Wildcardを許容した頻出部分グラフ列挙とその出力要約

“Frequent Subgraph Mining with Wildcards and its Output Summarization”

岡崎 文哉

平成28年2月

北海道大学工学部情報エレクトロニクス学科

コンピュータサイエンスコース

目次

第1章

はじめに

1.1 背景

データマイニングや知識発見の主要な研究題目のひとつである頻出パターンマイニングは，データベース中に高頻度に存在するパターンをすべて列挙する問題である．その対象はアイテムセット，文字列，木やグラフで表される多種多様な構造を含む．グラフデータベースから頻出する部分グラフパターンを発見する問題は，化合物データベース，タンパク質-タンパク質ネットワーク，XML 文節データベースなどのグラフデータベースから特徴的な部分構造を見つける多くの応用が存在するため，これまでデータマイニングの分野で，いくつものグラフマイニングアルゴリズムが提案されている (FSSM[6], FSG[9]，AGM[7, 8]，gSpan[14]，Gaston[10])．

扱うグラフデータベースでは辺や頂点にラベルが与えられており，頻出部分グラフマイニングにより見つかる部分グラフには一部のラベルが異なるだけの類似部分グラフが多数存在する．従ってラベルを区別しない wildcard ラベルを含んだ部分グラフを求めることで，よりデータの構造を反映した柔軟なパターンが発見できると考えられる．一般的に wildcard は文字列検索などの際に指定する特殊文字の 1 種で，どのような文字・文字列にもマッチするものとして扱われるが，本研究でにおいて wildcard は出現パターンにおいて辺や頂点のどのようなラベルにもマッチするラベルとして扱う．

関連研究として，化学の分野において化合物における wildcard の有用性の研究がある．Hofer ら [5] は化学の分野において同様の働きをしているとみなした複数の種類の頂点を wildcard として処理をすることで有用なパターンを見つけることができたとしている．ま

た，文字列における頻出パターンマイニングにおいて wildcard を考慮した研究 [13, 4, 12] も存在する．マイニングとは異なるが，グラフの検索 [3, 2] の分野でも wildcard を含むグラフを検索する研究もある．しかしマイニングの分野では頻出部分グラフマイニングにおける全列挙という点で既存研究はほぼ見受けられない．その一つの理由として wildcard 導入による出力パターン数の劇的な増大が実用を妨げる点があるものと思われる．

1.2 概要

本研究では，我々は wildcard を許容する部分グラフパターンが，化合物データのみならず様々な分野で有用であると考えた．頻出部分グラフマイニングに wildcard を許容することで列挙される部分グラフ数が膨大に増加する問題の解決のために，頻出パターンマイニングにおいてパターン集合要約に用いられる飽和パターン集合・極大パターン集合を用いることができる．そこで，頻出部分グラフマイニングにおいて wildcard を許容した部分グラフを全列挙したうえで，膨大に増加するパターン集合を要約するため飽和パターン集合と極大パターン集合を求める方法を提案する．実験により，wildcard を許容することによる頻出パターン数の増加，および，飽和パターン・極大パターンによるパターン集合要約の効果を検証する．さらに，wildcard を許容した頻出部分グラフを特徴としてグラフ分類を行うことで，wildcard を許容した頻出部分グラフがどれほど有効であるかを実験し，考察する．

1.3 構成

本論文では，前節の概要について，以降の章で以下の通り詳述する．

- 第 2 章では，頻出部分グラフマイニングの記法，定義について述べる．
- 第 3 章では，提案アルゴリズムのベースとなる既存手法を説明する．
- 第 4 章では，wildcard を許容した頻出部分グラフマイニングへの拡張を提案する．
- 第 5 章では，wildcard を許容した頻出部分グラフマイニングで求めたパターン集合を要約し飽和パターン集合と極大パターン集合を求める手法を提案する．

- 第 6 章では，提案した手法の実験結果を示す．
- 第 7 章では，結論と，本研究の今後の展望を述べる．

第2章

頻出部分グラフマイニング

本章では，本研究で扱う頻出部分グラフマイニングについて説明する．

2.1 グラフマイニング

グラフマイニングとは，グラフを対象としたデータマイニングを指す．化学式，WWWや社会ネットワークなどグラフで表される幅広い応用が存在するため，これまでにさまざまな研究が存在する．グラフマイニングにおける重要な課題のひとつが，本研究で扱う頻出部分グラフマイニングである．

頻出パターンマイニングは，データ集合中で，一定頻度以上現れるパターンを列挙する手法である．頻出パターンマイニングにおける列挙の対象となるパターンは様々で，アイテムセットや系列データ，時系列，パス，部分木，部分グラフなどが挙げられる．本研究で扱う頻出部分グラフマイニングは，頻出パターンマイニングにおける部分グラフを対象とした問題である．

2.2 グラフの定義

定義 1 (無向グラフ). 無向グラフは，頂点と呼ばれる要素の有限集合 V, 辺と呼ばれる要素の有限集合 E，関数 $\psi : E \to \{X \subseteq V| \ |X| = 2\}$ の 3つの組 (V, E, ψ) で定義され，グラフ G の頂点集合や辺集合を $V(G), E(G)$ と表し，$G = (V(G), E(G))$ とも書く．

本研究においてグラフは無向グラフを表しており，有向グラフについては考慮しない．

定義 2 (部分グラフ). *2*つのグラフ $G = (V(G), E(G))$ に対して，$V(G') \subset V(G)$ かつ $E(G') \subset E(G)$ となる $G' = (V(G'), E(G'))$ が与えられたとき，G' を G の部分グラフと呼ぶ.

本研究で扱うグラフでは，多重辺や自己ループを含まず，頂点・辺にラベルが付与された無向グラフとし，本研究 (および従来研究) での列挙対象は連結な部分グラフである.

2.3 問題定義

頻出部分グラフマイニングとは，グラフデータセット $GS = \{G_i \mid i = 1, 2, 3, \ldots, n\}$ に対して，閾値 σ が与えられた時，ある部分グラフ g の支持度 (以下，support) $\mathrm{support}(g) = |\{G_i \mid G_i \in GS, g \text{ が } G_i \text{のある部分グラフに同型 }\}|$ について $\mathrm{support}(g) \geq \sigma$ となる部分グラフ g を全列挙するという問題である．この指定される閾値 σ を最小支持度 (以下，minsup) と呼ぶ．頻出部分グラフマイニングは部分グラフ同型問題に密接に関係している．部分グラフ同型問題の計算複雑さは NP 完全であり，この問題を効率よく解くアルゴリズムは知られていない [1] が頻出部分グラフマイニングは前述したように実データの特性を活かして高速に処理できるアルゴリズムが開発されている.

