# EMPIRICAL COMPARISON OF GRAPH CLASSIFICATION AND REGRESSION ALGORITHMS

By

NIKHIL S. KETKAR

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTORATE OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of NIKHIL S. KETKAR find it satisfactory and recommend that it be accepted.

_____

Lawrence B. Holder, Ph.D.

_____

Diane J. Cook, Ph.D.

_____

Ananth Kalayaraman, Ph.D.

# EMPIRICAL COMPARISON OF GRAPH CLASSIFICATION AND REGRESSION ALGORITHMS

## Abstract

by Nikhil S. Ketkar, Ph.D,
Washington State University
May 2009

Chair: Lawrence B. Holder

Several domains are inherently structural; relevant data cannot be represented as a single table without significant loss of information. The development of predictive models in such domains becomes a challenge as traditional machine learning algorithms which deal with attribute-valued data cannot be used. One approach to develop predictive models in such domains is to represent the relevant data as labeled graphs and treat subgraphs of these graphs as features on which to base the predictive model.

The general area of this research is the development of predictive models for such domains. Specifically, we target domains which are readily modeled as sets of separate graphs (rather than a single graph) and on the tasks of binary classification and regression on such graphs. An example would be learning a binary classification model that distinguishes between aliphatic and aromatic compounds or a regression model for predicting the melting points of chemical compounds.

The contributions of this work include a comprehensive comparison of current approaches to graph classification and regression to identify their strengths and weaknesses, the development of novel pruning mechanisms in the search for subgraph features for the graph regression problem, the development of a new algorithm for graph regression called gRegress and the application of current approaches in graph classification and regression to various problems in computational chemistry.

Our empirical results indicate that our pruning mechanisms can bring about a significant improvement in the search for relevant subgraph features based on their correlation with each other and the target, sometimes by an order of magnitude. Our empirical results also indicate that gRegress addresses a key weakness in the current work on graph regression, namely, the need for a combination of linear models.

Table of Contents

# 1. Introduction

This chapter presents an introduction to the graph classification and regression problems. Before we introduce the formal definitions of these problems, we present some background on the subject.

The twenty-first century has seen an explosion of data in almost every field of human endeavor. As the amount of data grows, it becomes impossible for domain experts to analyze the data manually. To address these problems, machine learning and data mining techniques have found their way into a variety of application domains and in a number of cases have become indispensible research tools. The characterizing aspect of initial research in machine learning and data mining was the focus on developing models from attribute-valued data. Attribute-valued data is data represented as a single table consisting of a number of examples, each associated with a number of attributes and the values of these attributes. While techniques dealing with attribute-valued data are effective in many domains, they are inadequate when applied to problems from domains that are inherently structural.

The general area of this research is dealing with such domains. Specifically, we target domains which are readily modeled as sets of separate graphs (rather than a single graph) and on the tasks of binary classification and regression on such graphs. An example would be learning a binary classification model that distinguishes between aliphatic and aromatic compounds or a regression model for predicting the melting points of chemical compounds.

### 1.1. Formal Definitions

Our graphs are defined as $G = (V_G, E_G, L_G, \mathcal{L}_G)$, where $V_G$ is the set of vertices, $E_G \subseteq V_G \times V_G$ is a set of edges, $L_G$ is the set of labels and $\mathcal{L}_G$ is the labeling function $\mathcal{L}_G: V_G \cup E_G \rightarrow L_G$. The notions of subgraph (denoted by $G \subseteq G'$), supergraph, graph isomorphism (denoted by $G = G'$) and subgraph isomorphism in the case of labeled graphs are intuitively similar to the notions of simple graphs with the additional condition that the labels on the vertexes and edges should match.

Given a set of examples $E = \{(X_1, Y_1), (X_2, Y_2), \ldots (X_n, Y_n)\}$ where $X_i$ are labeled graphs and $Y_i \in \{+1, -1\}$ which represents the class, the graph classification problem is to induce a function which predicts the class from the labeled graph for unseen examples.

Given a set of examples $E = \{(X_1, Y_1), (X_2, Y_2), \ldots (X_n, Y_n)\}$ where $X_i$ are labeled graphs and $Y_i \in \mathbb{R}$ which represents the output value, the graph regression problem is to induce a function which predicts the output value from the labeled graph for unseen examples.

### 1.2 Key Issues in Graph Classification and Regression

Given the formal definitions of the graph classification and regression problems we can now identify the key issues involved in developing solutions for these problems. The purpose of identifying these issues is to present the contributions of this research, which are presented in the next section, in perspective.

Figure 1 illustrates a set of toy examples for the graph classification or regression problem. Given that the predictive models are to be based on subgraph features, we must take note of the search space of features for these examples. Figure 2 shows all the subgraphs for the toy example. It is easy to see that this search space is exponential

both in the size of the examples and the number of examples. A key issue in addressing the graph classification and regression problems is how this feature space is to be ordered. Figure 3 illustrates a simple 'is-subgraph-of' ordering of the feature space. Given such an ordering, the next issue is how this search space is to be searched so as to maximize a particular measure and how it might be possible to prune sections of this search space. Figure 4 shows how the search space might be drastically reduced by pruning mechanisms. Finally, after good features are identified, there is the issue of developing a predictive model based on these features. Figure 5 shows a simple, weighted, threshold based model for classification.

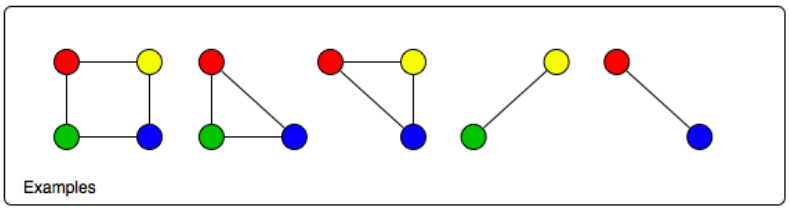


**Figure 1. Examples for the Graph Classification or Regression Task**

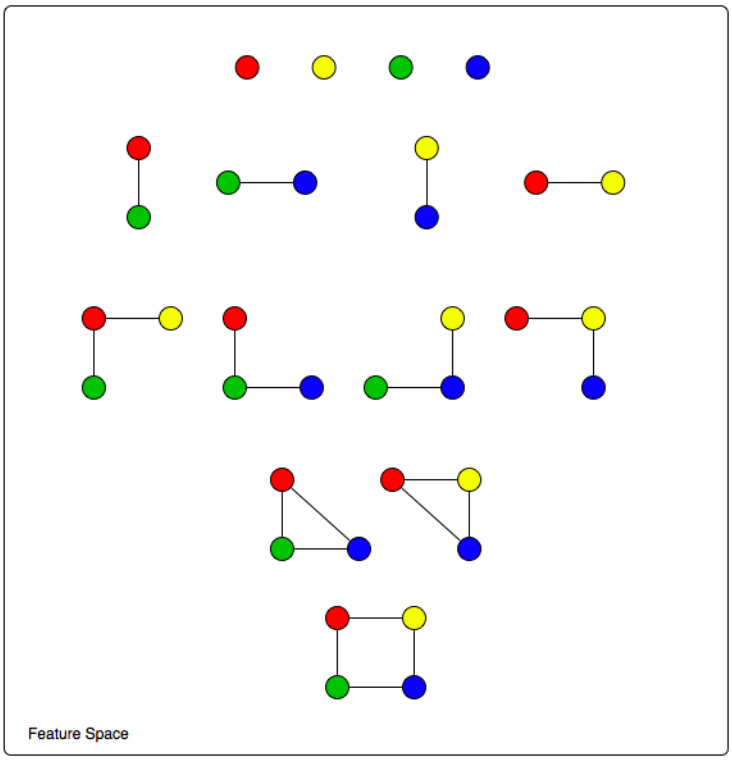


**Figure 2. Feature Space for the Examples**

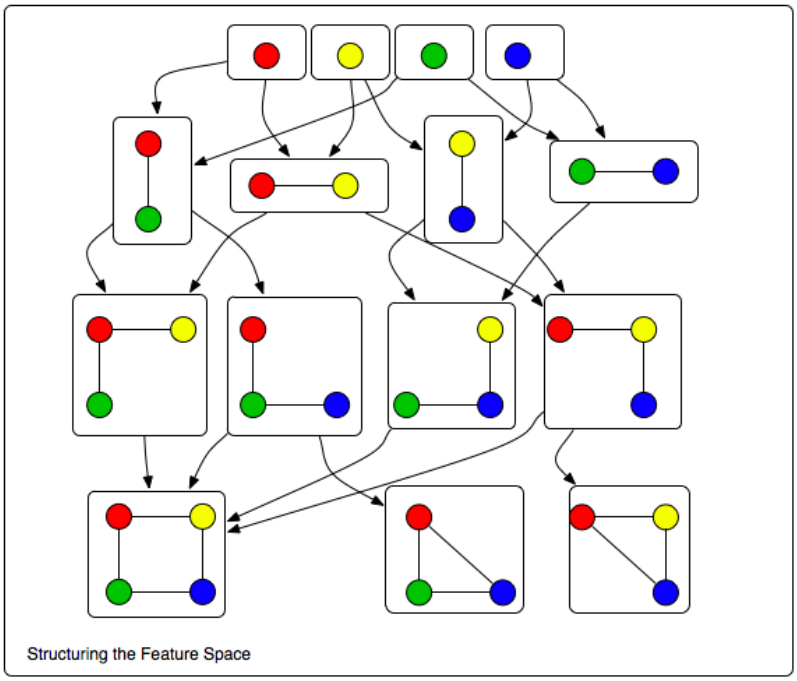


**Figure 3. Structuring the Feature space**

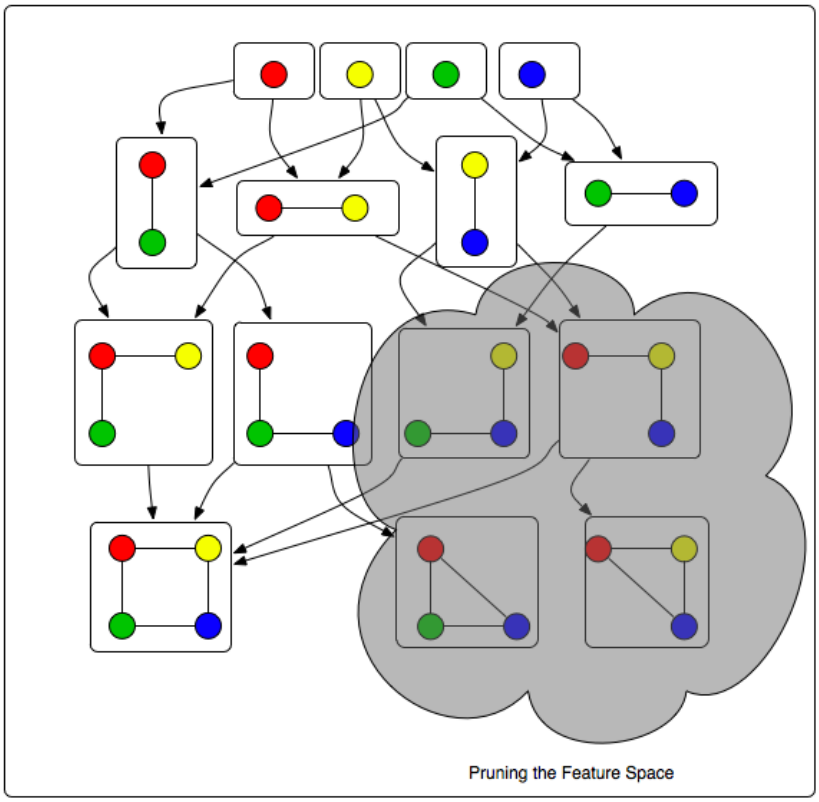


**Figure 4. Pruning the Feature Space**

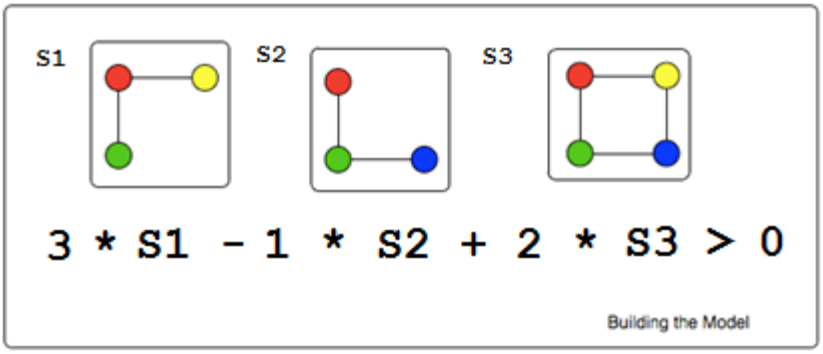


**Figure 5. Building the Model based on Selected Features**

## 1.3. Thesis Statement and Contributions

We now present our thesis statement which is as follows. *The state of the art in graph classification and regression algorithms can be improved both in terms of runtime and predictive accuracy by understanding and combining their strengths and by understanding and addressing their weaknesses.*

The rest of the document is devoted to supporting the thesis statement. The key contributions of our research, which are stepping stones towards validating the thesis statement, can be categorized as theoretical, empirical and application oriented.

### 1.3.1. Theoretical Results

1.3.1.1 Pruning Mechanisms for Feature Search

We developed a novel class of pruning mechanisms for searching the space of subgraph features. These pruning mechanisms are independent of the model building step and can be applied to a number of approaches to graph classification and regression.

#### 1.3.1.2 gRegress: Algorithm for Graph Regression

We developed a novel algorithm for graph regression called gRegress. Our algorithm outperforms previously introduced algorithms on graph regression and represents the current state of the art in graph regression.

#### 1.3.1.3 Faster Computation of Walk-based Kernel

We develop an alternative approach for the computation of the Walk-based kernel for graph classification. Although this approach was not able to outperform the current approach for the kernel computation, it is an interesting direction in the faster computation of graph kernels.

### 1.3.2. Empirical Results

#### 1.3.2.1 Strengths and weaknesses of graph classification algorithms

We conducted a comprehensive comparison of the major approaches to graph classification and identified the strengths and weaknesses of these approaches.

#### 1.3.2.2 Pruning mechanism can drastically improve run times

We conducted a performance study of our pruning mechanisms and demonstrated that pruning mechanisms can drastically improve the running time of graph regression algorithms.

#### 1.3.2.3 Certain problem domains need multiple linear models

We demonstrate that certain problem domains with respect to graph regression require multiple linear models. Models which are at their core based on a single linear function perform poorly as compared to models based on combinations of linear models.

1.3.2.4 Structure can improve performance of regression models

We demonstrate that in certain cases, incorporating structural features can drastically improve the performance of predictive models as compared to models based only on attribute valued features.

### 1.3.3. Applications

We successfully applied our observations, approaches and algorithms to various classification and regression problems from the domain of computational chemistry with promising results.

## 1.4. Organization of the Thesis

The rest of the document is organized as follows. Chapter 2 surveys the related work on the graph classification and regression problems. Chapter 3 introduces pruning mechanisms for the feature search in graph classification and regression. Chapter 4 discusses the gRegress algorithm and presents an empirical comparison of the gRegress algorithm with other approaches to graph regression. Chapter 5 presents a comprehensive empirical comparison of graph classification algorithms. Chapter 6 presents an alternative approach for the computation of the walk-based kernel. Chapter 7 presents the conclusions and the future work.

While all the chapters of the document contribute in some way towards supporting our thesis, the most significant of results are presented in Chapters 3 and 4. Table 1 summarizes the empirical observations and theoretical results in each chapter.

**Table 1. Organization of the Thesis**

| Chapter | Empirical Observations | Theoretical Results | Contribution towards validating the thesis |
|---|---|---|---|
| **Chapter 3** | Massive redundancy in search for subgraph features. | Developed pruning mechanisms in the search for subgraph features. | Order of magnitude improvement in runtime. |
| **Chapter 4** | 1) Need for combination of linear models in certain domains.<br><br>2) Including structure in models can improve predictive accuracy. | Developed gRegress, an algorithm that induces a tree based combination of linear models. | Significant improvement in predictive accuracy. |
| **Chapter 5** | 1) Differences in the behavior of graph classification algorithms despite similar performance.<br><br>2) Walk-based Graph Kernels cannot capture structure. | | Motivation behind thesis. |
| **Chapter 6** | | Language of Walks in a graph is a Regular Language. | Important development in the direction of improving runtime. |

# 2. Related Work

Learning from structured data such as graphs, first-order logic and relational databases emerged as a sub-field of machine learning and data mining and received a significant amount of attention by researchers in the time period between 1989 and 2009. The distinguishing aspect of this body of research from previous work in machine learning and data mining was that it considered the relationships between entities in the development of predictive models. This body of research has produced a variety of tasks, approaches and algorithms to address these tasks, and the application of these approaches to real-world problems with promising results. Here, we first present a brief overview of this work and then survey the particular literature relevant to this research.

We begin by making the following observations about learning from structured data.

1) Learning from structured data requires the expansion of the hypothesis space to include structural features.
2) Learning from structured data requires the expansion of the hypothesis space to include semantic features over structures.
3) Learning from structured data must consider the violation of the independent and identically distributed (I.I.D) assumption and revisit traditional machine learning algorithms.

Based on these observations, three major categories of models, approaches and algorithms have emerged. The first major category of models, approaches and algorithms is Inductive Logic Programming. The distinguishing aspect of this category is the use of first-order logic both for the representation of structured data as well as the induced models. Cheng et al. [7] survey all the

major developments in this category. The major strength of this category is its rich semantic power, and its major weakness is its computational complexity which restricts the induction of models with large structures [51].

The second major category of models, approaches and algorithms is Structural Relational Learning. The distinguishing aspect of this category is the identification, correction and exploitation of biases that cause pathological behavior in traditional machine learning algorithms when the assumption that the examples are independent and identically distributed (I.I.D.) is violated, as in the case of structured data. Getoor et al. [59] survey all the major developments in this category. The major strength of this category is the significant improvement of traditional machine learning algorithms when applied to structural data, and its major weakness is that complicated structural features are not captured by these models which typically use simple aggregation to generate structural features.

The third major category of models, approaches and algorithms is Graph Mining. The distinguishing aspect of this category is the use of a graph-based representation of structured data and the use of rich structural features like subgraphs in model building. Cook et al. [53] survey all the major developments in this category. The major strength of this category is that models can capture large structural features, and the major weakness of this category is the lack of formal semantics and the ad-hoc nature of the developed models [51].

The work conducted in our research belongs to the third category (Graph Mining), on which we now present a brief overview.

## 2.1. Graph Mining

The key tasks researched in Graph Mining are Frequent Subgraph Mining, Graph Classification, Graph Regression and their variants. We now present a concise overview of the same.

### 2.2.1. Frequent Subgraph Mining

The specific problem of mining frequent subgraphs was initially formulated in the work done by [19,22]. Significant improvements were made by introducing canonical labeling with DFS codes in the work done by [28]. The algorithm introduced in this work, called gSpan, remains to date, one of the best algorithms for the task. A novel form of canonical labeling using CAMs (canonical adjacency matrices) was introduced by [36]. The intuition is that the algorithm introduced in this work, FFSM, will outperform gSpan when the size of the graph transactions is large. This is because unlike gSpan, FFSM does not have to perform subgraph isomorphism to check the presence of a subgraph in graph transactions; this can be checked by operations on embedding lists of subgraphs when a new subgraph is generated by operations on CAMs. However, operations on embedding lists are quite costly. So in the case of moderately sized graph transactions, performing subgraph isomorphism might turn out to be cheaper and gSpan may perform better. Gaston [43] is another novel approach to the task which is based on the observation that most frequent graphs are not actually graphs but paths and trees. Searching for paths first, then trees and then finally graphs, the so called quick start, is a better way to organize and search the space efficiently. Various frequent subgraph mining algorithms are experimentally compared in the work done by [58] and [48]. Experimental results show that gSpan, FFSM and Gaston are the top performing frequent subgraph mining algorithms, but none

of them is clearly better than the other. A theoretical analysis of canonical labeling forms has been conducted in [46] and shows how the seemingly diverse canonical labeling systems in fact belong to the same general family of canonical forms.

Related to the task of frequent subgraph mining are tasks like mining closed frequent graphs [38], mining geometric subgraphs [60], maximal frequent subgraphs [41], induced subgraphs [28], trees [29], and the more recent work on mining from tenuous outerplanar graphs [56]. The work on outerplanar graphs is extremely significant from a theoretical standpoint as it is the only known class of graphs beyond trees for which frequent subgraph mining can be performed in incremental polynomial time.

### 2.1.2. Graph Classification

We now survey all the major approaches to graph classification.

#### 2.1.2.1 SubdueCL

The SubdueCL algorithm [25] is the pioneering algorithm for the graph classification problem. The key aspect of the algorithm is the greedy, heuristic search for subgraphs present in the positive examples and absent in the negative examples. The hypothesis space of SubdueCL consists of all the connected subgraphs of all the example graphs labeled positive. SubdueCL performs a beam search which begins from subgraphs consisting of all vertices with unique labels. The subgraphs are extended by one vertex and one edge or one edge in all possible ways, as guided by the input graphs, to generate candidate subgraphs. SubdueCL maintains the instances of subgraphs (in order to avoid subgraph isomorphism) and uses graph isomorphism to determine the instances of the candidate substructure in the input graph. Candidate substructures

are evaluated according to classification accuracy or the minimum description length principle. The length of the search beam determines the number of candidate substructures retained for further expansion. This procedure repeats until all substructures are considered or the user-imposed computational constraints are exceeded. At the end of this procedure the positive examples covered by the best substructure are removed.

The process of finding substructures and removing positive examples continues until all the positive examples are covered. The model learned by SubdueCL thus consists of a decision list, each member of which is a connected graph. Applying this model to classify unseen examples involves conducting a subgraph isomorphism test; if any of the graphs in the decision list are present in the example, it is predicted as positive, if all the graphs in the decision list are absent it the example, it is predicted as negative.

2.2.2.2 FSG+SVM

The FSG+SVM approach [49] for graph classification involves the combination of work done in two diverse fields of study, namely, frequent subgraph mining and support vector machines (SVM). The key idea in combining frequent subgraph miners and SVMs in order to perform graph classification is to use a frequent subgraph mining system to identify frequent subgraphs in the given examples, then to construct feature vectors for each example where each feature is the presence or absence of a particular subgraph, and finally to train a support vector machine to classify these feature vectors. The model produced by this approach thus consists of a list of graphs and a model produced by the SVM. Applying this model to classify unseen examples involves conducting a subgraph isomorphism test. First, a feature vector for the unseen example is produced where each feature is a Boolean representing the presence or absence of the graph in

the list. Then, this feature vector is classified as positive or negative by the model produced by the SVM.

#### 2.2.2.3 CLDT-GBI

The CLDT-GBI [26] approach involves combining aspects of frequent subgraph mining system GBI [19] and decision trees [4]. The approach induces a decision tree where every node is associated with a graph and represents an existence/nonexistence test in an example to be classified. The GBI system performs a heuristic, greedy search. In this approach, a variant of the GBI system, B-GBI which deals with overlapping candidate subgraphs, is used for feature generation. Broadly speaking, the approach involves a typical decision tree algorithm except that B-GBI is invoked to generate features at each node. The gain of each feature is computed on the basis of how the existence of the feature graph splits the examples at that node. This procedure is recursively applied until pure nodes with examples only from a single class are reached. In order to avoid overfitting, pessimistic pruning identical to C4.5 [4] is performed.

#### 2.2.2.4 Graph Kernels

Another approach to applying SVMs to graph classification is to use graph kernels, which, given two input graphs, output a similarity measure between the two graphs. This similarity measure is basically the inner product of the feature vectors of these graphs over a high dimensional feature space which can be feasibly computed without actually having to generate the feature vectors. The key work on this approach is the walk-based (direct-product) kernel [33] and the cycle-based graph kernel [33]. Recently, another kernel based on random walks on graphs [34] has been introduced.

#### 2.2.2.5 FSG+ADABoost

The FSG+ADABoost [47] approach involves combining aspects of frequent subgraph miners and AdaBoost [55] in a more integrated way than that of the FSG+SVM approach discussed earlier. Broadly speaking, the approach involves boosting decision stumps, where a decision stump is associated with a graph and represents an existence/nonexistence test in an example to be classified.

The novelty of this work is that the authors have adapted the search mechanism of gSpan which is based on canonical labeling and the DFS code tree for the search for such decision stumps. The key idea behind canonical labeling and the DFS code tree in gSpan is to prune the search space by avoiding the further expansion of candidate subgraphs that have a frequency below the user specified threshold as no supergraph of a candidate can have a higher frequency than itself. This idea cannot be directly applied to graph classification as the objective of the search is not to find frequent subgraphs but subgraphs whose presence or absence distinguishes positive examples from the negative ones. The authors prove a tight upper bound on the gain any supergraph of a candidate subgraph can have. Using this result, the proposed algorithm uses the search mechanism of gSpan, calculates and maintains the current highest upper bound on gain T and prunes the search space by avoiding the further expansion of candidate subgraphs that have a gain lower that T. The boosting of these decision stumps is identical to the meta-learning algorithm AdaBoost.

### 2.2.4. Graph Regression

The problem of developing regression models from graph transactions is relatively new as compared to the related problem of graph classification. While the graph classification problem involves learning to predict the (binary) class of unseen examples, the graph regression problem involves learning to predict the real-valued output associated with unseen examples.

The key work on graph regression thus far is that of applying partial least square regression [68] to the problem. Work related to the task of developing regression models from graph transactions also includes [61] in which the authors investigate the search from subgraphs with high correlation for a database of graph transactions. This work mainly dealt with the task of feature extraction; any attribute-valued regression algorithm could be applied to the extracted features.

## 2.3. Subgraph Isomorphism

Subgraph isomorphism is a fundamental building block of graph mining. Many approaches require performing subgraph isomorphism during the model building phase and almost all the approaches in graph mining require performing subgraph isomorphism during the prediction stage (except graph kernels). The subgraph isomorphism problem is NP-complete [70]. The subgraph isomorphism problem also has many other applications and has been studied to a great depth by researchers mainly in computer science and mathematics, and also by researchers in application specific areas like biochemistry and bioinformatics. Our research does not address the subgraph isomorphism problem directly, but we mention it here due to its fundamental role in graph classification and regression.

The pioneering algorithm for this task, proposed in [67], consists of a backtracking search. The large body of literature on this problem can be classified into two distinct categories. The first would be that of identifying and rigorously formulating special cases of subgraph isomorphism and establishing (by proof) the complexity class of such special cases. Examples of such work would be the work done on planar [68] and outerplaner [69] subgraph isomorphism. The second would be that of identifying special cases of subgraph isomorphism, designing algorithms to address these special cases and establishing, by experimental comparison, the improvement achieved by the approach. Examples of such work can be found in [71] and [72].

No major breakthroughs have been achieved in this area and are not to be expected in the recent future. While the NP-Completeness of subgraph isomorphism remains an obstacle to the successful application of graph classification and regression, research in this area will continue to search for tractable approaches for restricted cases of the problem.

# 3. Pruning Mechanisms

In this chapter we propose pruning mechanisms (in the search) for the task of extracting subgraph features from graph transactions. Given a set of graph transactions and a real value associated with each graph transaction the task is to extract the complete set of subgraphs such that

- Each subgraph in this set has correlation with the real value above a user-specified threshold.
- Each subgraph in this set has correlation with any other subgraph in the set below a user-specified threshold.

