Subgraph-Feature Search for Learning Classifiers and Regressors under Fixed Budget Constraint

Graduate school of Infomation Science and Technology, Hokkaido Universisy

Ryo Shirakawa

Contents

1 Introduction **3**

1.1 Related Work and Our Motivation 4

2 Preliminaries **5**

2.1 Notations 5

2.2 Search Space for Subgraphs 6

2.3 Discriminative Criterion 6

3 Problem Setting and Challenges **7**

4 Depth-first Search with Branch and Bound **8**

5 Proposed Method **9**

5.1 Best-first Search 10

5.2 Monte Carlo Tree Search 11

6 Experiments **16**

6.1 Datasets 16

6.2 Comparison of Respective Search Efficiency 17

6.3 Gradient Tree Boosting with Approximate Search for QSAR . 17

6.4 Approximate Search vs. Exact Search for Learning 21

7 Conclusion **23**

List of Figures

1 DFS code tree. Search space for subgraphs that appear in the training dataset. 7
2 Depth-first vs. Best-first. The bold circles show already searched nodes and double circles show the candidate nodes for next expansion. 10
3 One circle of processes in UCT algorithm. 12
4 Cost comparison until finding the best solution. 18
5 Comparison of update of provisional solution. 18
6 Training loss to the number of search nodes for QSAR. 20
7 Test accuracy to the number of search nodes for QSAR. . . . 21
8 Test accuracy to execution time for QSAR. 22
9 Relationship between execution time and frequency of searched subgraphs for CAS. 23
10 The size ratio of the features used for learning the MCTS (approximate search) and the exact Depth-first search. 25

List of Tables

1 Real Dataset summary 16
2 Artificial Dataset summary 17
3 exact Depth-first vs. MCTS (approximate search). 24
4 Best budget constraint and exploration strength parameter. . 24

Abstract

Recently, in various fields such as computational chemistry and bioinformatics, attention has been paid to classification and regression of data whose inputs are graphs of arbitrary size and shape, sometimes called graph classification/regression. In these problems, whether a certain subgraph is included or not is a fundamental feature for graphs that have no direct descriptors available. The simplest solution, enumerating subgraph patterns and use them as features, is so expensive since the number of possible subgraph patterns are intractably large due to the combinatorial explosion. Currently, discriminative pattern mining has been studied and given a good solution to this problem. The previous search method for such discriminative subgraph patterns tries to find the best one by the depth-first search with some pruning rules. Such an exact search, however, is costly for heavily repeated use, which is necessary to obtain the number of features enough for high performance classifiers or regressors. To overcome this computational cost issue, we propose two approximate algorithms based on (*i*) Best-first Search, (*ii*) Monte Carlo Tree Search(MCTS). We demonstrate effectiveness of our proposed methods by comparing performance in real problems using QSAR datasets.

1 Introduction

Graphs are fundamental data structures for representing combinatorial objects such as chemical compounds, networks and others. However, precisely because of their combinatorial nature, it is usually difficult to understand the underlying trends in large datasets of graphs. The rapid increase in data in recent years also includes data represented as graphs, and thus supervised learning in which the inputs are graphs of arbitrary size and shape has gained considerable attention in the fields of computer vision [1–4], chemoinformatics [5–13], bioinformatics [14–16] and computational chemistry [17, 18] and natural language processing [19].

Some of these problems have graph descriptors prepared in advance, such

as molecular dataset [20,21], but others have no direct descriptors available. For these problems, the most fundamental features are subgraph indicators, i.e., if a subgraph is embedded in the given graph or not, because many reactions and events are attributed to those substructures. However, the number of all possible subgraph patterns is intractably large due to the combinatorial explosion, so that it is difficult to enumerate all subgraphs in a realistic time for a given dataset. Therefore, it is necessary to restrict the subgraph candidate on the basis of some criterion. Frequent subgraph mining [22, 23] is one method in this approach. However, since it chooses subgraphs according to the frequencies, it is possible to overlook the importance of subgraphs. On the other hand, discriminative subgraph mining [8, 24–26] choose subgraphs according to some discriminative criteria, and not likely to overlook important subgraphs for learning. Because of this merit, the present paper also adopts this approach.

1.1 Related Work and Our Motivation

Frequent subgraph mining methods, such as gSpan [22] or Gaston [23], are one of the solution of graph classification/regression with subgraph features. These methods make feature vector by enumerating frequently appearing subgraph, and an existing machine learning model is applied to the vector. This learning method is called two step approach, because the mining part and learning part are completely separated. However Wale et al. [27] demonstrated experimentally that two step approach is costly in time and memory and it is harmful in accuracy to restrict features by frequency.

Discriminative subgraph mining methods are used instead to solve these problems. Fan et al. [25] proposed direct mining method that searches a significant subgraph based on information gain and simultaneously learns decision tree. This method is combined with frequent pattern mining and not consider all possible subgraph yet. Saigo et al. [8] removed frequency constraint by using model-based discriminative criteria and designing the

pruning of search space using anti-monotonic of frequency. It learned decision stump directly, and improved the accuracy by taking an ensemble. Shirakawa et al. [26] directly learned regression trees instead of decision stumps to solve non-linear problems. Yan et al. devised Leap Search [24] that uses horizontal pruning instead of vertical pruning as above. This method prunes sibling nodes using the similarity of appearance locations.

The previous discriminative subgraph mining methods have focused on reducing the mining costs by pruning, and searched exactly using them. However, since exact search is very expensive, there are many problems that do not scale even if pruning is used. Therefore, we consider approximate search under fixed budget constraint instead of exact search. Although the approximate search method is inferior to the strict search in accuracy, it is attracting attention as a means for obtaining a quick solution. In this problem, our motivation is to come up with a search method to find out how effective features are at a given cost.