2.4 既存手法

Yan と Han により開発された gSpan[14] は，深さ優先探索木を拡張したグラフマイニングアルゴリズムであり，AGM[7, 8] や FSG[9] のような隣接行列に基づいた Apriori 風の探索を行わず，パターングラフを深さ優先で探索する点を特徴とし，メモリ効率がよい．また，グラフ上の深さ優先探索により求められる DFS コードという符号をグラフの表現として用いる.

2.4.1 DFS コード

G を頂点の数が k のパターングラフとする．まず，G の頂点を 1 から k まで番号をつけながら深さ優先探索を行い，未探索の頂点に向かう辺を forward edge，探索済みの頂点に向かう辺を backward edge と呼ぶ．ここで，E^f を G の forward edge の集合，E^b を

G の backward edge の集合と表記する．T の辺全体 $E = E^f \cup E^b$ 上の 2 項関係 $\prec_T$ を以下のように定義する．

任意の $e_1, e_2 \in E$ に対して，$e_1 = (i_1, j_1), e_2 = (i_2, j_2)$ (i_1, i_2, j_1, j_2は頂点番号) とおくと，$e_1 \prec_T e_2$ が成り立つのは以下の条件が成り立つときである．

(i) $e_1, e_2 \in E^f$ のとき，$j_1 < j_2$

(ii) $e_1, e_2 \in E^b$ のとき，$i_1 < i_2$または $i_1 = i_2 \wedge j_1 < j_2$

(iii) $e_1 \in E^b, e_2 \in E^f$ のとき，$i_1 < j_2$

(iv) $e_1 \in E^f, e_2 \in E^b$ のとき，$j_1 \leq i_2$

2 項関係 $\prec_T$ は E 上の全順序になる．本研究において G は辺ラベルをもつとしているため，$e = (i, j) \in E$ の符号を $e = (i, j, \mathrm{label}_i, \mathrm{edgelabel}_{i,\,j}, \mathrm{label}_j)$ とする．

G の深さ優先探索順 1 つに対し DFS コードが一意に決まり，$\mathrm{code}(G) = (e_1, e_2, \ldots, e_k)$(ただし $e_1 \prec_T \cdots \prec_T e_k$) と表せる．

2.4.2 DFS 辞書順

あるグラフにおいて深さ優先探索順は多数存在するため，ここではそれぞれの DFS コードに辞書的順序を与えるため DFS コードの線形順序を定義する．まずグラフ G に対して考えられる DFS コードすべての集合を Z とする．任意の 2 つの DFS コード $\alpha = (a_0, a_1, \ldots, a_m)$，$\beta = (b_0, b_1, \ldots, b_n)$，$\alpha$,$\beta \in Z$ が与えられた時，DFS 辞書順 $\alpha \leq \beta$ となるための条件は以下のどちらかを満たすときである．

(i) $\exists t, 0 \leq t \leq \min(m, n), 0 \leq k < t$ である k について $a_k = b_k$かつ $a_t \prec_e b_t$

(ii) $0 \leq k \leq m$ である k について $a_k = b_k$かつ $n \geq m$

ここで使用した $a_t \prec_e b_t$ が成り立つのは，$a_t = (i_a, j_a, l_{i_a}, l_{i_a,\,j_a}, l_{j_a}), b_t = (i_b, j_b, l_{i_b}, l_{i_b,\,j_b}, l_{j_b})$ としたとき，以下のいずれかが成り立つときである．

(i) $a_t \in E^b_\alpha$かつ $b_t \in E^f_\beta$

(ii) $a_t \in E^b_\alpha, b_t \in E^b_\beta$, かつ $j_a < j_b$

(iii) $a_t \in E^b_\alpha, b_t \in E^b_\beta$, かつ $j_a = j_b, l_{i_a,\, j_a} < l_{i_b,\, j_b}$

(iv) $a_t \in E^f_\alpha, b_t \in E^f_\beta$, かつ $i_b < i_a$

(v) $a_t \in E^f_\alpha, b_t \in E^f_\beta$, かつ $i_a = i_b, l_{i_a} < l_{i_b}$

(vi) $a_t \in E^f_\alpha, b_t \in E^f_\beta$, かつ $i_a = i_b, l_{i_a} = l_{i_b}, l_{i_a,\, j_a} < l_{i_b,\, j_b}$

(vii) $a_t \in E^f_\alpha, b_t \in E^f_\beta$, かつ $i_a = i_b, l_{i_a} = l_{i_b}, l_{i_a,\, j_a} = l_{i_b,\, j_b}, l_{j_a} < l_{j_b}$

2.4.3 最小 DFS コード

グラフ G の DFS コードは一意には決まらないが，DFS 辞書順で最小である DFS コードは一意に決定する．このような DFS コードを最小 DFS コード (minimum DFS code) と呼ぶ.

2.4.4 DFS コード木と gSpan アルゴリズム

DFS コード $\alpha = (a_0, a_1, \ldots, a_m)$ に対して，$\beta = (a_0, a_1, \ldots, a_m, b)$ となるよう α に 1 つの辺を加えたものを DFS コード α の子，あるいは α を β の親と呼ぶ．DFS コード木はノードが最小 DFS コードであり，親子関係が上記を満たし，同一の親をもつ子ノードが DFS 辞書順に並んだ木構造である．gSpan アルゴリズムでは探索木が DFS コード木であり，あるグラフよりその部分構造のほうがサポートが大きい，つまり親より子のほうがサポートが小さくなる性質を用い，枝刈りをしながら深さ優先探索をするアルゴリズムである．グラフ G と同型なグラフ G' の最小 DFS コードは同じである [14] ため，DFS コードを用いた探索について，ある DFS コードが最小 DFS コードでないとき，その DFS コードが表すグラフと同型なものが探索木において存在する．そのため DFS コードの最小性を調べることで枝刈りが可能である.

Yan らは，最小でない DFS コードを根に持つ部分木をすべて枝刈りした木が出力すべきすべての部分構造を持つことを理論的に示した [14].

