Entire Regularization Paths for Graph Data

Koji Tsuda KOJI.TSUDA@TUEBINGEN.MPG.DE
Max Planck Institute for Biological Cybernetics, Spemannstr. 38, 72076 Tübingen, Germany

Abstract

Graph data such as chemical compounds and XML documents are getting more common in many application domains. A main difficulty of graph data processing lies in the intrinsic high dimensionality of graphs, namely, when a graph is represented as a binary feature vector of indicators of all possible subgraph patterns, the dimensionality gets too large for usual statistical methods. We propose an efficient method to select a small number of salient patterns by regularization path tracking. The generation of useless patterns is minimized by progressive extension of the search space. In experiments, it is shown that our technique is considerably more efficient than a simpler approach based on frequent substructure mining.

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, semi-structured texts such as HTML and XML, chemical compounds, RNA secondary structures, and so forth. In supervised learning for graph data, one can rely on the similarity measures derived from graph alignment (Sanfeliu & Fu, 1983) or graph kernels (Kashima et al., 2003). However, one drawback is that the features used in learning are implicitly defined, and the derived prediction rules are hard to interpret. Another approach is based on *graph mining*, where a set of small graphs (i.e., patterns) is used to represent a graph (Figure 1). The graph mining approach is especially popular in chemoinformatics, where the task is to predict activity values of chemical compounds (Kazius et al., 2006; Helma et al., 2004), because it is crucial to understand which parts of the compound contribute to its activity.

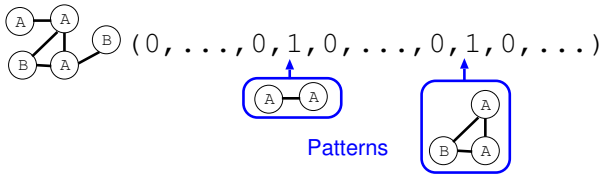

Figure 1. Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

For interpretability of prediction rules, it is required to select a small number of salient patterns necessary for a given learning task. A naive approach is to first use a frequent substructure mining algorithm such as gSpan (Yan & Han, 2002a) or Gaston (Nijssen & Kok, 2004) to collect frequently appearing patterns, and then apply a conventional feature selection algorithm. Actually, this approach was employed by Helma et al. (2004) and Kazius et al. (2006), where a linear support vector machine is used for feature selection. A frequent substructure mining algorithm has two parameters: minimum support (*minsup*) and maximum pattern size (*maxpat*). The output of graph mining is the set of all patterns satisfying the following two constraints: (1) the number of edges is less than *maxpat*, and (2) the pattern is included in more than *minsup* graphs in the database.

In principle, salient features should be selected from the full set of patterns, i.e., the set of all subgraphs of the graphs in the database. However, in the naive approach, one needs to restrict the pattern set a priori using *minsup* and *maxpat* constraints, because it takes a prohibitive amount of time to enumerate the full pattern set. Too much restriction can result in the loss of informative patterns and poor prediction accuracy. Another practical problem is that appropriate values of the parameters are data-dependent, and one has to devise a reasonable procedure to set them.

In this paper, we propose an efficient algorithm for

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

pattern selection based on a regularization path tracking algorithm, called LAR-LASSO (Efron et al., 2004). LAR-LASSO is a forward feature selection algorithm that adds or removes a numerical feature in each step. Instead of numerical features, a pattern is added or removed in our case. The central issue is how to implement the two kinds of feature searching steps contained in the algorithm. We present a method based on the data structure called DFS Code Tree (Yan & Han, 2002a). Especially, an effective tree pruning condition is proposed, which results in efficient discovery of optimal patterns. Our algorithm draws the entire regularization path in a very high dimensional space of pattern indicators. In experiments, we show that the computational cost is kept small, even if patterns are selected from the full set.

For pattern selection in supervised learning tasks, several filter-type methods have been proposed (Morishita & Sese, 2000; Bringmann et al., 2006). They can select the patterns with high correlation to the output variable quickly. However, salient patterns depend on the optimal parameters of the subsequent learning algorithm, and it is difficult to obtain a small number of features informative for any learning algorithm (Kohavi & John, 1997). Our algorithm is more related to graph boosting methods (Nowozin et al., 2007; Saigo et al., 2006) that select features by the L1-norm regularizer with a prespecified regularization parameter. Our novelty is in that the whole regularization path is obtained efficiently.