2 Preliminaries

2.1 Notations

Let $[n]$ be a set of integers $\{1, 2, \ldots, n\}$ and let $\mathbb{I}(P)$ be the indicator of a predicate P, i.e., $\mathbb{I}(P) = 1$ if P is true, else 0. A labeled graph is represented by a 4-tuple $G = (V, E, \mathcal{L}, l)$, where V is a set of vertices, $E \subset V \times V$ is a set of undirected edges, $\mathcal{L}$ is a set of labels, and $l : V \cup E \to \mathcal{L}$ is a mapping that assigns a label to each vertex or edge. We denote a subgraph isomorphism, i.e., if g is a subgraph of G, as $G \sqsupseteq g$ and its negation as $G \not\sqsupseteq g$. Thus, a subgraph indicator $\mathbb{I}(G \sqsupseteq g) = 1$ if $G \sqsupseteq g$, otherwise 0. We also denote the training set of pairs of input graph $G \in \mathcal{G}$ and its output responses $y \in \mathcal{Y}$ as

$$\mathcal{D} = \{(G_1, y_1), (G_2, y_2), \ldots, (G_N, y_N)\}. \tag{1}$$

We assume that $\mathcal{G}$ is a set of all finite-size, connected, discretely-labeled, undirected graphs. We denote any set of graphs of size N by $\mathcal{G}_N = \{G_i \mid i \in$

$[N]\}$, and the set of all possible connected subgraphs as $\mathcal{S}_N = \bigcup_{G \in \mathcal{G}_N} \{g \mid G \sqsupseteq g\}$.

Definition 1 (subgraph isomorphism) For two graphs, $G' = (V', E', \mathcal{L}', l')$ and $G = (V, E, \mathcal{L}, l)$, G' is subgraph isomorpic to G iff there exist an injective mapping $\phi : V' \to V$, 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) $l'(v_1, v_2) = l(\phi(v_1), \phi(v_2))$.

2.2 Search Space for Subgraphs

In supervised learning from graphs, we represent each input graph $G_i \in \mathcal{G}_N$ by the characteristic vector $(\mathbb{I}(G_i \sqsupseteq g) \mid g \in \mathcal{S})$ with a set $\mathcal{S}$ of relevant subgraph features. However, since $\mathcal{S}$ is not explicitly available when the learning phase starts, we need to simultaneously do searching and constructing of $\mathcal{S}$ during the learning process. In order to define an efficient search space for $\mathcal{S}_N$, i.e., any subgraphs occurring in $\mathcal{G}_N$, the techniques for *frequent subgraph mining* [22,23] are useful. Note that any subgraph feature $g \in \mathcal{S}_N$ can occur multiple times at multiple locations in a single graph, even if $\mathbb{I}(G_i \sqsupseteq g) = 1$.

In the present paper, we use the search space of the gSpan algorithm [22], which performs a depth-first search on the tree-shaped search spaces on $\mathcal{S}_N$, referred to collectively as an *DFS code tree*, as shown in Figure 1. Each node of the DFS code tree holds a subgraph feature g' that extends the subgraph feature g at the parent node by one edge, namely, $g' \sqsupseteq g$.

The following *anti-monotone* property of subgraph isomorphism over the DFS code tree on $\mathcal{S}_N$ can be used to derive the efficient search-space pruning of the gSpan algorithm:

$$g' \sqsupseteq g, \quad G_i \not\sqsupseteq g \Rightarrow G_i \not\sqsupseteq g'. \tag{2}$$

2.3 Discriminative Criterion

In the previous methods, the discriminating power are calculated by model-based criterioa or statistical criterioa such as information gain. In present

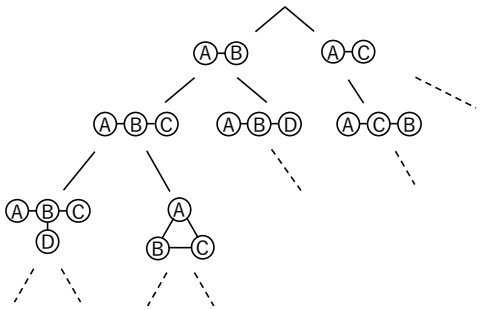


Figure 1: DFS code tree. Search space for subgraphs that appear in the training dataset.

paper, the discriminating power ($DiscriminativeScore : DScore$) is defined by the following equation.

$$\begin{aligned} DScore(g) &= \mathrm{TSS}(\mathcal{D}_1(g)) + \mathrm{TSS}(\mathcal{D}_0(g)) \\ \mathrm{TSS}(\mathcal{D}) &= \sum_{i\in[N]} (y_i - \bar{y})^2,\ \bar{y} = \frac{1}{N}\sum_{i\in[N]} y_i. \end{aligned} \tag{3}$$

where TSS is the total sum of squared error and $\mathcal{D}_1(g) = \{(G_i, y_i) \in \mathcal{D} \mid G \sqsupseteq g\}$ and $\mathcal{D}_0(g) = \{(G_i, y_i) \in \mathcal{D} \mid G \not\sqsupseteq g\}$. Since this score represents an error, a larger value indicates lower discriminating power, and a smaller value indicates higher discriminating power. This score does not depend on the learning model, and can be used even when the labels are discrete values.

3 Problem Setting and Challenges

Our problem is to search for a subgraph with a smaller value of (3) under fixed budget constraint. As mentioned abobe (Section 2.2), the search space of subgraphs is intractably large due to the combinatorial explosion. Therefore, the searchable space is limited, and where to search is an important issue. Simple search policies do not predict the direction of the search and it is difficult to find the better solution. On the other hand, search policies

that are too complex require a cost to determine the direction of the search and are inefficient. Our challenge is to design an efficient search policy for a better solution with as little computation as possible.

4 Depth-first Search with Branch and Bound

The state of the art methods [8, 24, 26] searching for a discriminative subgraph pattern is typically based on the depth-first search, as shown in Figure 2 (a), with a branch and bound technique. [26] searches the best subgraph pattern based on $DScore$. This method calculates the lower bound of $DScore$ obtained by child nodes of search space, and if this lower bound does not reach the provisional solution the child is pruned.

Theorem 1 *Given $\mathcal{D}_1(g)$ and $\mathcal{D}_0(g)$, for any subgraph $g' \sqsupseteq g$,*

$$DScore(g') \quad \geq \quad \min_{(\diamond, k)} \Big[\mathrm{TSS}(\mathcal{D}_1(g) \setminus S_{\diamond,k}) + \mathrm{TSS}(\mathcal{D}_0(g) \cup S_{\diamond,k}) \Big] \quad (4)$$

where $(\diamond, k) \in \{\leq, >\} \times \{2, \ldots, |\mathcal{D}_1(g) - 1|\}$, and $S_{\diamond,k} \subset \mathcal{D}_1(g)$, such that $S_{\leq,k}$ is a set of k pair (G_i, r_i) selected from $\mathcal{D}_1(g)$ in descending order of residual error r_i, and $S_{>,k}$ is that in increasing order. Note that $\setminus$, $\cup$ are set difference and set union respectively.