第3章

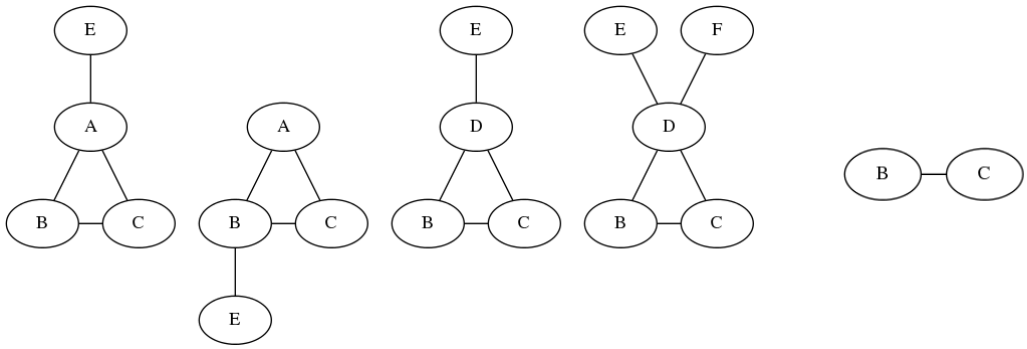
Wildcard許容頻出部分グラフマイニング

本章で，wildcardを許容した頻出部分グラフマイニングに対するアルゴリズムを提案する．

3.1 Wildcardを許容した部分グラフ

本研究においてwildcardをk個許容した部分グラフとは，任意のラベルにマッチするwildcardをラベルとして持つ頂点をk個まで含んで良いとした時に出力されるべきグラフである．一般的なグラフデータにおいては辺のラベルにおけるwildcardが意味を持つことがあると考えられるが，列挙方法については頂点の場合と同様であるため，本研究では頂点ラベルのwildcardのみについて述べる．

まず，wildcardを1つ許容した場合の簡単な例を紹介する．図3.1(a)には4つのグラフが存在する．この中から3つ以上のグラフに含まれる部分グラフを列挙する問題を考える．ここで，minsup=3として通常の頻出部分グラフマイニングで求まる解は図3.1(b)の1つである．図3.1(a)のグラフデータベースでは，すべてのグラフがサイクルを含んでいたり，ラベルの付け方が似ている部分もある．そのため，3.1(b)の1つの解の他にも求めたい特徴があると考えられる．これらの特徴を持つ部分グラフを出力するためには，単純にminsupを小さくする方法が考えられる．しかし，misup=2として求めたパターンは2つ以上のグラフに現れるパターンとしての特徴であるため，求めていたパターンではない．そこでwildcardを許容したパターンを考えたい．しかし，wildcardを1つ許容することで出現するパターンが増えることは容易に想像がつく．その結果が図3.2である．実線と点線はいずれも包含関係を表しており上が下の部分グラフになっている．実線と点線の違いは5節で述べる．この結果より，求めたものの中で一番辺の数が多い結果



(a) (b)

図 3.1: (a) グラフデータベースの例図 (b)(a) のデータベースに対し，minsup=3 として頻出部分グラフマイニングを行ったときの出力パターン

を見ると 4 本の辺で出来た部分グラフがあり，サイクルやラベルの特徴を捉えたパターンが出力されていると言える．このように wildcard を許容することで minsup をあまり下げることなく，出力数を抑えながら，より意味のあるパターンを出力することが期待できる．

本研究において wildcard を許容した部分グラフを全列挙する方法を提案する．

3.2 Wildcard 許容版 gSpan アルゴリズム

ここでは gSpan アルゴリズムで wildcard を許容するためのアルゴリズムを提案する．まず，アルゴリズムに必要な定義を行う．

定義 3 (1-edge extension)**.** あるグラフ G が与えられたとき，G に隣接するノード *1* つとその辺を加えたグラフを *1-edge extension* と呼び，$G + e$ と表記する．このとき，e を *extended edge* と呼ぶ.

定義 4 (Blanket)**.** あるグラフ G が与えられたとき，G の可能な *1-edge extension* すべての集合を G の *blanket* と呼び，パターン集合 $B(G)$ と表記する．G の *blanket* は *2* 種類，*right blanket* $B_R(G)$，*left blanket* $B_L(G)$ に分けることができ，$B(G)$ のうち，探索木において G の *DFS* コードの子になりうる *1-edge extension* を *right blanket* $B_R(G)$，それ以外を *left blanket* $B_L(G)$ と定義する．

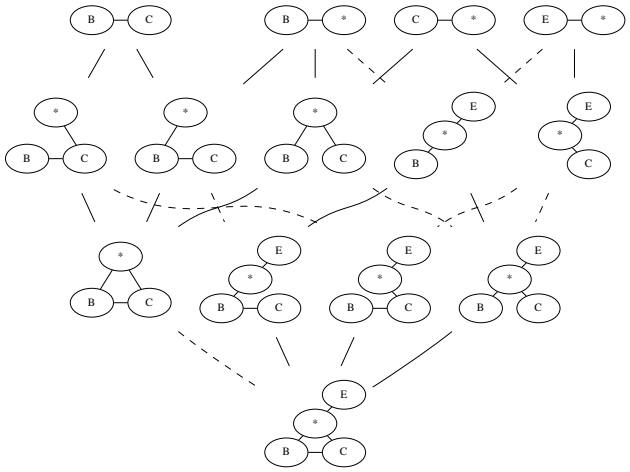


図 3.2: 図 3.1 のデータベースに対し，minsup=3 として wildcard1 つ許容した頻出部分グラフマイニングを行ったときの出力パターンとその包含関係

定義 5 (Wildcard-edge extension)**.** あるグラフ G が与えられたとき，G の可能な *1-edge extension* それぞれに対し，追加する頂点のラベルが *wildcard* である *1-edge extension* を考えることができ，*wildcard-edge extension* と定義する．*blanket* についても同様に，$B^W(G)$, $B^W_R(G)$, $B^W_L(G)$ と定義する．このとき，e を *wildcard edge* と呼び，*wildcard edge* を含むグラフを *wildcard graph* と呼ぶ.

定義 6 (1-edge グラフ)**.** *2*つの頂点とそれを結ぶ辺のみからなるグラフを *1-edge* グラフと呼ぶ．またその *1-edge* グラフが *wildcard graph* であるとき *1-edge wildcard graph* と呼ぶ.