This paper is organized as follows. In Section 2, the LAR-LASSO algorithm is briefly reviewed. Section 3 and 4 explain how to solve the pattern search problems efficiently. In Section 5, we show experimental results about the computational cost of our method. We conclude this paper in Section 6.

2. Path Tracking Algorithms

The entire regularization path (ERP) algorithms offer a principled way of feature selection (Efron et al., 2004; Rosset & Zhu, 2003). The idea is to trace the solution vector of the L1-norm regularized regression algorithm LASSO (Tibshirani, 1996), as the regularization parameter moves from infinity to zero.

Denote by $X \in \Re^{n \times d}$ the design matrix and by $\boldsymbol{y} \in \Re^n$ the target variables. A learning problem with the L1 regularization is written as

$$\boldsymbol{\beta}(\lambda) = \operatorname*{argmin}_{\boldsymbol{\beta}} L(\boldsymbol{y}, X\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_1. \tag{1}$$

where $\boldsymbol{\beta} \in \Re^d$ is a weight vector. In this paper, we assume that the loss function is quadratic[1],

$$L(\boldsymbol{y}, X\boldsymbol{\beta}) = \frac{1}{2}\sum_i (y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i)^2. \tag{2}$$

Due to the sparsity inducing property of the L1 norm, most of the weights are exactly zero. Denote by $\mathcal{A}$ the active set, i.e., the set of indices of nonzero weights in $\boldsymbol{\beta}$. If the regularization strength λ is set to infinity, the active set is empty. As λ is decreased towards zero, the number of elements in the active set gradually increases to d. This trajectory $\boldsymbol{\beta}(\lambda)$ is called the regularization path. Efron et al. (Efron et al., 2004) pointed out that this path is *piecewise linear* if the loss function is piecewise quadratic (Figure 2). Therefore, to draw the entire path, one does not need to take infinitesimally small steps in λ.

The active set changes itself as the regularization parameter decreases. At a given parameter value, either of the two events can happen: (1) inclusion of a feature to the active set, (2) exclusion of a feature from the active set. The ERP algorithm captures each event of feature inclusion and exclusion so that the entire path of solutions is obtained.

To solve the LASSO problem (1) with a fixed regularization parameter, one needs to solve a quadratic program. In machine learning literature, it is often the case that a grid search is performed to find the optimal regularization parameter in terms of, e.g., the cross validation error. To obtain a better regularization parameter, one has to repeat parameter learnings to try many candidate values, which is quite time consuming. By using the path tracking algorithm, one can find exactly the best regularization parameter in a relatively small computational cost.

The path tracking algorithm is shown in Algorithm 1. The derivative of the loss in the pseudo code is described as

$$\nabla L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} (y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i)\boldsymbol{x}_i. \tag{3}$$

First of all, the first element of the active feature set $\mathcal{A}$ is found by the initial search and the initial direction vector $\boldsymbol{\gamma}$ is set (lines 1 and 2). Next, one has to determine the step length d. As the weight $\boldsymbol{\beta}$ is moved to the direction of $\boldsymbol{\gamma}$, one of the following two events can occur: (1) inclusion of a new feature, or (2) exclusion of an existing feature. The search at line 4 is

[1] To apply our algorithm to other loss functions, it is necessary to extend the algorithm considerably. Especially, we need a new pruning condition for the search tree (see Theorem 1).

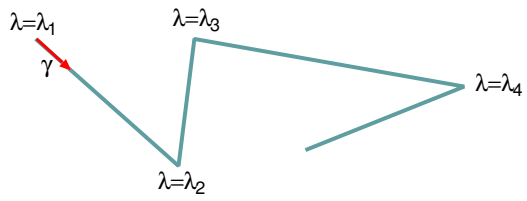

Figure 2. Schematic figure of the regularization paths in the space of the weight vector $\boldsymbol{\beta}(\lambda)$. To follow the path from the starting point $\lambda = \lambda_1$, the direction vector $\boldsymbol{\gamma}$ and the step size d are computed and then one jumps to the next turning point. By repeating this, one can follow the entire regularization path without taking small steps.

performed to determine the step size that the first inclusion event occurs. Due to the Karush-Kuhn-Tucker (KKT) condition, every element in the active set $\mathcal{A}$ has the same gradient value (Rosset & Zhu, 2003),

$$\nabla L(\beta + d\gamma)_j = \nabla L(\beta + d\gamma)_{j'}, \quad \forall j, j' \in \mathcal{A},$$

where $\nabla L(\cdot)_j$ denotes the derivative with respect to the j-th variable. This constant is denoted as $\nabla L(\beta + d\gamma)_{\mathcal{A}}$. The direction $\boldsymbol{\gamma}$ is called an equiangular direction.