Proof 1 *Given $\mathcal{D}_1(g)$ and $\mathcal{D}_0(g)$,*

$$\begin{aligned} bound(g) &= \min_{g'} \big[\mathrm{TSS}(\mathcal{D}_1(g')) + \mathrm{TSS}(\mathcal{D}_0(g')) \big] \\ &= \min_{S \subset \mathcal{D}_1(g)} \big[\mathrm{TSS}(\mathcal{D}_1(g) \setminus S) + \mathrm{TSS}(\mathcal{D}_0(g) \cup S) \big] \quad (5) \\ &= \min_{(\diamond, k)} \Big[\mathrm{TSS}(\mathcal{D}_1(g) \setminus S_{\diamond,k}) + \mathrm{TSS}(\mathcal{D}_0(g) \cup S_{\diamond,k}) \Big] \quad (6) \end{aligned}$$

where $(\diamond, k) \in \{\leq, >\} \times \{2, \ldots, |\mathcal{D}_1(g) - 1|\}$. From the anti-monotone property (2), we have $\mathcal{D}_1(g') \subseteq \mathcal{D}_1(g)$ for $g' \sqsupseteq g$ for the training set $\mathcal{D}$ from which the equation (5) directly follows. Thus, we show (6) in detail. For simplicity, let $A = , \{a_1, \ldots, a_n \mid a_i \in \mathbb{R}\}$ denote $\mathcal{D}_1(g)$, and $B = \{b_1, \ldots, b_m \mid b_i \in \mathbb{R}\}$

denote $\mathcal{D}_0(g)$*. Then, the goal of* (5) *is to minimize the total sum of squares* $\mathrm{TSS}(A \setminus S) + \mathrm{TSS}(B \cup S)$ *by tweaking* $S = \{s_1, \ldots, s_k\} \subset A$*. The key fact is that* $\mathrm{TSS}(A \setminus S) + \mathrm{TSS}(B \cup S)$ *can be regarded as a quadratic equation of* $\sum_{i=1}^{k} s_i$ *when the size of* S *is fixed to* k*. More precisely,*

$$\mathrm{TSS}(A) = \sum_{i\in[n]} \Big(a_i - \frac{\sum_{i\in[n]} a_i}{|A|}\Big)^2 = \sum_{i\in[n]} a_i^2 - \frac{\big(\sum_{i\in[n]} a_i\big)^2}{|A|}$$

$$\mathrm{TSS}(A) + \mathrm{TSS}(B) = \sum_{i\in[n]} a_i^2 + \sum_{i\in[m]} b_i^2 - \frac{\big(\sum_{i\in[n]} a_i\big)^2}{|A|} - \frac{\big(\sum_{i\in[m]} b_i\big)^2}{|B|}$$

$$\begin{aligned}
&\mathrm{TSS}(A\setminus S) + \mathrm{TSS}(B \cup S) \\
&= \sum_{i\in[n]} a_i^2 + \sum_{i\in[m]} b_i^2 - \frac{\big(\sum_{i\in[n]} a_i - \sum_{i\in[k]} s_i\big)^2}{|A\setminus S|} - \frac{\big(\sum_{i\in[m]} b_i - \sum_{i\in[k]} s_i\big)^2}{|B \cup S|} \\
&= \sum_{i\in[n]} a_i^2 + \sum_{i\in[m]} b_i^2 - \frac{\big(\sum_{i\in[n]} a_i - \sum_{i\in[k]} s_i\big)^2}{n-k} - \frac{\big(\sum_{i\in[m]} b_i - \sum_{i\in[k]} s_i\big)^2}{m+k}
\end{aligned}$$

Therefore, $\mathrm{TSS}(A\setminus S)+\mathrm{TSS}(B\cup S)$ *is minimized when* $\sum_{i=1}^{k} s_i$ *is maximized or minimized. In other words,* (5) *becomes minimum when the mean of* $S \subset \mathcal{D}_1(g)$ *is maximized or minimized.* □

This computational cost is $O(|\mathcal{D}_1(g)|)$, i.e. it takes linear time for the frequency of the subgraph.

5 Proposed Method

The previous method [26] employs a naive depth-first search policy. This policy does not capture factors such as the nature of the dataset or the situation of current search. The previous method calculate *DScore* and its lower bound for every subgraph, so there is no additional cost for using these values in search policies. In this section, using these values, we propose two

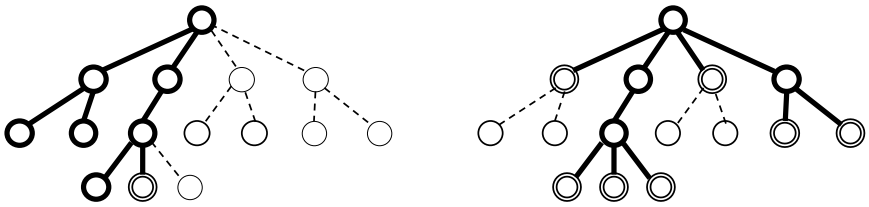


Figure 2: Depth-first vs. Best-first. The bold circles show already searched nodes and double circles show the candidate nodes for next expansion.

efficient search policies based on (*i*) Best-first Search and (*ii*) Monte Carlo Tree Search.

5.1 Best-first Search

Best-first searching [28] is a search policy that expands from the best node based on some evaluation value, shown as Figure 2 (b). The A* algorithm [29] and Dijkstra algorithm [30] are representative.

The previous method designed the pruning rule by calculating the bound value in (4). In the proposed method, this bound value is set as the search priority of each node, and is applied to the best-first search. The pseudocode is shown in Algorithm. 1. The search starts from the root node, and selects the node with the highest bound from the expansion candidate set. The child nodes of the selected node are expanded, and the score is calculated and added to the expansion candidate set. The above operation is repeated until pruning is possible for all expansion candidate set.