Algorithm 1，2 に wildcard を許容した頻出部分グラフマイニングの擬似コードを示す．このアルゴリズムは gSpan をベースとしている．拡張が必要な点は，まず gSpan で最初に探索する 1-edge グラフに対して全てに wildcard を適用した場合の 1-edge グラフを考える．その 1-edge グラフから深さ優先で関数を呼び出し，その中で再帰的に辺を拡張して探索を進め，support<minsup となるときに探索を打ち切る．

Algorithm 1 wildcard 許容版 gSpan algorithm

Input: GraphDatabase $\mathbb{GS}$, minsup, wildcard 許容数 W

Output: wildcard を許容したすべての頻出部分グラフ

1: $\mathbb{GS}$ 内の頂点と辺を出現頻度順にソート
2: 頻出でない頂点と辺を削除
3: 残った頂点と辺を出現頻度の降順にラベル付
4: $\mathbb{S}^1 \leftarrow \mathbb{GS}$ 内のすべての頻出1-edge グラフ
5: $\mathbb{S}^1_W \leftarrow \mathbb{GS}$ 内のすべての頻出1-edge wildcard graph
6: **for all** グラフ $g \in \mathbb{S}^1, \mathbb{S}^1_W$ in DFS 辞書順 **do**
7: **if** g is wildcard graph **then**
8: SubgraphMining($\mathbb{GS}$, g, $W-1$)
9: **else if** g is not wildcard graph **then**
10: SubgraphMining($\mathbb{GS}$, g, W)
11: **end if**
12: **end for**

Algorithm 2 SubgraphMining($\mathbb{GS}$, g, W)

1: $s : g$ の DFS コード
2: **if** $s \neq \min(code(g))$ **then**
3: return
4: **end if**
5: 現在のパターン G を出力
6: $B_R(G) \leftarrow$ 現在のパターンから計算したextended edge
7: **if** $W > 0$ **then**
8: $B_R^W(G) \leftarrow$ 現在のパターンから計算したwildcard edge
9: **end if**
10: **for all** $e \in B_R(G), B_R^W(G)$ **do**
11: **if** support(c) $\geq$ minsup **then**
12: $g \leftarrow g + e$
13: **if** e is wildcard edge **then**
14: SubgraphMining($\mathbb{GS}$, g, $W - 1$)
15: **else if** e is not wildcard edge **then**
16: SubgraphMining($\mathbb{GS}$, g, W)
17: **end if**
18: **end if**
19: **end for**

第4章

Wildcard許容頻出飽和・極大部分グラフマイニング

前章で述べた通り，wildcard を許容することによってデータが増加するという問題がある．そのため出現したパターンを更に有益に扱うために，冗長なパターンを減らす必要がある．本章では，頻出飽和パターン集合（frequent closed pattern set）と頻出極大パターン集合 (frequent maximal pattern set) について説明し，それを求めるアルゴリズムを提案する．

4.1 頻出飽和パターン集合と頻出極大パターン集合

頻出飽和パターン集合とは，あるパターン X, Y について，support(X)=support(Y) であり，X が Y に含まれるとき，つまり頻出部分グラフマイニングにおいて，グラフ X がグラフ Y の部分グラフになっている場合，頻出パターン集合の中からそういった X を除いたパターンからなる集合を頻出飽和パターン集合という．これは，X が Y に含まれ，support(X)=support(Y) であるとき，support(X)=support(Y)，X が起こっている場所には必ず Y が起こっているため，X を不要なパターンとみなしても情報は失われないからである．

頻出極大パターン集合とは，あるパターン X, Y について，support(X) $>$ minsup かつ support(Y) $>$ minsup であり，X が Y に含まれるとき，X を不要なパターンとみなし，頻出パターン集合の中から X のようなパターンを除いた集合を頻出極大パターン集合という．

4.2 既存手法

頻出パターン集合，頻出飽和パターン集合，頻出極大パターン集合の関係は，頻出パターン集合 $\supseteq$ 頻出飽和パターン集合 $\supseteq$ 頻出極大パターン集合となる．しかし，頻出飽和パターン集合はまだ余分なパターンを含んでいることが多く，頻出極大パターン集合においては削減されすぎていることがある．この問題に対して Takigawa らが提案した方法に δ-tolerance closed frequent subgraphs[11] がある．これは 1 つのパラメータを与えることで飽和と極大の間を単調に削減するためのアルゴリズムであり，本研究においてこれを wildcard を許容した頻出部分グラフマイニングに用いる方法を提案する．

定義 7 (δ-tolerance closed 頻出部分グラフ)**.** ある部分グラフ X を含む部分グラフ Y について $\mathrm{support}(Y) > \max((1-\delta)\cdot \mathrm{support}(X), \mathrm{minsup})$ を満たす Y が存在しないとき，X は *δ-tolerance closed* 頻出部分グラフである．

これを 4 節で使用した図 3.1 のデータベースにおいて考える．図 3.2 において実線はサポート数の変わらない包含関係，点線はサポート数が変わる包含関係であり，この場合，出現するパターンのサポート数は 3 か 4 である（minsup=3 より）．この結果より飽和パターン集合を求めることで 3 パターンまで出現を抑えることができている．

これを求める方法としてパターンをすべて出力した後，パターンと support を比べるというナイーブな手法を考えることができる．しかし，部分グラフ同型のアルゴリズムで十分効率のよいものは見つけられていないため，このナイーブな手法では現実的に計算時間がかかりすぎてしまう．一方で，Takigawa らが提案した手法は gSpan がいわゆる逆探索的なアプローチであることに着目しており， 探索をしながら飽和や極大の判定を行う．加えて枝刈りを導入することで，ナイーブなものに比べて十分高速化できるとしている [11].