We also need to determine the step size d_2 that the first exclusion event occurs (line 5). If $d_1 < d_2$, then the next event is inclusion, otherwise exclusion. Then, the actual step is taken (line 7), the active set is updated (lines 8,9), a new direction is set for a new iteration.

3. Main Search

In applying LAR-LASSO to graph data, one has to solve two kinds of pattern search problems: the initial search (line 1) and the main search (line 4). The main search is called many times while the initial search is done only once. Thus, in this section, we focus on the main search problem that mainly affects the overall computational time.

Let us define some notations. Given a graph database $\mathcal{G} = \{G_i\}_{i=1}^n$, let $\mathcal{T}$ denote the set of all patterns, i.e., the set of all subgraphs included in at least one graph in $\mathcal{G}$. Then, each graph G_i is encoded as a feature vector $\boldsymbol{x}_i = (x_{it})_{t \in T}$,

$$x_{it} = I(t \subseteq G_i),$$

where $t \subseteq G_i$ denotes that t is a subgraph of G_i, and $I(\cdot)$ is 1 if the condition inside is true and 0 otherwise.

In the main search problem, we need to find the optimal pattern t that attains the minimum value of d_t,

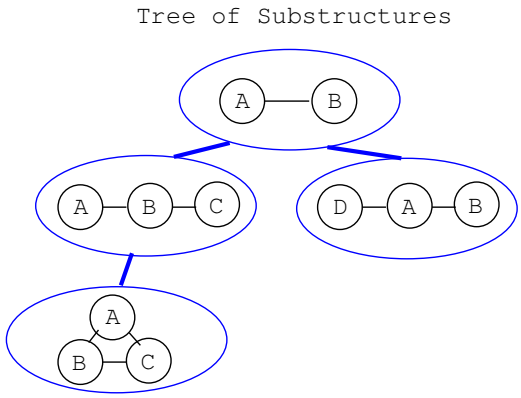

Figure 3. Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree)

implicitly defined as

$$|\nabla L(\boldsymbol{\beta} + d_t\boldsymbol{\gamma})_t| = |\nabla L(\boldsymbol{\beta} + d_t\boldsymbol{\gamma})_{\mathcal{A}}|. \tag{4}$$

The right hand side is equal for any element in $\mathcal{A}$ due to the KKT condition. The above equation is rewritten as

$$|\sum_{i=1}^n w_i x_{it} - d_t \sum_{i=1}^n v_i x_{it}| = |\rho_0 - d_t \eta_0| \tag{5}$$

where

$$w_i = (y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i), \quad v_i = \boldsymbol{\gamma}^\top \boldsymbol{x}_i,$$

and $\rho_0 = \sum_i w_i x_{ik}$, $\eta_0 = \sum_i v_i x_{ik}$ for any $k \in \mathcal{A}$. The equation (5) has the solution,

$$d_t = \min_+ \left\{ \frac{\rho_0 - \sum_i w_i x_{it}}{\eta_0 - \sum_i v_i x_{it}}, \quad \frac{\rho_0 + \sum_i w_i x_{it}}{\eta_0 + \sum_i v_i x_{it}} \right\}. \tag{6}$$

where min_+ stands for the operation of taking minimum among strictly positive elements.

3.1. Search Algorithm

Our search strategy requires a canonical search space in which a whole set of patterns are enumerated without duplication. As the search space, we adopt the DFS code tree (Yan & 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 (Figure 3). A pattern is represented as a text string called the DFS (depth first search) 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. Algorithm 2 shows the pseudo code for the recursive algorithm. See Appendix for details about the DFS code and the rightmost extension.