Our approach, embodied as a modification of gSpan, referred to as gSpanPrune, incorporates novel pruning mechanisms based on correlation of a subgraph feature with the output and correlation with other subgraph features. These pruning mechanisms lead to significant speedup. Experimental results indicate that in terms of runtime, gSpanPrune substantially outperforms gSpan, often by an order of magnitude while the regression models produced by both approaches have comparable accuracy.

## 3.1 Motivation and Intuition

Regression models are the trusted workhorse for predictive modeling in a variety of application domains. The problem of mining subgraph features from a database of graph transactions for building regression models is critical when an attribute-valued representation is insufficient to capture the domain of study. An example of such a scenario would be the case where we are trying to build a regression model for the toxicity of chemical compounds which is a real value collected from in-vivo experiments. The chemical compounds are represented as

graph transactions and the real value of interest associated with each transaction is the toxicity. Is such a scenario, the question is how to extract relevant features for building a regression model. Currently the state of the art in this regard is the large body of work on the problem of frequent subgraph mining (relevant literature on this topic was reviewed in Chapter 2). A typical frequent subgraph mining algorithm will mine the complete set of subgraphs with a user-defined frequency threshold and these subgraphs can be used as features to build a regression model. Such an approach involving feature extraction using a frequent subgraph mining algorithm has been studied in the context of the graph classification problem and has been applied to the task of classifying chemical compounds [49] and proteins [40] with promising results. However, this approach is plagued with a number of problems which we now illustrate by describing a small case study. The objective of this case study is to motivate our approach and set the stage for the rest of the chapter.

The case study involves building regression models for predicting the melting point of a set of chemical compounds (details on the data set can be found later in the thesis) based solely on subgraph features extracted by the frequent subgraph mining system gSpan [28] using support vector regression (SVR) [45]. We ran gSpan on the dataset at thresholds ranging from 20% to 5% in 1% decrements with a maximum subgraph feature size of 10. Regression models were built using the feature vectors based on the presence/absence of subgraph features using SVR (the particular implementation used was SVMlite [11]) and were evaluated using the $Q^2$ score on a 5-fold cross validation. The $Q^2$ score (which was used to evaluate gPLS [63]) is defined as follows.

$$Q^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{\sum_{i=1}^{n} \left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2}$$

The $Q^2$ score is close to 1 when the regression function *f* fits well, and is close to 0 when it does not. The $Q^2$ score for the model, the number of subgraphs discovered and the runtimes of gSpan for each threshold setting are illustrated in Figures 6, 7 and 8. We can observe the following. The predictive accuracy of the regression model improves as the threshold frequency reduces. This is an expected result [49] and has been observed earlier. It can be explained by the fact that additional relevant subgraph features are available on which the model can be based. The number of frequent subgraphs and the runtime also increase as the threshold decreases (as expected and observed earlier [49]) which in the worst case is expected to grow exponentially.

These observations raise the question of how many of the newly considered subgraph features actually contribute to increased predictive accuracy of the regression model. To answer this question we analyzed the frequent subgraphs generated at the threshold of 10%. Figure 9 shows the absolute pairwise correlations between subgraph features for those subgraphs whose absolute correlation with the output is at least 0.20. Pairwise correlation lower than 0.20 is denoted in blue, while pairwise correlation greater than 0.20 is denoted in red. The subgraphs in blue are the ones that contribute most to the predictive accuracy of the regression model based on these thresholds on correlation with the output and the pairwise correlations. While these thresholds are somewhat arbitrary, they do give a certain measure of the redundancy of the subgraphs generated. Typically, feature selection for building regression models considers the tradeoff between how much a feature correlates with the output and how much the feature correlates with the features already selected. Our intuition is that mining features based on their frequency

produces useful features but also produces additional redundant features at an added cost. Of course, redundant features could be eliminated by a simple post processing step, but this is computationally expensive as the redundant subgraphs are still generated in the first place. We should prefer to mine for a set of subgraphs such that each member of this set has high correlation with the output value and that the members of this set have low correlation with each other. Mining a complete set of subgraphs based on two thresholds, correlation with the output and correlation with other features, is an intuitive approach for building regression models and is also computationally efficient.

For a given subgraph feature, we prove an upper bound on the correlation with the output that can be achieved by any supergraph for this subgraph feature. For a given subgraph feature, we prove a lower bound on the correlation that can be achieved by any supergraph for this subgraph feature with any other subgraph feature. Using these two bounds we design a new algorithm called gSpanPrune, which extracts the complete set of subgraphs such that a) each subgraph in this set has correlation with the real value above a user-specified threshold, and b) each subgraph has correlation with any other subgraph in the set below a user-specified threshold.

We conduct an experimental validation on a number of real-world datasets showing that in terms of runtime, gGraphPrune substantially outperforms gSpan, often by an order of magnitude while the regression models produced by both approaches have comparable accuracy.

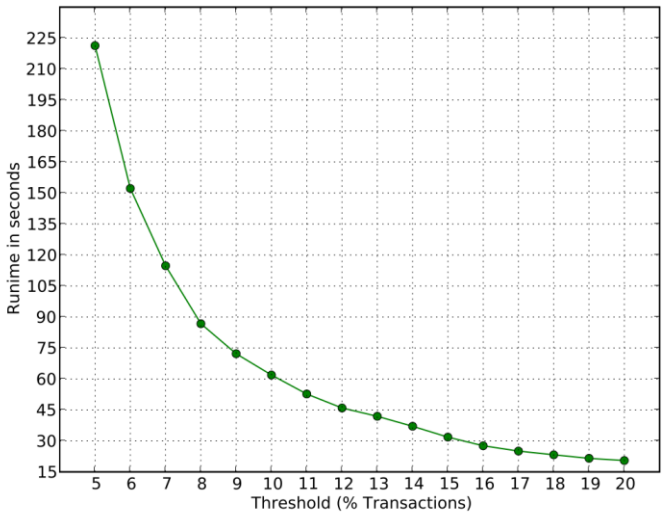


**Figure 6. Effect of Varying the Threshold on the Runtime of a Frequent Subgraph Mining System**

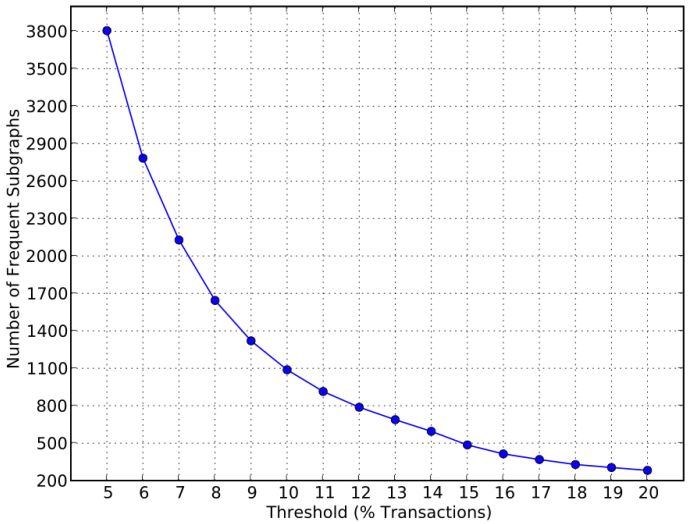


**Figure 7. Effect of Varying the Threshold on the Number of Subgraphs produced by a Frequent Subgraph Mining System**

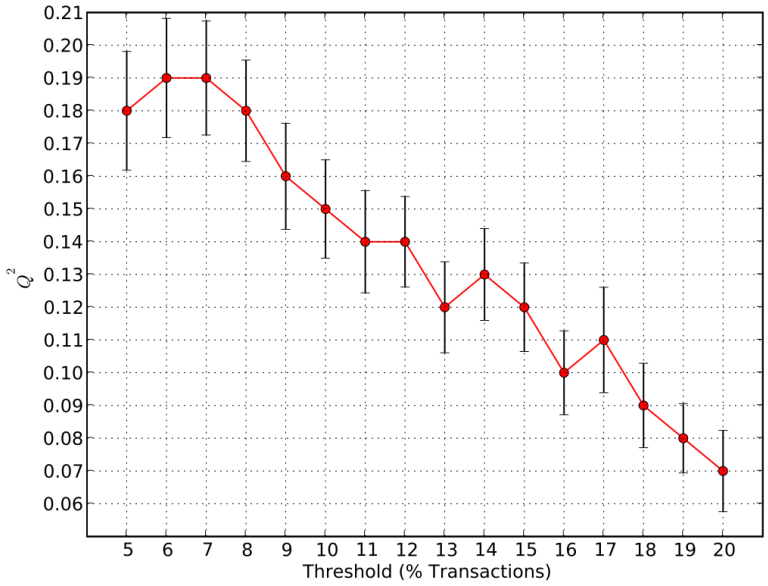


**Figure 8. Effect of Varying the Threshold on the Predictive Performance of the Model based on the Subgraphs Produced by a Frequent Subgraph Mining System**

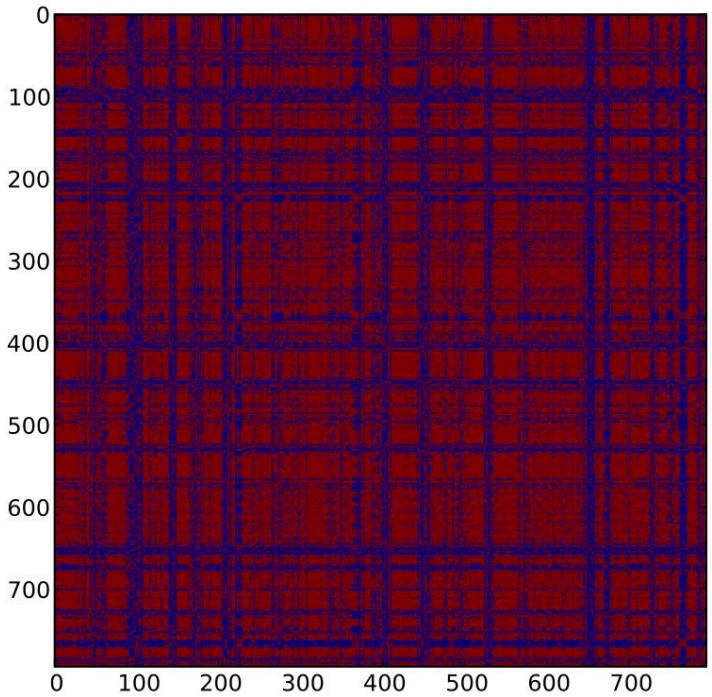


**Figure 9. Pairwise Correlations between the Subgraph Features, blue denotes pairwise correlations lesser than 0.2, red denotes pairwise correlations greater than 0.2.**

### 3.1 Theoretical Results

As discussed earlier, our graphs are defined as $G = (V, E, L, \mathcal{L})$, where $V$ is the set of vertices, $E \subseteq V \times V$ is a set of edges, $L$ is the set of labels and $\mathcal{L}$ is the labeling function $\mathcal{L}: V \cup E \rightarrow L$. The notions of subgraph (denoted by $G \subseteq G'$), supergraph, graph isomorphism (denoted $G = G'$) and subgraph isomorphism in the case of labeled graphs are intuitively similar to the notions of simple graphs with the additional condition that the labels on the vertices and edges should match.

Our examples consist of pairs, $E = \{< x_1, y_1 >, < x_2, y_2 >, \ldots, < x_n, y_n >\}$ where $x_i$ is a labeled graph and $y_i \in \mathbb{R}$ and is assumed to be centered, that is, $\sum_{y_i} = 0$. We define the set $\mathcal{S}$ to contain every distinct subgraph of every graph in $E$. For any subgraph feature $g$ we define the vector of existence of $g$ in each example $x_i$ as,

$$h_g(x) = \begin{cases} 1 \; if \; g \subseteq x \\ -1 \; otherwise \end{cases}$$

We define the indicator vector $I$ as the vector of target output values $y_i$ for each example $x_i$. The absolute correlation of a subgraph feature $g_i$ with the output is given by,

$$\rho_{g_i, I} = \left| \frac{h_{g_i} . I}{\left\| h_{g_i} \right\| . \|I\|} \right|$$

The absolute correlation of a subgraph feature $g_i$ with another subgraph feature $g_j$ is given by,

$$\rho_{g_i, g_j} = \left| \frac{h_{g_i} . h_{g_j}}{\left\| h_{g_i} \right\| . \left\| h_{g_j} \right\|} \right|$$

We can now define the problem as follows.

**Given:**

1. A set of examples $E$
2. A threshold on the correlation with the output, $\alpha \in \mathbb{R}, 0 \leq \alpha \leq 1$.
3. A threshold on the correlation between subgraph features, $\beta \in \mathbb{R}, 0 \leq \beta \leq 1$.

**Find:**

A maximal set $H = \{g_1, g_2, \ldots, g_k\}$ such that,

1. For each $g_i \in H$,

$$\rho_{g_i,I} = \left| \frac{h_{g_i} . I}{\left\| h_{g_i} \right\| . \|I\|} \right| \geq \alpha$$

2. For each $g_i, g_j \in H$,

$$\rho_{g_i,g_j} = \left| \frac{h_{g_i} . h_{g_j}}{\left\| h_{g_i} \right\| . \left\| h_{g_j} \right\|} \right| \leq \beta$$

We now discuss why it makes intuitive sense to mine for the set $H$. First, note that the formulation is in terms of absolute correlations. This is simply because we are interested in mining subgraph features with high correlations either positive or negative. Negative correlation, implying that the absence of a subgraph correlates with the output is equivalent to positive correlation as the regression model will simply learn negative weights for such a feature. Next, note that the set $H$ is the maximal or the largest possible set of subgraphs such that,

- Each subgraph in this set has correlation with the real value above a user-specified threshold $\alpha$
- Each subgraph has correlation with any other subgraph in the set below a user-specified threshold $\beta$.

Feature selection for building regression models considers the tradeoff between how much a feature correlates with the output and how much the feature correlates with the features already selected. The problem definition intuitively captures this trade off.

We now prove some properties about $H$. We define $\mathcal{S}_\alpha$ to be the maximal subset of $\mathcal{S}$ for which $\rho_{g_i,I} \geq \alpha$ and $g_i \in S$. We also define $H^*$ to be the set containing all possible $H$ sets.

**Lemma 1**

For any $\alpha, \beta \in \mathbb{R}, 0 < \alpha, \beta < 1$, $H \subseteq \mathcal{S}_\alpha \subseteq \mathcal{S}$.

**Proof**

As $\alpha < \rho_{g_i,I}$, $g_i \in \mathcal{S}$ $\mathcal{S}_\alpha \subseteq \mathcal{S}$ must hold true. Any choice of $\beta$ further constrains $H$ so that $H \subseteq \mathcal{S}_\alpha \subseteq \mathcal{S}$. The result follows. ■

**Lemma 2**

For $\alpha = 0$ and $\beta = 1$, $H = \mathcal{S}$.

**Proof**

As $\alpha = 0$, $\mathcal{S}_\alpha = \mathcal{S}$. As $\beta = 1$, $\rho_{g_i g_j} \leq \beta$ is true for any pair of subgraph features. The result follows. ■

**Lemma 3**

If $\alpha = 1$ and $\beta = 0$, either $|H| = 0$ or $|H| = 1$.

**Proof**

There are two cases, first where there are no subgraph features $g_i$ such that $\alpha < \rho_{g_i}$, in which case $|H| = 0$. The other case is where there exists one or more subgraph features $g_i$ for which $\alpha < \rho_{g_i}$ . In this case only one subgraph feature can be in $H$ as $\beta = 0$ and for any two features $\rho_{g_i g_j} \geq \beta$. The result follows. ■

**Lemma 4**

For some $\alpha_i$, $\alpha_j$, $\beta$ if $\alpha_i \geq \alpha_j$ then $|H_i| \leq |H_j|$.

**Proof**

Follows from the fact that $\alpha_j$ is more permissive than $\alpha_i$ while constraining $H$. The $\beta$ being identical, the result follows.■

**Lemma 5**

For some $\alpha$, $\beta_i$,$\beta_j$. If $\beta_i \geq \beta_j$ then $|H_i| \geq |H_j|$.

**Proof**

Follows from the fact that $\beta_i$ is more permissive than $\beta_j$ while constraining $H$. The $\alpha$ being identical, the result follows.■

**Lemma 6**

For any $\alpha$ and $\beta$, $|H^*| \leq |\mathcal{S}_\alpha|$.

**Proof**

Under the given constraints of $\alpha$ and $\beta$ suppose we have a $\mathcal{S}_\alpha$ . In the extreme case, for every pair of subgraph features $\rho_{g_i g_j} \geq \beta$. Then $|H^*| = |\mathcal{S}_\alpha|$. In the other extreme case for every pair of subgraph features $\rho_{g_i g_j} \leq \beta$. Then $|H^*| = 1$. The result follows.■

These results characterize the relation between the size of $H$, $\alpha$ and $\beta$ in broad strokes. Figure 10 illustrates this relationship. Note that the hypothesis space H is small for small values of $\alpha$ and large values of $\beta$. H grows as $\alpha$ increases and $\beta$ decreases. H is at its maximum (equal to S) when $\alpha$ has the maximum value of 1 and $\beta$ has the minimum value of 0. These results are limited in the sense that they characterize the relationship at extreme cases of $\alpha$ and $\beta$ and do not give a fine grained picture. A better characterization of this relationship is important as the size of $H$ directly relates to the computational complexity of the problem.

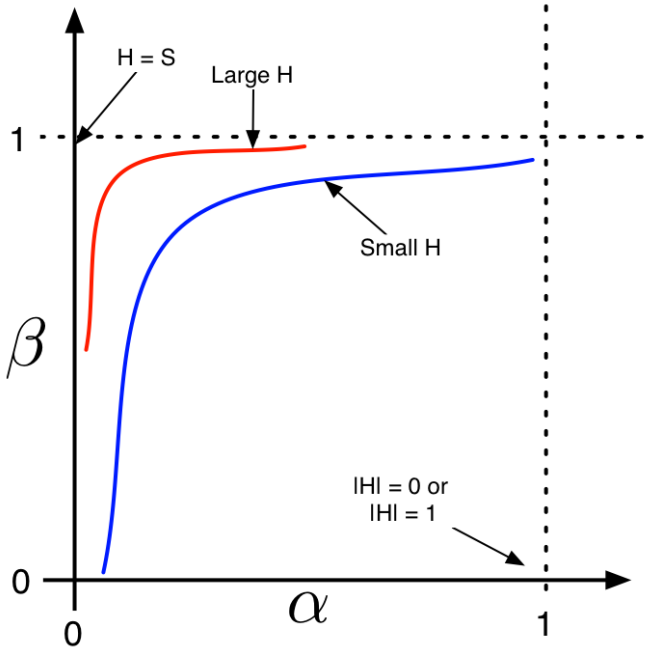


**Figure 10. Effect of Pruning Parameters on the Feature Space**

Given the formulation of the problem earlier, a naïve solution would be an algorithm that searches the complete space of subgraph features (of the graph transactions) checking for each subgraph feature conditions (1) and (2) retaining only those subgraph features that satisfy both of them. Of course, one of the many canonical labeling schemes introduced in frequent subgraph mining systems could be incorporated to prevent the generation of duplicate subgraphs.

The critical problem here is determining pruning conditions corresponding to the frequency antimonotone pruning condition used by all frequent subgraph mining systems. The frequency antimonotone pruning condition is a simple observation that if a subgraph feature has frequency below the user specified threshold, no supergraph of this subgraph can be frequent given this threshold. This simple observation allows for massive pruning of the search space in the case of frequent subgraph mining. Thus the key problem is to answer the following two questions.

- Given a subgraph feature, what is the highest possible correlation any supergraph of this subgraph feature can achieve with the output?
- Given a subgraph feature what is the lowest possible correlation any supergraph of this subgraph feature can achieve with some other subgraph feature?

It must be noted that once we have a quantitative measure for these two questions, it becomes very easy to adapt any frequent subgraph mining system to solve the problem at hand. Quantitative measures for these questions in a sense correspond the frequency antimonotone condition in the case of frequent subgraph mining.

For a graph $g$ we define

$$m_g(x) = \begin{cases} -1 \; if \; I(x) \leq 0 \\ h_g(x) \; if \; I(x) > 0 \end{cases}$$

Intutively, $m_g(x)$ is a vector with every position corresponding to a training example $x$. If the training example has output less than or equal to zero, the value corresponding to this example is -1. If the training example has output greater than zero the value corresponding to this example is $h_g(x)$.

We have the following upper bound on the correlation any supergraph of a subgraph feature can achieve with the output.

**Theorem 1**

For some subgraph features $g_i$ and $g_j$ if $g_j \subseteq g_i$ then

$$\rho_{g_i,I} = \left| \frac{h_{g_i} . I}{\left\| h_{g_i} \right\| . \|I\|} \right| \leq \left| \frac{m_{g_i} . I}{\left\| m_{g_i} \right\| . \|I\|} \right|$$

**Proof**

It is easy to see that for any subgraph feature say $g_i$ if $h_{g_i}(x) = -1$ then for no subgraph feature $g_i \subseteq g_j$, $h_{g_j}(x) = 1$. That is, all those $x$ such that $h_{g_i}(x) = -1$, for any $g_i \subseteq g_j$, $h_{g_j}(x) = -1$ (this is captured by $m_{g_i}$). Furthermore, only for those $x$ where $h_{g_i}(x) = 1$ can $h_{g_j}(x) = -1$ for some $g_i \subseteq g_j$. The highest possible $\rho_{g_j I}$ can occur in the case where for all $x$ such that $I(x) \leq 0$, $h_{g_j}(x) = -1$ (this is captured by $m_{g_i}$). The result follows.■

For a graph $g$ we define

$$n_g(x) = \begin{cases} -1 \; if \; h_g(x) > 0 \\ h_g(x) \; if \; h_g(x) \leq 0 \end{cases}$$

Intutively, $n_g(x)$ is a vector with every position corresponding to a training example $x$. If the training example has $h_g(x)$ greater that zero the value corresponding to this example is -1. If the training example has $h_g(x)$ lesser than zero the value corresponding to this example is $h_g(x)$.

We have the following lower bound on the correlation any supergraph of a subgraph feature can achieve with some other subgraph feature.

**Theorem 2**

For some subgraph features $g_i$, $g_j$ and $g_k$ if $g_j \subseteq g_i$ then,

$$\rho_{g_j,g_k} = \left| \frac{h_{g_j} . h_{g_k}}{\left\| h_{g_j} \right\| . \left\| h_{g_k} \right\|} \right| \geq \left| \frac{n_{g_i} . h_{g_k}}{\left\| n_{g_i} \right\| . \left\| h_{g_k} \right\|} \right|$$

**Proof**

As before it is easy to see that for any subgraph feature say $g_i$, if $h_{g_i}(x) = -1$ then for no subgraph feature $g_i \subseteq g_k$, $h_{g_k}(x) = 1$. That is, all those $x$ such that $h_{g_i}(x) = -1$,

for any $g_i \subseteq g_k$, $h_{g_k}(x) = -1$. Furthermore, only for those $x$ where $h_{g_i}(x) = 1$ can $h_{g_k}(x) = -1$ for some $g_i \subseteq g_k$ (this is captured by $n_{g_i}$). The lowest possible $\rho_{g_j g_k}$ can occur in the case where for all $x$ such that $h_{g_i}(x) > 0$, $h_{g_k}(x) = -1$ (this is captured by $n_{g_i}$). The result follows. ■

Using these bounds it is now possible to adapt any subgraph enumeration scheme to the task of finding a set H. To demonstrate this, we adopt the DFS search and DFS canonical labeling used by gSpan, which we refer to as gSpanPrune. The key steps of our algorithm are summarized as follows, and Figure 11 illustrates how the pruning mechanism works. Note that E refers to examples as discussed earlier.

**Algorithm:** gSpanPrune $(E, \alpha, \beta)$

1. $H \leftarrow \phi$
2. $P \leftarrow$ DFS codes of 1-vertex subgraphs in E
3. **for all** $g_i$ such that $g_i \in P$ **do:**
4. Extend$(E, \alpha, \beta, H, g_i)$
5. **return** H

**Procedure:** Extend$(E, \alpha, \beta, H, g_i)$

1. **if** $g_i$ is not minimum DFS code: **return**
2. **if** $\left| \frac{h_{g_i}.I}{\|h_{g_i}\|.\|I\|} \right| < \alpha$ :
3. **return**
4. **for all** $g_j$ such that $g_j \in H$ :
5. **if** $\left| \frac{h_{g_i}.h_{g_j}}{\|h_{g_i}\|.\|h_{g_j}\|} \right| > \beta$ :
6. **return**
7. **else** :
8. $H \leftarrow H \cup g_i$
9. $P \leftarrow$ DFS codes of rightmost extensions of $g_i$

10. **for every** $g_j \in H$, $g_k \in P$,such that $\left|\frac{n_{g_j} . h_{g_k}}{\|n_{g_j}\| . \|h_{g_k}\|}\right| \leq \beta$ :

11. Extend$(E, \alpha, \beta, H, g_k)$

It should be noted that there can be several ways we can select among a number of subgraph features each of which satisfy the $\alpha$ constraints but no two of them together satisfy the $\beta$ constraint. Our algorithm selects the smallest of the subgraphs (this is because the smaller subgraph will be added to H earlier than the larger subgraph feature and if the $\beta$ constraint is violated the subgraph is never added to H). But there are other methods for choosing between two features that violated the $\beta$ constraint, e.g., choose the feature that has higher correlation with the output, or randomly choose. One of the next steps for this research will be to investigate the effect different selection methods have on performance.

