Mach Learn (2009) 75: 69–89
DOI 10.1007/s10994-008-5089-z

gBoost: a mathematical programming approach to graph classification and regression

Hiroto Saigo · Sebastian Nowozin · Tadashi Kadowaki · Taku Kudo · Koji Tsuda

Received: 9 March 2007 / Revised: 2 October 2008 / Accepted: 9 October 2008 / Published online: 12 November 2008

Abstract Graph mining methods enumerate frequently appearing subgraph patterns, which can be used as features for subsequent classification or regression. However, frequent patterns are not necessarily informative for the given learning problem. We propose a mathematical programming boosting method (gBoost) that progressively collects informative patterns. Compared to AdaBoost, gBoost can build the prediction rule with fewer iterations. To apply the boosting method to graph data, a branch-and-bound pattern search algorithm is developed based on the DFS code tree. The constructed search space is reused in later iterations to minimize the computation time. Our method can learn more efficiently than the simpler method based on frequent substructure mining, because the output labels are used as an extra information source for pruning the search space. Furthermore, by engineering the mathematical program, a wide range of machine learning problems can be solved without modifying the pattern search algorithm.

Keywords Graph mining · Mathematical programming · Classification · Regression · QSAR

Editors: Thomas Gärtner and Gemma C. Garriga.

H. Saigo (✉) · S. Nowozin · K. Tsuda
Max Planck Institute for Biological Cybernetics, Spemannstrasse 38, 72076 Tübingen, Germany
e-mail: hiroto.saigo@tuebingen.mpg.de

Present address:
H. Saigo
Max Planck Institute for Informatics, Campus E1 4, 66123 Saarbrücken, Germany

T. Kadowaki
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan

Present address:
T. Kadowaki
Eisai Co., Ltd. 5-1-3, Tokodai, Tsukuba, Ibaraki 300-2638, Japan

T. Kudo
Google Japan Inc., Cerulean Tower 6F, 26-1 Sakuragaoka-cho, Shibuya-ku, Tokyo 150-8512, Japan

1 Introduction

Graphs are general and powerful data structures that can be used to represent diverse kinds of objects. Much of the real world data is represented not as vectors, but as graphs including sequences and trees, for example, biological sequences (Durbin et al. 1998), semi-structured texts such as HTML and XML (Abiteboul et al. 2000), RNA secondary structures (Hamada et al. 2006), and so forth. Probably graphs are most commonly used in chemoinformatics (Gasteiger and Engel 2003), where chemical compounds are represented as graphs and their bioactivities or chemical reactivities are predicted by a learning machine. When the activity is represented as a binary label, it is often called the Structure-Activity Relationship (SAR) analysis. On the other hand, if a real-valued label is used, it is called the Quantitative Structure-Activity Relationship (QSAR) analysis. To this aim, several off-the-shelf molecular property descriptors are developed (James et al. 2004; Duran et al. 2002), and used successfully with conventional learning machines such as the support vector machines (SVMs) (Schölkopf and Smola 2002).

However, in other situations such as protein (Borgwardt et al. 2006) or RNA (Hamada et al. 2006) graph mining, there are no graph descriptors available. Thus, it is a central issue how to build the interface between graph data and existing machine learning algorithms. One way is to design a pairwise similarity measure between two graphs. When the similarity function is proven to be positive definite, it is called a kernel function (Schölkopf and Smola 2002). Several kernels have been proposed so far (Kashima et al. 2003; Gärtner et al. 2003; Ralaivola et al. 2005; Fröhrich et al. 2006; Mahé et al. 2005; Mahé et al. 2006; Horváth et al. 2004). The basic idea of those kernel methods is to represent a graph as a very high dimensional space of indicators of substructures (i.e., walks or trees), and then compute the dot product between two feature vectors efficiently by a recursive algorithm. Therefore, the kernel methods take all substructures into account. This all-features property is not always beneficial. Especially, when only a few small substructures determine the output label completely, it is desirable to single out the substructures rather than taking other features into account. On the other hand, if the information is distributed over many substructures, it would be meaningless to try to select a few substructures. In drug discovery, it is desired to pick up important substructures to explain why the drug candidate (i.e., graph) is supposed to work as a drug. Thus, at least in this case, the prediction accuracy is not the only goal. It is required for humans to interpret the prediction rules. Prediction rules of kernel methods are certainly not easy to interpret, because the features are numerous and implicit.

Frequent substructure mining methods, such as AGM (Inokuchi 2005), Gaston (Nijssen and Kok 2004) or gSpan (Yan and Han 2002a), have been applied to enumerate frequently appearing subgraph patterns. Then, a graph is represented as a vector of binary indicators, and an existing machine learning method such as SVM (Schölkopf and Smola 2002) is applied to the vector. Especially when L1-SVM (Schölkopf and Smola 2002), LASSO (Tibshrani 1996) or related methods are used, one can identify a few salient substructures as well. Such an approach is commonly practiced in chemoinformatics (Helma et al. 2004; Kazius et al. 2006). We call it a "two step" approach, because the mining part (e.g., gSpan) and the learning part (e.g., SVM) are completely separated. However, Wale and Karypis (2006) recently argued that the two step approach is too time and memory consuming. To achieve the best accuracy in chemoinformatics data, one needs relatively large substructures up to 10 edges (James et al. 2004). It is harmful in accuracy to restrict the minimum support of substructures, because infrequent ones could also be crucially important for some graphs in the database.

In several cases (Morishita and Sese 2000; Takabayashi et al. 2006; Bringmann et al. 2006), the frequent mining is replaced with discriminative substructure mining. First, the

qualifying substructures are enumerated based on a statistical criterion such as information gain. Then, a machine learning method is applied to the feature space of substructure indicators. However, in principle, the salient patterns depend on the optimal solution of the subsequent learning problem. It is very difficult to theoretically guarantee that the statistical criterion provides good features for the subsequent learning algorithm. This is the problem of all filter methods. For related discussions, see (Kohavi and John 1997). Additionally, one has to redesign the statistical criterion to deal with a different machine learning problem. For example, classification and regression problems require different criteria and thus pattern search algorithms should be based on different pruning conditions.

Our method learns from graph data using mathematical programming. We employ LPBoost (Demiriz et al. 2002) as a base algorithm and combine it with a substructure mining algorithm. Due to the large number of possible patterns, our mathematical program has an intractable number of variables. To solve the large problem, the column generation technique (Luenberger 1969) is applied: A pattern feature is added in each iteration, and the restricted program of growing size is solved repeatedly. By observing the duality gap, we can monitor how close the current solution is to the optimal solution. This allows us to define the termination condition in a theoretically controlled way. In an iteration, a new pattern that optimizes a gain function is searched in the space of graphs. We employ a branch-and-bound search method to find it efficiently. As a canonical search space, we use the DFS (Depth First Search) code tree (Yan and Han 2002a). In each search, the optimal pattern is searched for based on the gain function with different parameters. In large datasets, we found it too time consuming to reconstruct the search space from scratch in each iteration. To alleviate this problem, the search space is stored in memory and progressively extended. In comparison to the naïve method that enumerates all the patterns first and solves the mathematical programming afterwards, our progressive method is more efficient in time and in the size of search space.

For numerical vectors, a number of mathematical programming-based methods (Boyd and Vandenberghe 2004) have been proposed and commonly used. Binary classification (Demiriz et al. 2002; Schölkopf and Smola 2002) and least squares regression (Tibshrani 1996) are among the most fundamental examples. We will show that, using the same pattern search algorithm, both classification and regression problems can be solved by engineering the mathematical program. It is regarded as an advantage over the filter methods (Bringmann et al. 2006) that our approach is easily extendable to a wide range of machine learning problems.

This paper is based on two previous conference papers (Kudo et al. 2005; Saigo et al. 2006). Kudo et al. (2005) proposed to combine AdaBoost and the optimal pattern search algorithm. However, AdaBoost is less efficient, because it takes significantly more iterations than our mathematical programming method. Empirical comparison will be reported in Sect. 6. The direct predecessor of this paper is the MLG'06 paper by Saigo et al. (2006) that includes a linear programming-based regression method for the QSAR analysis. This paper, however, has significantly more contents such as the empirical comparison to AdaBoost, the quadratic programming-based regression method, in-depth description of pattern search and so forth.