δ-tolerance closed frequent subgraphs において Takigawa らは以下の枝刈りを提案した．探索木において，あるパターン α とその子ノード β のグラフデータベースにおける出現数が同じことを Occurence matched であるという（以下，OM）．support と OM の違いは，データベースにおいてパターン α を含むグラフ数を表す support に対して，OM はデータベース全体での出現数であり，1 つのグラフに 2 つ以上出現する場合についても

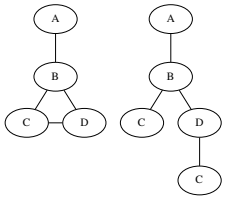


図 4.1: 枝刈りに対する feasible edge の条件の例

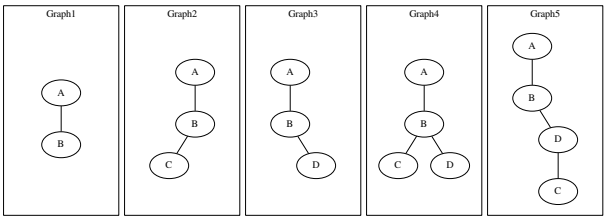


図 4.2: 図 4.1 のデータベースに対し，minsup=1 として頻出部分グラフマイニングをした際の出力グラフ例

すべてカウントするという違いがある．α と β が OM ということは，α があるとき，常に β にできることを意味するので，一見，α の兄弟の探索を打ち切って β の兄弟の探索にすれば良いように思える．しかし，Takigawa らはこの枝刈りでは必要なパターンも枝刈りしてしまうことを示した．

必要なパターンを刈ってしまう例を示す．図 4.1 の 2 つのグラフデータベースに対して minsup=1 として頻出部分グラフマイニングをした際の飽和パターン集合を考える．すると図 4.2 のようなパターン例が出現する．ここで Graph1 と Graph2 は OM であるために，Graph1 の子ノード Graph3 は Graph2 の子ノード Graph4 があるために飽和パターンではなくなる．それ故に，Graph1 と Graph2 が OM であるという条件から Graph1 の他の子ノード（今回は Graph3）が枝刈りができるはずである．しかし，実際に Graph3 を枝刈りすることで，Graph5 のような出現が必要なパターンも同時に枝刈りをしてしまっている．

この問題に対し，Takigawa らは次のような条件と枝刈りを提案した．

定義 8 (Feasible edge)．*1* つの辺 e を加えた際，枝刈りが実行可能である場合は e が以下のいずれかを満たすときである．

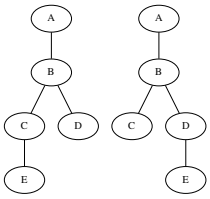


図 4.3: Wildcard pruning と枝刈りをしてはいけない例を示すためのデータベース

(i) e が backward edge である．

(ii) 探索している部分グラフが木構造でありかつグラフデータベース中の e がすべてブリッジになっている．

定理 1 (Right-blanket pruning)**.** あるグラフ G が与えられた時，$G' = G + e \in B_R(G)$ が G と OM であり，かつ e が *feasible edge* であるならば探索木において $G + e', e \neq e'$ とその子孫すべてを枝刈りすることができる．

定理 2 (Left-blanket pruning)**.** あるグラフ G が与えられた時，$G' = G + e \in B_L(G)$ が G と OM であり，かつ e が *feasible edge* であるならば探索木において G とその子孫すべてを枝刈りすることができる．

4.3 Wildcard 許容版 δ-tolerance closed frequent subgraphs

本節では，wildcard 許容版 δ-tolerance closed frequent subgraphs を求めるアルゴリズムを設計する．それに当たって，まず，wildcard を含めた飽和と極大について新しく定義を追加する．

4.3.1 Wildcard を含むパターンに対する飽和集合・極大集合

定義 9 (Wildcard graph を含むグラフ集合の飽和集合・極大集合)**.** *wildcard graph* を含むグラフ集合の飽和集合・極大集合とは，*wildcard* を通常のラベルと見なして求めた飽和集合・極大集合から，*wildcard* が *1* つの文字のみにマッチする *wildcard graph* を除いた集合とする．

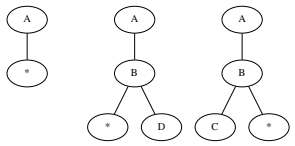


図 4.4: Wildcard が 1 種類のラベルのみを許容しているパターン例

この定義は，wildcard graph において，wildcard が表すラベルがすべて同じで 1 種類のみである時，これを wildcard を対応するラベルで置き換えた同一のパターンが存在し，support が同じであるため冗長な情報であるとしたものである．これを図 4.3 のデータベースを用いて説明する．データベースにおいて misup=2 で wildcard を 1 つ許容した頻出部分グラフマイニングを行うと，合計 24 個の部分グラフが出力される．図 4.4 に示したグラフは頻出なものの中で，wildcard が 1 種類のラベルのみを許容しているパターンの例である．ここに示したグラフ中の wildcard は，図 4.3 のデータベースよりそれぞれ 1 種類のラベルのみにマッチする．したがって，定義 5.3 より図 4.4 の例のようなグラフは飽和集合・極大集合に含まれない．このようなグラフの除外は，次の節の更なる枝刈りによって効率化が可能である．

4.3.2 Wildcard pruning

命題 4.3.1 (Wildcard pruning)**.** ある *wildcard graph* G において，含まれる *wildcard* が *1* 種類のラベルしか保持しない場合，G と以下の子ノードを枝刈りすることができる．

証明 . ある *wildcard* 許容グラフ G において，含まれる *wildcard* が *1* 種類のラベルしか持たない時，その *wildcard* を許容しているラベルで置き換えたグラフ G' を見た場合，G と G' はデータベース中において指すものが同じなので，*OM* であり，その *support* も等しい．よって G は出力をしなくてよい．また，G の *DFS* コードの子を考えた時，その *wildcard graph* $G+e$ において含まれる *wildcard* がもつラベルは同様の *1* 種類のラベルのみである．そのため，G と以下すべての子ノードを枝刈りすることができる． □

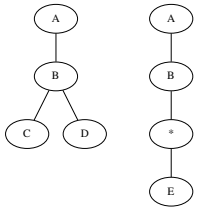


図 4.5: minsup=2 における wildcard 1 つ許容した頻出飽和部分グラフ

4.3.3 提案アルゴリズム

また，wildcard-edge extension を探索する際に wildcard edge は extended edge と同様に扱ってはならない．その例を図 4.3 のデータベースを用いて示す．まず，このデータベースにおいて minsup=2 として wildcard を 1 つ許容した頻出飽和部分グラフが図 4.5 のグラフである．また，この探索を行ったとき，1-edge グラフ A-B から始まる探索木 (DFS コード木) が図 4.6 である．この探索木を見ると，A-B に対して A-B-C が OM であり，かつ feasible である．そのため，Right-blanket pruning が適応でき，A-B-D とそのすべての子ノードが枝刈りされる．しかし，A-B-∗(∗:wildcard) のノードは枝刈りすることができない．仮に，A-B-∗ を枝刈りしてしまうと，図 4.5 に含まれる A-B-∗-E が出力されない．これは，A-B-∗ のグラフ中の wildcard が許容するラベルが 2 種類以上あるためである．つまり δ-tolerance closed frequent subgraphs の枝刈りにおいて，探索を打ち切ることができるのは通常の extended edge であって，wildcard edge を同時に枝刈りすることはできない．これは，追加した辺に関して OM かつ feasible であるときに通常の extended edge と同様に wildcard edge を枝刈りをしてしまうと，以上のようなパターンを探索せず，見逃してしまうためである．