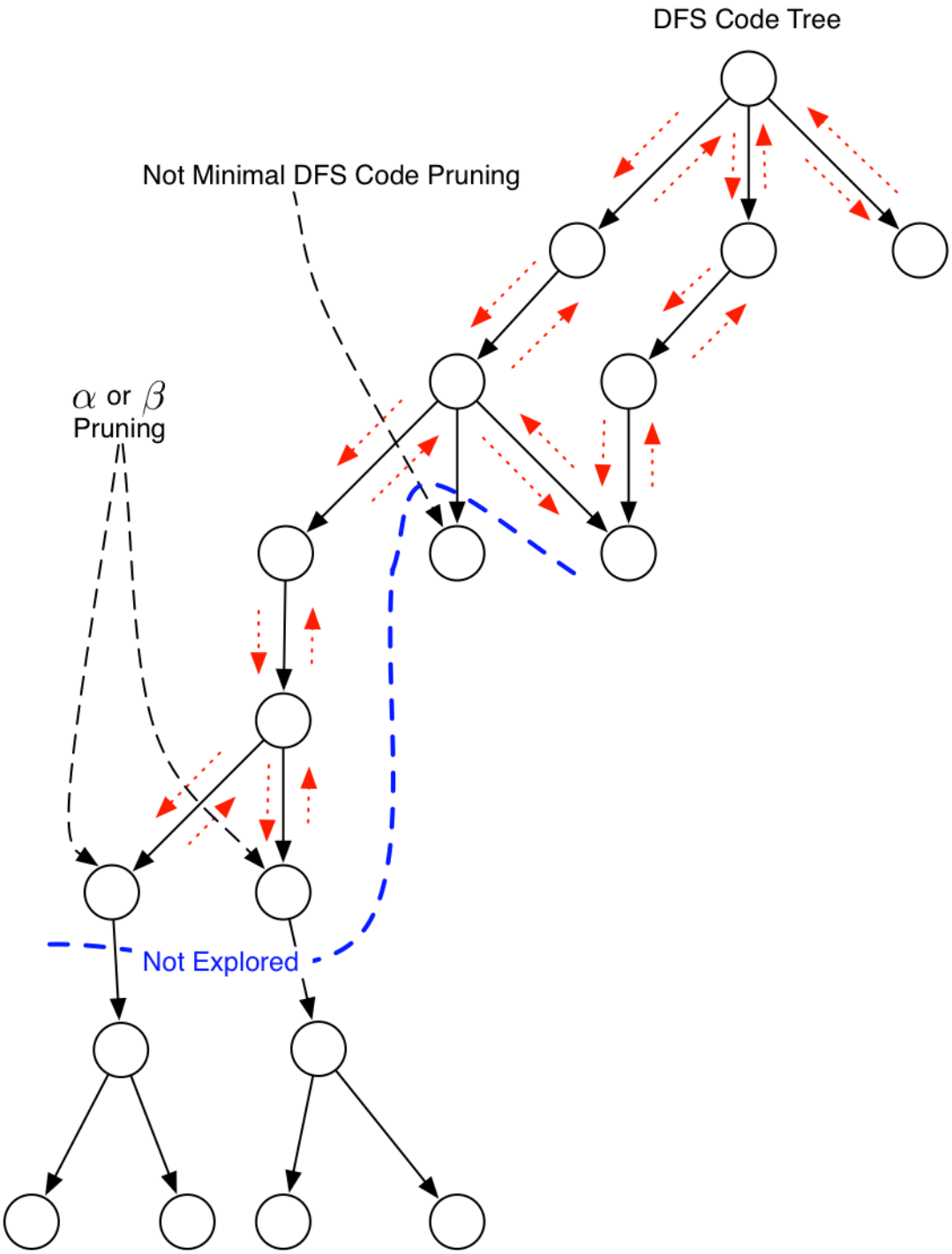


**Figure 11. Pruning Mechanisms in gSpanPrune. Every subgraph considered by gSpanPrune represents a node in the DFS code tree. The entire DFS code tree is not explored, the non-minimal pruning eliminates duplicate subgraphs and the $\alpha, \beta$ pruning eliminates subgraphs which do not obey the $\alpha, \beta$ constraints.**

## 3.2 Experimental Evaluation

Our experimental evaluation of the proposed pruning mechanisms seeks to answer the following questions.

1. How do the subgraph features extracted using gSpanPrune compare with frequent subgraph mining algorithms with respect to predictive accuracy of the regression model developed based on these features?
2. How does gSpanPrune compare with frequent subgraph mining algorithms in terms of runtime when applied to the task of feature extraction for building regression models?
3. How does the runtime of gSpanPrune vary for various choices of $\alpha$ and $\beta$ parameters?

In order to answer these questions we collected a number of data sets. All the data sets are publicly available and are from the domain of computational chemistry. They consist of chemical compounds with a specific property of interest associated with each compound. In every case we use a simple graph representation for the chemical compounds with element symbols as vertex labels and bond types (single bond, double bond, aromatic bond) as edge labels. This is illustrated in Figure 12. The value for which the regression model was to be built was centered to have a mean of zero. No information other than the subgraph features are used to build the regression models for these experiments.

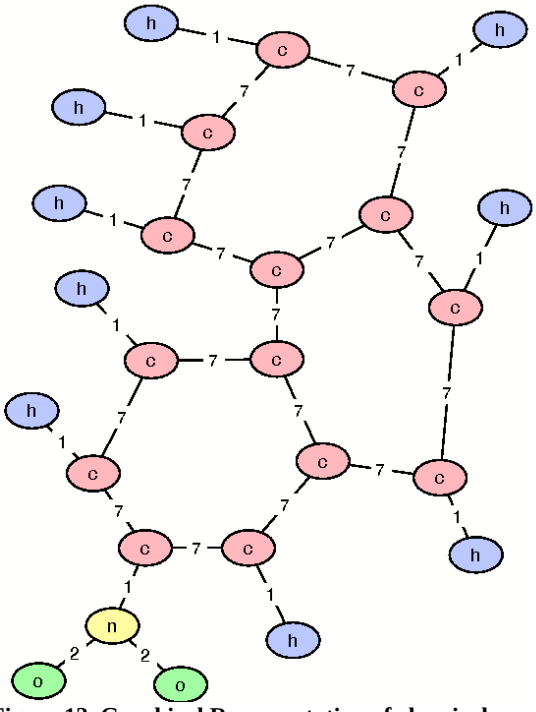


**Figure 12. Graphical Representation of chemical compounds**

The Karthikeyan [52] data set was originally collected to build a regression model for the prediction of melting points of chemical compounds. The Bergstrom [30] data set was originally collected to develop a model for the melting point of solid drugs. The Huuskonen [18] data set was originally collected to develop a model for the aqueous solubility of chemical compounds. The Delaney [39] data set was originally collected to develop a model for the aqueous solubility of compounds with low molecular weights. The ERBD (Estrogen Receptor Binding Dataset) [15] was originally collected for developing predictive models of the estrogen receptor binding activity. The ARBD (Androgen Receptor Binding Data) [24] was originally collected for developing predictive models of the androgen receptor binding activity. Summary statistics for all these datasets are presented in Table 2.

**Table 2. Properties of the data sets**

| Data Set | ARBD | Bergstrom | Delaney | ERBD | Huuskonen | Karthikeyan |
|---|---|---|---|---|---|---|
| Transactions | 146 | 277 | 1144 | 131 | 1380 | 4450 |
| Edge Labels | 3 | 3 | 3 | 3 | 3 | 3 |
| Vertex Labels | 9 | 9 | 10 | 8 | 10 | 13 |
| Avg. Edges in Transaction | 21 | 22 | 14 | 21 | 13 | 24 |
| Avg. Vertices in Transaction | 20 | 21 | 13 | 19 | 13 | 22 |
| Max. Edges in Transaction | 38 | 53 | 62 | 47 | 50 | 61 |
| Max. Vertices in Transaction | 34 | 51 | 55 | 43 | 47 | 59 |

Among the various frequent subgraph mining systems to compare gSpanPrune with, we chose gSpan [28]. While it is unclear whether gSpan is the best frequent subgraph mining, it can definitely be considered to be among the state of the art as far as the frequent subgraph mining problem is concerned. In order to ensure that our results generalize to frequent subgraph mining algorithms in general, we compare the number of subgraphs considered by gSpanPrune and gSpan. This is simply a count of all minimal DFS codes considered by each of the approaches. The difference between the number of minimal DFS codes considered by gSpanPrune and gSpan gives us a measure of how the pruning approach compares with any other frequent subgraph mining system. This is because different frequent subgraph mining systems may use other forms of canonical labeling, and search mechanisms will prevent the generation of duplicate subgraph features better than gSpanPrune and gSpan, but every subgraph feature (the minimal code in the case of gSpan and gSpanPrune) will have to be considered at least once. If gSpanPrune considers significantly fewer subgraphs, the speedup in terms of runtime would most likely apply to other frequent subgraph mining systems.

Among the various approaches to regression we chose SVR (Support Vector Regression) [45], which can be considered among the state of the art as far as the regression problem is concerned. In particular, we use the SVMLite package [11]. While it is possible that in certain situations other regression algorithms might outperform SVR, we find it unlikely to get opposite

results while comparing the quality of the regression models based on the subgraph features produced by gSpan and gSpanPrune with any regression algorithm.

We use the $Q^2$ score (defined earlier on page 22) to evaluate the predictive accuracy of the regression models. Note that the $Q^2$ score is a real number between 0 and 1 and its interpretation is similar to the Pearson correlation coefficient. The closer it is to 1, the better the regression function fits the testing data.

In order to answer question (1) and (2) we conducted experiments on gSpan and gSpanPrune on the six data sets described above. The subgraph features produced by each algorithm were used to build a regression model using SVR. The predictive accuracy of the models was evaluated based on the $Q^2$ score using a 5-fold cross validation. Additionally the runtimes and the number of subgraphs considered by each algorithm were also recorded. The maximum subgraph size for each system was set to 10 (our initial experimentation indicated that features greater than a size of 10 do not contribute much to the predictive model. Such a constraint was also placed on the experimentation conducted by other researchers on gPLS [63]). The parameters of each system (threshold frequency in the case of gSpan and the $\alpha$ and $\beta$ parameters in the case of gSpanPrune) were systematically varied. While comparing results on the various runs of the algorithms, we select the highest $Q^2$ scores achieved by each system and then compare the lowest possible runtimes and the subgraphs considered for this $Q^2$ score. The intuition behind this is to compare the lowest computational cost for the best possible predictive accuracy. The results of these experiments are reported in Figures 13, 14 and 15.

In order to answer question (3) we ran gSpanPrune on the Karthikeyan and ARBD data set (we chose this data set as this was the largest data set in terms of transactions) with $\alpha$ and $\beta$ parameters systematically varied in small increments of 0.05. Figures 16, 17, 18, 19, 20 and 21 illustrate these results with contour plots.

We can observe the following from the experimental results. The predictive accuracy of the regression models based on the features generated by gSpan and gSpanPrune is comparable. GSpanPrune substantially outperforms gSpan in terms of runtime and the number of subgraphs explored. The runtime and the number of subgraphs explored by gSpanPrune increases for small values of $\alpha$ and large values of $\beta$.

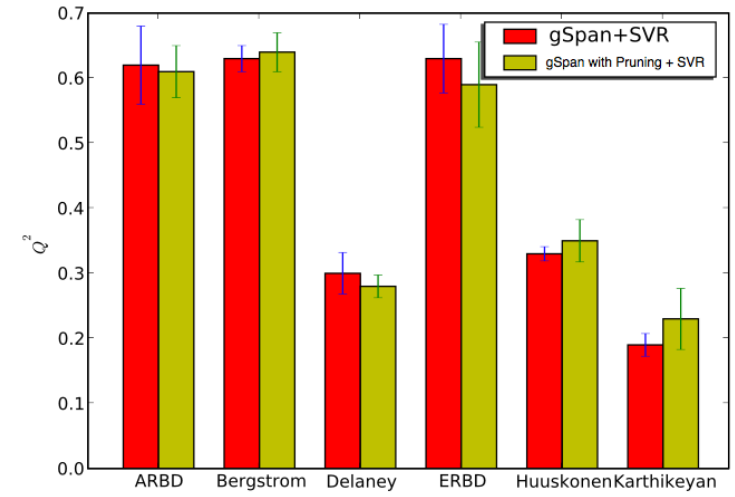


**Figure 13. Predictive Performance of gSpan and gSpanPrune**

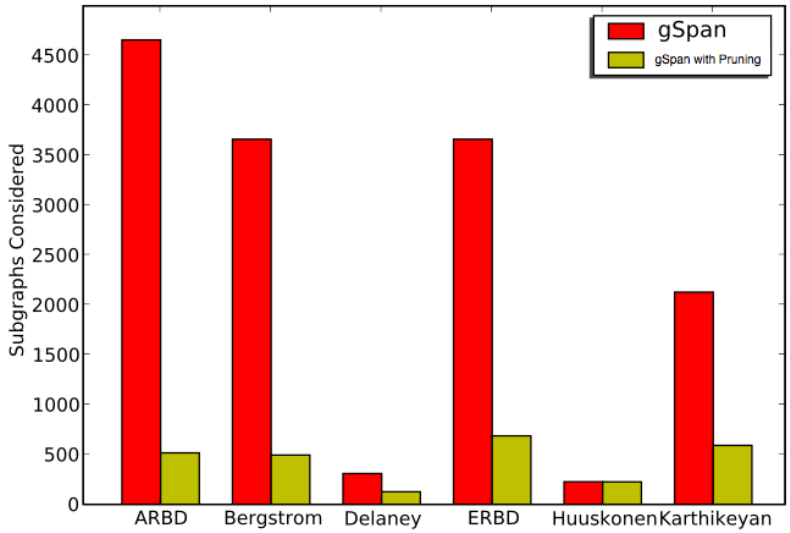


**Figure 14. Subgraphs considered by gSpan and gSpanPrune**

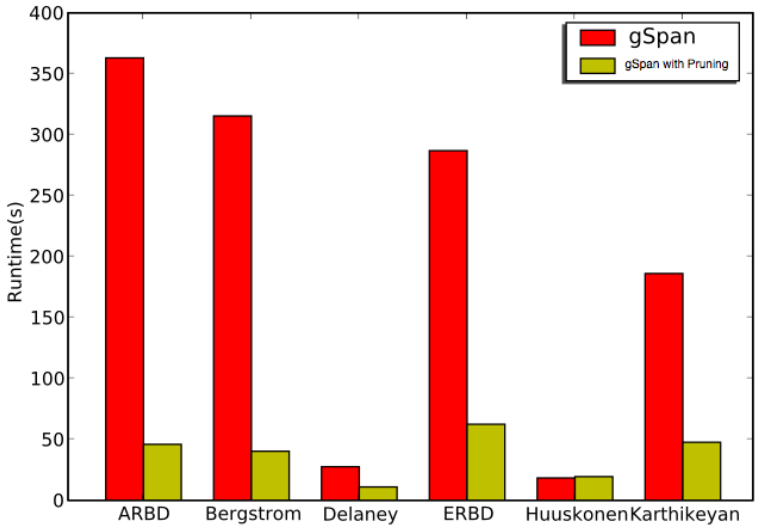


**Figure 15. Runtimes of gSpan and gSpanPrune**

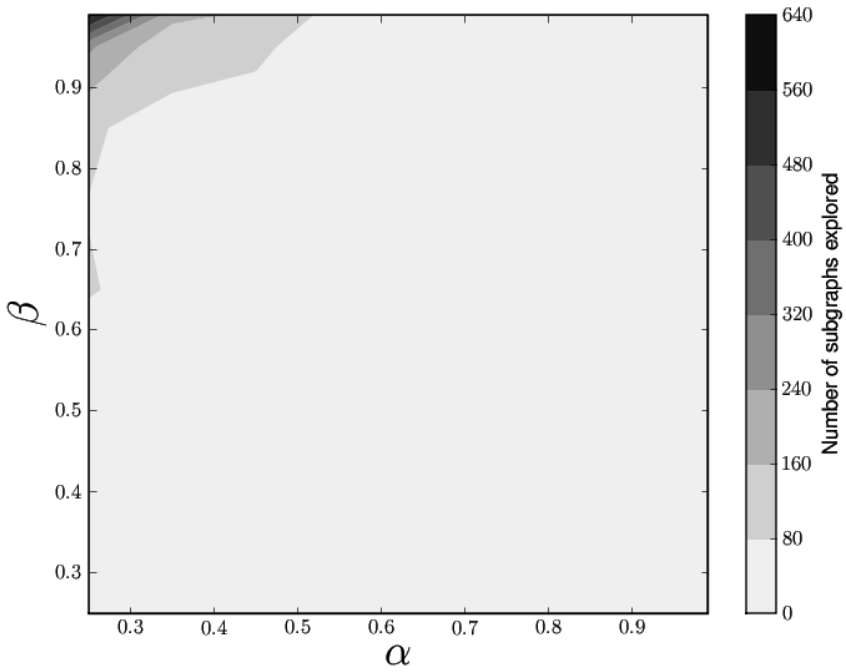


**Figure 16. Effect of Pruning Parameters on the Feature Space, Number of Subgraphs Considered (Karthikeyan dataset)**

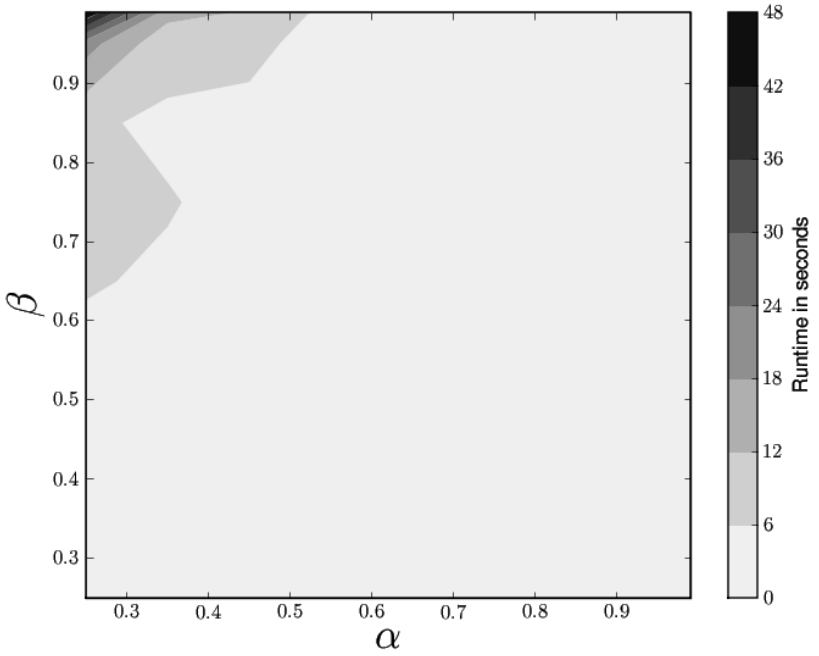


**Figure 17. Effect of Pruning Parameters on the Feature Space, Runtime (Karthikeyan dataset)**

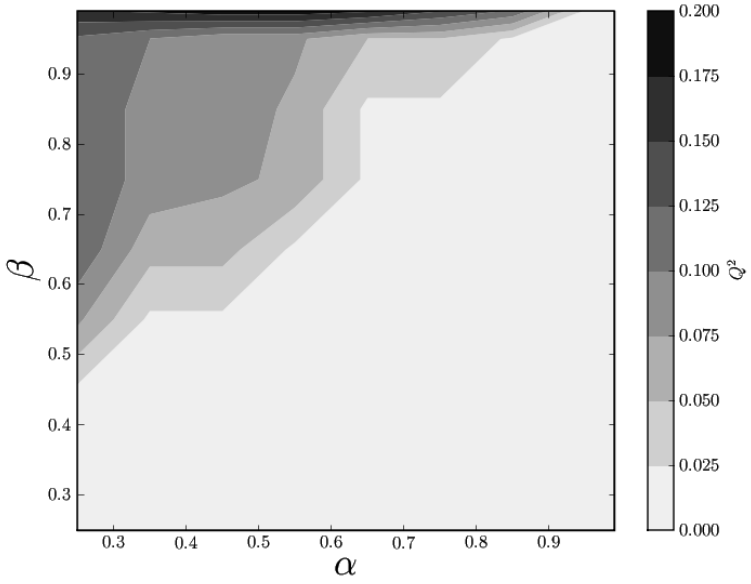


**Figure 18. Effect of Pruning Parameters on the Predictive Accuracy (Karthikeyan dataset)**

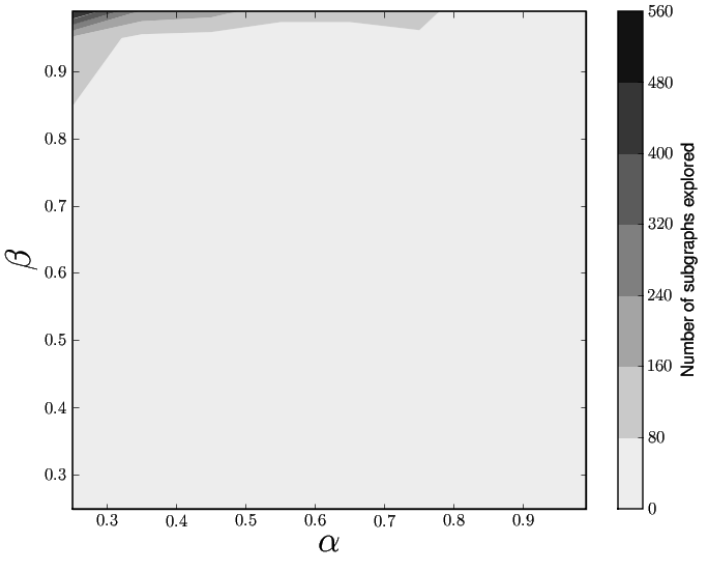


**Figure 19. Effect of Pruning Parameters on the Feature Space, Number of Subgraphs Considered (ARBD dataset)**

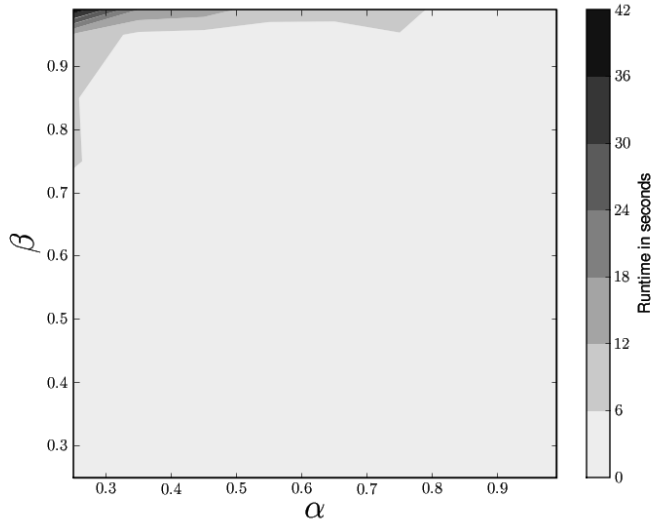


**Figure 20. Effect of Pruning Parameters on the Feature Space, Runtime (ARBD dataset)**

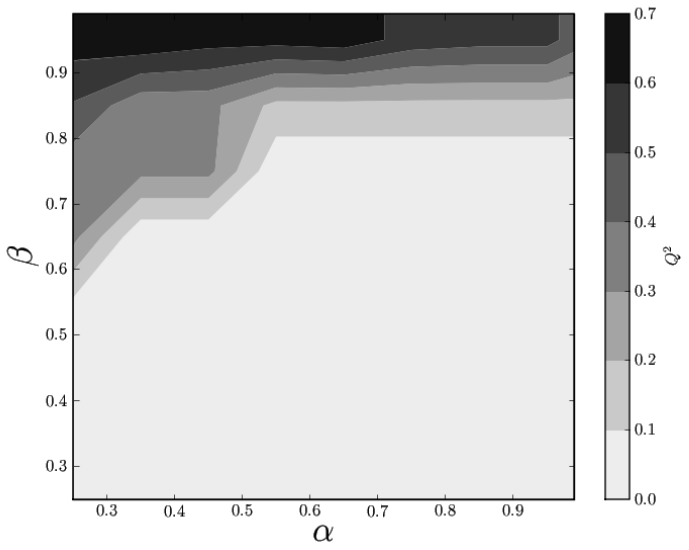


**Figure 21. Effect of Pruning Parameters on the Predictive Accuracy (ARBD dataset)**

## 3.4 Limitations, Opportunities for Further Improvements and Alternative Evaluation Measures

There are a number of limitations, opportunities for further improving the search for subgraph features, and alternative evaluation mechanisms for the search of subgraph features. Here, we present a brief discussion of these.

1. The testing for the β constraint (line 12 in the Expand procedure) which involves computing and checking pairwise correlations for the subgraph features under consideration against all subgraph features included in H, can be a significant overhead. This is quite apparent in the Husskonnen dataset. Generally there is a vast difference between the number of subgraphs considered by gSpan and gSpanPrune. However in certain cases where there is a minor difference, as in the case of Husskonnen dataset, gSpanPrune consumes more time than gSpan. This discrepancy can be attributed to the overhead of computing the β constraint. In certain cases, as in the Husskonnen dataset,

this can lead to a worse performance as compared to running gSpan without the pruning mechanisms.

2. There is an inherent relationship between the pairwise correlations of subgraph features. For example given three different subgraph features $g_i, g_j$ and $g_k$ we have three different pairwise correlations $\rho_{g_i,g_j}$ , $\rho_{g_j,g_k}$ and $\rho_{g_i,g_k}$ . If we can prove that these pairwise correlations are transitive, it will not be necessary to check the β constraint for the new subgraph feature under consideration against every subgraph feature already included H (step 12 in the expand procedure). As the size of H grows, checking the β constraint can become quite expensive and the transitivity between pairwise correlations (if true) can be exploited to address this issue.
3. Certain application domains may not require unconstrained graphs to fully capture the relevant data. Constrained cases such as trees or planar graphs might suffice. This raises the important question on the relevance of our pruning mechanisms in such cases. It must be pointed out that when the subgraph isomorphism test is not expensive (as in the case of trees or planar graphs) the pruning mechanisms may not give a speedup. It is also possible that the extra overhead of computing the $\beta$ constraint might make the pruning mechanisms expensive.
4. The experimentation focused on maximizing the predictive accuracy achieved by features generated by each system and then comparing the computational resources consumed by each system (runtime). The intuition behind this experimentation is to compare the computational cost of the best possible predictive model produced by each system. An alternative evaluation mechanism would be to compare the number of subgraph features produced by each system per unit computational resource. This would be similar to the

notion of measuring floating point operations per second as in the case of high performance computing. Such an evaluation mechanism based on subgraph features produced per unit computational time does give an objective measure of the speedup caused by the pruning mechanisms, but raises the question as to how good the produced subgraph features are with respect to predictive accuracy of the model. Although this is a weakness, this could be a valid evaluation mechanism. An additional, alternative evaluation mechanism would be to compare the improvement in predictive accuracy per unit time for each of the systems. Such a metric would capture the tradeoff between computational expenditure and predictive accuracy and might be well suited for the task.

# 4. gRegress: An algorithm for Graph Regression

In this chapter, we present a new algorithm for graph regression, which we refer to as gRegress. The intuition behind gRegress is the need for a combination of linear models in certain domains. gRegress addresses this need with a tree based combination of linear models. Our experiments indicate that gRegress outperforms the current state of the art gPLS in a number of application domains. We begin with the motivation behind our approach.

## 4.1 Motivation

As mentioned earlier graph regression is a relatively new problem and the only published work in this area is gPLS [63], the basic idea behind which is to adapt partial least square regression for the task. Our intuition is that there might be certain graph regression problems that cannot be modeled by a single linear function and would require a combination of linear regression models. An example of this would be the following hypothetical problem from chemistry. Suppose we are trying to learn a regression model for the boiling point of chemical compounds. It is a well known fact that the boiling point is largely dependent on the functional group [73]. Learning a separate, linear regression model for different classes of compounds (one for alcohols, one for ethers, one for esters etc.) could perform better than a linear regression model for all the compounds. Learning such a combination of linear models involves two sub-problems. The first is to categorize the examples into classes and then to learn a regression model for each of them.

Our approach, which builds on this intuition, is to learn a model which consists of a decision tree such that each non-leaf is a subgraph presence/absence test which splits the examples into categories and each leaf is a linear regression model based on weights on the presence/absence of subgraph features. Such a model is illustrated in Figure 22.