Algorithm 1 The LAR-LASSO algorithm

$\boldsymbol{\beta} = 0$, $\mathcal{A} = \text{argmax}_j |\nabla L(\boldsymbol{\beta})|_j$. ▷ Initial Search
$\boldsymbol{\gamma}_{\mathcal{A}} = -\text{sgn}(\nabla L(\boldsymbol{\beta}))_{\mathcal{A}}$, $\boldsymbol{\gamma}_{\mathcal{A}^C} = 0$.
while $\max|L(\boldsymbol{\beta})| > 0$ **do**
  $d_1 = \min\{d > 0 : |\nabla L(\boldsymbol{\beta} + d\boldsymbol{\gamma})_j| = |\nabla L(\boldsymbol{\beta} + d\boldsymbol{\gamma})_{\mathcal{A}}|, j \notin \mathcal{A}\}$ ▷ Main Search
  $d_2 = \min\{d > 0 : (\boldsymbol{\beta} + \boldsymbol{\gamma})_j = 0, j \in \mathcal{A}\}$.
  Find step length: $d = \min(d_1, d_2)$
  Take step: $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + d\boldsymbol{\gamma}$
  If $d = d_1$ then add the variable attaining the minimum to $\mathcal{A}$.
  If $d = d_2$ then remove the variables such that $\beta_j = 0$ from $\mathcal{A}$.
  $C = \frac{1}{2}\sum_i \boldsymbol{x}_{\mathcal{A},i}\boldsymbol{x}_{\mathcal{A},i}^\top$
  $\boldsymbol{\gamma}_{\mathcal{A}} = C^{-1}\text{sgn}(\boldsymbol{\beta}_{\mathcal{A}}), \boldsymbol{\gamma}_{\mathcal{A}^C} = 0$.
end while

Algorithm 2 Finding the Optimal Step Size d_1

procedure OPTIMAL STEPSIZE
  $d_1 = \infty$
  for $t \in$ DFS codes with single nodes **do**
    project(t)
  end for
  return d_1
end procedure
function PROJECT(t)
  if t is not a minimum DFS code **then**
    return
  end if
  if pruning condition (7) holds **then**
    return
  end if
  Calculate d_t as (6)
  if $d_t < d_1$ and $d_t \neq 0$ **then**
    $d_1 = d_t$
  end if
  for $t' \in$ rightmost extensions of t **do**
    project(t')
  end for
end function

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 d_t^* the minimum d_t among the ones observed so far. If it is guaranteed that $d_{t'}$ of any supergraph t' is not smaller than d_t^*, we can avoid the generation of downstream nodes without losing the optimal pattern.

We propose the following pruning condition.

Theorem 1. *Let us define*

$$b_{\boldsymbol{w}} = \max\left\{\sum_{w_i<0}|w_i|x_{it}, \sum_{w_i>0}|w_i|x_{it}\right\}.$$

If the following condition is satisfied,

$$b_{\boldsymbol{w}} + d_t^* b_{\boldsymbol{v}} < |\rho_0| - d_t^*|\eta_0|, \tag{7}$$

the inequality $d_{t'} > d_t^*$ holds for any t' such that $t \subset t'$.

(proof) Let us rewrite the equation (5) for $d_{t'}$,

$$|\sum_{i=1}^n w_i x_{it'} - d_{t'}\sum_{i=1}^n v_i x_{it'}| = |\rho_0 - d_{t'}\eta_0|. \tag{8}$$

The right hand side of (8) is bounded from below as

$$|\rho_0 - d_{t'}\eta_0| \geq |\rho_0| - d_{t'}|\eta_0|. \tag{9}$$

The left hand side of (8) is bounded from above as

$$|\sum_{i=1}^n w_i x_{it'} - d_{t'}\sum_{i=1}^n v_i x_{it'}| \leq |\sum_{i=1}^n w_i x_{it'}| + d_{t'}|\sum_{i=1}^n v_i x_{it'}| \tag{10}$$

Each term on the right hand size of (10) is further bounded as

$$\begin{aligned}|\sum_{i=1}^n w_i x_{it'}| &\leq \max\left\{\sum_{w_i<0}|w_i|x_{it'}, \sum_{w_i>0}|w_i|x_{it'}\right\} \\ &\leq \max\left\{\sum_{w_i<0}|w_i|x_{it}, \sum_{w_i>0}|w_i|x_{it}\right\}\end{aligned}$$

The second inequality is derived from the fact $t \subset t'$. So we have

$$|\sum_{i=1}^n w_i x_{it'} - d_{t'}\sum_{i=1}^n v_i x_{it'}| \leq b_{\boldsymbol{w}} + d_{t'} b_{\boldsymbol{v}}. \tag{11}$$

For the equation (8) to have a solution, the two ranges (9) and (11) must intersect. If (7) holds, no solution exists in the domain $d_{t'} \leq d_t^*$, because the two ranges have no intersection. □

3.2. Reusing the search space

In LAR-LASSO, the main search problem is solved repeatedly with different parameters w_i, v_i, ρ_0, η_0. Since the search spaces have large overlap, it is more efficient to reuse the generated search space in next iterations. The whole tree of patterns is kept in the memory, and, in each iteration, the search space is progressively extended.