Alg. 1: Subgraph Search by Best-first Search

Input: Training data $\mathcal{D} = \{(G_1, r_1), (G_2, r_2), \ldots, (G_N, r_N)\}$

Output: Best score subgraph g^*

Function *Best-first Search($\mathcal{D}$)*

```
candidate ← {root} repeat
    g ← arg max_{candidate c} bound(c) ;
    candidate ← candidate/g ;
    forall child c of g do
        if DScore(c) is better than score* then
            score* ← DScore(c) ;
            g* ← c ;
        end
        candidate ← candidate ∪ c ;
    end
until all candidate can be pruned;
return g* ;
```

5.2 Monte Carlo Tree Search

We consider to apply Monte Carlo tree search (MCTS) [31–33] to our problem. MCTS is widely known as the method used in computer Go. It is also applied in various fields such as feature selection [32]. One of them, Upper Confidence Bounds for Tree (UCT) algorithm [31] is empirically highly evaluated as a way to find good solutions within a limited budget. This method resolves the exploration-exploitation problem using Upper Confidence Bound (UCB), which allows to estimate the expected reward of each child. The simplest UCB policy is called UCB1 and does not require the prior specific knowledge. The policy selects to search child node i that maximizes

$$UCB1 = \bar{X}_i + C \times \sqrt{\frac{\ln n}{2n_i}} \tag{7}$$

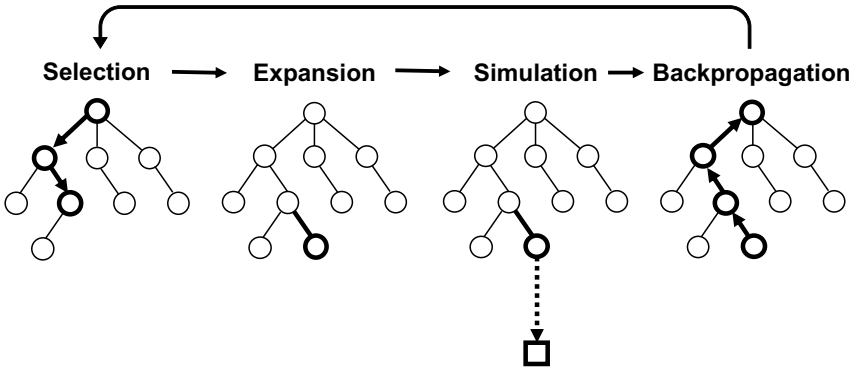


Figure 3: One circle of processes in UCT algorithm.

where $\bar{X}_i$ is the average reward from child node i, n_i is the number of times child node i was selected, n is the number of times parent node of i was selected, and C is an exploration constant. The left term encourages the exploitation of high expected reward child, and the right term encourages the exploration of less visited child.

The UCT algorithm is an iterative method doing the four operations of selection, expansion, simulation, and backpropagation up to a predefined computational cost, shown as Figure 3.

1) Selection:

   Start from the root node, and select the child node repeatedly based on the value of $UCB1$ until the expandable node is reached. A node is expandable if it represents a non-terminal state and has unvisited children.

2) Expansion:

   One random child node that is unvisited is expanded. In this operation, we consider the same pruning condition as (4). If the random select node can be pruned, another node is reselected and expanded.

3) Simulation:

A Monte Carlo simulation is run from the new expanded node, and repeatedly select a child randomly from the available children. This simulation stops if the simulation node has no children or is based on a stochastic condition known as a stopping feature used in [32]. The stochastic condition is $1 - q^d$ depending on the simulation node level d, where $q < 1$ is a parameter and $q = 1 - 10^{-1}$ in the present paper.

4) Backpropagation:

Calculate the reward of the node g obtained by simulation. To align the range to $[0, 1]$, reward is defined,

$$reward(g) = -DScore(g)/\mathrm{TSS}(\mathcal{D}) \tag{8}$$

this value will be large if discrimination is successful, otherwise small. Then backpropagate this reward to the nodes selected in selection and expansion phase.

Monte Carlo Simulation in Subgraph Feature Space

UCT algorithm is often used in game-tree space or item set space. The major difference between these situations and the subgraph feature space is the difficulty of constructing the search space. In this problem setting, the search space is not given in advance. Therefore, we need to construct a search space that matches the given dataset and search for a discriminative subgraph at the same time. When Monte Carlo simulation is performed in this problem, naive way is to enumerate all the child nodes for each simulation node, which requires a large cost. We design a low-cost simulation method since the simulation cost has a big influence on the search efficiency. First, a graph including simulation node is randomly selected from the dataset. Expand the subgraph represented by the simulation node on the selected graph. If an expanded subgraph is available(minimum order defined by [22]), this simulation is taken, otherwise it is repeated.

The entire procedure of the proposed algorithm is illustrated with pseudocode in Algorithm. 2, 3.

Alg. 2: Subgraph Search by UCT

Input: Training data $\mathcal{D} = \{(G_1, r_1), (G_2, r_2), \ldots, (G_N, r_N)\}$

Output: Best score subgraph g^*

Function *UCT($\mathcal{D}$)*

```
repeat
    g ← Selection(root) ;
    g ← Expansion(g) ;
    if DScore(g) is better than score* then
        score* ← DScore(g) ;
        g* ← g;
    end
    s ← Simulation(g) ;
    Backpropagation(s, g) ;
until predefined cost is exhausted;
return g* ;
```

Alg. 3: Four Basic Operations in UCT

Function *Selection(g)*

  while g *is expandable* **do**

    $g \leftarrow \underset{children\ c\ of\ g}{\arg\max}\ \bar{X}_c + C \times \sqrt{\frac{\ln n_g}{2n_c}}$;

  end

  return g ;

Function *Expansion(g)*

  repeat

    $g' \leftarrow random(\{children\ c\ of\ g\})$;

  until *g' is not pruned*;

  return g' ;

Function *Simulation(g)*

  $s \leftarrow g$;

  while s *has some child and not enough stochastic condition* **do**

    repeat

      $G \leftarrow random(\mathcal{D})$;

      $s' \leftarrow random\ expansion\ from\ s\ on\ G$;

    until s' *is enough to minimum order*;

    $s \leftarrow s'$;

  end

  return s ;

Function *Backpropagation(s, g)*

  $X = reward(s)$;

  while $g \neq NULL$ **do**

    $n_g \leftarrow n_g + 1$;

    $\bar{X}_g \leftarrow \frac{\bar{X}_g \times (n_g - 1) + X}{n_g}$;

    $g \leftarrow parent\ of\ g$;

  end

6 Experiments

6.1 Datasets

We also evaluate the performance based on the most typical benchmark on real datasets: the quantitative structure-activity relationship (QSAR). We select Four binary-classification datasets (CPDB, Mutag, AIDS(CAvsCM), CAS) in Table 1: CPDB and Mutag for mutagenicity tests and AIDS(CAvsCM) for antiviral tests. All chemical structures are encoded as molecular graphs using RDKit[1], and some structures in the raw data are removed by chemical sanitization[2]. We simply apply a node labeling by the RDKit default atom invariants (edges not labeled), i.e., atom type, # of non-H neighbors, # of Hs, charge, isotope, and inRing properties. These default atom invariants use connectivity information similar to that used for the well-known ECFP family of fingerprints [34]. See [17] for more elaborate encodings.