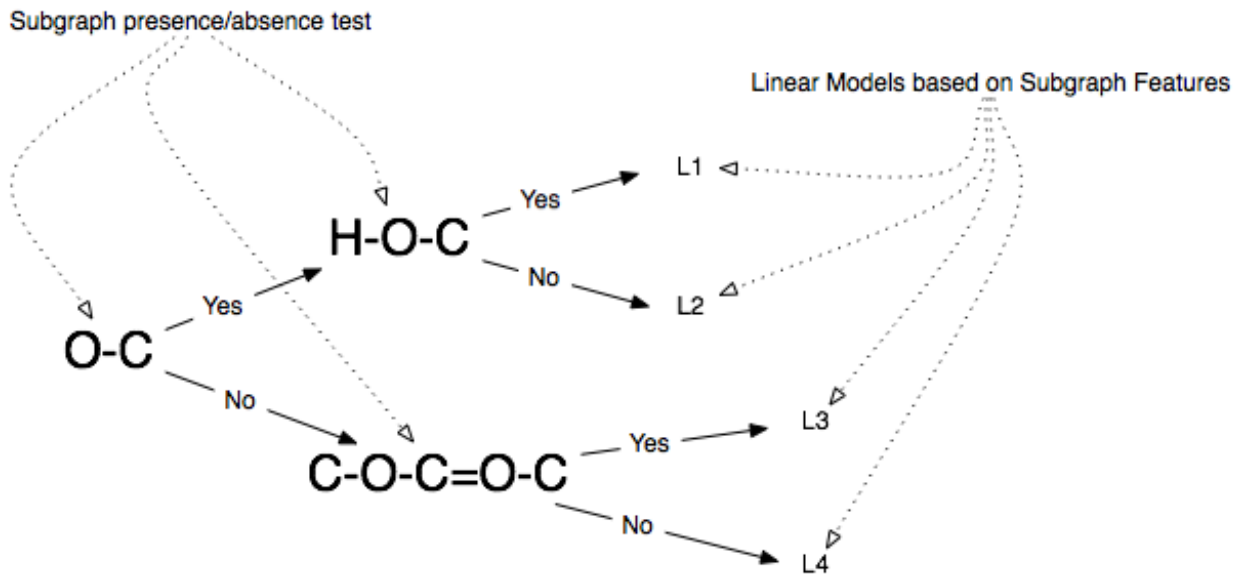


**Figure 22. A Decision Tree with Linear Models as Leaves**

### 4.1 Algorithm

The key steps of the gRegress algorithm are summarized as follows.

1. Build a decision tree recursively, splitting on presence/absence of subgraph features which minimize the following error measure,

$$\Delta e = sd(E) - \sum_i \frac{|E_i|}{|E|} \times sd(E_i)$$

until the number of examples at a leaf node is less than or equal to a user-defined threshold L.

2. Build a multiple regression model for examples remaining at each leaf.

Note that $sd$ refers to standard deviation. Intuitively, the error measure evaluates the improvement in predictive accuracy due to the use of separate regression models based on a split on the current feature.It is important to distinguish gRegress from the previously discussed DT-CLGBI [26]. While both of these approaches are based on decision trees, there are some

important differences. First, DT-CLGBI produces a pure decision tree where each node represents a subgraph presence/absence test while gRegress is a combination of linear models combined with a decision tree. Leaf nodes in DT-CLGBI are pure nodes (belonging to a single class) while leaf nodes in gRegress are linear models. Second, the splitting tests in DT-CLGBI are based on an entropy measure with respect to binary classification while those in gRegress are based on an error measure based on standard deviation with respect to regression. In the case where gRegress is run with L = 1 (number of examples at leaf) no linear model at the leaf is necessary (or possible) and the induced tree is quite similar although there might be a difference due to the difference in the splitting condition. The key difference between DT-CLGBI and gRegress is that DT-CLGBI produces a decision tree of subgraph presence/absence tests while gRegress produces a combination of linear models organized as a tree.

### 4.2 Experimental Evaluation

In order to evaluate the performance of the gRegress algorithm we compared it to the following approaches.

1. gPLS
2. Frequent Subgraph Mining and Support Vector Regression with the Linear Kernel
3. Frequent Subgraph Mining and Support Vector Regression with the Polynomial Kernel
4. Frequent Subgraph Mining and Support Vector Regression with the Radial Basis Function (RBF) Kernel
5. Frequent Subgraph Mining and Simple Linear Regression
6. Frequent Subgraph Mining and Additive Regression
7. Frequent Subgraph Mining and K-Nearest Neighbor

It must noted here that gPLS is the current state of the art as far as the graph regression problem is concerned. While the other approaches have not been studied in literature, except for gRegress, they are straightforward combinations of existing work. Also note that additive regression is a simple combination of regression models. After the initial regression model based on the output, additional regression models are developed based on the residuals (errors). The final model is a simple addition of the values predicted by the models.

We conducted experiments with the Karthikeyan, Bergstrom, Huuskonen, Delaney, ERBD and ARBD datasets discussed earlier. Mean correlations on the 5-fold cross validation are reported in Table 3. Note that correlation is different from $Q^2$ and is a more common evaluation measure for regression (we used $Q^2$ earlier to compare our results to gPLS [63]) and is defined as follows.

$$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\ \sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

**Table 3. Comparison of approaches to Graph Regression. Values are mean correlations on a 5-fold cross validation.**

| | gRegress | gPLS | SVM Linear | SVM Poly | SVM RBF | Simple Linear | Additive | KNN |
|---|---|---|---|---|---|---|---|---|
| ERBD | 0.7046 | 0.6979 | 0.598 | 0.7007 | 0.7146 | 0.4458 | 0.6743 | 0.7036 |
| ARBD | 0.7789 | 0.7146 | 0.6193 | 0.728 | 0.7668 | 0.6756 | 0.7087 | 0.721 |
| Bergstrom | 0.285 | 0.3952 | 0.3109 | 0.2523 | 0.3426 | 0.1024 | 0.3295 | 0.3185 |
| Delaney | 0.7676 | 0.6936 | 0.6832 | 0.6832 | 0.7484 | 0.4982 | 0.6637 | 0.7481 |
| Huuskonen | 0.7644 | 0.7019 | 0.69 | 0.6923 | 0.7609 | 0.4953 | 0.6806 | 0.7409 |
| Karthikeyan | 0.5234 | 0.4705 | 0.4998 | 0.49 | 0.5747 | 0.2929 | 0.4199 | 0.5113 |

We can make the following observations from the results.

1. gRegress and SVM-RBF are the best performing approaches; they outperform gPLS, which is the current state of the art, in all cases except for the Bergstrom dataset.
2. The performance of gRegress and SVM-RBF is found to be comparable in most cases.
3. gPLS, SVM-Linear and Simple Linear Regression perform poorly in a number of cases supporting our intuition on the need for a combination of linear models.
4. The KNN approach performs surprisingly well given its simplicity.
5. In the case of the Bergstrom dataset, gPLS performs better than most approaches. This is an unexpected result and requires further analysis and experimentation.

Another important question in regards to applying these approaches to real world problems is if incorporating structure can improve the predictive accuracy of existing attribute-valued models. To investigate this issue we use the 2C9 dataset introduced in [62]. This dataset consists of a number of attribute values associated with chemical compounds (other datasets which we experimented with earlier did not have such features or they were not available as a part of the public dataset) Each of these can be treated as a separate graph regression problem. Each of the approaches can be modified to handle attribute-valued features in addition to subgraph features. In the case of gPLS and gRegress, attribute threshold tests can replace subgraph presence/absence tests in the case of attribute-valued features. In the case of SVM, simple linear regression, additive regression and KNN, attribute valued features can simply be added to the set of selected subgraph features. We conducted experiments with only the attributes, only the subgraph features and a combination of both. Tables 4, 5 and 6 present the results of these experiments.

**Table 4. Performance on 2C9 dataset with Attribute Features Only. Values are mean correlations on a 5-fold cross validation.**

| | gRegress | gPLS | SVM Linear | SVM Poly | SVM RBF | Simple Linear | Additive | KNN |
|---|---|---|---|---|---|---|---|---|
| 2c9 | 0.7664 | 0.75 | 0.7652 | 0.7641 | 0.7387 | 0.7373 | 0.7194 | 0.7363 |
| weinerPol | 0.9755 | 0.9818 | 0.9757 | 0.9764 | 0.9518 | 0.9722 | 0.9278 | 0.9095 |
| zagreb | 0.9872 | 0.993 | 0.9891 | 0.9898 | 0.9769 | 0.9719 | 0.9369 | 0.9489 |
| PEOP_VSA_NEG | 0.8415 | 0.8391 | 0.8621 | 0.8654 | 0.8019 | 0.6927 | 0.7792 | 0.8234 |
| apol | 0.9993 | 0.994 | 0.9992 | 0.9993 | 0.9938 | 0.9993 | 0.9596 | 0.9843 |
| bpol | 0.9632 | 0.9612 | 0.9626 | 0.9658 | 0.9471 | 0.9632 | 0.8917 | 0.9302 |
| vsa_hyd | 0.9601 | 0.9538 | 0.9573 | 0.9574 | 0.9343 | 0.9136 | 0.8984 | 0.9267 |
| SlogP | 0.8617 | 0.8756 | 0.8675 | 0.871 | 0.8077 | 0.6912 | 0.7362 | 0.7977 |
| vdw_area | 0.9848 | 0.9826 | 0.9901 | 0.9925 | 0.9839 | 0.9848 | 0.951 | 0.9704 |
| vdw_vol | 0.9994 | 0.9933 | 0.9996 | 0.9996 | 0.9974 | 0.9996 | 0.9645 | 0.9889 |

**Table 5. Performance on 2C9 dataset with Subgraph Features Only. Values are mean correlations on a 5-fold cross validation.**

| | gRegress | gPLS | SVM Linear | SVM Poly | SVM RBF | Simple Linear | Additive | KNN |
|---|---|---|---|---|---|---|---|---|
| 2c9 | 0.5132 | 0.5677 | 0.315 | 0.61 | 0.6357 | 0.1993 | 0.5064 | 0.5857 |
| weinerPol | 0.5884 | 0.6851 | 0.4607 | 0.7408 | 0.7285 | 0.334 | 0.591 | 0.5759 |
| zagreb | 0.6375 | 0.7507 | 0.524 | 0.5241 | 0.776 | 0.4007 | 0.6272 | 0.6361 |
| PEOP_VSA_NEG | 0.6825 | 0.7292 | 0.4606 | 0.4597 | 0.7124 | 0.4402 | 0.7046 | 0.6336 |
| apol | 0.7165 | 0.7882 | 0.517 | 0.517 | 0.8123 | 0.3842 | 0.6961 | 0.6774 |
| bpol | 0.8208 | 0.8161 | 0.5927 | 0.8657 | 0.8353 | 0.5742 | 0.7929 | 0.6918 |
| vsa_hyd | 0.7801 | 0.8052 | 0.5678 | 0.8364 | 0.8232 | 0.4567 | 0.7254 | 0.724 |
| SlogP | 0.7168 | 0.7767 | 0.5787 | 0.7375 | 0.7105 | 0.2331 | 0.7239 | 0.6652 |
| vdw_area | 0.7342 | 0.7688 | 0.5064 | 0.8188 | 0.7884 | 0.4544 | 0.63 | 0.651 |
| vdw_vol | 0.7009 | 0.7793 | 0.5085 | 0.8236 | 0.8028 | 0.3743 | 0.6403 | 0.6682 |

**Table 6. Performance on 2C9 dataset with Attribute and Subgraph Features. Values are mean correlations on a 5-fold cross validation.**

| | gRegress | gPLS | SVM Linear | SVM Poly | SVM RBF | Simple Linear | Additive | KNN |
|---|---|---|---|---|---|---|---|---|
| 2c9 | 0.7459 | 0.733 | 0.4717 | 0.6374 | 0.6418 | 0.7373 | 0.7398 | 0.5853 |
| weinerPol | 0.977 | 0.9838 | 0.9062 | 0.8704 | 0.7439 | 0.9722 | 0.9181 | 0.5843 |
| zagreb | 0.9908 | 0.9948 | 0.9527 | 0.9259 | 0.7913 | 0.9719 | 0.935 | 0.6454 |
| PEOP_VSA_NEG | 0.9244 | 0.9524 | 0.8598 | 0.8562 | 0.7262 | 0.6927 | 0.817 | 0.6422 |
| apol | 0.9992 | 0.9995 | 0.9655 | 0.9344 | 0.8234 | 0.9958 | 0.9594 | 0.6863 |
| bpol | 0.9765 | 0.9903 | 0.9655 | 0.9148 | 0.8413 | 0.9086 | 0.9019 | 0.6976 |
| vsa_hyd | 0.9779 | 0.9875 | 0.8984 | 0.9186 | 0.8314 | 0.9136 | 0.9006 | 0.731 |
| SlogP | 0.8798 | 0.9317 | 0.9575 | 0.8157 | 0.72 | 0.6912 | 0.7852 | 0.6749 |
| vdw_area | 0.99 | 0.9968 | 0.9689 | 0.9512 | 0.7997 | 0.979 | 0.9498 | 0.6614 |
| vdw_vol | 0.9994 | 0.9997 | 0.9791 | 0.9442 | 0.8148 | 0.9958 | 0.9645 | 0.678 |

We observe a significant improvement in performance in the model developed for predicting the PEOP_VSA_NEG property when both the attribute and subgraph features are used. In most other cases the performance is found to be comparable. This is a clear indicator that in certain cases the incorporation of structural features can lead to significant improvements in the predictive accuracy of the regression models.

In addition to improving the predictive accuracy of the regression model, the incorporation of structural features can further the understanding of the underlying problem. For example, correlations between attribute-valued features and structural features can lead to a better understanding of the attribute-valued feature, or the correlations between structural features can raise important questions (For example, why do two structural features correlate with each other? Or, does this correlation imply that the presence of one feature causes another?) about the properties of the domain. In general, structural information represents a relevant body of features, and incorporating them into model building can lead to better models as well as a better understanding of the domain.

# 5. Empirical Comparison of Graph Classification Algorithms

We performed an empirical comparison of the major approaches for graph classification introduced in the literature, namely, SubdueCL, frequent subgraph mining in conjunction with SVMs, walk-based graph kernel, frequent subgraph mining in conjunction with AdaBoost and DT-CLGBI. The objective of this comparison was to identify the strengths and weaknesses of these approaches. Experiments were performed on five real world data sets from the Mutagenesis and Predictive Toxicology domains, and a corpus of artificial data sets. This chapter presents the experiments and the results.

## 5.1 Experiments with Real World Datasets

For experiments with real world data, we selected the Mutagenesis data set introduced by [21] and the Predictive Toxicology Challenge (PTC) data introduced by [6]. The Mutagenesis data set has been used as a benchmark data set in graph classification for many years. The data set has been collected to identify mutagenic activity in a compound based on its molecular structure. The Predictive Toxicology Challenge (PTC) data set has also been used in the literature for several years. The PTC carcinogenesis databases contain information about chemical compounds and the results of laboratory tests made on rodents in order to determine if the chemical induces cancer. The data consists of four categories: male rats MR, female rats FR, male mice MM, or female mice FM. Each of the data sets were represented as graphs by introducing a vertex for every atom, labeled with its compound and by introducing an edge for every bond, labeled with its type. An example of such a representation for a compound in the Mutagenesis dataset is shown in Figure 23.

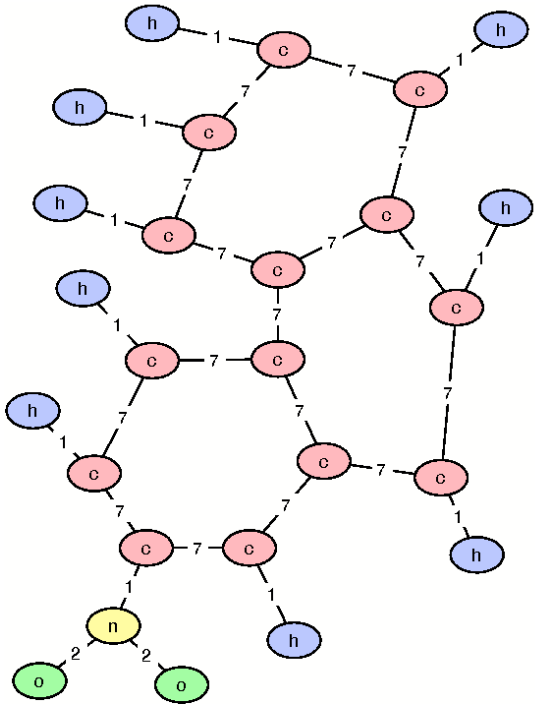


**Figure 23. Graph Representation of Chemical Compounds**

As mentioned earlier, the empirical comparison included SubdueCL [17], frequent subgraph mining in conjunction with SVMs, walk-based graph kernel [32], frequent subgraph mining in conjunction with AdaBoost [47] and DT-CLGBI [26]. For SubdueCL, we use the implementation from the Subdue described in [50]. For frequent subgraph mining in conjunction with SVMs we use the Gaston system [43] and SVMLite [11]. The walk-based (direct product) graph kernel was implemented using SVMLite. The frequent subgraph mining in conjunction with AdaBoost approach was implemented using code from Gaston and the Weka machine learning framework [13]. The DT-CLGBI was also implemented using code form Gaston and the Weka machine learning framework.

We compared the performance of the five algorithms/approaches on the five datasets using a five-fold cross validation. We measured accuracy for all the five algorithms/approaches and the area under the ROC curve (AUC) discussed in [9] for all the algorithms/approaches. Computing the AUC for SubdueCL involves some difficulties which are described next.

In order to plot ROC curves and compute the AUC it is essential to produce a real-valued prediction between zero and one representing the degree of membership (where zero implies the negative class and one the positive class). Binary predictions made by classifiers can be trivially converted to a real-valued predictions, by considering the learning algorithm and the way in which the model is applied. For example, for decision trees, the class distribution at the leaf can be translated to such a real-valued prediction. For SubdueCL, producing such a value is not straightforward. As mentioned before, the model learned by Subdue consists of a decision list each member of which is a connected graph. If any of the graphs in the decision list are present in the example, it is predicted as positive. If all the graphs in the decision list are absent from the example, it is predicted as negative. We considered applying approaches studied by [20] for scenarios like this, all of which are based on combining and reporting the confidence of the rules (in the decision list) on the training set. Strategies for combining the confidence include voting, weighted voting, using the first matching rule, applying a random rule and applying the rule with lowest false positive rate. None of these is in accordance with the particular way in which the model learned by SubdueCL is supposed to be applied. The key problem here is that absence of a single graph (in the decision list) in an unseen example, does not imply that the example is predicted to be negative. All the graphs in the decision list have to be absent from the example for it to be predicted as negative. We therefore approximate the AUC for SubdueCL using a

single point in the ROC space. This does not accurately reflect the performance of SubdueCL, but is a reasonable approximation of the same.

The results of the experiments are reported in Figures 24 and 25. None of the approaches/algorithms had significantly better performance than the others on the real world datasets we considered. Revisiting our original question on the comparative strengths and weaknesses of the algorithms/approaches, we decided to further compare their actual predictions and the agreements/disagreements among predictions on specific examples. The intuition behind this comparison is that comparable performance does not imply similar behavior. Two algorithms could differ in predictions, make different mistakes and be correct on different examples and end up in having comparable performance, overall.

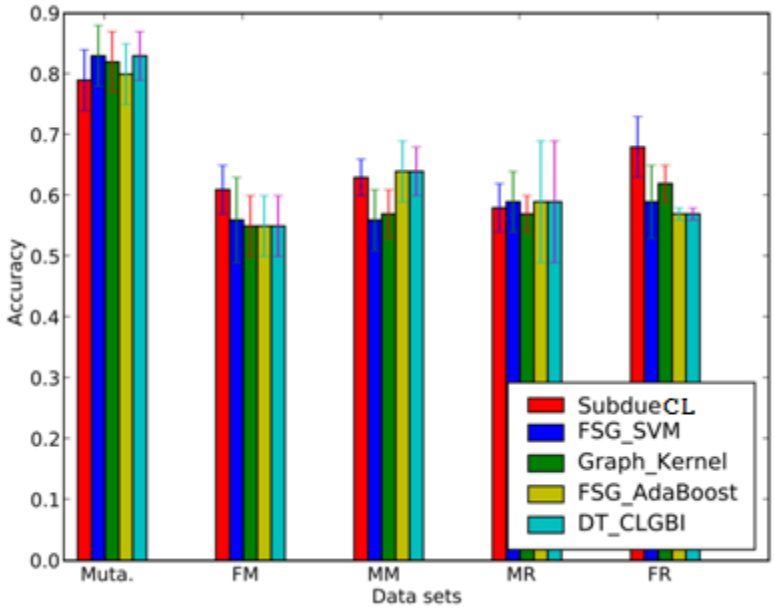


**Figure 24. Accuracy on Real World Data Sets**

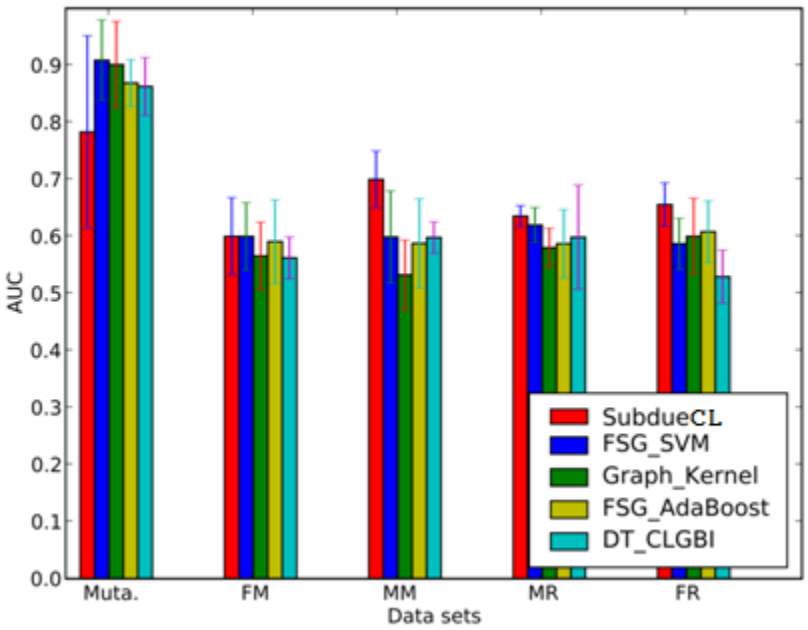


**Figure 25. AUC on Real Word Data Sets**

We conducted a literature survey on how such a comparison can be performed and to our knowledge no framework for such analysis was found in literature. We therefore introduced the following visual mechanism to perform such an analysis. Note here that the purpose of the mechanism is exploratory and not confirmatory.

The key idea is to generate a scatterplot between the predictions made by two classifiers on a set of test examples. Note here that all the algorithms/approaches except for SubdueCL can be made to predict a real value between zero and one representing the degree of membership (where zero implies the negative class and one the positive class). Given a set of test examples, we first separate them according to their class and for two classifiers, say, A and B, get a scatterplot of their predictions on the positive set and the negative set. When such a scatterplot is generated with both the X-axis and the Y-axis having a range from zero to one, we get two plots as shown in Figures 26 and 27.

Every point on such scatterplots would fall in one of the following four zones assuming the threshold to be at 0.5: both classifiers make a similar correct prediction, both classifiers make similar incorrect predictions, A is correct and B is incorrect and vice versa. Note here that the positive and negative examples need to be separated as the four zones are different for each case. Furthermore, by looking at how far a point is from each axis, we can get an indication of how much the predictions are similar or dissimilar.

Figures 28 up to 42 illustrate ROC curves and matrices of such scatterplots for each of the five datasets (note that SubdueCL was omitted from this analysis). The matrix of scatterplots has

each row and each column devoted to one graph classification algorithm. For example, row 1, column 1 correspond to FSG+SVM and row 2, column 2 correspond to the Graph Kernel. The agreement between the predictions made by FSG+SVM and Graph Kernel can be found in row 1, column 2 and row 2, column 1. Note that each matrix of scatteplot corresponds to one dataset and either the positive examples or the negative examples. The ROC curves accompany the plots so that the true positive and the false positive rates at different thresholds can be compared. Note that in our analysis involving the comparison of the predictions (the scatterplots) we chose the threshold of 0.5.

On all the datasets, a higher agreement is observed between the predictions made by the graph kernel and the FSG+SVM approach. There is a much larger disagreement observed among all other algorithms/approaches. We stress here that this observation is exploratory in nature, we cannot conclude disagreement at a statistically significant level as this is a null hypothesis as in correlation analysis and can only be rejected. However, for the purposes of our study this was sufficient indication that a further comparison on artificial datasets wherein we could evaluate the algorithms/approaches on specific parameters of interest was necessary.