For less memory consumption, each search can be performed independently without reusing the search space. But in this case, the computation time would be larger by orders of magnitude. In the search algorithm, the most time consuming part is the minimum DFS code check (line 9). The required time is exponential to the size of pattern t (Yan & Han, 2002a). If the search space is not reused, one has to check the minimality of the same DFS code many times.

4. Initial Search

In the initial search (line 1 in Algorithm 1), the task is to find the first pattern to get into the active set. It is proven (Rosset & Zhu, 2003) that the first pattern is found by maximizing the following gain function: $\text{argmax}_{t\in T}\, g_t$, where

$$g_t = \left| \sum_{i=1} y_i x_{it} \right| . \tag{12}$$

The search is conducted basically in the same way, but with a different pruning condition. We used the following pruning condition that has been previously employed in graph boosting (Kudo et al., 2005).

Theorem 2. *For any t' such that $t \subset t'$, $g_{t'} < g_t^*$, if*

$$g_t^* > \max \left\{ \sum_{y_i<0} |y_i| x_{it}, \sum_{y_i>0} |y_i| x_{it} \right\} . \tag{13}$$

For the proof, see (Kudo et al., 2005).

5. Experiments

First, we illustrate how our method works using the estrogen receptor dataset from the Endocrine Disruptors Knowledge Base[2]. It contains 131 chemical compounds suspected to work as environmental hormones, and our task is to predict its toxicity. The first 10 events of LAR-LASSO are shown in Figure 4. As it is just the beginning of the regularization path, most events are inclusions. However, within this short period, an exclusion happens at the 7th event. The evolution of weight parameters is shown in Figure 5. The excluded pattern corresponds to the green curve that converges to zero at the 7th event.

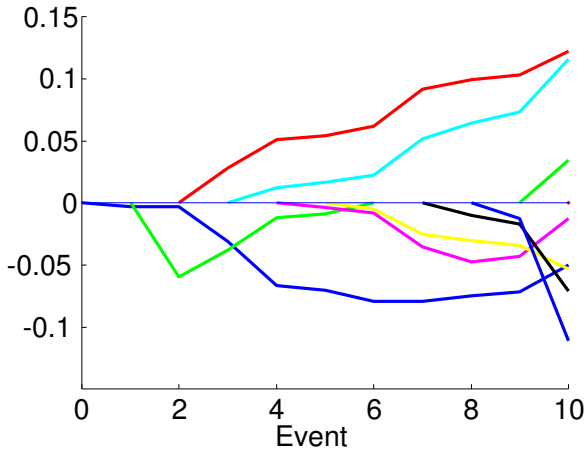

Figure 5. Solution paths for the EDKB dataset. Each curve shows the evolution of the weight parameter of a pattern. A curve can be terminated by an exclusion event. For example, the green curve emerging from the 2nd event disappears at the 7th event.

Next, the computation cost of our progressive method is compared with that of the *naive method*, where all patterns satisfying *minsup* and *maxpat* constraints are enumerated first, and LAR-LASSO is applied subsequently. To show the difference clearly, we used a larger chemical dataset called CPDB with 683 graphs (Helma et al., 2004). This is a classification dataset, but we used the quadratic loss, and the two classes are represented as -1 and 1 in the target variable $\boldsymbol{y}$. The dataset is divided into 90% training set and 10% validation set. In each event of LAR-LASSO, the classification error in the validation set is measured and the event index that achieves the smallest error is recorded (Figure 6). As mentioned in Section 3.2, the number of nodes grows as the algorithm proceeds (Figure 7). In real situations, the LAR-LASSO algorithm is terminated shortly after the minimum error point to avoid overfitting. In this experiment, we recorded the computation time and the number of nodes at the minimum error point.