Table 1: Real Dataset summary

Dataset	CPDB	Mutag	AIDS(CAvsCM)	CAS
# data	684	188	1503	4337
# $(y = -1, +1)$	(343, 341)	(63, 125)	(1081, 422)	(1936, 2401)
# nodes	25.2	26.3	59.0	30.3
# edges	25.6	28.1	61.6	31.3

# of nodes and edges are average.

In addition to the real datasets, we also have an artificial datasets. These artificial datasets are made by randomly sampling 100 graphs from CAS, which is one of the real data, and assigning a random label to each graph. Artificial1 is assigned as a discrete value labels ($y \in \{-1, +1\}$), and Artificial2 is assigned as a real value labels ($y \in [-1, +1]$). Prepare 100 similar datasets for each. These summary is shown in Table 2. These are used to

[1] `http://www.rdkit.org/`

[2] Due to this pre-processing, the number of datasets differs from that in the simple molecular graphs in the literature, where the nodes are labeled by atom type, and the edges are labeled by bond type.

more generally evaluate each search policy, taking into account the settings and properties of various problems.

Table 2: Artificial Dataset summary

Dataset	Artificial1	Artificial2
# data	100	100
# dataset	100	100
y	$\{-1, +1\}$	$[-1, +1]$
sampled from	CAS	CAS

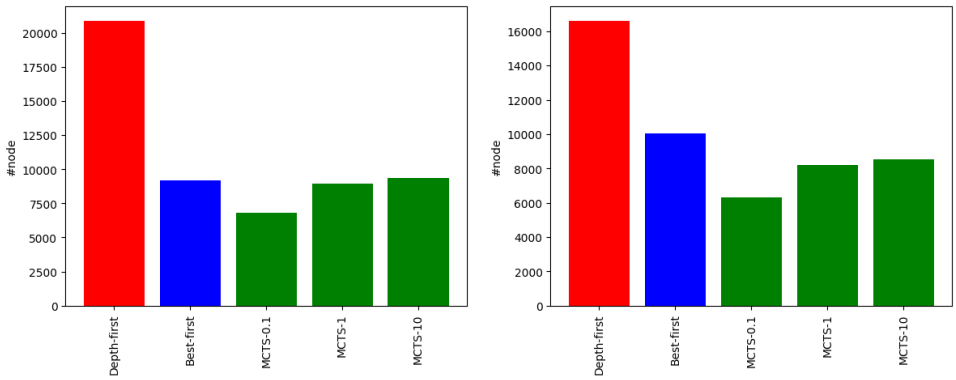
6.2 Comparison of Respective Search Efficiency

First, a feature search is performed once for each artificial dataset without any budget constraint, thereby comparing the respective search efficiencies. Exploration strength parameter of MCTS is set with three type of values (0.1, 1, 10). Figure 4, 5 are the average results for all 100 datasets, which show the number of nodes searched until finding the best subgraph pattern and how the provisional solution is updated in each search. Figure 4 shows that MCTS find the best subgraph with the minimum number of searches, and the best-first search continues at that point. In addition, it was found that each search was not significantly affected by the type of graph label, and the smaller the value of the MCTS parameter, the better the result.

In Figure 5, both of the proposed methods can find a good solution faster than the existing depth-first search. Also the solution is rapidly improved and converged at an early step, there is a sufficient expectation for an approximate search under fixed budget constraint.

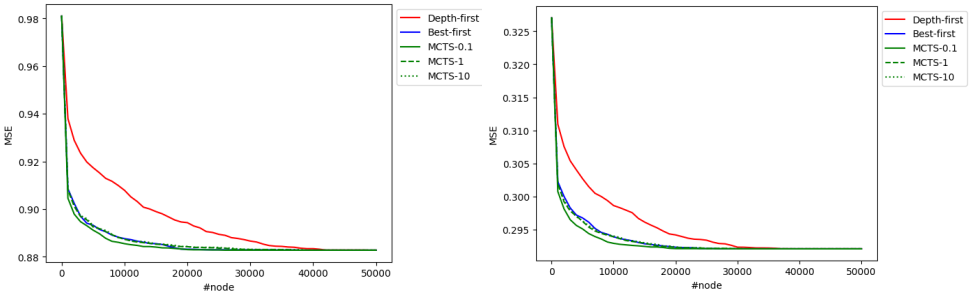
6.3 Gradient Tree Boosting with Approximate Search for QSAR

Now that the search efficiency of the proposed method has been confirmed, in this part, we construct gradient tree boosting (GTB) [26] model for QSAR



(a) Artificial1 ($y \in \{-1, +1\}$) (b) Artificial2 ($y \in [-1, +1]$)

Figure 4: Cost comparison until finding the best solution.



(a) Artificial1 ($y \in \{-1, +1\}$) (b) Artificial2 ($y \in [-1, +1]$)

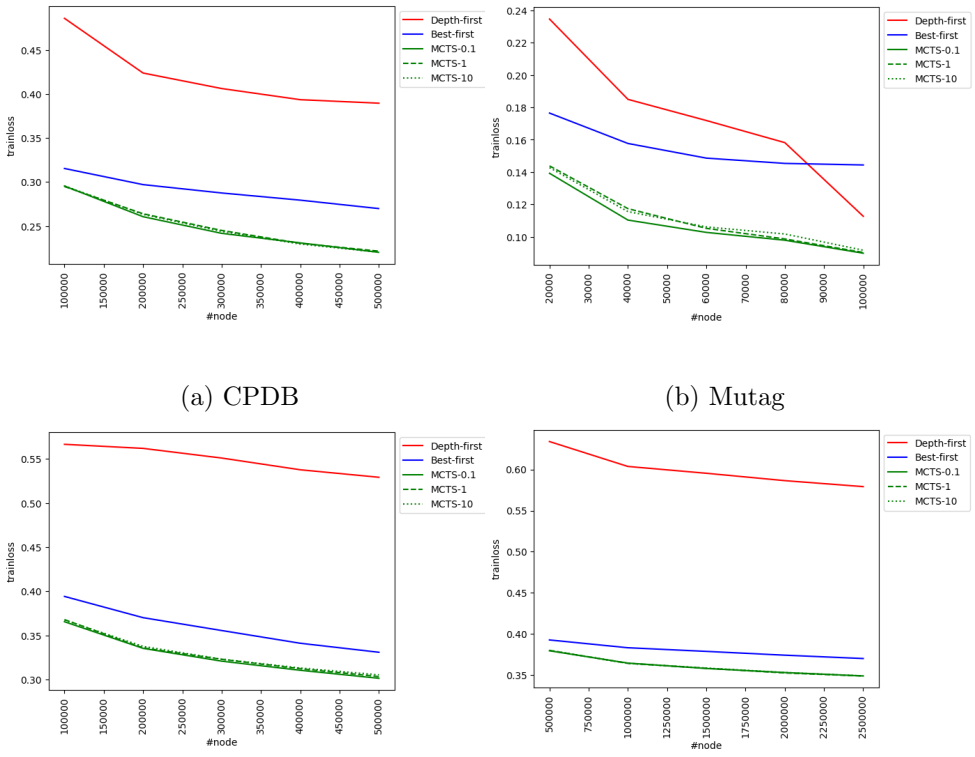
Figure 5: Comparison of update of provisional solution.