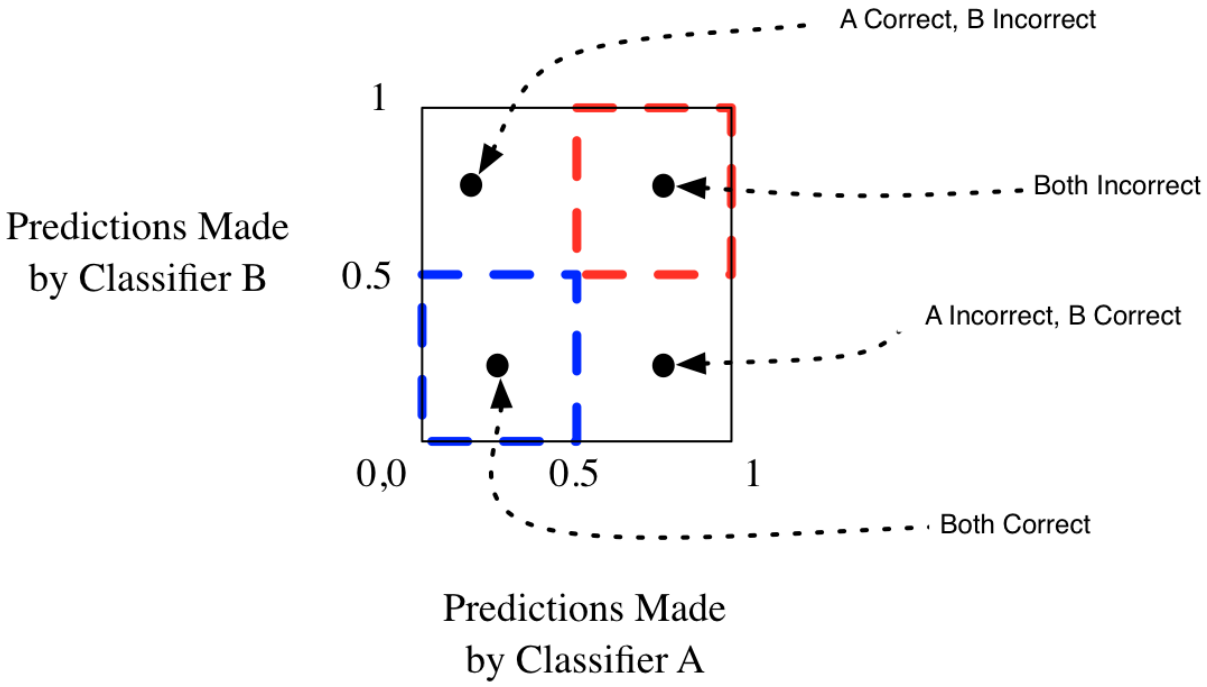


**Figure 26. Comparing Predictions on Positive Examples**

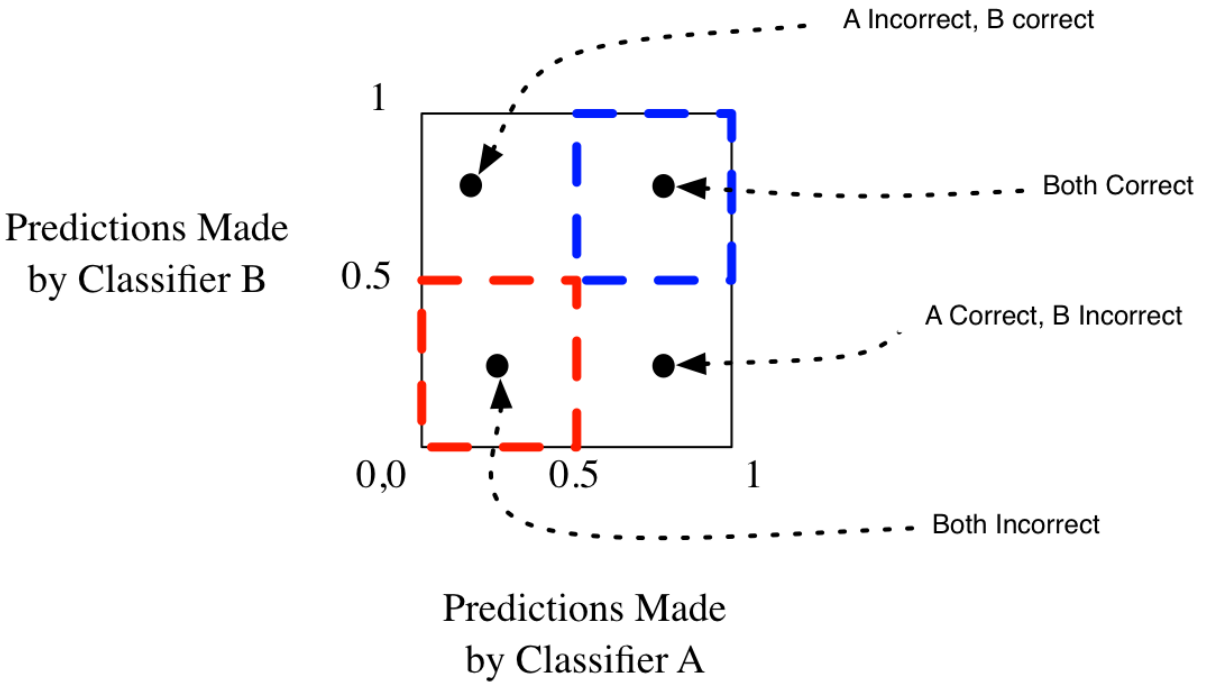


**Figure 27. Comparing Predictions on Negative Examples**

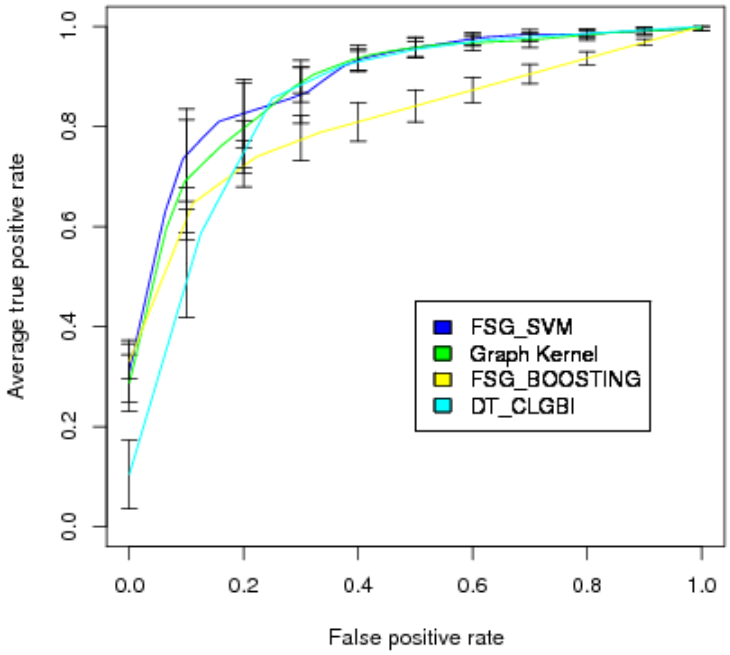


**Figure 28. ROC Curves for the Mutagenesis Data Set**

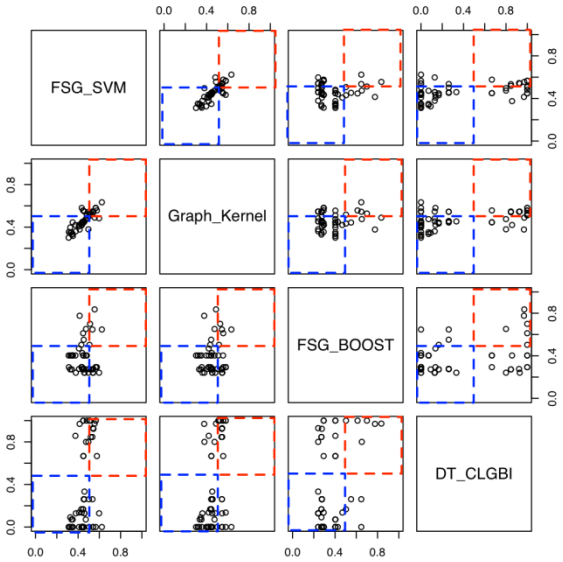


**Figure 29. Comparing predictions on Positive Examples for Mutagenesis Data Set**

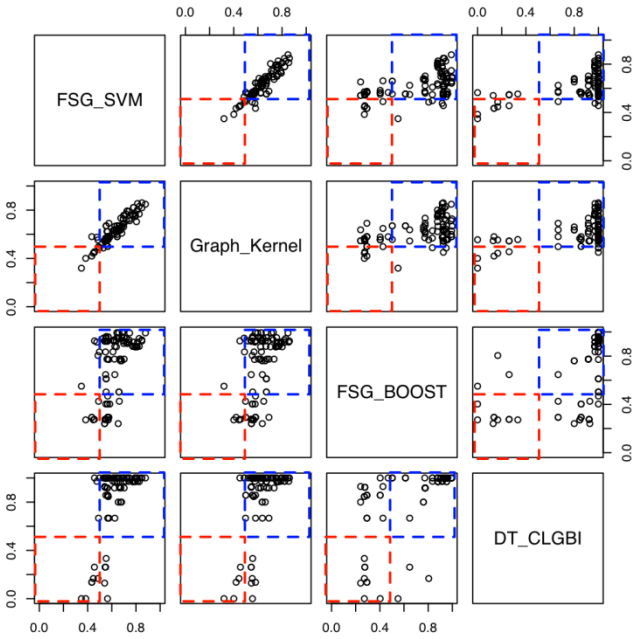


**Figure 30. Comparing Predictions on the Negative Examples in the Mutagenesis Data Set**

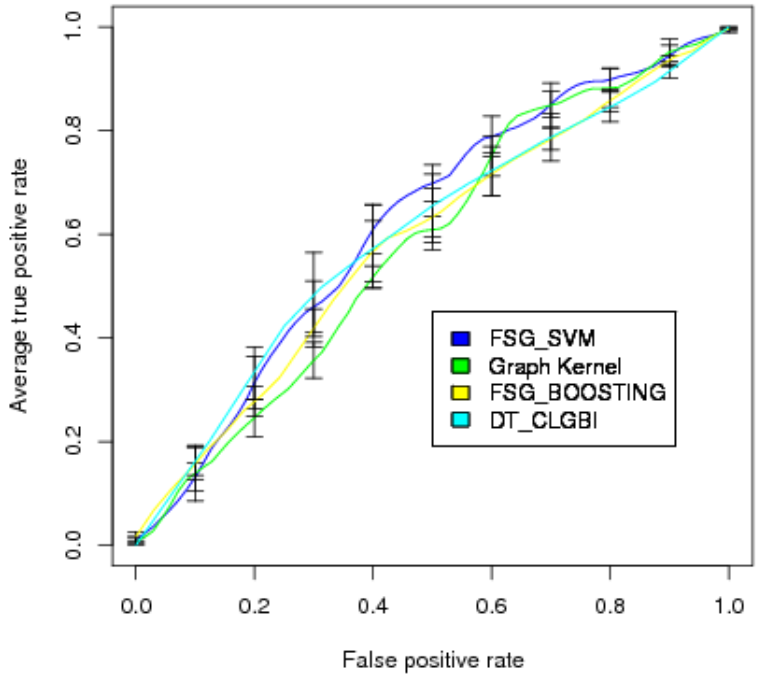


**Figure 31. ROC curves for the MR Data Set**

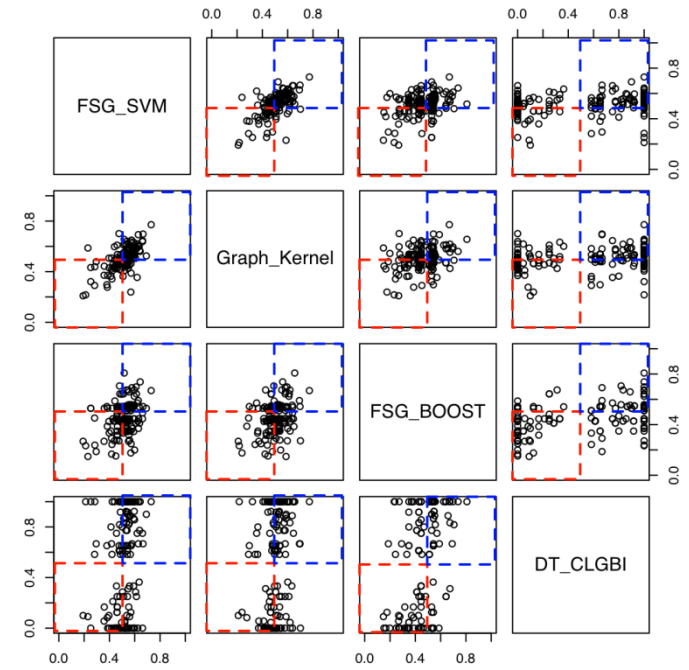


**Figure 32. Comparing the Predictions on the Positive examples in the MR Data Set**

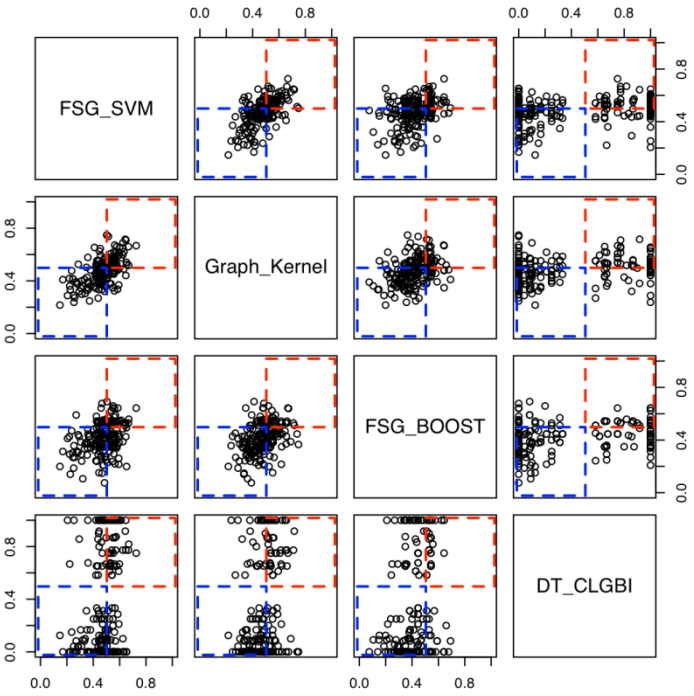


**Figure 33. Comparing the Predictions on the Negative Examples in the MR Data Set**

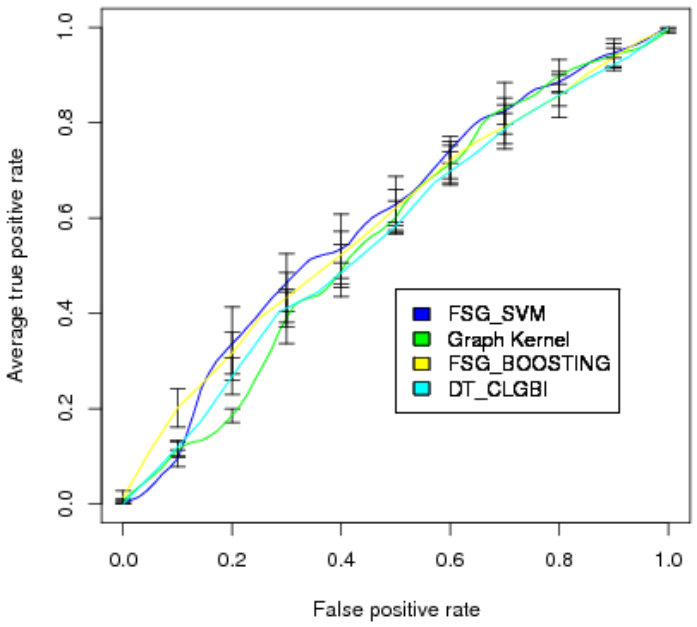


**Figure 34. ROC curves for the MM Data Set**

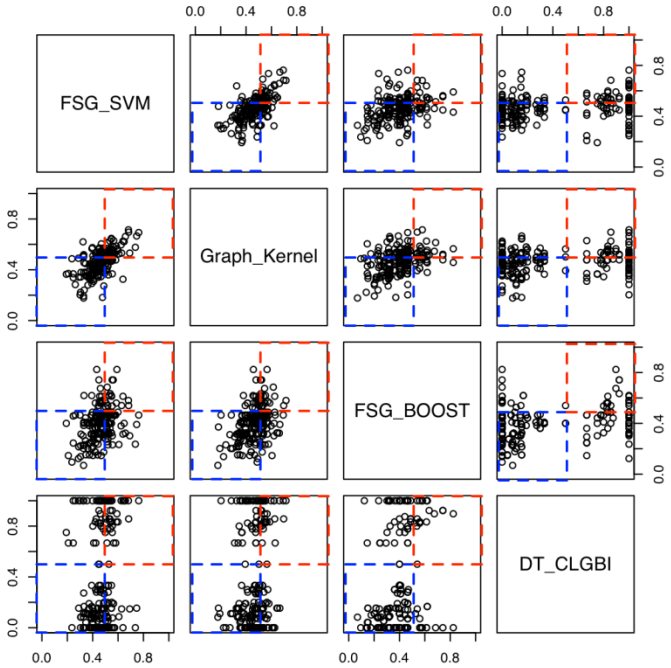


**Figure 35. Comparing the Predictions on the Positive examples in the MM Data Set**

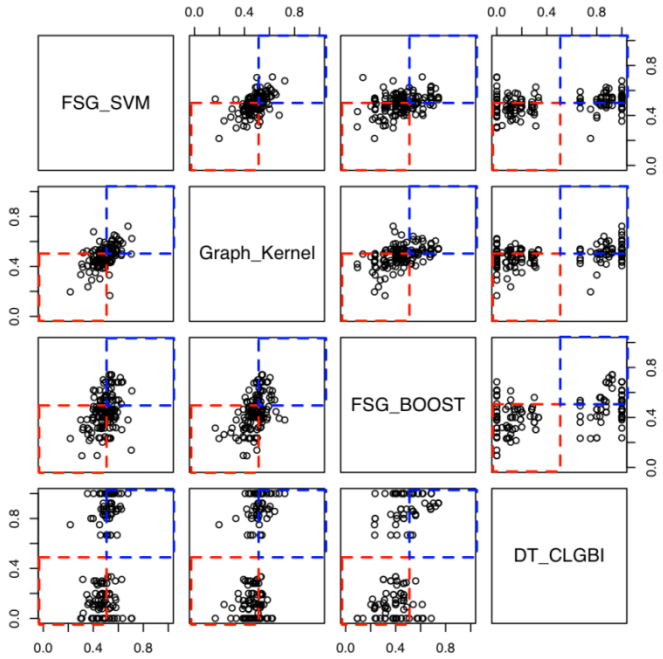


**Figure 36. Comparing the Predictions on the Negative Examples in the MM Data Set**

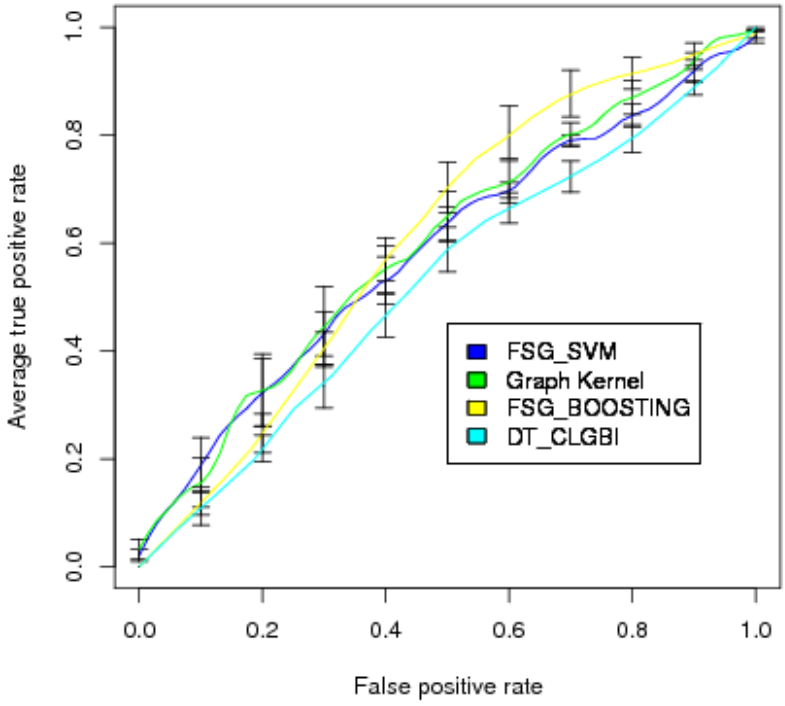


**Figure 37. ROC curves for the FM Data Set**

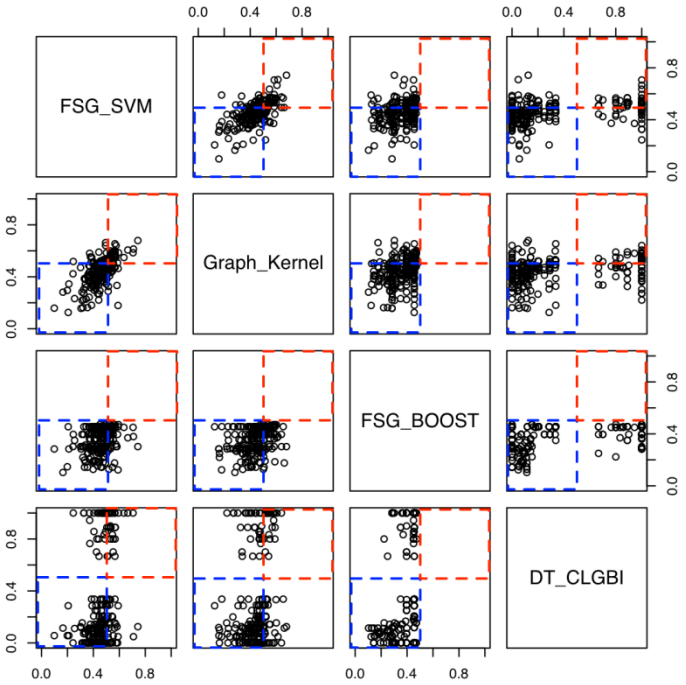


**Figure 38. Comparing the Predictions on the Positive examples in the FM Data Set**

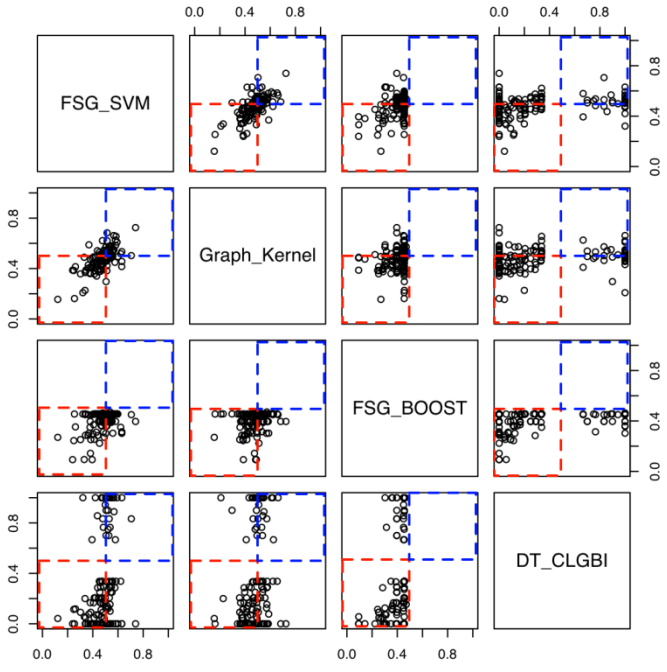


**Figure 39. Comparing the Predictions on the Negative Examples in the FM Data Set**

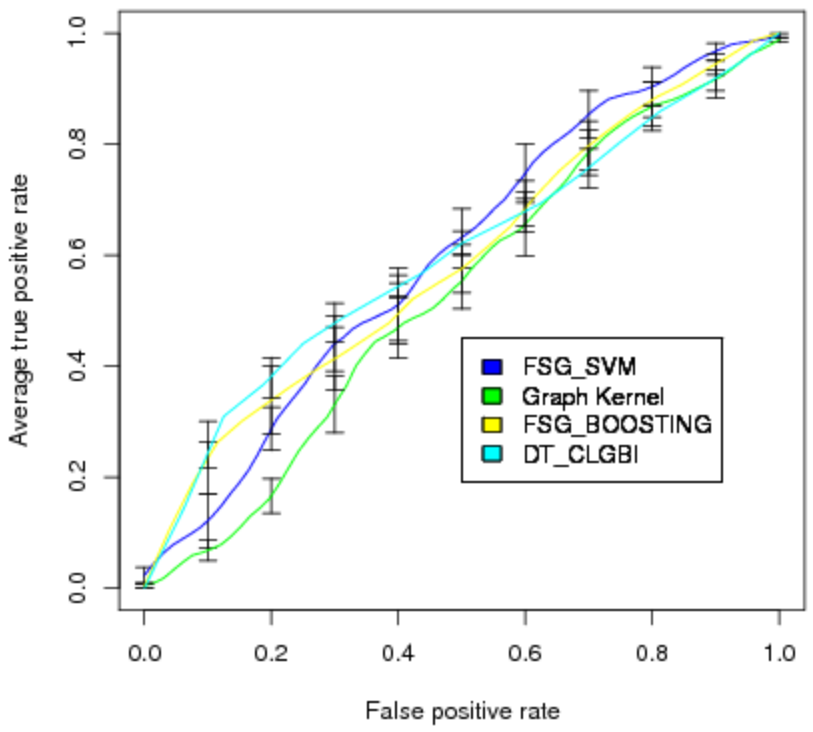


**Figure 40. ROC curves for the FR Data Set**

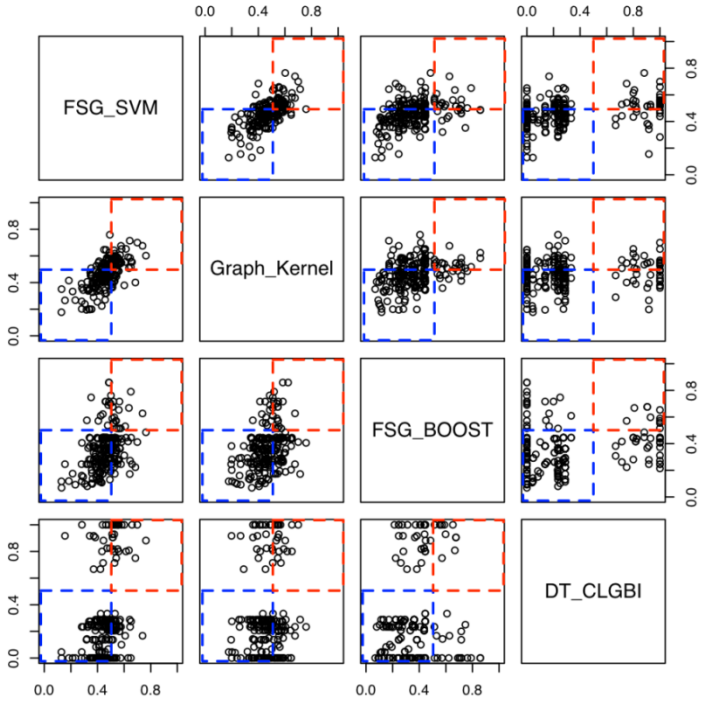


**Figure 41. Comparing the Predictions on the Positive examples in the FR Data Set**

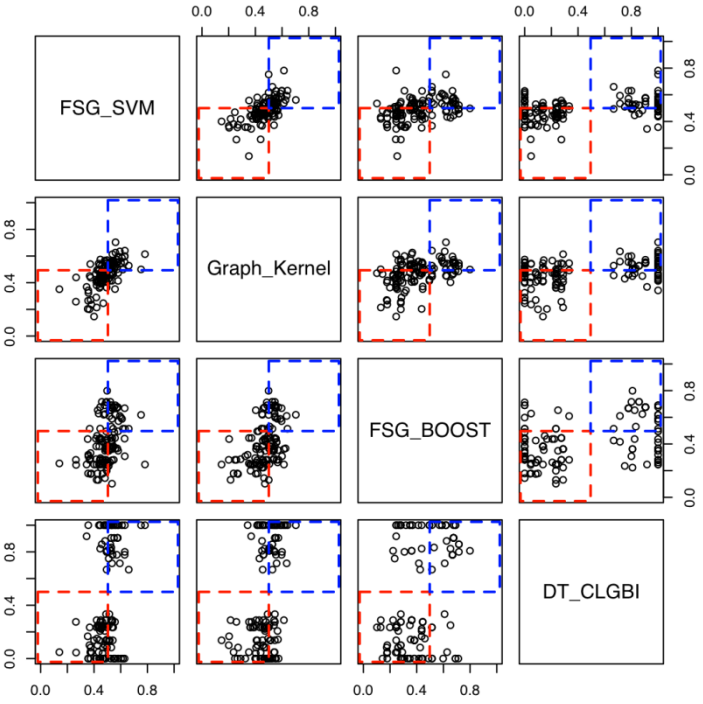


**Figure 42. Comparing the Predictions on the Negative Examples in the FR Data Set**

## 5.2 Experiments with Artificial Datasets

In this section we discuss our experiments and results with artificial datasets. First, we discuss our artificial dataset generator.

Our artificial dataset generator comprises two major components, namely, the graph generator and concept generator. The graph generator generates connected graphs based on five user specified parameters, namely, $N_V$ which is the number of vertices, $N_E$ which is the number of edges, $N_\alpha$ which is the number of vertex labels, $N_\beta$ which is the number of edge labels, and S which is the seed for the random number generator. It is required that $N_E \geq N_V - 1$ , to ensure a connected graph. Based on these parameters, a connected graph $g_S = (V_S, E_S, \alpha_S, \beta_S)$ is generated as follows.

1. $N_V$ vertices are generated and assigned labels from $\alpha_S$ (uniform) randomly.
2. These vertices are connected by adding $N_V - 1$ edges, forcing them to form a path of length $N_V$. These edges are assigned labels from $\beta_S$ (uniform) randomly.
3. Lastly, $N_E - N_V + 1$ edges are added to the graph by first picking two vertices from $V_S$, (uniform) randomly, and adding an edge between these two vertices with a label from $\beta_S$ selected (uniform) randomly.

The concept generator is similar to the graph generator except that the concept is a graph that is not assumed to be connected. So the generation process is identical except that step 2 is not performed and in step 3, $N_E$ edges are added.

For any dataset generation, given user specified parameters for the graphs and the concepts, first a concept is generated. A negative example is simply any graph generated by the user specified parameters. A positive example is any graph in which the concept is embedded.

We describe the embedding procedure below. It is required that the number of vertices in the concept are less than or equal to the number of vertices in the input graph.