The rest of this paper is organized as follows: Sect. 2 introduces the binary graph classification method based on a linear program. In Sect. 2.1, the pattern search algorithm repeatedly called from the mathematical program is presented. In Sect. 3, our mathematical programming approach is applied to least squares regression. Sect. 4 shows the experimental results on chemoinformatics benchmark datasets. The experiments for investigating computational costs are summarized in Sect. 5. In Sect. 6, the difference between our method and AdaBoost is discussed. Section 7 concludes the paper with general discussions.

2 Graph classification with linear programming

In this paper, we deal with undirected, labeled and connected graphs. To be more precise, we define the graph and *subgraph isomorphism* as follows:

Definition 1 (Labeled connected graph) A labeled graph is represented in a 4-tuple $G = (V, E, \mathcal{L}, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\mathcal{L}$ is a set of labels, and $l : V \cup E \to \mathcal{L}$ is a mapping that assigns labels to the vertices and edges. A labeled connected graph is a labeled graph such that there is a path between any pair of vertices.

Definition 2 (Subgraph isomorphism) Let $G' = (V', E', \mathcal{L}', l')$ and $G = (V, E, \mathcal{L}, l)$ be labeled connected graphs. A graph G' is subgraph-isomorphic to a graph G ($G' \subseteq G$) iff there exists an injective function $\phi : V' \to V$, s.t., (1) $\forall v \in V, l(v) = l'(\phi(v))$, (2) $\forall (v_1, v_2) \in E, (\phi(v_1), \phi(v_2)) \in E'$ and (3) $l(v_1, v_2) = l'(\phi(v_1), \phi(v_2))$. If G' is a subgraph of G, then G is a supergraph of G'.

We start from defining the binary classification problem of graphs, and the regression problem will be discussed in Sect. 3. In graph classification, the task is to learn a prediction rule from the training examples $\{(G_n, y_n)\}_{n=1}^{\ell}$, where G_n is a training graph and $y_n \in \{+1, -1\}$ is the associated class label. Let $\mathcal{T}$ be the set of all patterns, i.e., the set of all subgraphs included in at least one training graph. Then, each graph G_n is encoded as a $|\mathcal{T}|$-dimensional vector $\boldsymbol{x}_n$,

$$x_{n,t} = I(t \subseteq G_n), \quad \forall t \in \mathcal{T},$$

where $I(\cdot)$ is 1 if the condition inside is true and 0 otherwise. This feature space is illustrated in Fig. 1.

Based on the binary representation of $x_{n,t}$, individual stumps, or *hypotheses* is defined as:

$$h(\boldsymbol{x}; t, \omega) = \omega(2x_t - 1),$$

where $\omega \in \Omega = \{-1, 1\}$ is a parameter. This parameter allows us to check not only the presence of subgraphs, but also the absence of subgraphs in training graphs.

Our prediction rule is a convex combination of simple classification stumps $h(\boldsymbol{x}; t, \omega)$, and has the form

$$f(\boldsymbol{x}) = \sum_{(t,\omega) \in \mathcal{T} \times \Omega} \alpha_{t,\omega} h(\boldsymbol{x}; t, \omega), \tag{1}$$

where $\alpha_{t,\omega}$ is a weight such that $\sum_{(t,\omega) \in \mathcal{T} \times \Omega} \alpha_{t,\omega} = 1$ and $\alpha_{t,\omega} \geq 0$.

This is a linear discriminant function in an intractably large dimensional space. To obtain an interpretable rule, we need to obtain a *sparse* weight vector $\boldsymbol{\alpha}$, where only a few weights

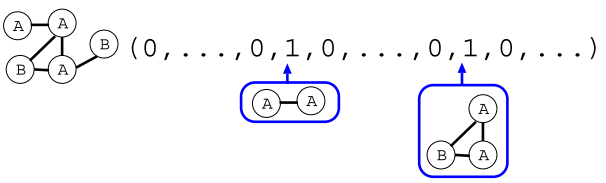

Fig. 1 Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators

are nonzero. In the following, we will present a linear programming approach for efficiently capturing patterns with non-zero weights.

To obtain a sparse weight vector, we use the formulation of LPBoost (Demiriz et al. 2002). Given the training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{\ell}$, the training problem is formulated as

$$\min_{\boldsymbol{\alpha},\boldsymbol{\xi},\rho} -\rho + D\sum_{n=1}^{\ell}\xi_n \tag{2}$$

$$\text{s.t.} \sum_{(t,\omega)\in\mathcal{T}\times\Omega} y_n\alpha_{t,\omega}h(\boldsymbol{x}_n;t,\omega) + \xi_n \geq \rho, \quad \xi_n \geq 0,\ n=1,\ldots,\ell, \tag{3}$$

$$\sum_{(t,\omega)\in\mathcal{T}\times\Omega} \alpha_{t,\omega} = 1, \quad \alpha_{t,\omega} \geq 0,$$

where ρ is the soft-margin, separating negative from positive examples, $D = \frac{1}{\nu\ell}$, $\nu \in (0,1)$ is a parameter controlling the cost of misclassification which has to be found using model selection techniques, such as cross-validation. Nevertheless, it is known that the optimal solution has the following ν-property:

Theorem 1 (Rätsch et al. 2002) *Assume that the solution of* (2) *satisfies* $\rho \geq 0$. *The following statements hold*:

1. ν *is an upperbound of the fraction of* margin errors, *i.e.*, *the examples with*

$$\sum_{(t,\omega)\in\mathcal{T}\times\Omega} y_n\alpha_{t,\omega}h(\boldsymbol{x}_n;t,\omega) < \rho.$$

2. ν *is a lowerbound of the fraction of the examples such that*

$$\sum_{(t,\omega)\in\mathcal{T}\times\Omega} y_n\alpha_{t,\omega}h(\boldsymbol{x}_n;t,\omega) \leq \rho.$$

Directly solving this optimization problem is intractable due to the large number of variables in $\boldsymbol{\alpha}$. So we solve the following *equivalent* dual problem instead.

$$\min_{\boldsymbol{\lambda},\gamma} \gamma \tag{4}$$

$$\text{s.t.} \sum_{n=1}^{\ell} \lambda_n y_n h(\boldsymbol{x}_n;t,\omega) \leq \gamma, \quad \forall (t,\omega)\in\mathcal{T}\times\Omega, \tag{5}$$

$$\sum_{n=1}^{\ell}\lambda_n = 1, \quad 0 \leq \lambda_n \leq D,\ n=1,\ldots,\ell.$$

After solving the dual problem, the primal solution $\boldsymbol{\alpha}$ is obtained from the Lagrange multipliers (Demiriz et al. 2002).

The dual problem has a limited number of variables, but a huge number of constraints. Such a linear program can be solved by the *column generation* technique (Luenberger 1969): Starting with an empty pattern set, the pattern whose corresponding constraint is violated the most is identified and added iteratively. Each time a pattern is added, the optimal solution is updated by solving the restricted dual problem. Denote by $\boldsymbol{\lambda}^{(k)}, \gamma^{(k)}$ the optimal solution of

the restricted problem at iteration $k = 0, 1, \ldots$, and denote by $\mathcal{H}^{(k)} \subseteq \mathcal{T} \times \Omega$ the hypothesis set at iteration k. Initially, $\mathcal{H}^{(0)}$ is empty and $\lambda_n^{(0)} = 1/\ell$. The restricted problem is defined by replacing the set of constraints (5) with

$$\sum_{n=1}^{l} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \omega) \leq \gamma, \quad \forall (t, \omega) \in \mathcal{H}^{(k)}.$$

After solving the problem, the hypothesis set $\mathcal{H}^{(k)}$ is updated to $\mathcal{H}^{(k+1)}$ by adding a hypothesis. Several criteria have been proposed to select the new hypothesis (du Merle et al. 1999), but we adopt the most simple rule that is amenable to graph mining: We select the hypothesis which violates the constraint with the largest margin.

$$(t^*, \omega^*) = \operatorname*{argmax}_{t \in \mathcal{T}, \omega \in \Omega} \sum_{n=1}^{\ell} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \omega). \tag{6}$$

The hypothesis set is updated as $\mathcal{H}^{(k+1)} = \mathcal{H}^{(k)} \cup \{(t^*, \omega^*)\}$. In the next section, we discuss how to efficiently find the optimal hypothesis in detail.