datasets using iterative approximate search of the discriminating subgraphs under fixed budget constraint. GTB parameters are fixed at the following values, max subgraph size is 10, the number of trees is 100, depth of each trees is 1 and shrinkage is 1. The budget constraint on the search is set to CPDB=(1000, 2000, 3000, 4000, 5000), Mutag=(200, 400, 600, 800, 1000), AIDS(CAvsCM)=(1000, 2000, 3000, 4000, 5000) and CAS=(5000, 10000, 15000, 20000, 25000), according to the scale of the dataset, and the search is terminated when the number of search nodes exceeds the budget. The results in Figures 6 7 and 8 were measured by 10-fold cross validation, and Figure 9 was measured using all CAS data.

Figure 6 shows the change in training loss of the final ensemble model to the search budget constraint. Comparing the learning of the model with the same number of searches, it can be seen that the model using MCTS performed the best. For the MCTS exploration strength parameter, the same performance is shown, despite the fact that the difference is up to 100 times, indicating that the search is robust to this parameter. The result of the accuracy, is shown in Figure 7, is similar to the result of the training loss, and the accuracy of the model using the MCTS is the highest. On the other hand, the approximation search based on the existing depth-first policy resulted in poor model accuracy.

Then, we consider the execution time. Figure 8 indicates that the best-first search requires the longest execution time. On the other hand, the execution time of the depth-first search is very fast for the same number of searches. In order to verify this difference in execution time, consider the properties of the subgraphs searched by each method. Figure 9 shows the average frequency of the subgraphs searched by each method and the execution time. From Figure 9 (a), it can be seen that the subgraphs searched by best-first policy have an average frequency 30 times higher than depth-first and 4 times higher than MCTS. Looking at the lower bound (4) in the previous section 4, the graph with the higher frequency has a larger maximum

value of k, and the lower bound is likely to be larger because the movement pattern of many sets can be considered. In addition, the depth-first policy likely to search for many large subgraphs, but the frequency of those is generally low. Since the calculation of the lower bound requires a linear cost for the frequency, there is a large difference in the execution time like Figure 9 (b). The hatched portion of Figure 9 (b) indicates the computational cost other than the lower bound, and the other portion indicates the computational cost of the bound. Even if the execution time differs for such a reason, the MCTS model showed the best performance when the accuracy of the model was evaluated at the same execution time.



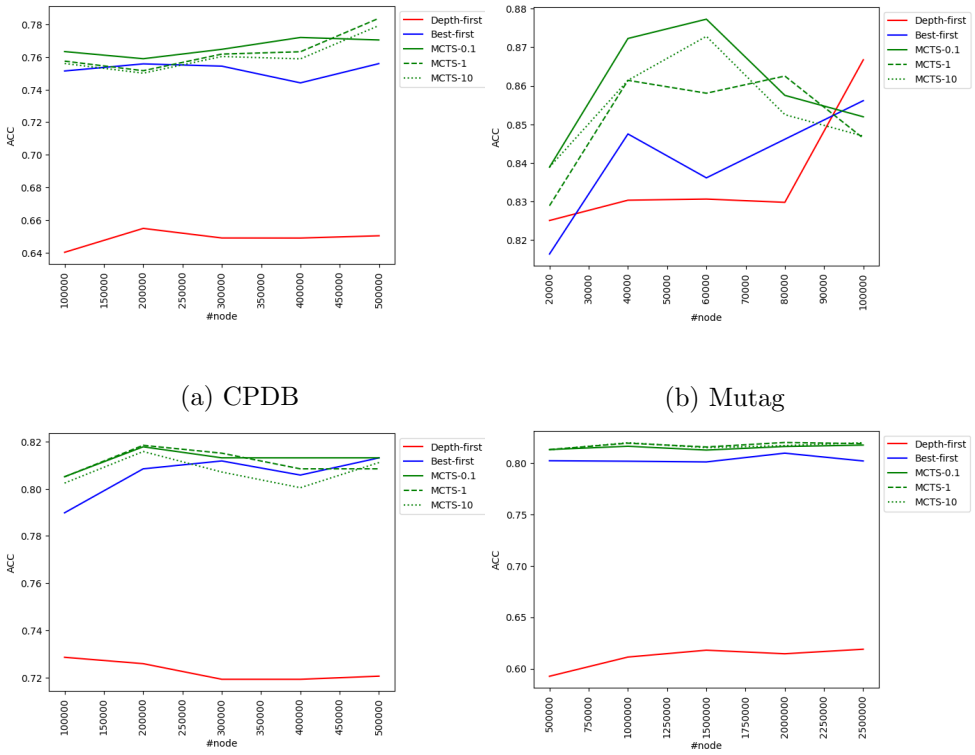
(a) CPDB

(b) Mutag

(c) AIDS

(d) CAS

Figure 6: Training loss to the number of search nodes for QSAR.