1. Select n vertices randomly from the example graph where n are the number of vertices in the concept. Each selected vertex in the example graph is assigned to a vertex in the concept.
2. Change the labels of the n selected vertices in the graph so that they match the vertices in the concept.
3. For every edge between two vertices in the concept introduce an edge between the corresponding vertices in the example graph. If such an edge already exists, only a label change is required.

Overall, the assumptions underlying the generation process are as follows.

1. The distribution of vertex labels, edge labels and the degree distribution is assumed to be uniform and independent of each other, both in the concept and the example graph.
2. Examples are generated using a random process, based on the user specified parameters and are connected graphs.
3. Concepts are generated using a random process, based on the user specified parameters.
4. Positive examples are generated by embedding the concept in an example; negative examples are those examples in which the concept has not been embedded.

We use the artificial dataset generator to compare the performance of the five algorithms/approaches across various parameters of interest. For each experiment, five training sets and five test sets are generated, using different seeds. The algorithms are trained on the

training set and prediction accuracy is measured on the test set. All plots show mean accuracy on the five test sets versus a parameter of interest.

### 5.2.1 Varying Number of Concept Vertices

We vary the number of concept vertices from 1 to 10 adding a single edge with every additional vertex, with the example set to 100 vertices and 100 edges, 50 positive examples and 50 negative examples both in the training and test sets. Figures 43, 44, 45 and 46 show the mean accuracies for different numbers of vertex and edge labels. It can be observed in all the plots that initially when the number of vertices in the concept is small, all classifiers perform poorly as the training examples and test examples are indistinguishable. This changes as the number of vertices in the concept are gradually increased and the performance of all the classifiers improves. Eventually the concept is large enough to sufficiently distinguish the examples and all the classifiers achieve high accuracy. Another observation is that as the number of vertex and edge labels increases, the task becomes easier as it is possible to distinguish the examples by learning a small part of the concept. The difference in performance of the algorithms is noticeable only with both vertex labels and edge labels equal to 2. This difference was found to be significant at only concept size 3, 4, 5 and 6 and we conclude that overall the performance was found to be comparable.

### 5.2.2 Varying Concept Degree

We vary the concept degree by increasing the number of concept edges with the example set to 100 vertices and 100 edges, 50 positive examples and 50 negative examples both in the training and test sets. Figures 47, 48, and 49 show the mean accuracies for different number of vertices in the concept. It can be observed in all the plots that initially all classifiers perform poorly as the

training examples and test examples are indistinguishable. This changes as edges are gradually added to the concept. All the algorithms/approaches except for the graph kernel are able to use these additional distinguishing features to improve on their performance at a significant level. The graph kernel performs poorly and gains accuracy slowly as compared to the all the other algorithms/approaches.

As the degree of the concept increases, distinguishing the examples does become easier but capitalizing on this difference to improve the performance requires learning concepts with structure (like trees and graphs). We postulate that the hypothesis space of the kernel is walks and it is insufficient at capturing concepts involving structure.

### 5.2.3 Varying Number of Example Vertices

We vary the number of vertices in the example by increasing the number of vertices adding a single edge with every additional vertex, with the concept set to 10 vertices and 10 edges, 50 positive examples and 50 negative examples both in the training and test sets. Figure 50 shows the mean accuracies for different number of vertices in the concept. It can be observed that the performance of SubdueCL and the graph kernel drops with additional example size. This difference was found to be significant at all the cases with example vertices greater than 100.

As the number of vertices in the example increases, the concept which is a disconnected graph and embedded at random positions is spread over larger distances. We postulate that the

hypothesis space of SubdueCL is connected graphs and it demonstrates poor performance as it fails to learn disconnected concepts.

### 5.2.4 Varying Example Degree

We vary the degree of the example by increasing the number of edges with the concept set to 5 vertices and 5 edges, example vertices set to 10, 50 positive examples and 50 negative examples both in the training and test sets. Figure 51 shows the mean accuracies for different number of vertices in the concept. It can be observed that the performance of SubdueCL and the graph kernel drops with additional example size. This difference was found to be significant at all the cases with number of edges greater than 30.

In the case of SubdueCL, we postulate that this poor performance is due to the increased number of candidates in the search which causes the greedy search to miss relevant concepts. The graph kernel also has to consider a increased number of walks which causes poor performance.

### 5.2.5 Varying Concept Noise

Varying the concept noise involves varying two parameters: how many examples contain the noisy concept and by what amount the concept is made noisy. We vary the noise in the concept by changing the labels on the vertices and edges in the concept. We refer to this as the noise level. Noise level in the concept is measured as the fraction of the labels changed. Experiments are performed by introducing an increasing number of noisy examples for a variety of noise

levels. Figures 52, 53, 54, 55, 56 and 57 show the performance at different noise levels. Accuracy drops as the number of noisy examples increases and as the noise level increases.

The difference in performance of the algorithms is noticeable in all the cases. The graph kernel and SubdueCL perform poorly and lose accuracy faster as compared to the other algorithms/approaches. This difference was found to be significant at noise levels 0.1 and 0.2. It must be noted here that both SubdueCL and the graph kernel had poor performance even without noise. We postulate that their poor performance is due to difficulty learning the concept, even without noise.

### 5.2.6 Varying Mislabeled Examples

We vary the number of mislabeled examples, by mislabeling positive examples, mislabeling negative examples and swapping class labels on positive and negative examples. For the experiments involving mislabeling positive examples and mislabeling negative examples an additional amount of positive examples and negative examples were added to ensure a balanced dataset. This was because we wanted to analyze the effect of mislabeled examples and the increased negative and positive examples (mislabeled) would skew the training data, making it impossible to determine if the effect was due to the skewed training data or the noise (we also perform experiments with skewed training data which are reported later). The experiment involving swapping labels on positive and negative examples did not have this problem. The results are shown in Figures 58, 59 and 60. Performance drops as the number of mislabeled examples are increased.

The difference in performance of the algorithms is noticeable in all the cases. SubdueCL and DT-CLGBI outperform the others in the mislabeled positives case and the mislabeled negatives case. In the case where the labels are swapped, SubdueCL outperforms all the others.

The model learned by SubdueCL is a list of graphs present only in the positive examples. Due to this, mislabeled examples do not affect its performance as much as the other algorithms/approaches are affected. Eventually however as the number of mislabeled examples increases to 50% of the training data, its performance drops to that of random guessing.

### 5.2.7 Varying Number of Training Examples

We vary the number of training examples and an increase in performance is observed as shown in Figure 61. The difference in performance of the algorithms is noticeable in all the cases.

The best performance is achieved by FSG+AdaBoost and DT-CLGBI, the next better performance is achieved by FSG+SVM followed by SubdueCL and graph kernel.

### 5.2.8 Varying Class Skew

We vary the class skew in the training examples by increasing in turn, the positive or negative examples while holding the others constant. The results are presented in Figures 62 and 63. SubdueCL and graph kernels show poor performance as compared to others in both the cases.

It must be noted here that the graph kernel had poor performance even without the skew. We postulate that their poor performance is due to difficulty learning the concept, even without

the skew. In the case of SubdueCL the poor performance is because it learns a list of graphs present only in the positive examples. Due to this, it is affected more by the skew.