The computation time and the size of search space are summarized in Table 1. Here the minimum support constraint is not used in both methods, but the maxpat parameter is changed from 5 to ∞. When the maxpat parameter is set to ∞, the full set is used as a source of pattern selection. As observed in the table, the search space of the naive method grows rapidly as the maxpat parameter is increased. On the other hand, our progressive method can always keep

[2] http://edkb.fda.gov

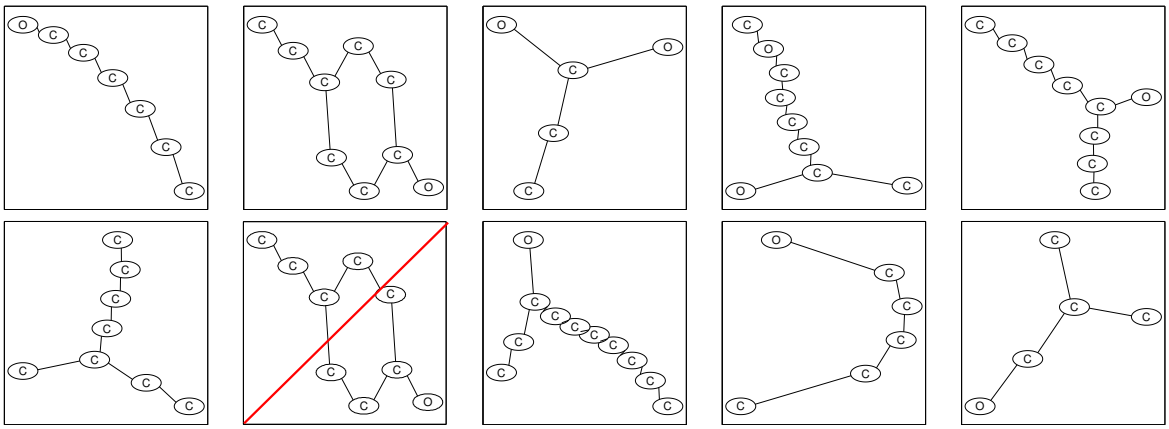

Figure 4. Patterns included or excluded in the first 10 events for the EDKB dataset. The 7th event (indicated by a red line) is an exclusion event.

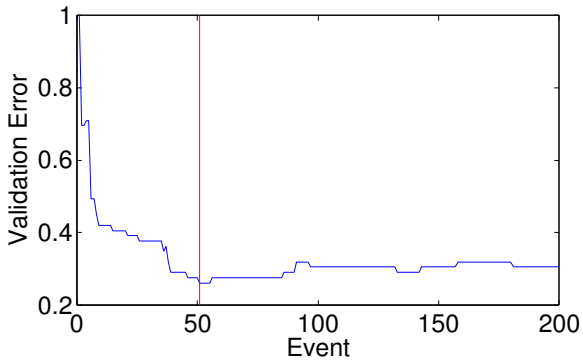

Figure 6. Validation errors of LAR-LASSO for the CPDB dataset (maxpat=10). The red vertical line indicates the event that achieved the smallest validation error.

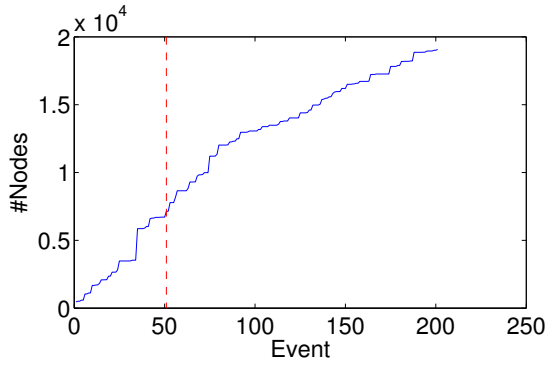

Figure 7. Number of nodes in the search space for the CPDB dataset (maxpat = 10). The red vertical line indicates the event that achieved the smallest validation error. In Table 1, the number of nodes at that point is shown.

the search space very small. Even if the full pattern set is used (maxpat=∞), the computational time is only about 3 seconds. A main reason of this efficiency is that the search space is pruned by the condition imposed by the regression algorithm (Theorem 1). Compared to the naive approach, we have the target values $\boldsymbol{y}$ as an extra information source that can be exploited for tree pruning.

6. Conclusion

We have presented an algorithm for applying LAR-LASSO to 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 (Algorithm 1) that updates the weight parameters and the graph mining part (Algorithm 2) that searches for optimal patterns. This paper has shown that the integration of graph mining and machine learning leads to significant improvement in speed.