これらの条件を考慮することで，wildcard 許容版 δ-tolerance closed frequent subgraphs において，条件を満たすものすべてを全列挙することが可能である．このアルゴリズムは Takigawa らの δ-tolerance closed frequent subgraphs を拡張した．Algorithm 3，4，5 に擬似コードを示す．

まず Algorithm 3 において，探索木のはじめのノードとなる 1-edge グラフにおいて考えうるすべての wildcard をもつグラフを追加する．

次に Algorithm 4 の SubgraphMining($\mathbb{GS}$, G, W) では，現在のパターンについて δ-

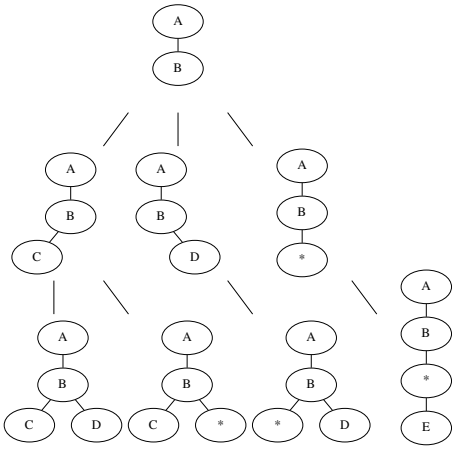


図 4.6: minsup=2 における wildcard 1 つ許容した頻出部分グラフマイニングの探索木 (DFS コード木) の一部

tolerance closed であるかを探索し，子ノードとして考えうるすべてのパターンを再帰的に呼び出すものである．Algorithm 4 の 2 行目で現在のパターンが wildcard を含むグラフであるとき，その wildcard の現れる場所全てにおいて同じラベルを表す，つまりすべての wildcard が 1 種類のラベルを表すならば，wildcard pruning を満たしそのパターンとその子ノードすべては必要なパターンではないので枝刈りをする．この枝刈りの判定するために WildcardEdgeExtension の際に調べた wildcard をその後の子ノードで保持しておく必要があるため，wildcard graph は wildcard を持つ場所とそのラベルをデータ構造上にもつ．また，δ-tolerance closed frequent subgraphs と同様に，Left-blanket pruning により枝刈りができるかを探索し，δ-tolerance closed であるかどうかを調べ，Right-blanket pruning による枝刈りの探索をする．しかし，前に述べたように Left-blanket pruning と Right-blanket pruning による枝刈りの条件を満たしていたとしても wildcard edge の探索はしなければならない．そのため，Algorithm 4 の 3 箇所に Algorithm 5 の WildcardEdgeExtension($\mathbb{GS}$, G, W) を入れる必要がある．Algorithm 5 は擬似コードが複雑にならないように分けたものである．

LeftPruning(G, $\mathbb{GS}$, W)，IsDTOLCLOSED(G, $\mathbb{GS}$, W) は δ-tolerance closed frequent subgraphs における定義 [11] に，現在のパターンに更に wildcard の拡張ができる場合，通常のものと同様に wildcard edge も調べるように拡張したものであるため，ここでは省略

Algorithm 3 wildcard 許容版 δ-tolerance closed frequent subgraphs algorithm

Input: GraphDatabase $\mathbb{GS}$, minsup, δ, wildcard 許容数 W

Output: wildcardを許容したすべてのδ-tolerance closed subgraphs

1: $\mathbb{GS}$ 内の頂点と辺を出現頻度順にソート
2: 頻出でない頂点と辺を削除
3: 残った頂点と辺を出現頻度の降順にラベル付け
4: $\mathbb{S}^1 \leftarrow \mathbb{GS}$ 内のすべての頻出1-edge グラフ
5: $\mathbb{S}^1_W \leftarrow \mathbb{GS}$ 内のすべての頻出1-edge wildcard graph
6: **for all** グラフ $G \in \mathbb{S}^1, \mathbb{S}^1_W$ in DFS 辞書順 **do**
7: **if** G is wildcard graph **then**
8: SubgraphMining($\mathbb{GS}$, G, $W-1$)
9: **else if** G is not wildcard graph **then**
10: SubgraphMining($\mathbb{GS}$, G, W)
11: **end if**
12: **end for**

する.

Algorithm 4 SubgraphMining($\mathbb{GS}$, G, W)

1: s : G の DFS コード
2: **if** G is wildcard graph **and** G のもつwildcard がすべて 1 種類のラベルを指す **then**
3: return
4: **end if**
5: **if** support(G) $\geq$ minsup **then**
6: return
7: **end if**
8: **if** s $\neq$ min(code(G)) **then**
9: return
10: **end if**
11: **if** LeftPruning(G, $\mathbb{GS}$, W) **then**
12: WildcardEdgeExtension($\mathbb{GS}$, G, W)
13: return
14: **end if**
15: **if** IsDTOLCLOSED(G, $\mathbb{GS}$, W) **then**
16: 現在のパターンを出力
17: **end if**
18: **for all** $e \in B_R(G)$ in DFS 辞書順 **do**
19: SubgraphMining($\mathbb{GS}$, $G + e$, W)
20: **if** $G \overset{\text{OM}}{\longleftrightarrow} G + e$ **and** e is feasible **then**
21: WildcardEdgeExtension($\mathbb{GS}$, G, W)
22: return
23: **end if**
24: **end for**
25: WildcardEdgeExtension($\mathbb{GS}$, G, W)

Algorithm 5 WildcardEdgeExtension($\mathbb{GS}$, G, W)

1: **if** $W = 0$ **then**
2: return
3: **end if**
4: **for all** $e \in B_R^W(G)$ in DFS 辞書順 **do**
5: SubgraphMining($\mathbb{GS}$, $G + e$, W)
6: **end for**