One of the big advantages of our method is that we have a stopping criterion that guarantees that the optimal solution of (2) is found: If there is no $(t, \omega) \in \mathcal{T} \times \Omega$ such that

$$\sum_{n=1}^{\ell} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \omega) > \gamma^{(k)}, \tag{7}$$

then the current solution is the optimal dual solution. Empirically, the patterns found in the last few iterations have negligibly small weights. The number of iterations can be decreased by relaxing the condition as

$$\sum_{n=1}^{\ell} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \omega) > \gamma^{(k)} + \epsilon. \tag{8}$$

Let us define the primal objective function as $V = -\rho + D \sum_{n=1}^{\ell} \xi_n$. Due to the convex duality, we can guarantee that, for the solution obtained from the early termination (8), the objective satisfies $V \leq V^* + \epsilon$, where V^* is the optimal value with the exact termination (7) (Demiriz et al. 2002). In our experiments, $\epsilon = 0.01$ is always used.

2.1 Optimal pattern search

Our search strategy is a branch-and-bound algorithm that requires a canonical search space in which a whole set of patterns are enumerated without duplication. As the search space, we adopt the DFS (depth first search) code tree (Yan and Han 2002a). The basic idea of the DFS code tree is to organize patterns as a tree, where a child node has a supergraph of the parent's pattern (Fig. 2). A pattern is represented as a text string called the DFS code. The patterns are enumerated by generating the tree from the root to leaves using a recursive algorithm. To avoid duplications, node generation is systematically done by rightmost extensions. See Appendix for details about the DFS code and the rightmost extension.

All embeddings of a pattern in the graphs $\{G_n\}_{n=1}^{\ell}$ are maintained in each node. If a pattern matches a graph in different ways, all such embeddings are stored. When a new pattern is created by adding an edge, it is not necessary to perform full isomorphism checks with

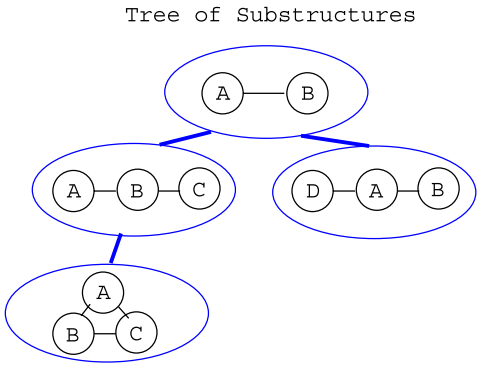

Fig. 2 Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree). To find the optimal pattern efficiently, the tree is systematically expanded by rightmost extensions

respect to all graphs in the database. A new list of embeddings are made by extending the embeddings of the parent (Yan and Han 2002a). Technically, it is necessary to devise a data structure such that the embeddings are stored incrementally, because it takes a prohibitive amount of memory to keep all embeddings independently in each node.

As mentioned in (6), our aim is to find the optimal hypothesis that maximizes the gain $g(t,\omega)$.

$$g(t,\omega)=\sum_{n=1}^{\ell}\lambda_n^{(k)}y_n h(\boldsymbol{x}_n;t,\omega). \tag{9}$$

For efficient search, it is important to minimize the size of the search space. To this aim, *tree pruning* is crucially important: Suppose the search tree is generated up to the pattern t and denote by g^* the maximum gain among the ones observed so far. If it is guaranteed that the gain of any supergraph t' is not larger than g^*, we can avoid the generation of downstream nodes without losing the optimal pattern. We employ the following pruning condition.

Theorem 2 (Morishita 2001; Kudo et al. 2005) *Let us define*

$$\mu(t)=\max\left\{2\sum_{\{n|y_n=+1,t\subseteq G_n\}}\lambda_n^{(k)}-\sum_{n=1}^{\ell}y_n\lambda_n^{(k)},\,2\sum_{\{n|y_n=-1,t\subseteq G_n\}}\lambda_n^{(k)}+\sum_{n=1}^{\ell}y_n\lambda_n^{(k)}\right\}.$$

If the following condition is satisfied,

$$g^*>\mu(t), \tag{10}$$

the inequality $g(t',\omega')<g^*$ holds for any t' such that $t\subseteq t'$ and any $\omega'\in\Omega$.

Proof By definition,

$$g(t',\omega')=\sum_{n=1}^{\ell}\lambda_n^{(k)}y_n\omega'(2I(t'\subseteq G_n)-1).$$

If we focus on the case $\omega'=+1$, then

$$g(t',+1)=2\sum_{\{n|t'\subseteq G_n\}}y_n\lambda_n^{(k)}-\sum_{n=1}^{\ell}y_n\lambda_n^{(k)}$$

$$\leq 2 \sum_{\{n|y_n=+1, t' \subseteq G_n\}} \lambda_n^{(k)} - \sum_{n=1}^{\ell} y_n \lambda_n^{(k)}$$

$$\leq 2 \sum_{\{n|y_n=+1, t \subseteq G_n\}} \lambda_n^{(k)} - \sum_{n=1}^{\ell} y_n \lambda_n^{(k)}.$$

The second inequality follows from the fact that

$$\{n|y_n = +1, t' \subseteq G_n\} \subseteq \{n|y_n = +1, t \subseteq G_n\}.$$

Similarly,

$$g(t', -1) \leq 2 \sum_{\{n|y_n=-1, t \subseteq G_n\}} \lambda_n^{(k)} + \sum_{n=1}^{\ell} y_n \lambda_n^{(k)}.$$

Therefore, $\mu(t)$ is an upperbound of $g(t', +1)$ and $g(t', -1)$. If the current maximum gain g^* is more than $\mu(t)$, it is guaranteed that there is no downstream pattern whose gain is larger than g^*. □

The gBoost algorithm is summarized in Algorithms 1 and 2. Algorithm 1 is exactly the same as LPBoost (Demiriz et al. 2002), so it does not contain anything novel. Our contribution is to combine it with the pattern search algorithm.

2.2 Reusing the search space

Our method calls the pattern search algorithm repeatedly with different parameters $\boldsymbol{\lambda}^{(k)}$. In each iteration, the search tree is generated until the pruning condition is satisfied. Creating a new node is time consuming, because the list of embeddings is updated, and the minimality of the DFS code has to be checked (see Appendix). In our previous paper (Saigo et al. 2006), the search tree is erased after the optimal pattern is found, and a new search tree is built from scratch in the next iteration. In this paper, we maintain the whole search tree, including all embeddings, in the main memory for better efficiency. Then, node creation is necessary only if it is not created in previous iterations. Naturally this strategy requires more memory, but we did not experience any overflow problems in our experiments with 8 GB memory.

Algorithm 1 gBoost algorithm: main part

```
1: H^(0) = ∅, λ_n^(0) = 1/ℓ, k = 0
2: loop
3:     Find the optimal hypothesis (t*, ω*) based on λ^(k)        ▷ Algorithm 2
4:     if termination condition (8) holds then
5:         break
6:     end if
7:     H^(k+1) = H^(k) ∪ {(t*, ω*)}
8:     Solve the restricted dual problem (4) to obtain λ^(k+1)
9:     k = k + 1
10: end loop
```

Algorithm 2 Finding the Optimal Pattern

```
1: procedure OPTIMAL PATTERN
2:     Global variables: g*, ω*, t*
3:     g* = −∞
4:     for t ∈ DFS codes with single nodes do
5:         project(t)
6:     end for
7:     return (t*, ω*)
8: end procedure
9: function PROJECT(t)
10:    if t is not a minimum DFS code then
11:        return
12:    end if
13:    if pruning condition (10) holds then                ▷ Theorem 2
14:        return
15:    end if
16:    if g(t, ω) > g* for any ω ∈ Ω then
17:        g* = g(t, ω), t* = t, ω* = ω
18:    end if
19:    for t′ ∈ rightmost extensions of t do
20:        project(t′)
21:    end for
22: end function
```

3 Extension to least squares regression

One merit of our mathematical programming-based approach is that a wide range of machine learning problems are solved based on the same pattern search. In this section, we particularly focus on least squares regression. It is possible to apply our approach to, e.g., one-class classification (Rätsch et al. 2002), multi-class classification (Freund and Schapire 1997), hierarchical classification (Cai and Hofmann 2004), 1.5-class classification (Yuan and Casasent 2003) and knowledge-based support vector machines (Le et al. 2006). Mathematical programs are commonly used in machine learning, so certainly there are more applications.