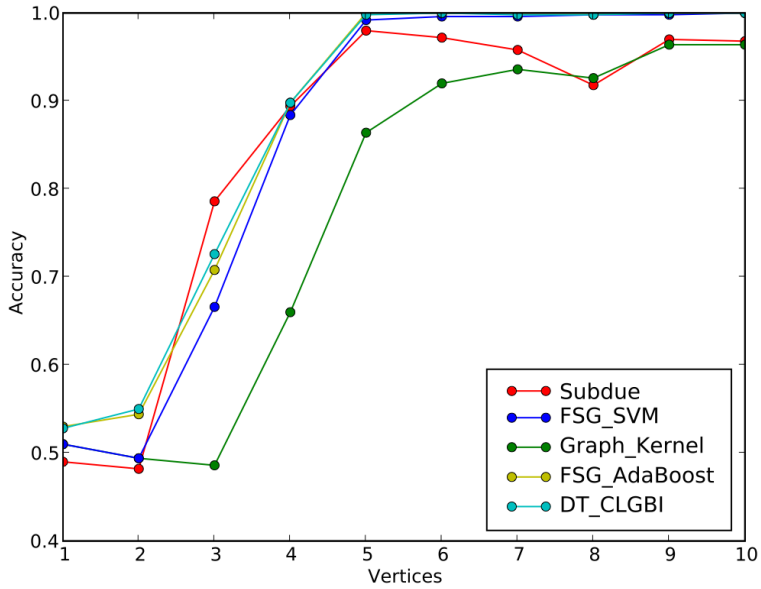


**Figure 43. Varying the Number of Concept Vertices with 2 Vertex Labels and 2 Edge Labels**

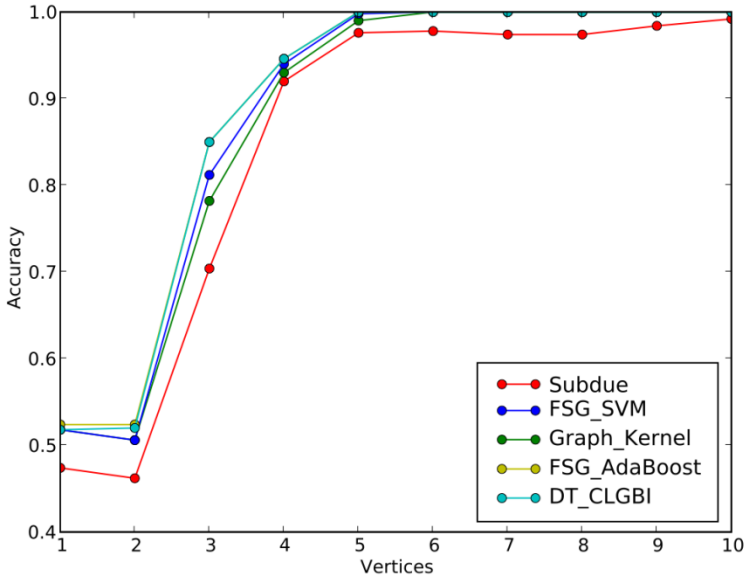


**Figure 44. Varying the Number of Concept Vertices with 3 Vertex Labels and 3 Edge Labels**

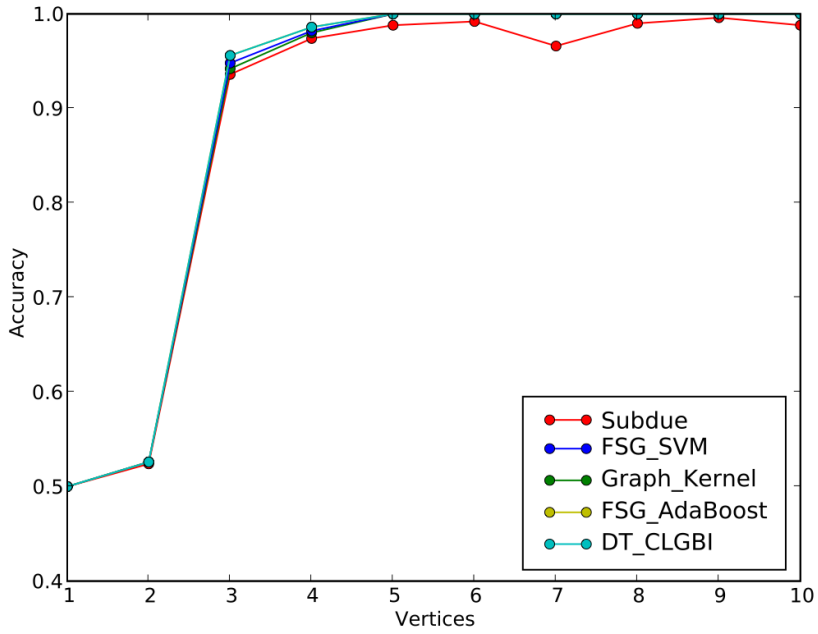


**Figure 45. Varying the Number of Concept Vertices with 4 Vertex Labels and 4 Edge Labels**

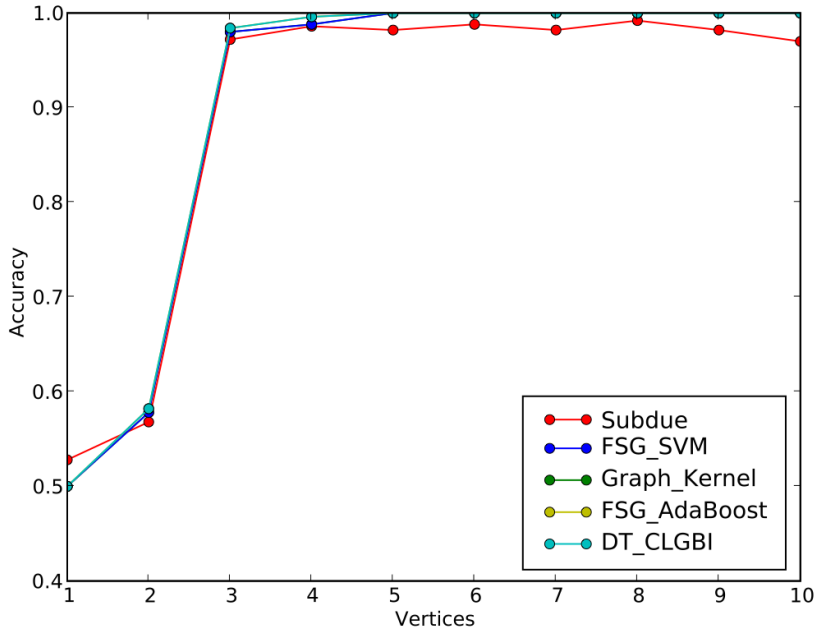


**Figure 46. Varying the Number of Concept Vertices with 5 Vertex Labels and 5 Edge Labels**

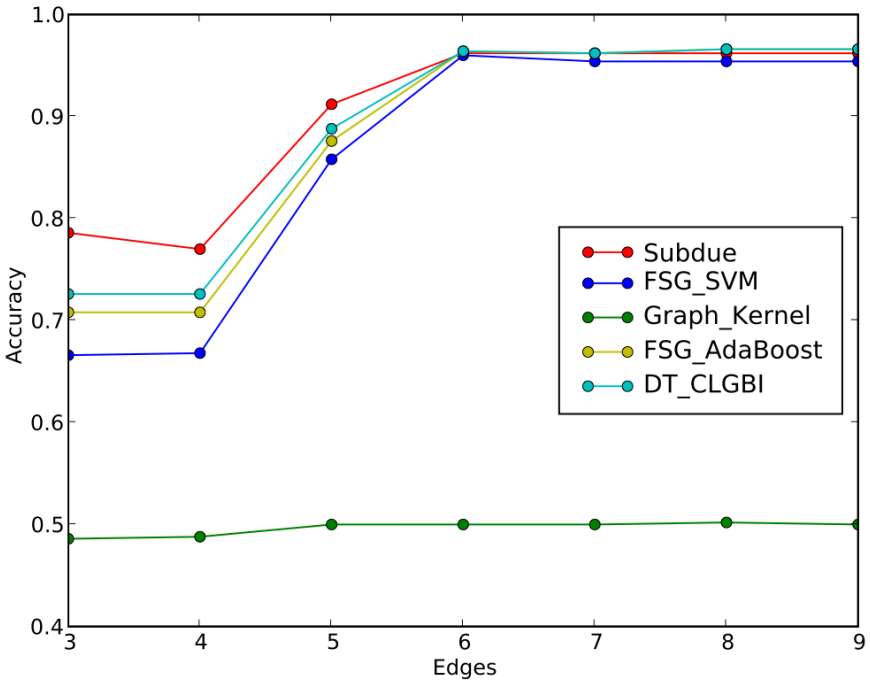


**Figure 47. Varying the Concept Degree with 4 Vertices**

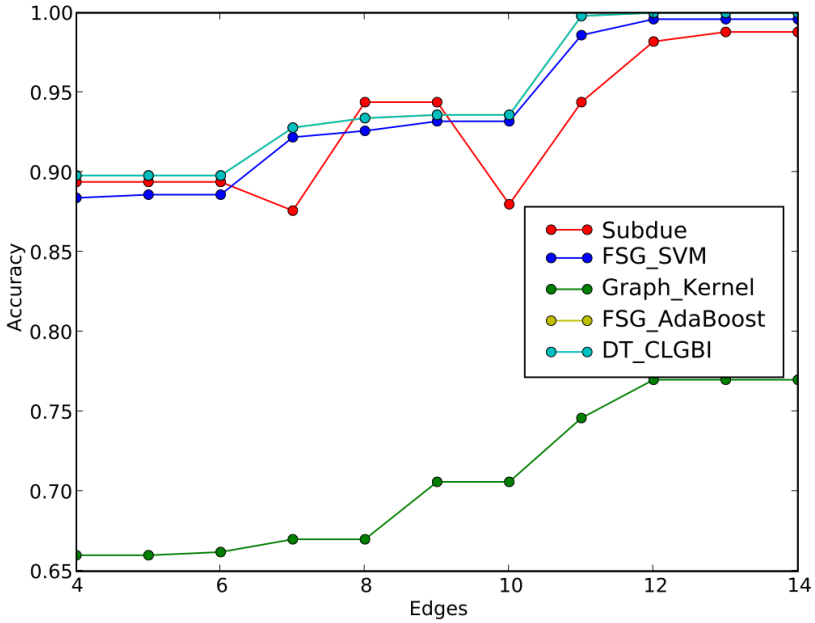


**Figure 48. Varying the Concept Degree with 5 Vertices**

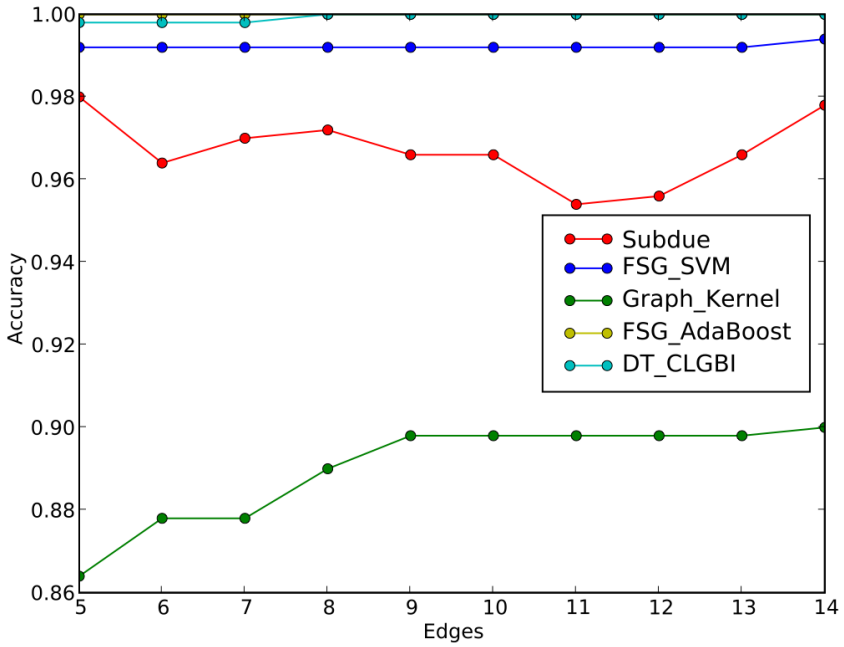


**Figure 49. Varying the Concept Degree with 6 Vertices**

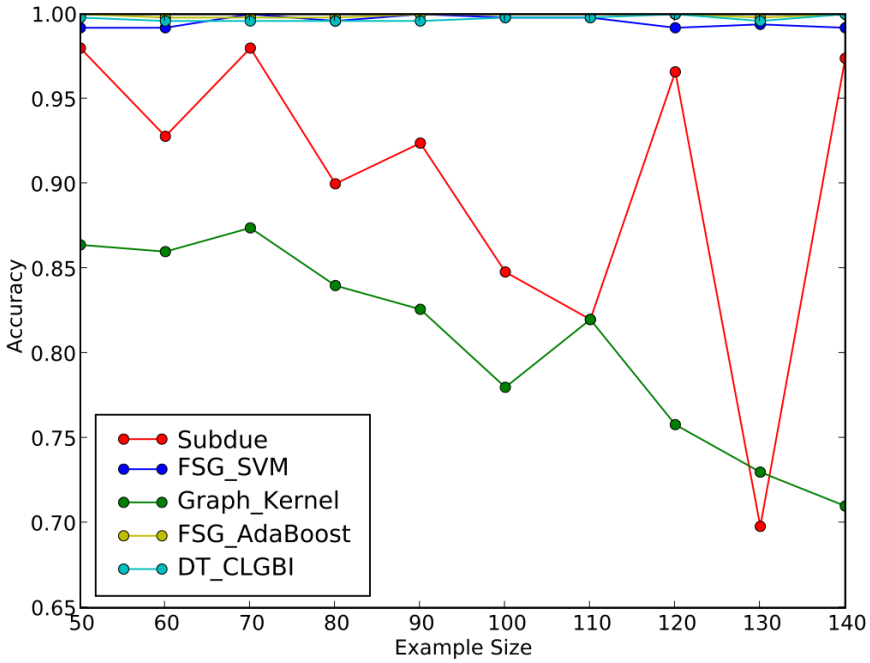


**Figure 50. Increasing the Example Size**

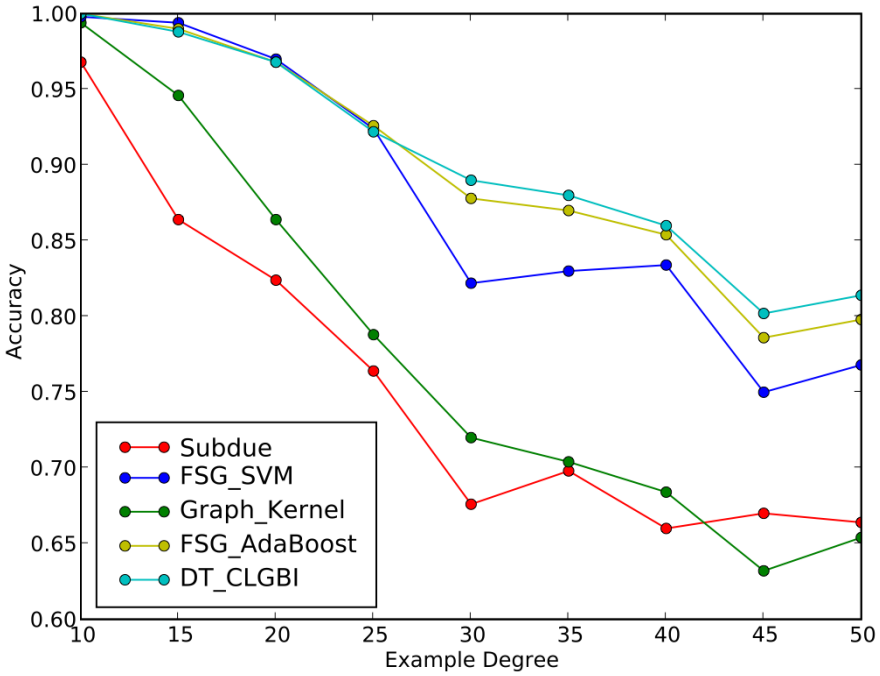


**Figure 51. Increasing the Example Degree**

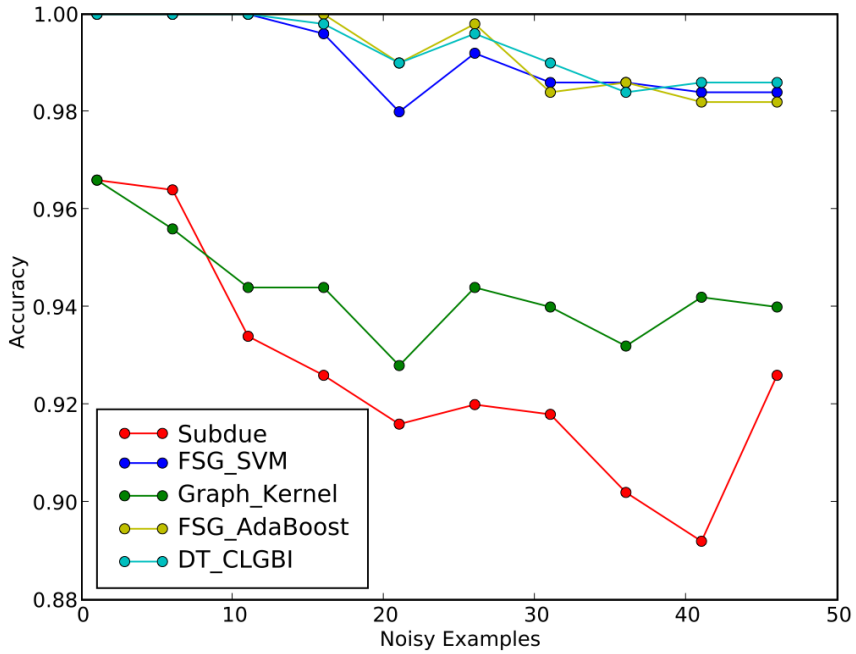


**Figure 52. Effect of Noise, Noise Level 0.1**

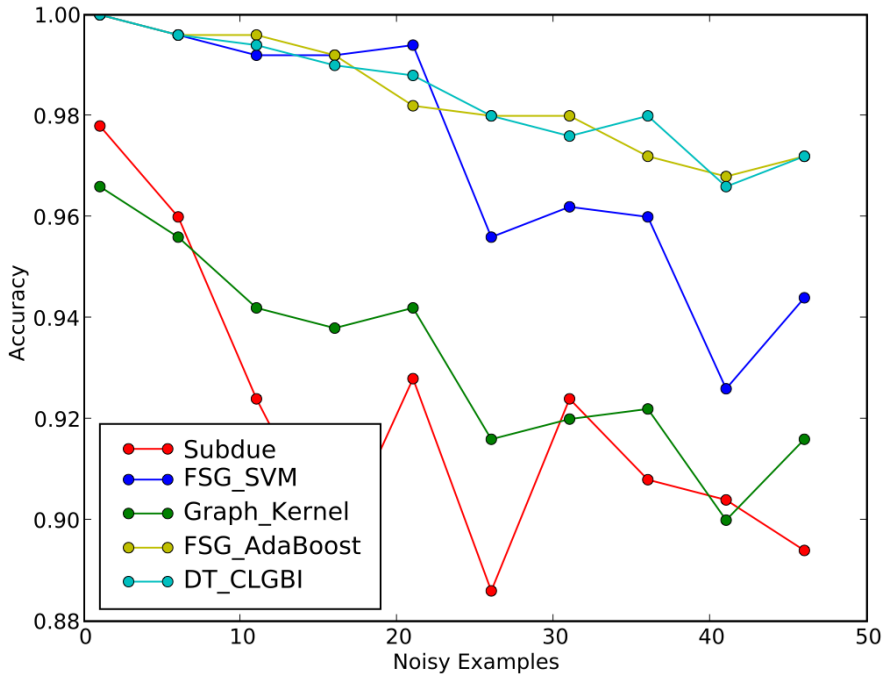


**Figure 53. Effect of Noise, Noise Level 0.2**

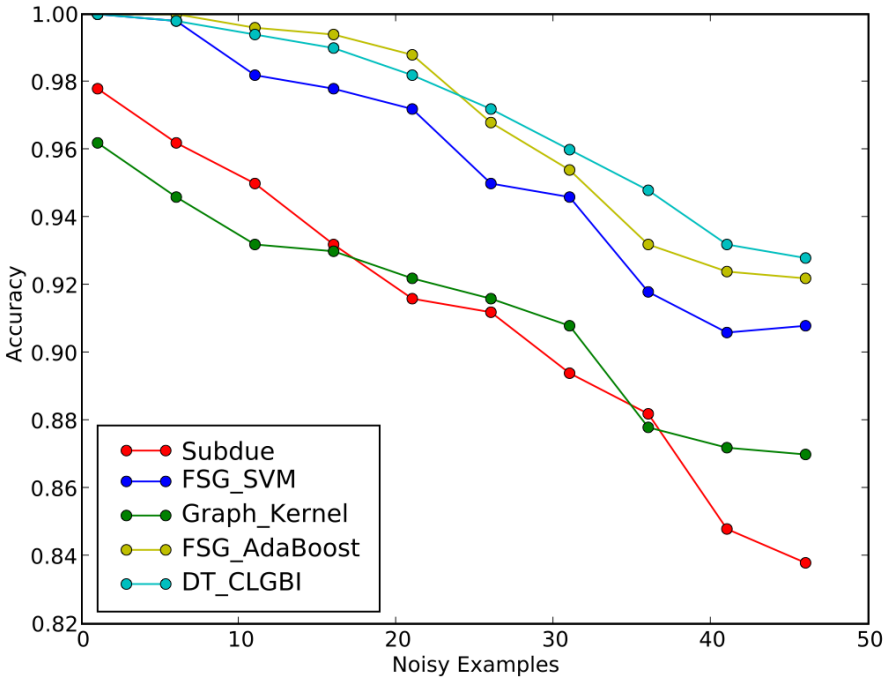


**Figure 54. Effect of Noise, Noise Level 0.3**

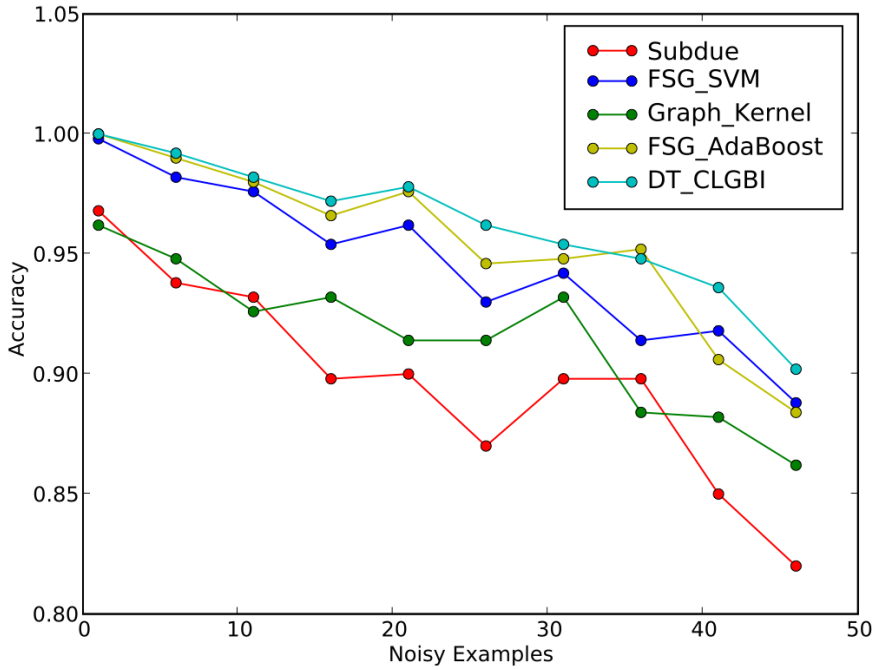


**Figure 55. Effect of Noise, Noise Level 0.4**

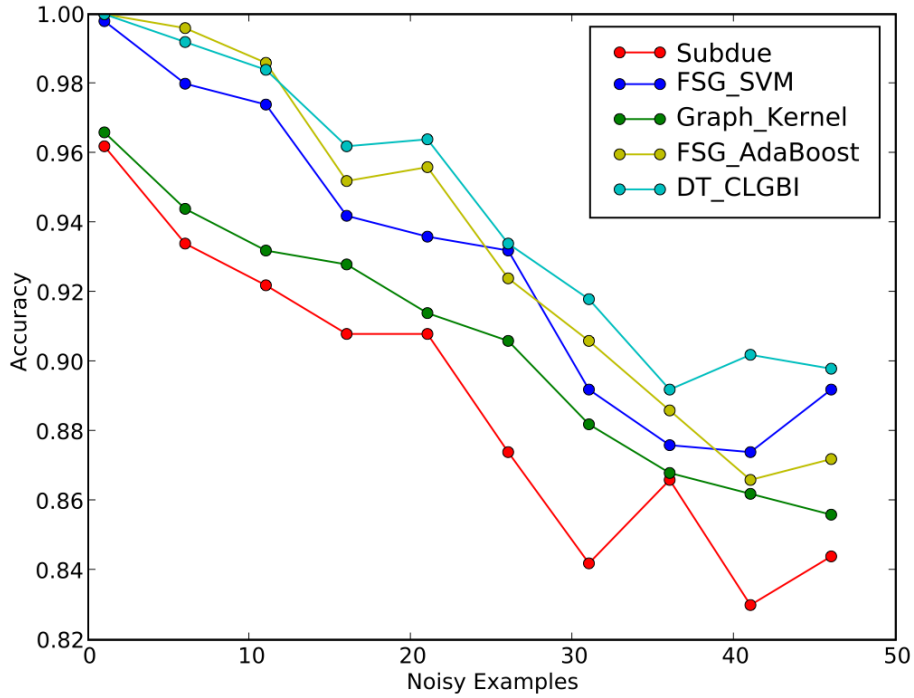


**Figure 56. Effect of Noise, Noise Level 0.5**

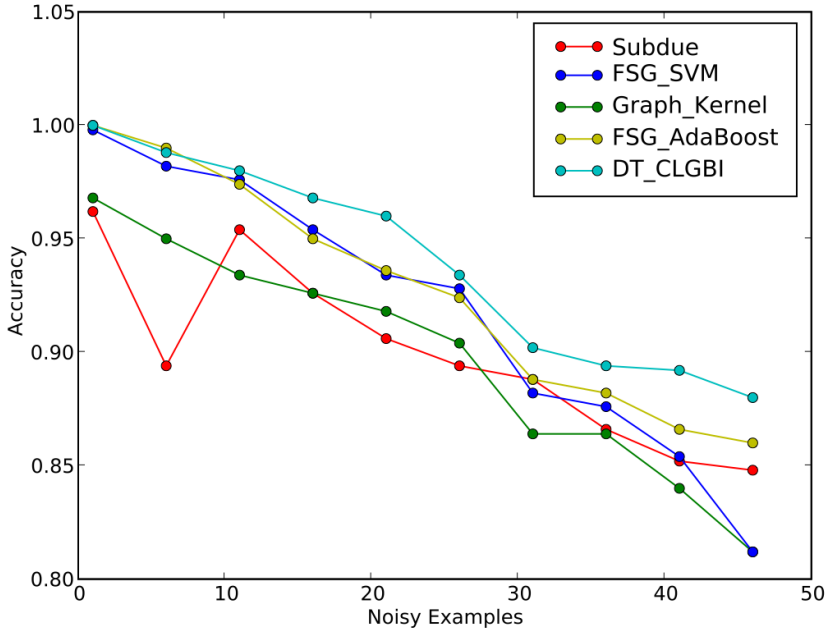


**Figure 57. Effect of Noise, Noise Level 0.6**

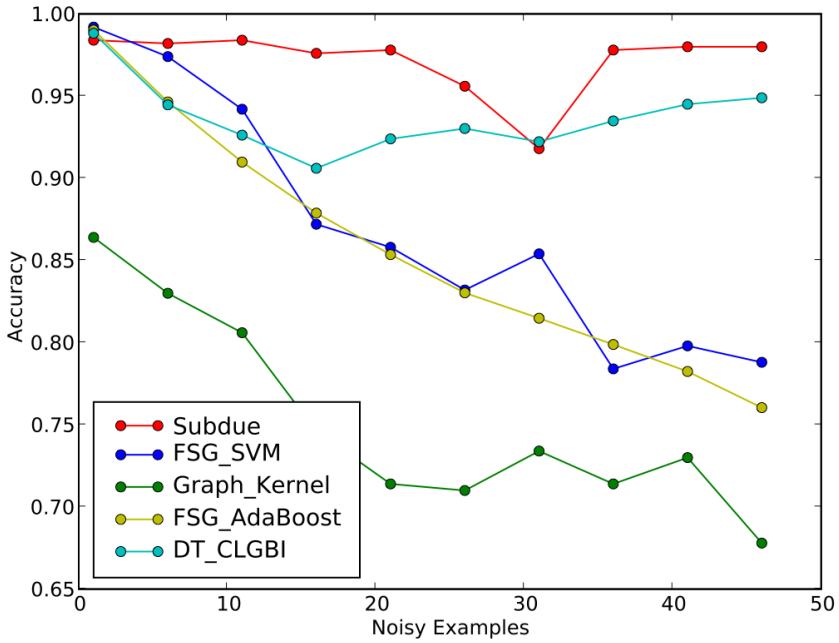


**Figure 58. Effect of Mislabeled Positive Examples**

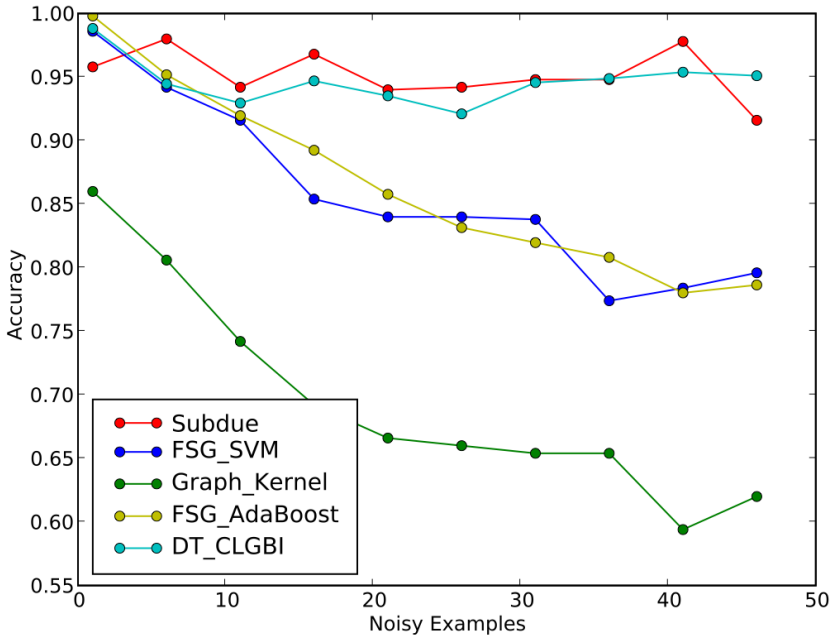


**Figure 59. Effect of Mislabeled Negative Examples**

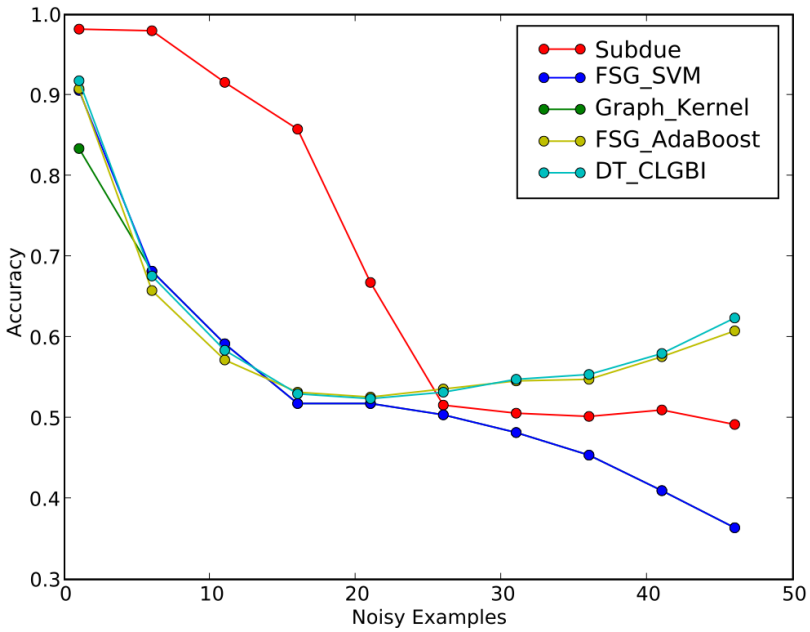


**Figure 60. Effect of Mislabeled examples, swapping of class labels**

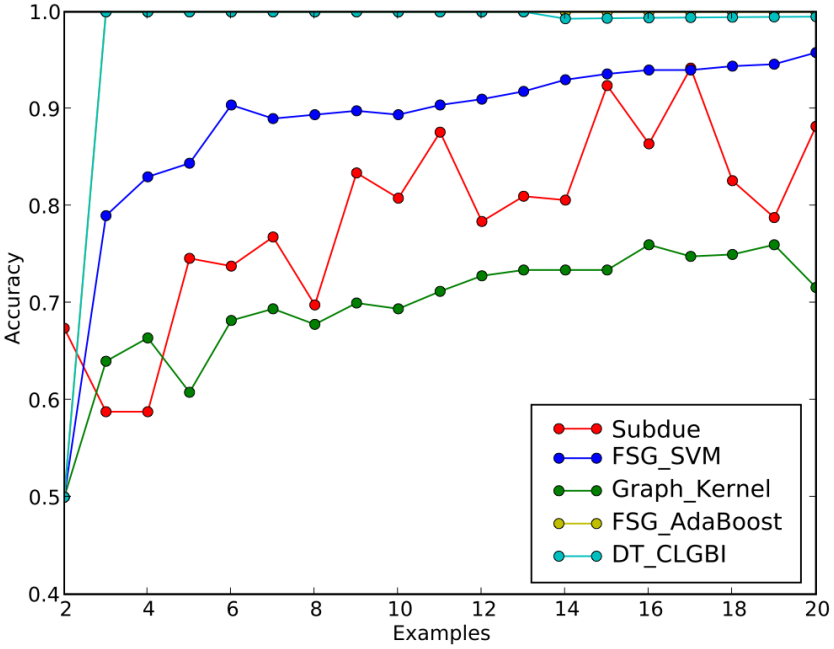


**Figure 61. Effect of Increasing the Number of Training Examples**

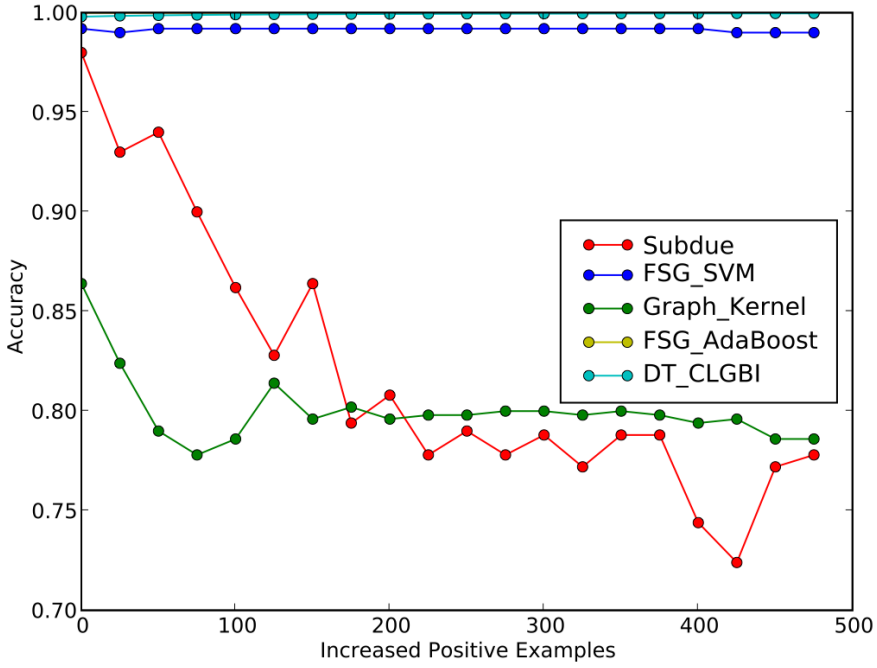


**Figure 62. Effect of Class Skew, Increased Positive Examples**

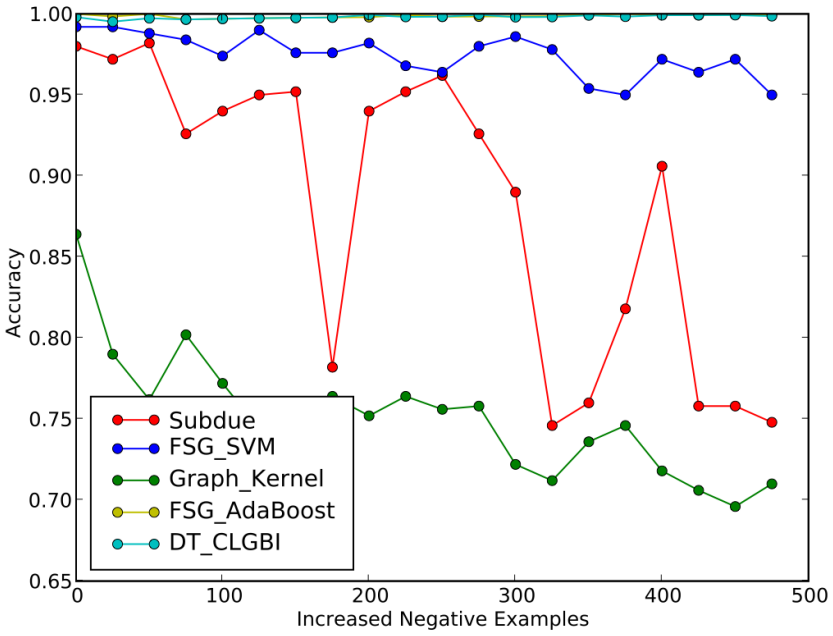


**Figure 63. Effect of Class Skew, Increased Negative Examples**

## 5.3 Conclusions of the Comparison

We performed an empirical comparison of the major approaches for graph classification introduced in the literature, namely, SubdueCL, frequent subgraph mining in conjunction with SVMs, walk-based graph kernel, frequent subgraph mining in conjunction with AdaBoost, and DT-CLGBI. Experiments were performed on five real world data sets from the Mutagenesis and Predictive Toxicology domain, and a corpus of artificial data sets. The conclusions of the comparison are as follows.

In datasets where the underlying concept has a high average degree, walk-based graph kernels perform poorly as compared to other approaches. The hypothesis space of the kernel is walks, and it is insufficient at capturing concepts involving significant structure. In datasets where the underlying concept is disconnected, SubdueCL performs poorly as compared to other

approaches. The hypothesis space of SubdueCL is connected graphs, and it is insufficient at capturing concepts which consist of a disconnected graph. FSG+SVM, FSG+AdaBoost, DT-CLGBI have comparable performance in most cases.

Given the overall goal of conducting a comprehensive empirical comparison of approaches for graph classification in order to identify their underlying strengths and weaknesses, our empirical comparison has two major limitations. First, the artificial datasets were generated according to a model that assumed a uniform distribution of

vertex labels, edge labels and a uniform degree distribution. Furthermore, we assumed that these distributions are independent. Secondly, a qualitative comparison of the learned models was not performed. An approach that learns a model involving fewer/smaller graphs is superior, because prediction involves performing subgraph isomorphism.

# 6. Alternative Computation of the Walk-Based Kernel

We now present our work on the computation of the direct product kernel. The motivation behind this work is to address the problem of computing the kernel for long walks. From the perspective of graph classification and regression, the use of graph kernels is an attractive approach as it avoids subgraph isomorphism. Any valid graph kernel could be used with support vector classification or support vector regression to extend these algorithms to operate on graph transactions.

The direct product kernel [32] for graph classification, is based on defining a feature for every possible label sequence in a labeled graph and counting how many label sequences in two given graphs are identical. Although the direct product kernel has achieved promising results in terms of accuracy, the kernel computation is not feasible for large graphs. This is because computing the direct product kernel for two graphs is essentially computing either the inverse of, or by diagonalizing, the adjacency matrix of the direct product of these two graphs. For two graphs with adjacency matrices of sizes m and n, the adjacency matrix of their direct product graph can be of size mn in the worst case. As both matrix inversion and matrix diagonalizing in the general case is $O(n^3)$, computing the direct product kernel is $O((mn)^3)$. Our survey of data sets in graph classification indicates that most graphs have adjacency matrices of sizes in the order of hundreds which often leads to adjacency matrices of direct product graphs (of two graphs) having sizes in the order of thousands.

In this work we show how the direct product kernel can be computed in $O((m+n)^3)$. The key insight behind our result is that the language of label sequences in a labeled graph is a regular language and that regular languages are closed under union and intersection.

Intuitively, the direct product kernel is based on defining a feature for every possible label sequence in a labeled graph and counting how many label sequences in two given graphs are identical. So basically, the direct product kernel takes as input two graphs and outputs a count of the identical walks that can be taken in both of the graphs (refer to Figure 64).

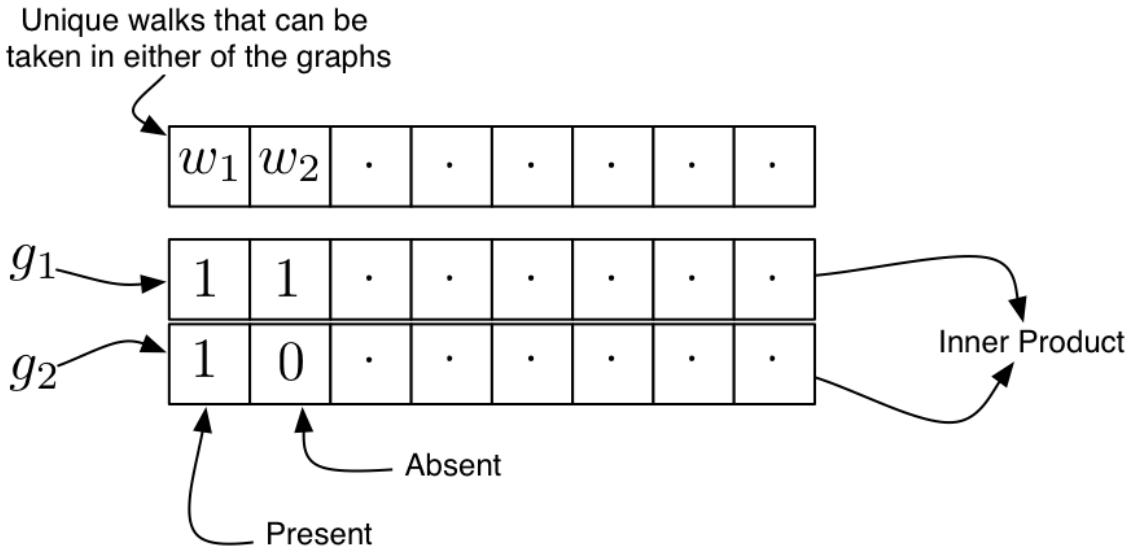


**Figure 64. Feature Space of the Walk-based Kernel**

### 6.1 Notions and Notation

To define the direct product kernel we need some more notions and notation. A walk w in a graph g is a sequence of edges $e_1, e_2, e_3, \ldots e_n$ such that for every $e_i = (u, v)$ and $e_{i+1} = (x, y)$, v = x is obeyed. Every walk is associated with a sequence of edge labels $\alpha_1, \alpha_2, \alpha_3 \ldots \alpha_n$. An adjacency matrix $M_g$ of graph g is defined as,

$$M_{g_{i,j}} = \begin{cases} 1 \; if \; (v_i, v_j) \in E \\ 0 \; otherwise \end{cases}$$

A direct product of two graphs $\mathrm{g}_1 = (\mathrm{V}_1, \mathrm{E}_1, \alpha_1)$ and $\mathrm{g}_2 = (\mathrm{V}_2, \mathrm{E}_2, \alpha_2)$ (with identical edge label alphabet $\sum$) $g1 \times g2$ is defined as,

1. $V_{g_1 \times g_2} = \{(v_1, v_2) \in V_1 \times V_2\}$
2. $E_{g_1 \times g_2} = \{(u_1, u_2) \in E_1 \times E_2\}$ such that
    a) $(u_1, v_1) \in E_1$
    b) $(u_2, v_2) \in E_2$
    c) $\alpha_1(u_1, v_1) = \alpha_2(u_2, v_2)$

An important observation here is that taking a walk on a direct product graph $\mathrm{g}_1 \times \mathrm{g}_2$ is equivalent to taking an identical walk on graphs $g_1$ and $g_2$. Stated differently, this means that we can take a certain walk on $\mathrm{g}_1 \times \mathrm{g}_2$ if and only if there exists a corresponding identical walk in both $g_1$ and $g_2$.

For two graphs $\mathrm{g}_1 = (\mathrm{V}_1, \mathrm{E}_1, \alpha_1)$ and $\mathrm{g}_2 = (\mathrm{V}_2, \mathrm{E}_2, \alpha_2)$ (with identical edge label alphabet $\sum$) let $M_{g_1 \times g_2}$ be the adjacency matrix of their direct product graph $g_1 \times g_2$ with a sequence of weights $\lambda_0, \lambda_1, \ldots$ such that $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 0$ for all $i \in \mathbb{N}$, the direct product kernel $k_{g_1 \times g_2}$ is defined as,

$$k_{g_1 \times g_2} = \sum_{i,j=1}^{V_{g_1 \times g_2}} \sum_{\ell=0}^{\infty} \left[M_{g_1 \times g_2}^{\ell}\right]_{ij}$$

if the limit exists.

Intuitively, the direct product kernel computes the powers of the adjacency matrix of the direct product graph $M_{g_1 \times g_2}$ and sums them. This is equivalent to counting the identical walks

that can be taken in both the input graphs. This is because any walk in $g_1 \times g_2$ corresponds to an identical walk in both $g_1$ and $g_2$ and the $\ell^{th}$ power of $M_{g_1 \times g_2}$ captures all walks of length $\ell$ in $M_{g_1 \times g_2}$.

### 6.2 Faster Alternative Computation

In this section we present a faster alternative to computing the direct product kernel. The key insight behind this is based on showing that the language of edge label sequences corresponding to walks in a graph is a regular language.

**Lemma 1**

Let $w = e_1, e_2, \ldots e_n$ be a walk in graph $g = (V, E, \alpha)$ and let $s = \alpha(e_1), \alpha(e_2), \ldots \alpha(e_n)$ be the sequence of edge labels corresponding to each of the edges in the walk. Let $\mathcal{L}$ be the language of all such sequences corresponding to walks in $g$. Then, $\mathcal{L}$ is a regular language.

**Proof**

Construct a finite automaton $M = (Q, \Sigma, \delta, q, F)$ as follows. The set of states $Q$ is constructed by introducing a state $s_v$ corresponding to every vertex $v$ in $g$. Additionally, two states $s_0$ and $s_F$ are introduced in $Q$. The alphabet $\Sigma$ is the same as the alphabet of the edge labels. The set of transitions $\delta$ is constructed by introducing a transition from state $s_u$ to $s_v$ on symbol $\alpha(e)$ for every directed edge $e = (u, v)$ in $g$. Additionally $\varepsilon$ transitions are introduced from the state $s_0$ to every other state in $s_v$ in $Q$ except $s_F$ each of which corresponds to a vertex in $g$. Also, $\varepsilon$ transitions are introduced from every state in $Q$ except for $s_0$ and $s_F$ , each of which corresponds to a vertex in $g$ to the state $s_F$ . Set the start state $q = s_0$ and the final state $F = s_F$. By construction, $M$ accepts $\mathcal{L}$ and hence $\mathcal{L}$ is regular. ■

We now show that the language of edge label sequences corresponding to walks in a direct product graph is basically the intersection of the two regular languages of edge labels corresponding to walks in the two graphs and is itself a regular language as regular languages are closed under intersection.