第5章

実験と考察

この章では，これまでに述べた wildcard を許容することで頻出部分グラフマイニングによる出力がどれほど増え，出力時間が長くなるかを実験する．さらに，提案した飽和・極大パターンにより，どれほど出力が抑えられるか，出力時間はどのように変化するかを実験，検証する．

また，wildcard を許容した部分グラフパターンがどれほど有用であるかを，機械学習の特徴として用いることで検証する．

5.1 実験環境

本手法を実装したものの実験環境として，以下を使用した．

CPU	Intel(R) Core(TM) i7-3770K 3.50GHz
メモリ	33GB
OS	Ubuntu 14.04.3 LTS

5.2 使用したデータベース

今回，Takigawa ら [11] によって用いられた Mutag と CPDB の 2 つのデータを使用し，実験を行った．それぞれのデータベースのグラフ数，平均ノード数，平均エッジ数を表 5.1 に示す．AIDS(CAvsCM) に関しては，機械学習に対する実験で使用した．

表 5.1: 使用したデータセット

	グラフ数	平均ノード数	平均エッジ数
Mutag	188	17.9	19.8
CPDB	684	14.1	14.6
AIDS(CAvsCM)	1503	59.0	61.6

5.3 出力数と出力時間に関する実験

頻出部分グラフマイニングに wildcard を許容した場合の出力数，CPUtime の比較を行う．さらに増加した出力数が，wildcard 許容版 δ-tolerance closed frequent subgraphs algorithm によって抑えられているのか実験した結果を出力数と CPUtime の観点から検証する．

この実験では頂点のラベルにのみ wildcard を許容した．これは使用するデータが化合物データであり，化合物データにおいて辺のラベルは結合数を指すことが多いため，辺のラベルに wildcard を考える必要はないと判断したためである．しかし，一般的なグラフデータベースにとって辺のラベルが重要な情報を持つことは少なくないため，辺のラベルに対して wildcard を許容するべき時もあるが，これは頂点のラベルと同様に拡張可能である．

図 5.1 に実験結果を示す．図 5.1 左が Mutag，図 5.1 右が CPDB の実験結果で，それぞれ上図が出力数，中央図が CPUtime，下図が 1 出力あたりの CPUtime の実験結果を表している．図中において gSpan は通常の頻出部分グラフマイニング，gSpan with wildcard は wildcard を 1 つ許容した頻出部分グラフマイニング，closed with wildcard，maximal with wildcard はそれぞれ wildcard を 1 つ許容した頻出部分グラフマイニングに対する頻出飽和パターン集合，頻出極大パターン集合の結果である．

5.3.1 考察

Mutag の実験結果では，図 5.1 左において，gSpan と gSpan with wildcard を比較すると，misup をさげるとどちらも指数的に出力が伸びているのに対し，minsup を固定して

比較すると，出力数はほぼ定数倍であることがわかる．CPDB の結果でも同様のことが言え，これはすべての頻出部分グラフのすべての頂点ラベルに対して wildcard を許容した場合を考えることができるためであり，通常の頻出部分グラフに対して出力部分グラフの平均ノード数倍の出力が得られるものと考えられる．また CPUtime において，gSpan と gSpan with wildcard を比較すると，図 5.1 の左右の中央図より，こちらもほぼ定数倍で増えていることがわかる．これは図 5.1 の下図より見て取れるように，gSpan と gSpan with wildcard の 1 つあたりの CPUtime はほぼ同じであるためである．

実験結果より，頻出部分グラフマイニングでは minsup を下げると指数倍以上に出力が増える．さらに，wildcard を許容することでほぼ定数倍ではあるが出力数が増え，扱いがより困難になると考えられる．そこで飽和・極大パターン集合による集合要約の効果を検証する．

まずは Mutag と CPDB のデータの違いによる差をみると，Mutag は CPDB に比べデータ数は少ないが平均のグラフサイズが大きく，頻出部分グラフマイニングから見られるように出力数が多いことから似たグラフが多いことがみてとれる．それに比べ CPDB ではグラフデータベースが疎であるためか，頻出部分グラフが Mutag に比べて少ない．

図 5.1 の左右の上図，中央図より，飽和・極大パターン集合を求めると通常の部分グラフパターンより時間がかかるが，出力はかなり抑えられていることが，実験結果よりわかる．さらに δ の指標により飽和と極大の集合間の出力を自由に決めることができる．今回の実験では，δ を変えた実験は追加していないが，原理上，飽和と極大の間になる．

図 5.1 の左右の下図より，Mutag と CPDB の両者に見られるように wildcard を許容した頻出部分グラフの飽和・極大パターン集合は頻出部分グラフの増加傾向ほど増えないため minsup をさげると通常の頻出部分グラフ数ほどの出力まで抑えられている．さらに Mutag は CPDB に比べ似たグラフが多いためその傾向が顕著にあらわれている．図 5.1 右の中央図からわかるように，CPUtime では出力数が増えた場合，枝刈りが有効で，wildcard を許容した頻出部分グラフマイニングに対して飽和・極大パターン集合のほうが時間がかからなくなることもある．