Suppose we are given a training data set $\{(G_n, y_n)\}_{n=1}^{\ell}$, but now y_n may take on any real value. We use the same definition for a hypothesis $h(\boldsymbol{x}, t, \omega)$ and its corresponding weight $\alpha_{t,\omega}$ as those in the classification case. The regression function is defined as

$$f(\boldsymbol{x}) = \sum_{(t,\omega)\in\mathcal{T}\times\Omega} \alpha_{t,\omega} h(\boldsymbol{x}; t, \omega) + b,$$

where b is a newly introduced bias term. The learning problem is written as

$$\min_{\boldsymbol{\alpha}, b} \; C \sum_{(t,\omega)\in\mathcal{T}\times\Omega} |\alpha_{t,\omega}| + \frac{1}{2}\sum_{n=1}^{l}\Bigg(\sum_{(t,\omega)\in\mathcal{T}\times\Omega} \alpha_{t,\omega} h(\boldsymbol{x}_n; t, \omega) + b - y_n \Bigg)^2.$$

Note that we introduced the L1-norm regularizer to enforce sparsity to parameter vectors. This is exactly the same learning problem as that of LASSO (Tibshrani 1996). The learning problem above translates to the following quadratic program.

$$\min_{\boldsymbol{\alpha},\xi,b} C \sum_{(t,\omega)\in\mathcal{T}\times\Omega} (\alpha^+_{t,\omega} + \alpha^-_{t,\omega}) + \frac{1}{2}\sum_{n=1}^{\ell} \xi_n^2 \tag{11}$$

$$\text{s.t.} \quad \sum_{(t,\omega)\in\mathcal{T}\times\Omega} \alpha_{t,\omega} h(\boldsymbol{x}_n; t, \omega) + b - y_n \le \xi_n, \quad n = 1, \ldots, \ell, \tag{12}$$

$$y_n - \sum_{(t,\omega)\in\mathcal{T}\times\Omega} \alpha_{t,\omega} h(\boldsymbol{x}_n; t, \omega) - b \le \xi_n, \quad n = 1, \ldots, \ell, \tag{13}$$

$$\alpha^+_{t,\omega}, \alpha^-_{t,\omega} \ge 0, \quad \forall (t,\omega) \in \mathcal{T}\times\Omega, \tag{14}$$

where ξ_n is a slack variable, $\alpha_{t,\omega} = \alpha^+_{t,\omega} - \alpha^-_{t,\omega}$. The dual problem is described as

$$\min_{\boldsymbol{u}} \frac{1}{2}\sum_{n=1}^{\ell} (u_n^+ + u_n^-)^2 - \sum_{n=1}^{\ell} y_n (u_n^+ - u_n^-) \tag{15}$$

$$\text{s.t.} \quad -C \le \sum_{n=1}^{\ell} (u_n^+ - u_n^-) h(\boldsymbol{x}_n; t, \omega) \le C, \quad \forall (t,\omega) \in \mathcal{T}\times\Omega, \tag{16}$$

$$\sum_{n=1}^{\ell} u_n^+ - u_n^- = 0, \tag{17}$$

$$u_n^+,\ u_n^- \ge 0, \quad n = 1, \ldots, \ell. \tag{18}$$

Unlike the classification case, the dual constraint (16) is two-sided. Therefore, the gain function for regression has a slightly different form:

$$g_{reg}(t,\omega) = \left| \sum_{n=1}^{\ell} u_n^{(k)} h(\boldsymbol{x}_n; t, \omega) \right|.$$

However, we can still use the pruning condition (Theorem 2), because the same proposition holds for g_{reg}.

4 Experiments

In this section, our method is benchmarked with publicly available chemical compound datasets in classification and regression problems.

Table 1 Datasets Summary. Mutag, CPDB, CAS and AIDS are classification problems and EDKB-AR, EDKB-ER, EDKB-ES are regression problems, respectively. The number of positive data (POS) and negative data (NEG) are only provided for classification datasets. Average number of atoms (ATOM) and bonds (BOND) are shown for each dataset

Classification	ALL	POS	NEG	ATOM	BOND
Mutag	188	125	63	45.1	47.1
CPDB	684	341	343	14.1	14.6
CAS	4337	2401	1936	29.9	30.9
AIDS (CAvsCM)	1503	422	1081	58.9	61.4
Regression	**ALL**			**ATOM**	**BOND**
EDKB-AR	146			19.5	21.1
EDKB-ER	131			19.2	20.7
EDKB-ES	59			18.2	19.7

4.1 Classification (SAR analyses)

For classification, we used three mutagenicity datasets, Mutag,[1] CPDB,[2] CAS[3] and the AIDS antiviral screen dataset.[4] The statistics of the datasets are summarized in Table 1. We compared our method (gBoost) with marginalized graph kernel (MGK) (Kashima et al. 2003) and SVM with frequent mining (freqSVM) in 10-fold cross validation experiments. In FreqSVM, the frequent patterns are mined first, and then SVM is applied to the feature space created by the patterns (Helma et al. 2004; Kazius et al. 2006; Wale and Karypis 2006). These three methods are implemented by ourselves. In addition, we quote the 10 fold cross validation results by Gaston (Kazius et al. 2006), Correlated Pattern Mining (CPM) (Bringmann et al. 2006), and MOLFEA (Helma et al. 2004) from respective papers. The quoted accuracies are all based on 10-fold cross validation. In literature (Kazius et al. 2006; Helma et al. 2004; Bringmann et al. 2006), the accuracies of Gaston, CPM, and MOLFEA are shown for all possible regularization parameters. For each method, the best test accuracy is taken. Notice that they might not be compared with our results precisely, because experimental settings are slightly different. Please see the end of this section for details.

For gBoost, the maximum pattern size (maxpat), which in our case corresponds to the maximum number of nodes in a subgraph, was constrained up to 10, and we did not use the minimum support constraint at all. The regularization parameter ν is chosen from {0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6}. We used chemcpp[5] for computing the MGK. For MGK, the termination probability was chosen from $\{0.1, 0.2, \ldots, 0.9\}$, and the C parameter of SVM was chosen from {0.1, 1, 10, 100, 1000, 10000}. For freqSVM, frequent patterns are first mined by gSpan with maximum pattern size 10 and minimum support threshold 1% for Mutag and CPDB, 10% for CAS and AIDS. Then SVM was trained with C parameter from the range {0.1, 1, 10, 100, 1000, 10000}. To compare with the quoted results as fairly as possible, the best test accuracy is shown for all methods.

Table 2 summarizes the results. Overall, gBoost was competitive among the other state-of-the-art methods. In Mutag, CAS and CPDB, gBoost was the best method, but in AIDS, CPM performed best in accuracy and freq-SVM performed best in AUC. Notice that CPM's

[1] http://www.predictive-toxicology.org.

[2] Available from the supplementary information of (Helma et al. 2004).

[3] http://www.cheminformatics.org/datasets/bursi/.

[4] http://dtp.nci.nih.gov/docs/aids/aids_screen.html.

[5] http://chemcpp.sourceforge.net/html/index.html.

