Ps11 Ps13 Ps14 Ps16 Ps17 Ps18 Ps3 Ps4 Ps5 Ps7 Ps8
Cluster 3: Ps12
Cluster 4: Ps19
Cluster 5: Ps9
Cluster 6: Co0 Co2 Co4 Co5 Co8
Cluster 7: Co10 Co11 Co12 Co13 Co14 Co16 Co17 Co18 Co19 Co20 Co6 Co9

Cluster 8: Co15
Cluster 9: Co3

D.3 Triple Category Tests

Total Number of articles in each test: 45
Similarity Threshold used: -5
Scale used: 20
Cluster Editing algorithm used: $O(2^k)$

Categories	# clusters	# mixed Clusters
Dog Types, Planetary Science, Game Shows	17	0
Dog Types, Coffee, Furniture	26	0
Dog Types, Furniture, Drag Racing	26	0
Military Technology, Dolls, Coffee	21	0
Military Technology, Dolls, Native American Leaders	28	0
Military Technology, Planetary Science, Furniture	26	2
Military Technology, Judaism, Algebra	25	0
Dolls, Planetary Science, Furniture	25	0
Dolls, Food Ingredients, Currency	26	0
Planetary Science, Water Transport, Lakes	18	0
Planetary Science, Game Shows, Judaism	13	0
Water Transport, Game Shows, Currency	19	0
Game Shows, Coffee, Judaism	20	0
Game Shows, Currency, Lakes	10	0
Game Shows, Cities in Somalia, Algebra	10	0
Food Ingredients, Judaism, Algebra	31	0
Food Ingredients, Cities in Somalia, Lakes	18	1
Coffee, Furniture, Alcohol	25	2
Judaism, Native American Leaders, Algebra	28	0
Judaism, Lakes, Alcohol	22	1
Currency, Native American Leaders, Algebra	21	0
Currency, Furniture, Lakes	19	0
Native American Leaders, Drag Racing, Alcohol	25	1
Furniture, Cities in Somalia, Alcohol	22	1

As an example: The test with texts from the *Cities, Towns and Villages in Somalia*, the *Algebra* and the *Game Shows* categories:
CsXX: Text XX from the Cities, Towns and Villages in Somalia category
AgXX: Text XX from the Algebra category
GsXX: Text XX from the Game Shows category
Total number of texts: 45
Results:

```
Cluster 0: Cs0 Cs10 Cs11 Cs12 Cs13 Cs14 Cs15 Cs16 Cs2 Cs3 Cs4 Cs5 Cs6 Cs8 Cs9
Cluster 1: Ag0 Ag10 Ag12 Ag14 Ag2 Ag4 Ag6 Ag8
Cluster 2: Ag1 Ag11 Ag13 Ag5
Cluster 3: Ag3
Cluster 4: Ag7
Cluster 5: Ag9
Cluster 6: Gs1 Gs11 Gs12 Gs14 Gs15 Gs2 Gs3 Gs4 Gs7 Gs9
Cluster 7: Gs10 Gs8
Cluster 8: Gs13 Gs6
Cluster 9: Gs5
```

E List of Acknowledgments

Anders Moberg:

Main responsibilities:
* The first version of the Data Structure
* The first version of the $O(3^k)$ algorithm
* Connected component splitting
* The Document Clustering algorithm
* The dynamic programming solution in the $O(1.82^k)$ algorithm
* Algorithm verification
Other significant contributions:
* Obtaining information (e.g. material concerning Kernelization)
* Writing of the Planning Report
* Writing of the Final Report
* Kernelization
The group's opinion. Throughout the project, Anders has:
* contributed with a wealth of ideas to the project
* shown a disposition for problem solving
* been very creative
* made relevant contributions to the group discussions
* shown a disposition for analysis of material related to the project

Christofer Chavanne:

Main responsibilities:
* The first version of the Data Structure
* The $O(1.82^k)$ algorithm
* The planning report, structure and writing
* The final report, structure and writing
Other significant contributions:
* Obtaining information (e.g concerning Kernelization)
* Code Debugging
The group's opinion. Throughout the project, Christofer has:
* been giving relevant critique in a positive way.
* shown a disposition for finding relevant information.
* been giving clear and thoughtful reviews.
* made relevant contributions to the group discussions.

Dan Engqvist:

Main responsibilities:
* The first version of the $O(3^k)$ algorithm
* The Document Clustering algorithm
* Producing data for the Document Clustering tests
Other significant contributions:

* Writing of the final report
The group's opinion. Throughout the project, Dan has:
...

Daniel Forsberg:

Main responsibilities:
* Random data set generator
* Refined Data Structure
* Kernelization
* Connected component splitting
* Wikipedia parsing
Other significant contributions:
* Writing of the planning report
* Writing of the final report
The group's opinion. Throughout the project, Daniel has:
* shown a disposition for analysis of material related to the project
* been very creative
* contributed with a wealth of ideas to the project
* been a team player
* shown a disposition for problem solving
* made relevant contributions to the group discussions.

David Gullmarsvik:

Main responsibilities:
* Project Leader
* Planning report, structure and writing
* Final report, structure and writing
* The $O(1.82^k)$ algorithm
* Producing data for the document clustering tests
Other significant contributions:
* Obtaining information
* First version of the data structure
The group's opinion. Throughout the project, David has:
* been progressive
* made relevant contributions to the group discussions
* been ambitious
* been good at producing drafts of texts

Gustav Lindqvist:

Main responsibilities:
* Project secretary
* Maintaining the group contract
* Random data set generator
* Refined Data Structure

* The $O(1.82^k)$ algorithm
Other significant contributions:
* Writing the planning report
* Writing the final report
The group's opinion. Throughout the project, Gustav has:
* made relevant contributions to the group discussions
* shown a disposition for problem solving
* been very creative

Max Sikström:

Main responsibilities:
* Source code management and project hosting
* Visual Object Feature Clustering
* Graph state
* File I/O
* Scheduler
* The second version of the $O(3^k)$ algorithm
* The $O(2.62^k)$ algorithm
* The $O(2^k)$ algorithm
* User interface/main file/makefiles/scripts
* Running time comparisons of the different algorithms
Other significant contributions:
* The refined Data Structure
* Kernelization
* Writing the planning report
* Writing the final report
The group's opinion. Throughout the project, Max has:
* shown a disposition for problem solving
* been very creative
* contributed with a wealth of ideas
* produced the model
* made relevant contributions to the group discussions
* shown alot of initiative