(a) CPDB

(b) Mutag

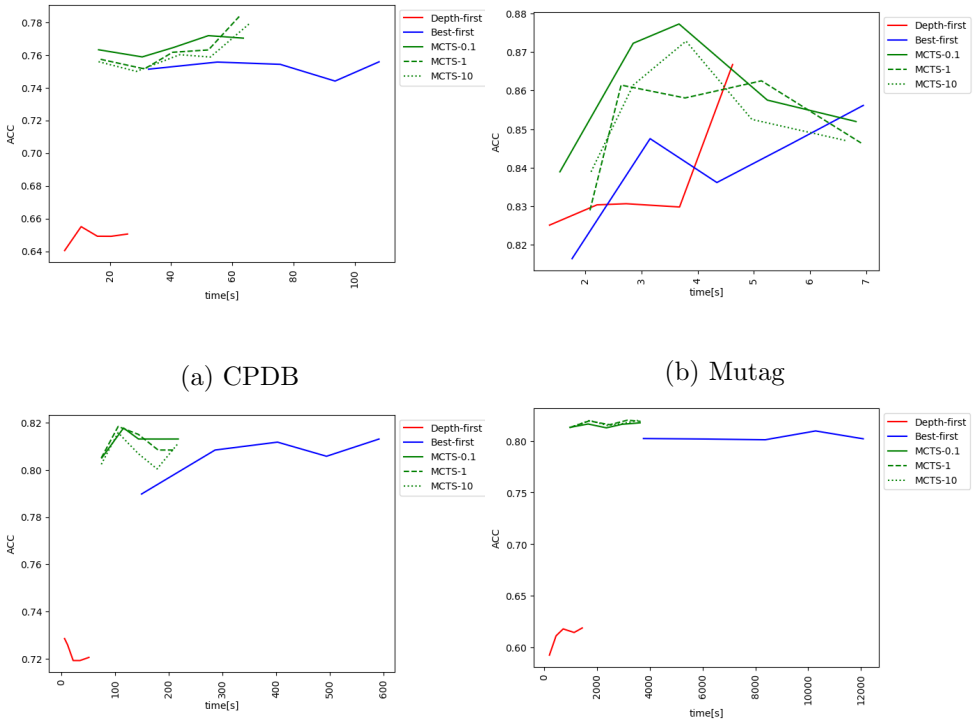
(c) AIDS

(d) CAS

Figure 7: Test accuracy to the number of search nodes for QSAR.

6.4 Approximate Search vs. Exact Search for Learning

So far, we have considered the efficiency of each approximate search. Then, we compare the approximate search with the exact search. Since approximate search based on MCTS policy performed the best compared with other approximate search methods, this section compares GTB+MCTS with the existing method [26], GTB+exact Depth-first The same parameters of GTB and budget constraint are used as for 6.3 and compared using the test accuracy of 10-fold cross-validation for QSAR datasets. The results are shown in Table 3 and the best MCTS parameters used in the results are shown in



(a) CPDB

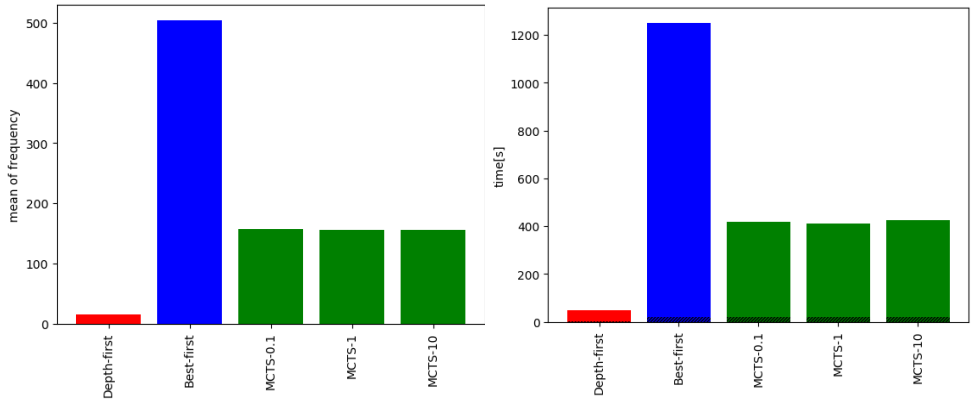
(b) Mutag

(c) AIDS

(d) CAS

Figure 8: Test accuracy to execution time for QSAR.

Table 4. Table 3 shows that, the accuracy of GTB+MCTS was 0.4% 2.7% higher than the accuracy of GTB+exact Depth-first. In addition, the number of search nodes and execution time were reduced from about 1/10 to 1/200. Why did the accuracy of learning improve despite under budget constrain? Figure 10 shows the answer. This figure indicates the percentage of subgraph size used for learning, and it can be seen that model based on the exact Depth-first is learned with almost large features. In particular, 40% or more of the maximum size feature is used. This is apparently over-fitting and led to a decrease in accuracy. On the other hand, model based on MCTS (approximate search) is built with a good balance of features. The latter is



(a) Average frequency of searched subgraphs.

(b) Execution time of each search.

Figure 9: Relationship between execution time and frequency of searched subgraphs for CAS.

considered to be a better model in terms of generalization performance.

7 Conclusion

In summary, we proposed an efficient approximate search method by applying the MCTS algorithm and the best-first search algorithm to the subgraph feature space. These approximate search methods were able to obtain better solutions compared to existing naive search policy. In addition, the learning model constructed by performing this search iteratively reduced the model construction cost from about 1/10 to 1/200 while improving the accuracy compared to the existing exact search model. This confirms that the approximate search leads to an improvement in the generalization performance, and may not only reduce the search cost but also provide a good solution for regularization.

However there are many challenges. First, regarding cost constraint,

Table 3: exact Depth-first vs. MCTS (approximate search).

Dataset	# Search nodes		
	GTB+exact Depth-first	GTB+MCTS	Ratio
CPDB	7.2×10^6	5.0×10^5	0.070
Mutag	3.8×10^5	6.0×10^4	0.015
AIDS(CAvsCM)	7.9×10^7	2.0×10^5	0.003
CAS	6.9×10^7	2.0×10^6	0.029
	Execution time [s]		
	GTB+exact Depth-first	GTB+MCTS	Ratio
CPDB	8.2×10^2	6.2×10	0.076
Mutag	2.3×10^2	3.7	0.016
AIDS(CAvsCM)	2.5×10^4	1.1×10^2	0.004
CAS	8.0×10^4	1.7×10^3	0.040
	Accuracy [%]		
	GTB+exact Depth-first	GTB+MCTS	Difference
CPDB	77.78	78.35	+0.57
Mutag	85.03	87.73	+2.70
AIDS(CAvsCM)	81.37	81.84	+0.47
CAS	80.82	81.99	+1.17

Table 4: Best budget constraint and exploration strength parameter.