Table 2 Classification performance obtained by 10-fold cross validation in the classification datasets measured by the accuracy (ACC) and the area under the ROC curve (AUC). We obtained the results of MGK, freqSVM and gBoost from our implementations, but the other results are quoted from the literature. The best results are highlighted in bold fonts

Method	Mutag		CAS		CPDB		AIDS (CAvsCM)	
	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC
Gaston (Kazius et al. 2006)	–	–	0.79	–	–	–	–	–
MOLFEA (Helma et al. 2004)	–	–	–	–	0.785	–	–	–
CPM (Bringmann et al. 2006)	–	–	0.801	–	0.760	–	**0.832**	–
MGK	0.808	0.901	0.771	0.763	0.765	0.756	0.762	0.760
freqSVM	0.808	0.906	0.773	0.843	0.778	0.845	0.782	**0.808**
gBoost	**0.852**	**0.926**	**0.825**	**0.889**	**0.788**	**0.854**	0.802	0.774

good result on AIDS is based on sequence patterns. When subgraph patterns are used, the best result was 0.767. The relatively poor result of gBoost in AIDS in comparison with freqSVM could be attributed to misselection of features by the L1 regularizer. It is known that the L1 regularizer selects too few features occasionally (Zou and Hastie 2005). One way to weaken sparsity is to introduce an L2 regularizer in addition to the L1 regularizer like the elastic net (Zou and Hastie 2005). The computation time of gBoost is decomposed into *mining time* and *LP time*. The former is used for expanding and traversing the pattern space, and the latter is to solve the series of restricted dual problems. For CAS, the mining time was 1370 seconds and the LP time was 1110 seconds, respectively, on a standard PC with AMD Opteron 2.2 GHz and 8 GB memory. The computation time can be reduced in several ways: (1) restricting the pattern to simpler ones, e.g., walks or trees, (2) limiting the pattern set a priori, e.g., by the correlation with class labels (Bringmann et al. 2006) or by the minimum support constraints (Kazius et al. 2006). However, in any case, informative patterns might be lost in exchange for better efficiency.

The top 20 discriminative subgraphs for CPDB are displayed in Fig. 3. We found that the top 3 substructures with positive weights (0.0672,0.0656, 0.0577) correspond to known *toxicophores* (Kazius et al. 2006). They correspond to *aromatic amine*, *aliphatic halide*, and *three-membered heterocycle*, respectively. In addition, the patterns with weights 0.0431, 0.0412, 0.0411 and 0.0318 seem to be related to *polycyclic aromatic systems*.

To characterize the influence of the regularization parameter ν, gBoost is applied to CPDB with $\nu = \{0.1, \ldots, 0.8\}$. Table 4 shows the number of "active patterns", the size of the searched space, the number of iterations until convergence, the margin, and the training accuracy. Here, active patterns are defined as those with non-zero weights,

$$\mathcal{A} = \left\{ t \in \mathcal{T} \mid \alpha_{t,\omega} \neq 0, \omega \in \{-1, 1\} \right\}.$$

For this experiment, we set the maximum pattern size to 10. When ν is low, gBoost creates a complex classification rule with many active patterns so that it can classify the training examples completely. As ν is increased, the regularization takes effect and the rule gets simpler with a smaller number of active patterns. At the same time, the active patterns become smaller in size (Fig. 4).

Experimental settings of quoted results We quoted the accuracies of MOLFEA, Gaston and CPM reported in literature in Table 2. In the following, we briefly summarize the experimental settings of these results.

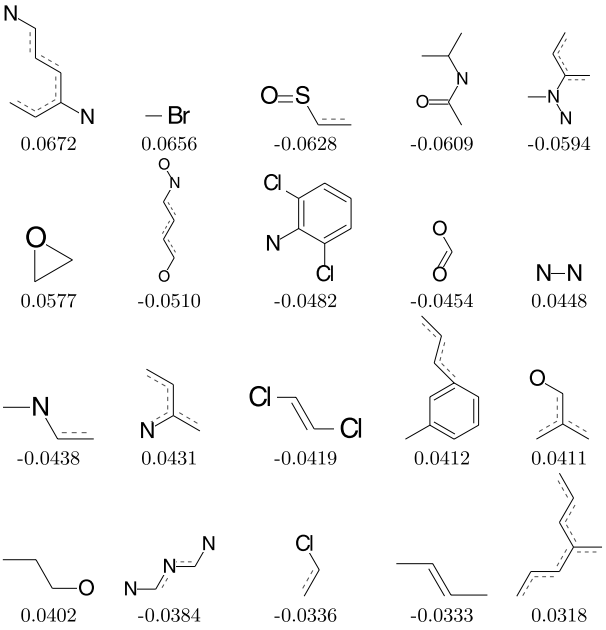

Fig. 3 Top 20 discriminative subgraphs from the CPDB dataset. Each subgraph is shown with the corresponding weight, and ordered by the absolute value from the *top left* to the *bottom right*. H atom is omitted, and C atom is represented as a dot for simplicity. Aromatic bonds appeared in an open form are displayed by the combination of *dashed* and *solid lines*

Table 3 Regression performance obtained by leave-one-out cross validation in three assays from the EDKB evaluated by mean absolute error (MAE) and Q^2. Note that for MAE, lower values indicate better prediction, which is vice versa for Q^2. We obtained the results of MGK, freqSVM and gBoost from our implementations, but the other results are quoted from the literature. The best results are highlighted in bold fonts

	Measure	CoMFA (Hong et al. 2003 Shi et al. 2001)	MGK	freqSVM	gBoost
EDKB-AR	MAE	–	0.229	0.193	**0.183**
	Q^2	0.571	0.346	0.465	**0.621**
EDKB-ER	MAE	–	0.320	0.268	0.263
	Q^2	**0.660**	0.267	0.532	0.541
EDKB-ES	MAE	–	0.322	0.248	**0.216**
	Q^2	–	0.522	0.588	**0.753**

MOLFEA Helma et al. (2004) used MOLFEA with four different minimum support thresholds {1%, 3%, 5%, 10%} without size constraints to generate path features. Four different machine learning algorithms are subsequently applied in 10-fold cross validation settings. Used algorithms are decision tree C4.5 (Quinlan 1993), PART rule learner (Frank and Witten 1998), SVM with the linear kernel and the second order polynomial kernel. At minimum support 5%, a linear SVM achieved the best accuracy 0.785 in 10-fold cross validation (Table 3 in Helma et al. 2004).

Gaston Kazius et al. (2006) used Gaston to find substructures of arbitrary size which appeared in more than 70 transactions. For classification, a series of decision trees is employed,