Naturally, our idea can be applied to any subclass of graphs. If tree mining is employed instead of graph mining, the computational time will be much shorter. The tree-based method would be useful for, e.g., semi-structured texts and phylogenetic trees.

Appendix: DFS Code Tree

In Algorithm 2, we need to find 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

Table 1. Number of nodes in the search tree and computation time against the maximum pattern size constraint (maxpat). The minimum support constraint is not used. The computation time is measured on a standard 3GHz PC with 1GB memory. The last row shows the computational cost when no constraints are imposed. In this case, the results of the naive method are not available due to memory overflow.

	Naive		Progressive	
maxpat	#nodes	time	#nodes	time
5	2142	0.51	1171	0.38
6	5717	1.39	2111	0.67
7	14309	3.79	3614	1.36
8	33862	9.93	4936	1.90
9	75814	25.64	6605	2.42
10	161858	65.60	7961	2.80
11	332553	164.20	8613	3.00
12	665202	397.95	8857	3.12
13	1302273	931.61	8964	3.11
∞	N/A	N/A	9088	3.15

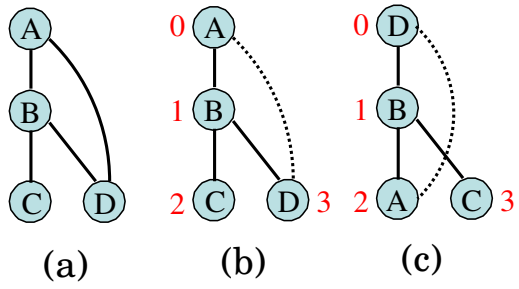

Figure 8. Depth first search and DFS code of graph. (a) A graph example. (b), (c) Two different depth-first-searches of the same graph. Red numbers represent the DFS indices. Bold edges and dashed edges represent the forward edges and the backward edges respectively.

loops: One has to avoid enumarating the same pattern again and again. In this section, we present a canonical search space of graph patterns called *DFS code tree* (Yan & Han, 2002b), that enumerates 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 graph G based on 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 show two different indexings 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 $V = \{0, \ldots, n-1\}$, 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 algorithm, 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 & Han, 2002b). Suppose a node has the DFS code $\alpha = (a_0, a_1, \cdots, a_n)$ 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.

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 & 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 Section 5.1 of (Yan & Han, 2002b), but still it is the most time consuming part of the algorithm.

Acknowledgments

I would like to thank Taku Kudo for offering his implementation of the DFS Code Tree. Discussions with Sebastian Nowozin and Hiroto Saigo were especially helpful.

References

Bringmann, B., Zimmermann, A., Raedt, L. D., & Nijssen, S. (2006). Don't be afraid of simpler patterns. *10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)* (pp. 55–66).

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *Ann. Statist.*, *32*, 407–499.

Helma, C., Cramer, T., Kramer, S., & Raedt, L. (2004). Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *J. Chem. Inf. Comput. Sci.*, *44*, 1402–1411.

Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *Proceedings of the 20th International Conference on Machine Learning* (pp. 321–328). Menlo Park, CA, AAAI Press.

Kazius, J., Nijssen, S., Kok, J., & Ijzerman, T. B. A. (2006). Substructure mining using elaborate chemical representation. *J. Chem. Inf. Model.*, *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 L. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*, 729–736. Cambridge, MA: MIT Press.

Morishita, S., & Sese, J. (2000). Traversing itemset lattices with statistical metric learning. *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems (PODS)* (pp. 226–236).

Nijssen, S., & Kok, J. (2004). A quickstart in frequent structure mining can make a difference. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 647–652). New York: ACM Press.

Nowozin, S., Tsuda, K., Uno, T., Kudo, T., & Bakır, G. (2007). Weighted substructure mining for image analysis. *CVPR '07: Proceedings of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. to appear.

Rosset, S., & Zhu, J. (2003). *Piecewise linear regularized solution paths* (Technical Report). Stanford University.

Saigo, H., Kadowaki, T., & Tsuda, K. (2006). A linear programming approach for molecular QSAR analysis. *International Workshop on Mining and Learning with Graphs (MLG)* (pp. 85–96).

Sanfeliu, A., & Fu, K. (1983). A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, *13*, 353–362.

Tibshirani, R. (1996). Regression shrinkage and selection via LASSO. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, *58*, 267–288.

Yan, X., & Han, J. (2002a). gspan: Graph-based substructure pattern mining. *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)* (pp. 721–724). 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.