	CPDB	Mutag	AID(CAvsCM)	CAS
budget constraint	5000	600	2000	20000
exploration strength	1	0.1	1	1

there is no standard as to how much cost should be spent on a problem of what scale. Currently, it is only experimentally determined, so it is easier to handle if the standard is set theoretically. Second, the MCTS method used this paper is the most basic one. Considering more advanced methods and problem-specific heuristics will likely lead to better searches.

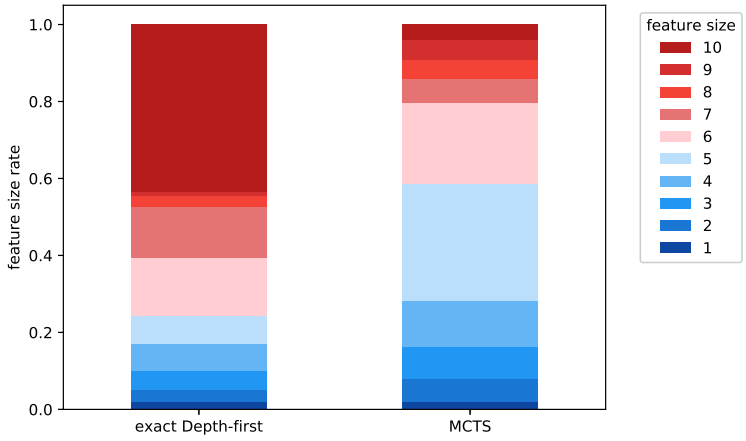


Figure 10: The size ratio of the features used for learning the MCTS (approximate search) and the exact Depth-first search.

acknowlege

In writing this paper, I would like to thank Associate Professor Nakamura for his very kind guidance to me who moved from another laboratory. I also thank Professor Kudo for his guidance from various perspectives too. Finally, I would like to express my gratitude to my family and all those who supported me for six years from undergraduate school.

References

[1] Zaïd Harchaoui and Francis Bach. Image classification with segmentation graph kernels. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, Minneapolis, Minnesota, USA, 2007.

[2] Sebastian Nowozin, Koji Tsuda, Takeaki Uno, Taku Kudo, and Gökhan Bakır. Weighted substructure mining for image analysis. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, Minneapolis, Minnesota, USA, 2007.

[3] Vincent Barra and Silvia Biasotti. 3D shape retrieval using kernels on extended reeb graphs. *Pattern Recognition*, Vol. 46, No. 11, pp. 2985–2999, 2013.

[4] Lu Bai, Luca Rossi, Horst Bunke, and Edwin R. Hancock. Attributed graph kernels using the jensen-tsallis q-differences. In *Proceedings of the 2014 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2014)*, pp. 99–114, Nancy, France, 2014.

[5] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pp. 321–328, Washington, DC, USA, 2003.

[6] Koji Tsuda. Entire regularization paths for graph data. In *Proceedings of the 24th International Conference on Machine learning (ICML)*, pp. 919–926, Banff, Alberta, Canada, 2007.

[7] Hiroto Saigo, Nicole Krämer, and Koji Tsuda. Partial least squares regression for graph mining. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 578–586, Las Vegas, Nevada, USA, 2008.

[8] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning*, Vol. 75, pp. 69–89, 2009.

[9] Pierre Mahé and Jean-Philippe Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, Vol. 75, No. 1, pp. 3–35, 2009.

[10] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, Vol. 11, pp. 1201–1242, 2010.

[11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, Vol. 12, No. Sep, pp. 2539–2561, 2011.

[12] Ichigaku Takigawa and Hiroshi Mamitsuka. Graph mining: procedure, application to drug discovery and recent advances. *Drug Discovery Today*, Vol. 18, No. 1-2, pp. 50–57, 2013.

[13] Ichigaku Takigawa and Hiroshi Mamitsuka. Generalized sparse learning of linear models over the complete subgraph feature set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 3, pp. 617–624, 2017.

[14] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, Vol. 21, No. 1, pp. i47–i56, 2005.

[15] Yan Karklin, Richard F. Meraz, and Stephen R. Holbrook. Classification of non-coding RNA using graph representations of secondary structure. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, pp. 4–15, Hawaii, USA, 2005.

[16] Ichigaku Takigawa, Koji Tsuda, and Hiroshi Mamitsuka. Mining significant substructure pairs for interpreting polypharmacology in drug-target network. *PLoS ONE*, Vol. 6, No. 2, p. e16999, 2011.

[17] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, Vol. 30, No. 8, pp. 595–608, Aug 2016.

[18] Justin Gilmer, Samuel Schoenholz, Patrick F. Riley, Oriol Vinyals, and George Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, Washington, DC, USA, 2017.

[19] Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An application of boosting to graph classification. In Lawrence K. Saul, Yair Weiss, and Léon Bottou,

editors, *Advances in Neural Information Processing Systems 17*, pp. 729–736. MIT Press, Cambridge, MA, 2005.

[20] C. A. James, Weininger, D., and J. Delany. Daylight theory manual. 2004.

[21] Joseph L Durant, Burton A Leland, Douglas R Henry, and James G Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of chemical information and computer sciences*, Vol. 42, No. 6, pp. 1273–1280, 2002.

[22] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, pp. 721–724, Washington, DC, USA, 2002.

[23] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceeding of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 647–652, Seattle, Washington, USA, 2004.

[24] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 433–444, Vancouver, Canada, 2008.

[25] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip Yu, and Olivier Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 230–238, New York, NY, USA, 2008. ACM.

[26] Ryo Shirakawa, Yusei Yokoyama, Fumiya Okazaki, and Ichigaku Takigawa. Jointly learning relevant subgraph patterns and nonlinear models of their indicators. *CoRR*, Vol. abs/1807.02963, , 2018.

[27] Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, Vol. 14, No. 3, pp. 347–375, 2008.

[28] J. Pearl. Heuristics: Intelligent search strategies for computer problem solving. 1 1984.

[29] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107, 1968.

[30] Edsger W Dijkstra, et al. A note on two problems in connexion with graphs. *Numerische mathematik*, Vol. 1, No. 1, pp. 269–271, 1959.

[31] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pp. 282–293, 2006.

[32] Romaric Gaudel and Michèle Sebag. Feature selection as a one-player game. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 359–366, 2010.

[33] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, Vol. 4, No. 1, pp. 1–43, 2012.

[34] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, Vol. 50, No. 5, pp. 742–754, 2010.