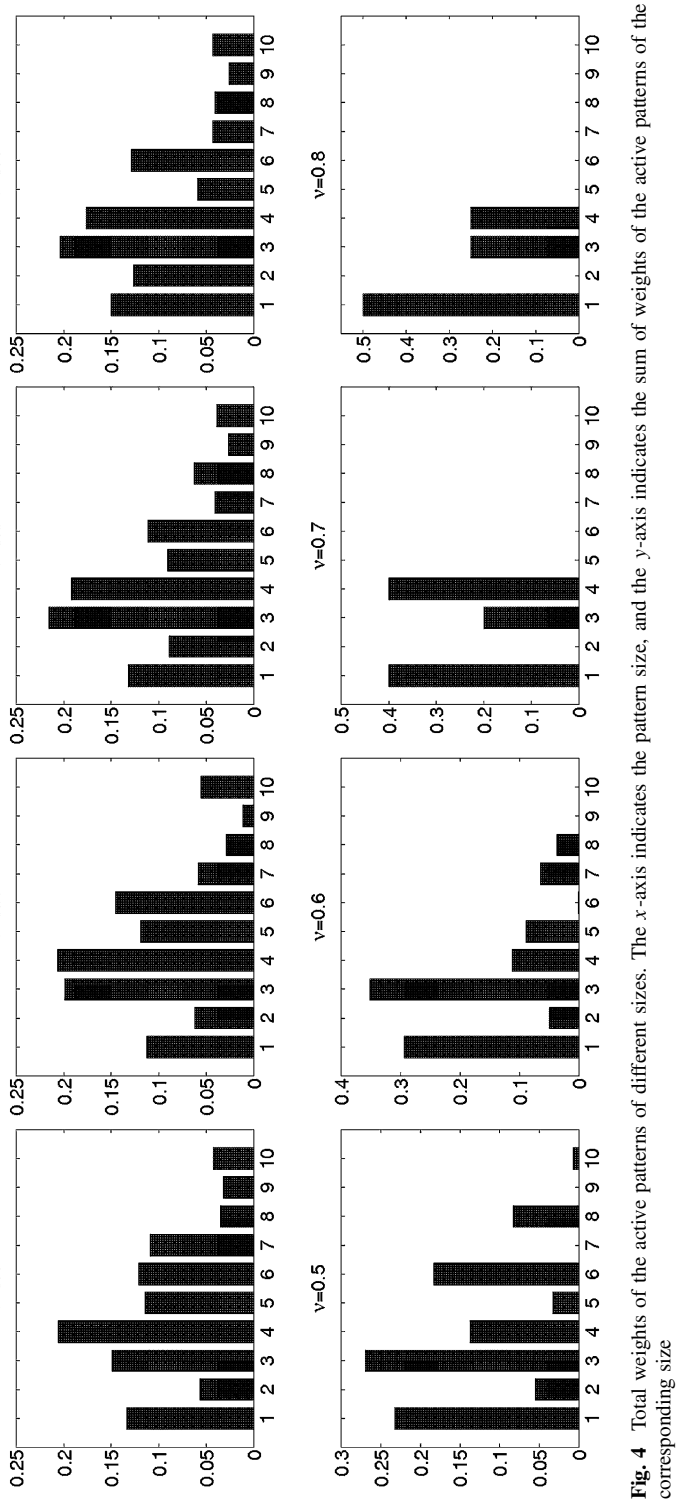

Fig. 4 Total weights of the active patterns of different sizes. The x-axis indicates the pattern size, and the y-axis indicates the sum of weights of the active patterns of the corresponding size

Table 4 Influence of the choice of ν parameter. P: the number of active patterns, T: the size of the tree-shaped search space, ITR: the number of iterations, ρ: the margin in (2), TrACC: the classification accuracy in the training set

ν	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
P	317	225	173	106	63	37	19	7
T	21766	21825	19083	16026	11330	6723	2753	1754
ITR	328	235	178	110	65	39	21	9
ρ	0.00859	0.0116	0.0200	0.0340	0.0695	0.121	0.4	0.6
TrACC	0.999	0.965	0.924	0.883	0.838	0.803	0.712	0.709

each of which corresponds to a subgraph pattern. The authors measured the 10-fold cross validation error, and the best accuracy was 0.79 (Table 3 in Kazius et al. 2006).

CPM In correlated pattern mining (CPM) by Bringmann et al. (2006), different pattern languages are compared to investigate the trade-off between the expressiveness and the performance. As a pattern language, (i) itemset, (ii) multi-itemset, (iii) sequence, (iv) tree and (v) graph are considered. For each pattern language, top 1, 10, 100, 1000 correlated features are mined according to the χ^2 measure. Then the performances of these features are tested by four algorithms in 10-fold cross validation. The algorithms are C4.5 (Quinlan 1993), SVM, Naive Cohen (1995). We selected the best accuracies from Table 4 and 5 of (Bringmann et al. 2006). SVM with sequence patterns achieved the best accuracy both in CAS and CPDB, leading to the accuracy 0.801 and 0.760, respectively. In AIDS, C4.5 with sequence patterns was the best (0.832).

4.2 Regression (QSAR analyses)

We used three datasets (AR, ER and ES) from Endocrine Disruptors Knowledge Base (EDKB).[6] A summary of the datasets in shown in Table 1. We compared gBoost with MGK and Comparative Molecular Field Analysis (CoMFA) (Hong et al. 2003; Shi et al. 2001). We evaluated the performance of MGK, freqSVM and gBoost with our own implementations, and quoted the reported performance of CoMFA from the literature. For gBoost, the regularization parameter C is chosen from $\{0.1, 1, 10, 100, 1000, 10000\}$. For freqSVM, minimum support threshold is set to 1, and the regularization parameter C is chosen from $\{0.1, 1, 10, 100, 1000, 10000\}$. For MGK, the termination probability is chosen from $\{0.1, \ldots, 0.9\}$, then obtained kernel is fed into SVR where the tube parameter ϵ is set to 0.1, the regularization parameter C is chosen from $\{0.1, 1, 10, 100, 1000, 10000\}$. gBoost performed best when setting C parameter to 100 for all three datasets, and built regressor consisting of 36, 50 and 30 patterns on average for AR, ER and ES, respectively.

Table 3 summarizes the experimental results. The regression models are evaluated by the mean absolute error (MAE) and the Q^2 score:

$$Q^2 = 1 - \frac{\sum_{n=1}^{l}(y_n - f(x_n))^2}{\sum_{n=1}^{l}(y_n - \frac{1}{l}\sum_{n=1}^{l} y_n)^2}. \tag{19}$$

Our method (gBoost) performed constantly better than MGK and freqSVM, and better than CoMFA in AR. On the other hand, in ER, CoMFA was superior to other methods.

[6] http://edkb.fda.gov/databasedoor.html.

In CoMFA, all training compounds are aligned to a template compound, which is manually picked up by an expert. Then, each compound is encoded into a high dimensional feature vector describing the steric and electrostatic intersections with the template compound. Thus CoMFA is not a fully automatic method, and it assumes that training compounds are close to each other. The result in ER suggests that the energy features are useful for learning, but at the same time, it seems difficult to incorporate such features into our framework without experts' intervention.

5 Computational costs

Our method tightly couples the mathematical programming and graph mining. However, the same prediction rule can be obtained by the following "naïve" method:

1. The feature space is completely constructed by frequent substructure mining.
2. The mathematical program is solved by column generation.

To motivate the use of our method, it is essential to empirically show that our method is more efficient in time. Certainly our method has a smaller search space due to the pruning condition, but the question is how much the pruning condition can contribute in reducing the computation time in real datasets.

We compared the computational time and the test set accuracy of the naïve method and the proposed progressive method. Those two methods produce exactly the same series of reduced linear programs. So we subtracted the time needed to solve the linear programs from the total computational time. The remaining time (i.e., *mining time*) is for constructing and traversing the search space. The results for a range of maxpat constraints in the CPDB dataset are summarized in Figs. 5 and 6. In this experiment, no minimum support constraints are employed. The test accuracy reaches the highest level around $maxpat = 4$, and already at this point, the mining time shows substantial difference. Notice that the time is plotted in the log-scale. Even in the case that small patterns can achieve good accuracy, it makes sense to explore larger patterns for better interpretability. In fact, the toxicophore contains many patterns having more than 4 edges (Kazius et al. 2006). We found that the maxpat is typically set to 10 or more in literature (Wale and Karypis 2006; Helma et al. 2004; Kazius et al. 2006). In this domain, our mining time is more than 10 times smaller.

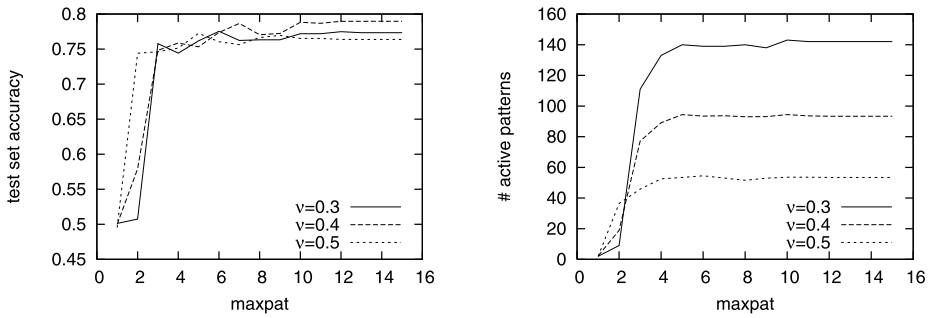

Fig. 5 Test set accuracy and the number of active patterns against the maxpat parameter. The accuracy is computed by 10-fold cross validation in the CPDB dataset