**Lemma 2**

Let $g_{g_1 \times g_2}$ be the direct product graph of $g_1$ and $g_2$. Let $\mathcal{L}_{g_1 \times g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$. Similarly, let $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ be languages of all sequences of edge labels corresponding to walks in $g_1$ and $g_2$ respectively. Then, $\mathcal{L}_{g_1 \times g_2} = \mathcal{L}_{g_1 \cap g_2}$ is regular.

**Proof**

From the definition of direct graph product, taking a walk on a direct product graph $g_1 \times g_2$ is equivalent to taking an identical walk on graphs $g_1$ and $g_2$. Each of these walks has a corresponding edge label sequence associated with it. As $\mathcal{L}_{g_1 \times g_2}$ is the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$, and $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ are languages of all sequences of edge labels corresponding to walks in $g_1$ and $g_2$ respectively, thus $\mathcal{L}_{g_1 \times g_2} = \mathcal{L}_{g_1 \cap g_2}$ follows. ■

We now introduce the notion of union of the two languages corresponding to the sequence of edge labels corresponding to walks in the two graphs. Let $\mathrm{g}_1 = (\mathrm{V}_1, \mathrm{E}_1, \alpha_1)$ and $\mathrm{g}_2 = (\mathrm{V}_2, \mathrm{E}_2, \alpha_2)$ be two graphs (with identical edge label alphabet $\sum$), then the union of the corresponding languages $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ is denoted by $\mathcal{L}_{g_1 \cup g_2}$

**Lemma 3**

$\mathcal{L}_{g_1 \cup g_2}$ is regular.

**Proof**

Follows from the definitions and the property that regular languages are closed under union. ■

We can now show a result that gives us a relation between the sizes of the languages considered so far.

**Lemma 4**

Let $g_{g_1 \times g_2}$ be the direct product graph of $g_1$ and $g_2$. Let $\mathcal{L}_{g_1 \times g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$. Let $g_{g_1 \cup g_2}$ be the union graph of $g_1$ and $g_2$. Let $\mathcal{L}_{g_1 \cup g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \cup g_2}$. Similarly, let $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ be languages of all sequences of edge labels corresponding to walks in $g_1$ and $g_2$ respectively. Then $|\mathcal{L}_{g_1 \times g_2}| = \left|\mathcal{L}_{g_1}\right| + \left|\mathcal{L}_{g_2}\right| - \left|\mathcal{L}_{g_{1 \times g_2}}\right|$.

**Proof**

Follows from Lemmas 1, 2 and 3 and the property that regular languages are closed under union and intersection. ■

This result can be easily extended to subsets of these languages which place a restriction on the size of the sequences in the language.

**Lemma 5**

Let $\mathcal{L}_{g_1}^{\ell}$ , $\mathcal{L}_{g_2}^{\ell}$ $\mathcal{L}_{g_1 \times g_2}^{\ell}$ and $\mathcal{L}_{g_1 \cup g_2}^{\ell}$ be subsets of languages $\mathcal{L}_{g_1}$, $\mathcal{L}_{g_2}$ , $\mathcal{L}_{g_1 \times g_2}$ and $\mathcal{L}_{g_1 \cup g_2}$ such that they do not contain sequences longer than $\ell$. Then $\mathcal{L}_{g_1}^{\ell}$ , $\mathcal{L}_{g_2}^{\ell}$ $\mathcal{L}_{g_1 \times g_2}^{\ell}$ and $\mathcal{L}_{g_1 \cup g_2}^{\ell}$ are regular and $|\mathcal{L}_{g_1 \times g_2}^{\ell}| = \left|\mathcal{L}_{g_1}^{\ell}\right| + \left|\mathcal{L}_{g_2}^{\ell}\right| - \left|\mathcal{L}_{g_1 \cup g_2}^{\ell}\right|$ holds.

**Proof**

Clearly $\mathcal{L}_{g_1}^{\ell}$ , $\mathcal{L}_{g_2}^{\ell}$ $\mathcal{L}_{g_1 \times g_2}^{\ell}$ and $\mathcal{L}_{g_1 \cup g_2}^{\ell}$ are finite languages by definition and hence regular. The result follows from Lemma 4 and the property that regular languages are closed under union and intersection. ■

The problem thus reduces to counting the number of strings in a regular language of length no more than $\ell$. It has been shown that this problem can be solved in $O(n^3)$ [44] where $n$ is the number of states in the finite automata for the regular language. This approach is also based on diagonalising the adjacency matrix of the

finite automata. Note here that in order to compute $|\mathcal{L}_{g_1 \times g_2}^{\ell}|$ we compute $\left|\mathcal{L}_{g_1}^{\ell}\right| + \left|\mathcal{L}_{g_2}^{\ell}\right| - \left|\mathcal{L}_{g_1 \cup g_2}^{\ell}\right|$. The largest finite automata and hence the largest adjacency matrix to be diagonalized is $\left|\mathcal{L}_{g_1 \cup g_2}^{\ell}\right|$. The previous approach in essence dealt with $|\mathcal{L}_{g_1 \times g_2}^{\ell}|$. The key difference from the previous computation is that now we are dealing with finite automata corresponding to the union of two regular languages which grows as $O(m+n)$ for automatas with sizes $m$ and $n$ (for the two input graphs) instead of the finite automata corresponding to the intersection of the two languages which grows as $O(mn)$. This implies that the direct product kernel can be computed in $O((m+n)^3)$ instead of $O((mn)^3)$.

### 6.3 Experimental Results

We compare the proposed alternative kernel computation to the approximation of the kernel by matrix multiplication. Figure 65 shows the training time for the Mutagenesis and PTC datasets while approximating the kernel value using matrix multiplication. Figure 67 shows the time for the alternative computation (evaluation of the inverse of the union matrix). The results indicate that although the alternative computation is expensive as compared to approximation for small walks, it is comparable when compared to approximation for long walks. Longer walks, in general, lead to higher accuracy (refer Figure 66), but after a certain walk length, we have diminishing returns in the sense that the extra expense of computation does not buy us better predictive accuracy. So the predictive accuracy flattens out at a particular walk length where approximation by matrix multiplication turns to be cheaper. In general, it must be noted that approximation of the kernel by matrix multiplication may turn out to be cheaper than the alternative computation in practice for smaller walk sizes.

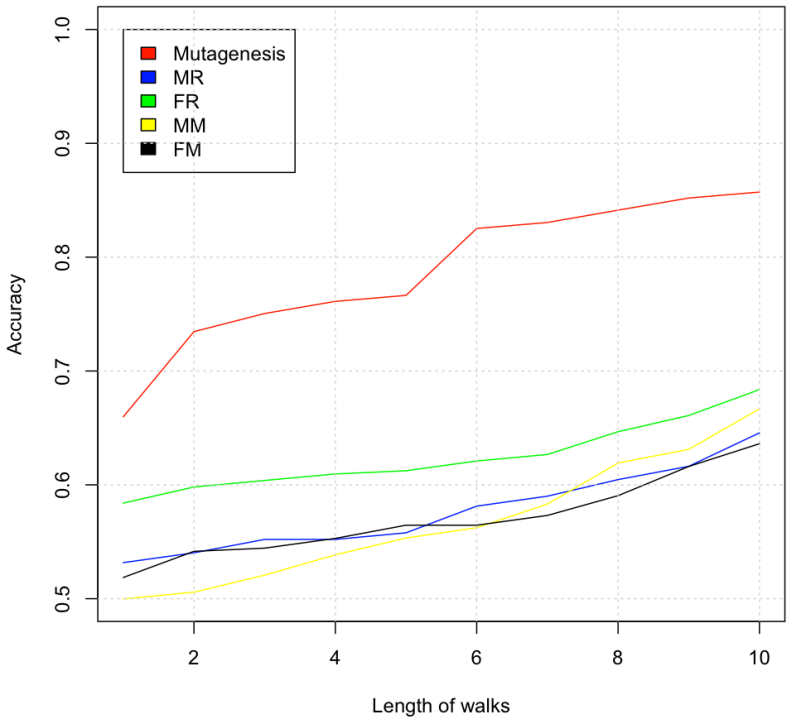


**Figure 65. Effect of the length of Walks on the Accuracy**

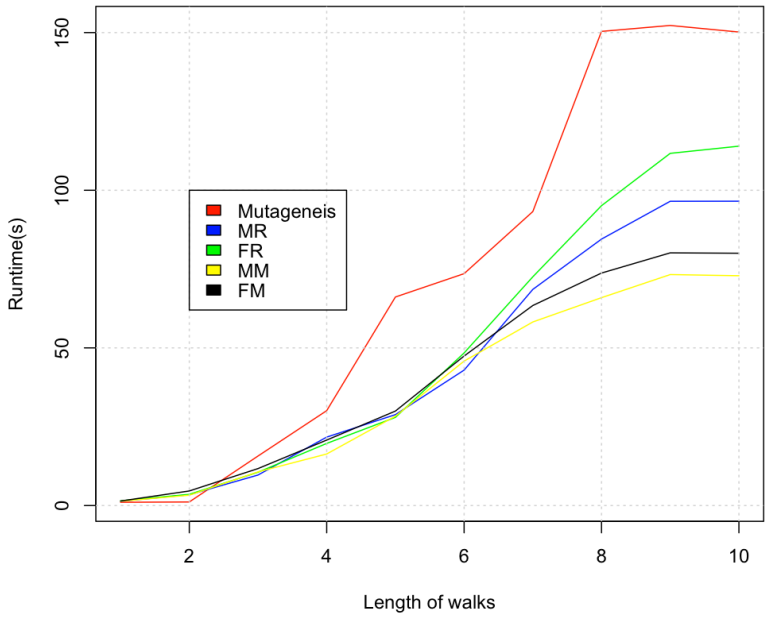


**Figure 66. Effect of the length of Walks on the Runtime**

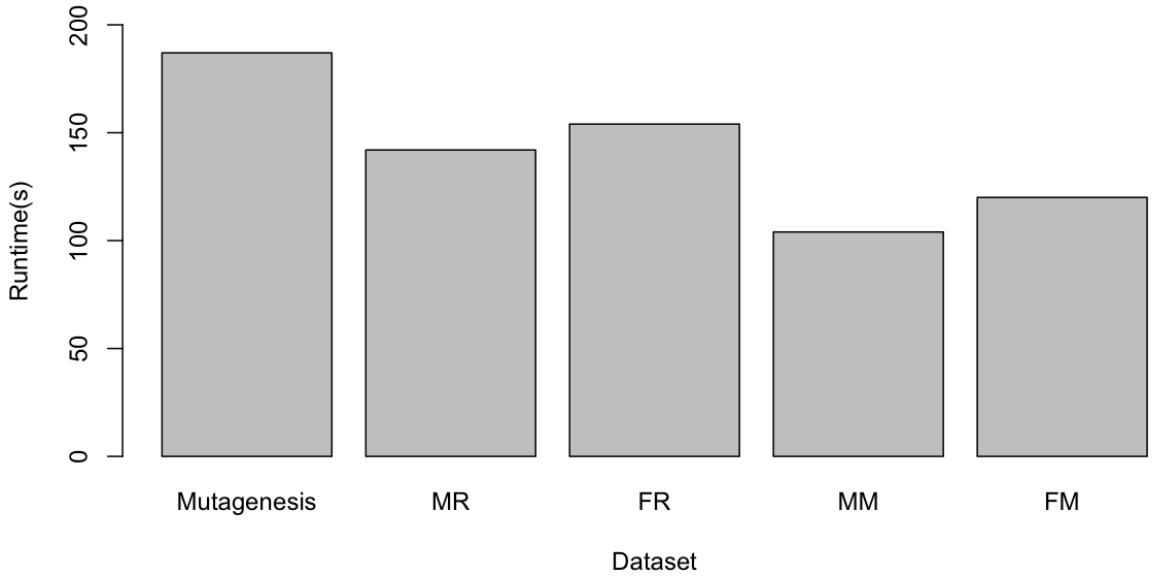


**Figure 67. Runtimes for Alternative Computation**

## 6.3 Discussion

The observation that the language of label sequences in a labeled graph is a regular language is important, as it can be the starting point for applying a number of results from automata theory to compute graph kernels.

# 7 Conclusions and Future Work

The conclusions of our work can be summarized as follows.

1. We developed a novel class of pruning mechanisms for searching the space of subgraph features. These pruning mechanisms are independent of the model building step and can be applied to a number of approaches to graph classification and regression. These pruning mechanisms drastically reduce the search space and reduce the computational time with no discernable loss of predictive accuracy.
2. We developed a novel algorithm for graph regression called gRegress. Our algorithm outperforms previously introduced algorithms on graph regression and represents the current state of the art in graph regression.
3. We developed an alternative approach for the computation of the Walk-based kernel for graph classification. Although this approach was not able to outperform the current approach for the computation, it is an interesting direction in the faster computation of graph kernels.
4. We demonstrated that certain problem domains with respect to graph regression, in a sense, require multiple linear models. Models which are at their core based on a single linear function perform poorly as compared to models based on combinations of linear models.
5. We demonstrate that in certain cases, incorporating structural features can drastically improve the performance of predictive models as compared to models based only on attribute valued features.

We plan to pursue the following as a part of our future work.

1. With regards to pruning mechanisms, the relationship between the $\alpha$, $\beta$ parameters and the size of the search space needs to be better characterized. We currently have a very coarse picture of how the search space varies with $\alpha$ and $\beta$. A better understanding of this relationship is central to the efficient extraction of subgraph features.
2. While we introduced pruning mechanisms for extracting one maximal set of features under the $\alpha$, $\beta$ constraints, there are many sets of subgraph features which satisfy these constraints. The quality (with respect to the regression model based on these features) of these sets needs to be further investigated.
3. The $\alpha$, $\beta$ parameters are user-defined parameters, and the user needs to select them appropriately. A significant improvement would be replacing these user-defined parameters with a parameter selection process. Relationships between the $\alpha$, $\beta$ parameters and the size of the search space could be exploited to perform this search efficiently.
4. The gRegress algorithm needs to be further tested on additional datasets. While we have demonstrated the need for a combination of linear models, results on additional data sets would strengthen our claim.
5. The formulation of graph kernels in terms of problems from automata theory is an important direction in developing new graph kernels. As we have established that the language of walks is regular and leveraged these results to facilitate faster computation, we can apply this idea to other features spaces such as trees to develop new graph kernels.

# Bibliography


[1] Holder, L.B., Discovering Substructure in Examples, DTIC Research Report ADA197778, 1988.

[2] Rissanen, J., Stochastic Complexity in Statistical Inquiry Theory, 1989, World Scientific Publishing Co., Inc., River Edge, NJ.

[3] Muggleton, S., Inductive Logic Programming, New Generation Computing, 1991, Pages:295-340.

[4] Quinlan, J.R., C4.5: Programs for Machine Learning, 1993, Morgan Kaufmann

[5] Cook, D.J., Holder, L.B., Substructure Discovery Using Minimum Description Length and Background Knowledge, Journal of Artificial Intelligence Research, Pages: 231-255, 1994.

[6] Srinivasan, A., Muggleton, S. H. and Sternberg, M. J. E., King, R. D., Theories for mutagenicity: a study in first-order and feature-based induction, Artificial Intelligence, 1996, Pages:277-299, Elsevier.

[7] Cheng, S. H. N., Wolf, R., Foundations of Inductive Logic Programming, 1997, Springer.

[8] Motoda, H., Yoshida, K., Machine learning techniques to make computers easier to use, Artificial Intelligence, 1998, Pages:295-321, Elsevier.

[9] Provost, F., Fawcett, T., Kohavi, R., The case against accuracy estimation for comparing induction algorithms, International Conference on Machine Learning, 1998, Pages:445-453.

[10] Chittimoori R. N., Holder L. B., Cook D. J., Applying the Subdue substructure discovery system to the chemical toxicity domain, FLAIRS, 1999, Pages:90-94.

[11] Joachims, T., Making large-scale support vector machine learning practical, 1999, MIT Press Cambridge, MA, USA.

[12] Joachims, T., SVMLight: Support Vector Machine, 1999, University of Dortmund.

[13] Witten, I.H., Frank, E. Trigg, L., Weka: Practical Machine Learning Tools and Techniques with Java Implementations, ICONIP/ANZIIS/ANNES, 1999, 192-196.

[14] Conte, L.L, Ailey, B., Hubbard, T. J. P, Brenner, S. E., Murzin A. G. Chothia C. SCOP: a Structural Classification of Proteins database, Nucleic Acids Research, 2000, Pages:275-259

[15] Blair, R.M., Fang, H., Branham, W.S., Hass, B.S., Dial, S.L., Moland, C.L., Tong, W., Shi, L., Perkins, R., Sheehan, D.M., The Estrogen Receptor Relative Binding Affinities of 188 Natural and Xenochemicals: Structural Diversity of Ligands, Toxicological Sciences, 2000, Pages:138-153, Soc Toxicology.

[16] Cristianini, N., Shawe-Taylor, J. An Introduction to Support Vector Machines, 2000, Cambridge University Press.

[17] Gonzalez, J.A., Holder, L.B., Cook, D.J., Graph Based Concept Learning, National Conference on Artificial Intelligence 2000, Pages: 1072-1072, AAAI Press.

[18] Huuskonen, J., Estimation of Aqueous Solubility for a Diverse Set of Organic Compounds Based on Molecular Topology, Journal of Chemical Information and Computer Sciences, 2000, Pages:773-777.

[19] Inokuchi, A., Washio, T., Motoda, H., An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, Lecture Notes in Computer Science, Pages:13-23, 2000, Springer.

[20] Fawcett, T., Using Rule Sets to Maximize ROC Performance, IEEE Conference on Data Mining, 2001, Pages:131-141, IEEE Computer Society.

[21] Helma, C., King, R. D., Kramer, S., Srinivasan, A., The Predictive Toxicology Challenge 2000-2001, Bioinformatics, 2001, Pages:107-108, Oxford University Press.

[22] Kuramochi, M. and Karypis, G., Frequent subgraph discovery, IEEE International Conference on Data Mining, 2001, Pages:313-320, IEEE Computer Society Washington, DC, USA.

[23] Cheng, J., Hatzis, C., Hayashi, H., Krogel, M. A., Morishita, S., Page, D., Sese, J., KDD Cup 2001 Report, SIGKDD Explorations, 2002, Pages:47-64.

[24] Branham, W.S., Dial, S.L., Moland, C.L., Hass, B.S., Blair, R.M., Fang, H., Shi, L., Tong, W., Perkins, R.G., Sheehan, D.M., Phytoestrogens and Mycoestrogens Bind to the Rat Uterine Estrogen Receptor, Journal of Nutrition, 2002, Pages:658-664, Am Soc Nutrition.

[25] Gonzalez, J. and Holder, L. and Cook, D., Graph-based relational concept learning, International Conference on Machine Learning, 2002.

[26] Matsuda, T., Motoda, H., Yoshida, T., Washio, T., Mining Patterns from Structured Data by Beam-Wise Graph-Based Induction, Discovery Science, 2002, Springer.

[27] Tong, W., Perkins, R., Fang, H., Hong, H., Xie, Q., Branham, SW, Sheehan, D., Anson, J., Development of quantitative structure-activity relationships (QSARs) and their use for priority setting in the testing strategy of endocrine disruptors, Regul Res Perspect, 2002.

[28] Yan, X., Han, J., gSpan: Graph-based substructure pattern mining, IEEE International Conference on Data Mining, Pages:721-731, 2002.

[29] Zaki, M.J., Efficiently mining frequent trees in a forest, Knowledge Discovery in Databases, Pages:71-80, 2002, ACM New York, NY, USA.

[30] Bergstrom, C.A.S., Norinder, U., Luthman, K., Artursson, P., Molecular Descriptors Influencing Melting Point and Their Role in Classification of Solid Drugs, Journal of Chemical Information and Computer Sciences, 2003, Pages:1177-1185.

[31] Inokuchi, A., Washio, T., Motoda, H., Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, Machine Learning, Pages:321-354, 2003, Springer.

[32] Gartner, T., Flach, P., Wrobel, S., On Graph Kernels: Hardness Results and Efficient Alternatives, Lecture Notes in Computer Science, 2003, Pages:129-143, Springer-Verlag.

[33] Gartner, T. and Flach, P. and Wrobel, S., On graph kernels: Hardness results and efficient alternatives, COLT 2003.

[34] Kashima, H., Tsuda, K., Inokuchi, A., Marginalized Kernels Between Labeled Graphs, International Conference on Machine Learning, 2003, Pages: 321-328.

[35] Washio, T., Motoda, H., State of the art of graph-based data mining, ACMSIGKDD Explorations Newsletter, Pages:59-68, 2003, ACM New York, NY, USA.

[36] Huan, J., Wang, W., Prins, J., Efficient mining of frequent subgraphs in the presence of isomorphism, IEEE International Conference on Data Mining, Pages:549-559, 2003.

[37] Ruckert, U. and Kramer, S., Generalized Version Space Trees, International Workshop on Knowledge Discovery in Inductive Databases, Page: 119-30, 2003.

[38] Yan, X., Han, J., CloseGraph: mining closed frequent graph patterns, Knowledge Discovery in Databases, 286-295, 2003, ACM Press New York, NY, USA.

[39] Delaney, J.S., ESOL: Estimating Aqueous Solubility Directly from Molecular Structure, Journal of Chemical Information and Computer Sciences, 2004, Pages: 1000-1005.

[40] Huan, J., Wang, W., Bandyopadhyay, D., Snoeyink, J., Prins, J., Tropsha, A., Mining protein family specific residue packing patterns from protein structure graphs, RECOMB, Pages:308-315, 2004, ACM, New York, NY, USA.

[41] Huan, J., Wang, W., Prins, J., Yang, J., SPIN: mining maximal frequent subgraphs from graph databases, Knowledge Discovery in Databases, 2004, Pages:581-586, ACM New York, NY, USA.

[42] Kuramochi, M., Karypis, G., An Efficient Algorithm for Discovering Frequent Subgraphs, IEEE Transactions of Knowledge and Data Engineering, Pages:1038-1051, 2004, IEEE Computer Society.

[43] Nijssen, S., Kok, J.N., A quickstart in frequent structure mining can make a difference, Knowledge Discovery in Databases, Pages:647-652, 2004, ACM New York, NY, USA.

[44] Ravikumar B., Eisman G., Weak minimization of DFA: an algorithm and applications, Theoretical Computer Science, 2004, Pages:113-133, Elsevier.

[45] Smola, A.J., Scholkopf, B., A tutorial on support vector regression, Statistics and Computing, Pages:199-222, 2004, Springer.

[46] Borgelt, C., On Canonical Forms for Frequent Graph Mining, MGTS, 2005, Springer.

[47] Kudo, T., Maeda, E., Matsumoto, Y., An application of boosting to graph classification, Advances in neural information processing systems, Pages:729-736, 2005.

[48] Worlein, M., Meinl, T., Fischer, I., Philippsen, M., A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston, Lecture Notes in Computer Science, Pages:392-400, 2005, Springer.

[49] Deshpande, M., Kuramochi, M., Wale, N., Karypis, G., Frequent Substructure-Based Approaches for Classifying Chemical Compounds, IEEE Transactions of Knowledge and Data Engineering, Pages:1036-1050, 2005, IEEE Computer Society.

[50] Ketkar, N. S., Holder L.B., Cook D. J., Qualitative Comparison of Graph-based and Logic-based Multi-Relational Data Mining: A Case Study, ACM KDD Workshop on Multi-relational Data Mining, 2005.

[51] Ketkar, N. S., Holder L. B., Cook, D.J., Shah R., Coble J., Subdue: Compression-based Frequent Pattern Discovery in Graph Data, ACM KDD Workshop on Open-Source Data Mining, 2005, ACM New York, NY, USA.

[52] Karthikeyan, M., Glen, R.C., Bender, A., General Melting Point Prediction Based on a Diverse Compound Data Set and Artificial Neural Networks, Journal of Chemical Information and Modeling, 2005, Pages:581-590.

[53] Cook, D.J., Holder, L.B., Mining Graph Data, 2006, John Wiley & Sons.

[54] Ketkar, N. S., Holder L.B., Cook D. J., Mining in the Proximity of Subgraphs, ACM KDD Workshop on Link Analysis: Dynamics and Statics of Large Networks, 2006.

[55] Freund, Y. and Schapire, R.E., Experiments with a new boosting algorithm, International Conference on Machine Learning, 1996, Pages:156-164.

[56] Horvath, T., Ramon, J., Wrobel, S., Frequent subgraph mining in outerplanar graphs, Knowledge Discovery in Databases, 2006, ACM New York, NY, USA

[57] Nguyen, P.C., Ohara, K., Mogi, A., Motoda, H., Washio, T., Constructing. Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction, PAKDD, 2006, 390-399, Springer.

[58] Nijssen, S., Kok, J.N., Frequent Subgraph Miners: Runtimes Don't Say Everything, MLG, 2006.

[59] Getoor, L., Taskar, B. Introduction to Statistical Relational Learning, 2007, MIT Press.

[60] Kuramochi, M., Karypis, G., Discovering frequent geometric subgraphs, Information Systems, 2007, Pages:1101-1120, Elsevier.

[61] Ke, Y., Cheng, J., Ng, W., Correlation search in graph databases, Knowledge Discovery in Databases, 2007, Pages:390-399, ACM Press New York, NY, USA.

[62] Hudelson, M., Ketkar N. K., Holder L. B., Carlson T.,, C. Peng, Waldher, B., Jones, J.,, High Confidence Predictions of Drug-Drug Interactions: Predicting Affinities for Cytochrome P450 2C9 with Multiple Computational Methods J. Med. Chem., 51, 3, 648 - 654, 2008.

[63] Hiroto. S., H., Kramer, N., Tsuda, K., Partial least squares regression for graph mining, Knowledge Discovery in Databases, Pages:578-586, 2008, ACM New York, NY, USA.

[64] Tilton J., Cook, D., Ketkar, N., The integration of graph based knowledge discovery with image segmentation hierarchies for data analysis, data mining and knowledge discovery, IEEE International Geoscience & Remote Sensing Symposium, 2008.

[65] Ketkar, N. S., Holder, L. B., Diane, D. J., Empirical Comparison of Graph Classification Algorithms CIDM, 2009.

[66] Ketkar, N. S., Holder, L. B., Diane, D. J., Computation of the Direct Product Kernel for Graph Classification CIDM, 2009.

[67] Ullmann, J. R. An Algorithm for Subgraph Isomorphism, Journal of the ACM, 1976.

[68] Eppstein, D., Subgraph isomorphism in planar graphs and related problems, Information and Computer Science, University of California, Irvine, 1994.

[69] Syslo, M., M, Subgraph Isomorphism Problem for Outerplanar Graphs, Journal of Theoretical Computer Science, 1982.

[70] Michael R. Garey and David S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.

[71] Messmer, BT and Bunke, H., Efficient subgraph isomorphism detection: A decomposition approach, IEEE TKDE 2000.

[72] Wang, Y.K. and Fan, K.C. and Horng, J.T., Genetic-based search for error-correcting graph isomorphism, IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1997.

[73] Isaacs, N.S., Physical Organic Chemistry, Longman Scientific & Technical, 1987.