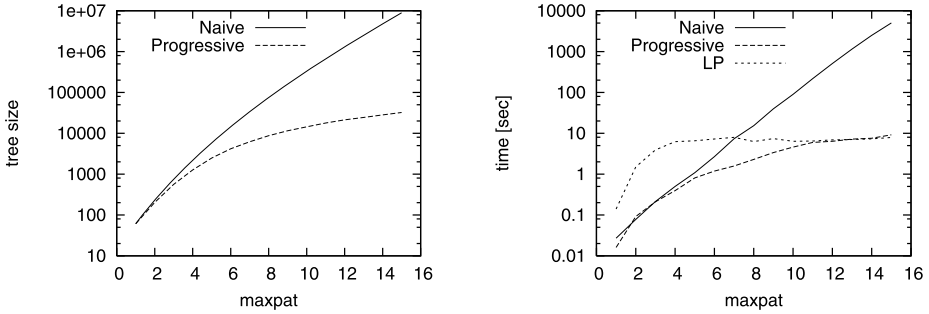

Fig. 6 Comparison in computational costs. In the *left panel*, the tree size refers to the number of nodes in the search space. In the *right panel*, the mining time is plotted respectively for the progressive and naïve methods. The LP time is identical for the two methods, thus depicted as a *single curve*

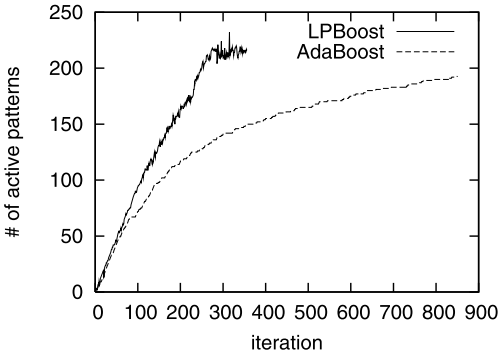

Fig. 7 Learning curves for AdaBoost and LPBoost in the CPDB dataset. Each curve shows the number of patterns included in the decision function. LPBoost converged at iteration 356. AdaBoost needs more iterations to select the same number of patterns since it often selects the same pattern

6 Comparison to AdaBoost

In our previous paper, we used AdaBoost to classify graphs (Kudo et al. 2005). However, since the AdaBoost updates only one weight parameter per iteration, it requires much more iterations to converge (Demiriz et al. 2002). In numerical data, the number of iterations is not a serious problem, because the optimal feature search is trivial. However, since a pattern search algorithm is called in our case, it is important to minimize the number of iterations. Figure 7 displays the faster convergence of LPBoost.

7 Conclusion

We have presented a mathematical programming-based learning method for graph data. Our algorithm is designed such that the search space is pruned autonomously, not by external constraints. It consists of two tightly-coupled components: the machine learning part that solves the mathematical program and the graph mining part that finds optimal patterns. It was shown that our tight integration of graph mining and machine learning leads to better efficiency in comparison with the naïve integration.

However, we have to point out that the search technique employed here can further be improved. Combinatorial search is a mature research field and there have been a lot of methods proposed so far. Compared to those sophisticated methods, our pattern search strategy

is still fundamental. For example, a standard A^* algorithm can be applied for even better results than reported in Sect. 5.

In this paper, we did not use *multiple pricing*, which was used in our previous paper (Saigo et al. 2006). In multiple pricing, the top k patterns are added to the dual problem, not just one. When the search space is not reused, the multiple pricing is effective in reducing the computational time. However, when the search space is progressively expanded, each pattern search becomes much more efficient and the effect of multiple pricing is no longer significant. It also leads to more constraints in the dual problem, which increases the LP time.

In (Saigo et al. 2006), we introduced additional constraints to the linear programming to describe chemical compounds with unobserved activities. After publication of (Saigo et al. 2006), we found that the accuracy gain by such additional constraints is not significant empirically. So, we did not include the topic in this paper. Nevertheless, it is an advantage of our mathematical programming-based framework that additional constraints can be incorporated easily.

Our approach is very general, because one can generalize any mining algorithm to a boosting algorithm. In future work, we would like to try many different combinations. From the mining side, one can try itemset mining, tree mining, sequence mining and so forth. From the learning side, many supervised and unsupervised algorithms are readily described as mathematical programs (Yuan and Casasent 2003; Cai and Hofmann 2004; Rätsch et al. 2002). Certainly many of them can be easily combined with mining algorithms.

The source code in C++ and Matlab is available from http://www.kyb.mpg.de/bs/people/nowozin/gboost/.

Acknowledgements The authors would like to thank anonymous reviewers for detailed comments. We also thank Pierre Mahé for his help in preparing the data.

Appendix: DFS Code Tree

Algorithm 2 finds the optimal pattern which optimizes a score function. To this end, we need an intelligent way of enumerating all subgraphs of a graph set. This problem is highly nontrivial due to loops: One has to avoid enumerating the same pattern again and again. In this section, we present a canonical search space of graph patterns called *DFS code tree* (Yan and Han 2002b), that allows to enumerate all subgraphs without duplication. In the following, we assume undirected graphs, but it is straightforward to extend the algorithm for directed graphs.

DFS code The *DFS code* is a string representation of a graph G based on a depth first search (DFS). According to different starting points and growing edges, there are many ways to perform the search. Therefore, the DFS code of a graph is not unique. To derive a DFS code, each node is indexed from 0 to $n-1$ according to the discovery time in the DFS. Denote by E^f the *forward edge* set containing all the edges traversed in the DFS, and by E^b the *backward edge* set containing the remaining edges. Figure 8 shows two different indexing of the same graph.

After the indexing, an edge is represented as a pair of indices (i, j) together with vertex and edge labels, $e = (i, j, l_i, l_{ij}, l_j) \in V \times V \times L_V \times L_E \times L_V$, where $V = \{0, \ldots, n-1\}$,

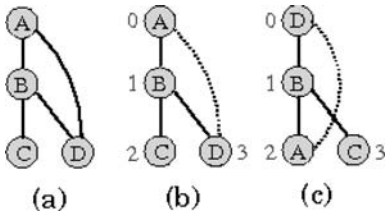

Fig. 8 Depth first search and DFS code of graph. (**a**) A graph example. (**b**), (**c**) Two different depth-first-searches of the same graph. Red numbers represent the DFS indices. *Bold edges* and *dashed edges* represent the forward edges and the backward edges respectively

L_V and L_E are the set of vertex and edge labels, respectively. The index pair is set as $i < j$, if it is an forward edge, and $i > j$ if backward. It is assumed that there are no self-loop edges. To define the DFS code, a linear order $\prec_T$ is defined among edges. For the two edges $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$, $e_1 \prec_T e_2$ if and only if one of the following statements is true:

1. $e_1, e_2 \in E^f$, and ($j_1 < j_2$ or $i_1 > i_2 \wedge j_1 = j_2$).
2. $e_1, e_2 \in E^b$, and ($i_1 < i_2$ or $i_1 = i_2 \wedge j_1 < j_2$).
3. $e_1 \in E^b, e_2 \in E^f$, and $i_1 < j_2$.
4. $e_1 \in E^f, e_2 \in E^b$, and $j_1 \leq i_2$.

The DFS code is a sequence of edges sorted according to the above order.

Minimum DFS Code Since there are many possible DFS codes, it is necessary to determine the minimum DFS code as a canonical representation of the graph. Let us define a linear order for two DFS codes $\alpha = (a_0, \ldots, a_m)$ and $\beta = (b_0, \ldots, b_n)$. By comparing the vertex and edge labels, we can easily build a lexicographical order of individual edges a_i and b_j. Then, the *DFS lexicographic order* for the two codes is defined as follows: $\alpha < \beta$ if and only if either of the following is true,

1. $\exists t, 0 \leq t \leq \min(m, n), a_k = b_k$ for $k < t, a_t < b_t$.
2. $a_k = b_k$ for $0 \leq k \leq m$ and $m \leq n$.

Given a set of DFS codes, the minimum code is defined as the smallest one according to the above order.

Right most extension As in most mining algorithms, we form a tree where each node has a DFS code, and the children of a node have the DFS codes corresponding to the supergraphs. The tree is generated in a depth-first manner and the generation of child nodes of a node is done according to the right most extension (Yan and Han 2002b). Suppose a node has the DFS code $\alpha = (a_0, a_1, \ldots, a_n)$ where $a_k = (i_k, j_k)$. The next edge a_{n+1} is chosen such that the following conditions are satisfied:

1. If a_n is a forward edge and a_{n+1} is a forward edge, then $i_{n+1} \leq j_n$ and $j_{n+1} = j_n + 1$.
2. If a_n is a forward edge and a_{n+1} is a backward edge, then $i_{n+1} = j_n$ and $j_{n+1} < i_n$.
3. If a_n is a backward edge and a_{n+1} is a forward edge, then $i_{n+1} \leq i_n$ and $j_{n+1} = i_n + 1$.
4. If a_n is a backward edge and a_{n+1} is a backward edge, then $i_{n+1} = i_n$ and $j_n < j_{n+1}$.

For every possible a_{n+1}, a child node is generated and the extended DFS code $(a_0, \ldots, a_{n+1})$ is stored. The extension is done such that the extended graph is included in at least one graph in the database.

For each pattern, its embeddings to all graphs in the database are maintained. Whenever a new pattern is created by adding an edge, the list of embeddings is updated. Therefore, it is not necessary to perform isomorphism tests whenever a pattern is extended.

DFS code tree The *DFS code tree*, denoted by $\mathbb{T}$, is a tree-structure whose node represents a DFS code, the relation between a node and its child nodes is given by the right most extension, and the child nodes of the same parent is sorted in the DFS lexicographic order.

It has the following completeness property. Let us remove from $\mathbb{T}$ the subtrees whose root nodes have non-minimum DFS codes, and denote by $\mathbb{T}_{min}$ the reduced tree. It is proven that all subgraphs of graphs in the database are still included in $\mathbb{T}_{min}$ (Yan and Han 2002b). This property allows us to prune the tree as soon as a non-minimum DFS code is found. In Algorithm 2, the minimality of the DFS code is checked in each node generation, and the tree is pruned if it is not minimum (line 9). This minimality check is basically done by exhaustively enumerating all DFS codes of the corresponding graph. Therefore, the computational time for the check is exponential to the pattern size. Techniques to avoid the total enumeration are described in Sect. 5.1 of (Yan and Han 2002b), but still it is the most time consuming part of the algorithm.

References

Abiteboul, S., Buneman, P., & Suciu, D. (2000). *Data on the web: from relations to semistructured data and XML*. San Mateo: Morgan Kaufmann.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., & Kriegel, H.-P. (2006). Protein function prediction via graph kernels. *Bioinformatics*, *21*(suppl. 1), i47–i56.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge: Cambridge University Press.

Bringmann, B., Zimmermann, A., Raedt, L. D., & Nijssen, S. (2006). Don't be afraid of simpler patterns. In *10th European conference on principles and practice of knowledge discovery in databases (PKDD)* (pp. 55–66).

Cai, L., & Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In *ACM 13th conference on information and knowledge management* (pp. 78–87). New York: ACM Press.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning* (pp. 115–123). San Mateo: Morgan Kaufmann.

Demiriz, A., Bennet, K. P., & Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning*, *46*(1–3), 225–254.

du Merle, O., Villeneuve, D., Desrosiers, J., & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, *194*, 229–237.

Duran, J. L., Leland, B. A., Henry, D. R., & Nourse, J. G. (2002). Reoptimization of MDL keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, *42*(6), 1273–1280.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University Press.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the 15th international conference on machine learning* (pp. 114–151). San Mateo: Morgan Kaufmann.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Fröhrich, H., Wegner, J., Sieker, F., & Zell, Z. (2006). Kernel functions for attributed molecular graphs—a new similarity based approach to ADME prediction in classification and regression. *QSAR & Combinatorial Science*, *25*(4), 317–326.

Gärtner, T., Flach, P., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th annual conference on computational learning theory and 7th kernel workshop* (pp. 129–143). Berlin: Springer.

Gasteiger, J., & Engel, T. (2003). *Chemoinformatics: a textbook*. New York: Wiley-VCH.

Hamada, M., Tsuda, K., Kudo, T., Kin, T., & Asai, K. (2006). Mining frequent stem patterns from unaligned RNA sequences. *Bioinformatics*, *22*, 2480–2487.

Helma, C., Cramer, T., Kramer, S., & Raedt, L. D. (2004). Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Sciences*, *44*, 1402–1411.

Hong, H., Fang, H., Xie, Q., Perkins, R., Sheehan, D. M., & Tong, W. (2003). Comparative molecular field analysis (CoMFA) model using a large diverse set of natural, synthetic and environmental chemicals for binding to the androgen receptor. *SAR and QSAR in Environmental Research*, *14*(5–6), 373–388.

Horváth, T., Gärtner, T., & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 158–167). New York: ACM Press.

Inokuchi, A. (2005). Mining generalized substructures from a set of labeled graphs. In *Proceedings of the 4th IEEE international conference on data mining* (pp. 415–418). Los Alamitos: IEEE Computer Society.

James, C. A., Weininger, D., & Delany, J. (2004). Daylight theory manual.

Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *Proceedings of the 21st international conference on machine learning* (pp. 321–328). Menlo Park: AAAI Press.

Kazius, J., Nijssen, S., Kok, J., Bäck, T., & Ijzerman, A. P. (2006). Substructure mining using elaborate chemical representation. *Journal of Chemical Information and Modeling*, *46*, 597–605.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *1–2*, 273–324.

Kudo, T., Maeda, E., & Matsumoto, Y. (2005). An application of boosting to graph classification. In *Advances in neural information processing systems 17* (pp. 729–736). Cambridge: MIT Press.

Le, Q. V., Smola, A. J., & Gärtner, T. (2006). Simpler knowledge-based support vector machines. In *Proceedings of the 23rd international conference on machine learning* (pp. 521–528). New York: ACM Press.

Luenberger, D. G. (1969). *Optimization by vector space methods*. New York: Wiley.

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., & Vert, J.-P. (2005). Graph kernels for molecular structure—activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, *45*, 939–951.

Mahé, P., Ralaivola, L., Stoven, V., & Vert, J.-P. (2006). The pharmacophore kernel for virtual screening with support vector machines. *Journal of Chemical Information and Modeling*, *46*(5), 2003–2014.

Morishita, S. (2001). Computing optimal hypotheses efficiently for boosting. In *Discovery science* (pp. 471–481).

Morishita, S., & Sese, J. (2000). Traversing itemset lattices with statistical metric pruning. In *Proceedings of ACM SIGACT-SIGMOD-SIGART symposium on database systems (PODS)* (pp. 226–236).

Nijssen, S., & Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 647–652). New York: ACM Press.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Mateo: Morgan Kaufmann.

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, *18*(8), 1093–1110.

Rätsch, G., Mika, S., Schölkopf, B., & Müller, K.-R. (2002). Constructing boosting algorithms from SVMs: an application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(9), 1184–1199.

Saigo, H., Kadowaki, T., & Tsuda, K. (2006). A linear programming approach for molecular QSAR analysis. In T. Gärtner, G.C. Garriga, & T. Meinl, (Eds.), *International workshop on mining and learning with graphs (MLG)* (pp. 85–96).

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge: MIT Press.

Shi, L. M., Fang, H., Tong, W., Wu, J., Perkins, R., & Blair, R. M. (2001). QSAR models using a large diverse set of estrogens. *Journal of Chemical Information and Computer Sciences*, *41*, 186–195.

Takabayashi, K., Nguyen, P. C., Ohara, K., Motoda, H., & Washio, T. (2006). Mining discriminative patterns from graph structured data with constrained search. In T. Gärtner, G.C. Garriga, & T. Meinl (Eds.), *Proceedings of the international workshop on mining and learning with graphs (MLG)* (pp. 205–212).

Tibshrani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B*, *58*(1), 267–288.

Wale, N., & Karypis, G. (2006). Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the 2006 IEEE international conference on data mining* (pp. 678–689).

Yan, X., & Han, J. (2002a). gSpan: graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE international conference on data mining* (pp. 721–724). Los Alamitos: IEEE Computer Society.

Yan, X., & Han, J. (2002b). *gSpan: graph-based substructure pattern mining* (Technical report). Department of Computer Science, University of Illinois at Urbana-Champaign.

Yuan, C., & Casasent, D. (2003). A novel support vector classifier with better rejection performance. In *Proceedings of 2003 IEEE computer society conference on pattern recognition and computer vision (CVPR)* (pp. 419–424).

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, *67*(2), 301–320.