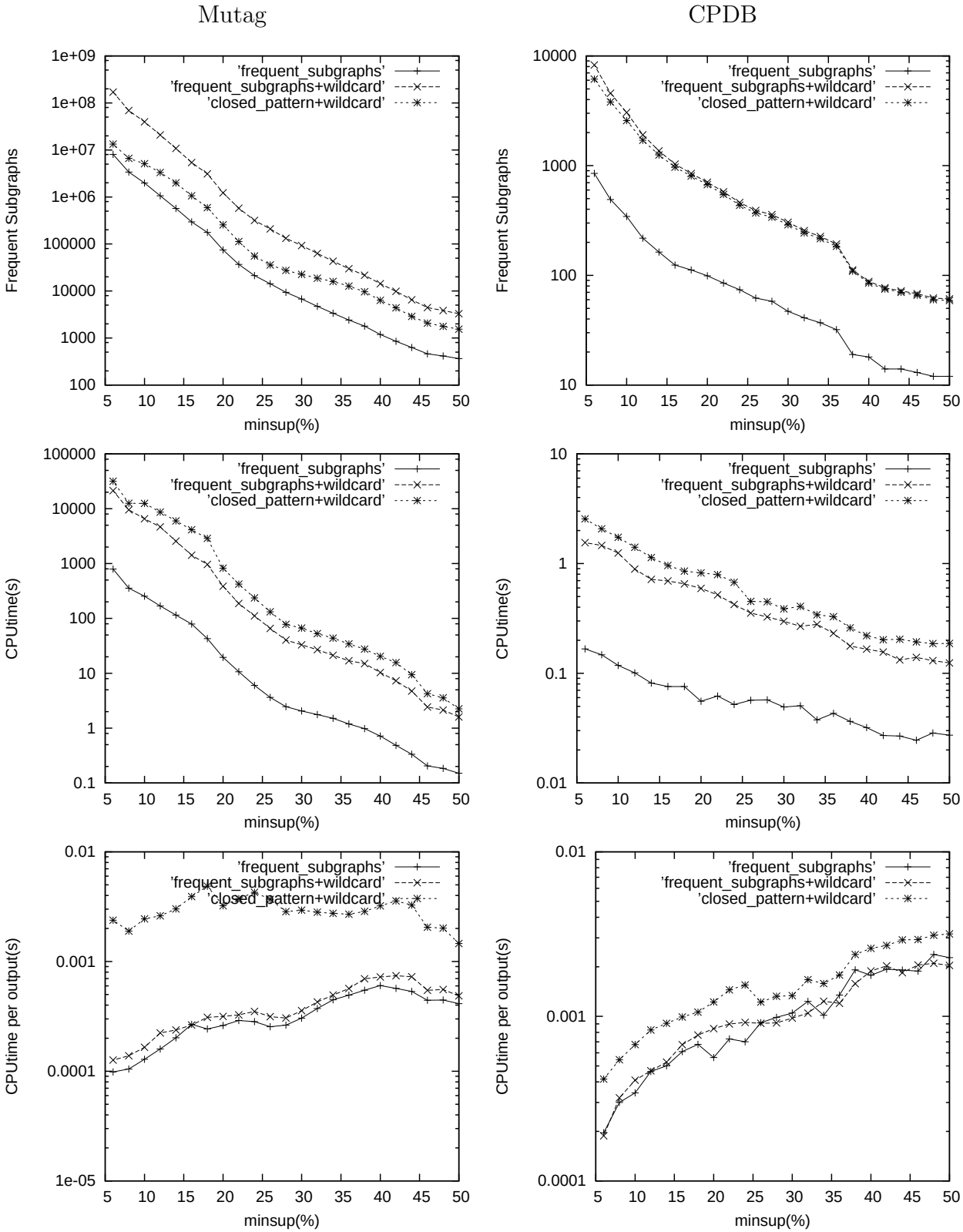


図 5.1: Mutag(左)CPDB(右) の実験結果：上から順に出力数，CPUtime，1 出力あたりの CPUtime

5.4 機械学習に対する応用実験

本節では，実験に用いたデータセットに対して，マイニングにより出力されたパターンの 0，1 ベクトルを 2 クラス分類の機械学習の特徴ベクトルとして機械学習を行う．さらにその精度を比べることで，wildcard を許容したパターンがどれほど有効であるかを検証する．機械学習として用いた手法は SVM，Random Forest である．

SVM(サポートベクターマシン) は 2 クラスの分類問題を解くための学習器である．SVM は線形の識別器で，カーネルを組み合わせることによって非線形に拡張することができる．これまで，文字認識や画像認識などの分野で利用され，高い識別能力を示している．

Random Forest は決定木を弱学習器とする集団学習アルゴリズムで，ランダムサンプリングされたトレーニングデータによって学習した多数の決定木を使用し分類器を形成する．Random Forest は説明変数 (ここでは特徴パターン) が多数であってもうまく動き，学習・評価が高速である．

5.4.1 実験準備

今回行う，2 クラス分類では，次のようなデータセット $D = \{(\boldsymbol{x}_i, y_i) \mid \boldsymbol{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}$ を与える．ここで，p は次元数 (特徴パターン数) を指す．

この特徴ベクトル $\boldsymbol{x}$ を通常の頻出部分グラフ，wildcard 許容頻出部分グラフそれぞれに対して求め，比較を行う．実験 1 として，膨大なパターン数を minsup で枝刈りすることで求める．これを 5.4.2 で，実験結果を示し，考察を行う．wildcard 許容に対して，出力増加の問題があることを 5.3 で示した．通常の頻出部分グラフに対する飽和・極大パターンによる機械学習は，本研究においては問題視しない．しかし，wildcard を含むパターンに対して，冗長なパターンを削除することは，大変重要な問題である．本論文において，wildcard に対する飽和・極大に対して，すでに存在しているパターンと冗長なものを削減する方法を 4.3.2 で提案した．実験 2 として，頻出部分グラフマイニングに対して wildcard を許容したパターンとそれに対して wildcard pruning による枝刈りのみを行ったパターンに対して，出力と出力時間の比較を行う．さらに，これに対して機械学習を行った精度の比較を行う．これを 5.4.3 で示し，考察を行う．

本研究では，定めた手法に対して 10-分割交差検証を行い，精度を比較する．まず，与え

られたデータセット D に対して１０通りの分割を行いトレーニングデータとテストデータに分ける．そして，それぞれのトレーニングデータに対して，通常の頻出部分グラフ，wildcard 許容頻出部分グラフを求め，グラフデータベース中のそれぞれのグラフにおいて，求めた特徴を持つか持たないかを表す 0，1 ベクトルを特徴ベクトル $\boldsymbol{x}$ とする．以上を入力とし，それぞれのデータセットに対して分類器を学習し，そのテストデータに対する正解率の平均値をその手法の精度としている．

使用した SVM のパラメータでは，$C = 1, 10, 100, 1000, 10000, kernel = linear$ と $C = 1, 10, 100, 1000, gamma = 0.001, 0.0001, kernel = rbf$ でグリッドサーチを行っており，時間の都合上 AIDS のデータでは決め打ちで C=1,gamma=0.001,kernel=rbf で実験を行った．

5.4.2　実験 1:結果と考察

図 5.2 に wildcard 許容に対する minsup を変動させた時の実験結果を示す．上段が Mutag，中央段が CPDB，下段が AIDS(CAvsCM) に対する実験結果であり，それぞれ左に SVM，右に Random Forest の実験結果である．

図 5.2 に示した実験結果のうち Mutag，AIDS(CAvsCM) は minsup を下げられる範囲のもので，これ以上さげると，使用した実行環境では実行できない．

図 5.2 上段の Mutag では，SVM に対して minsup35-50 では良い結果が出ている部分があり，Random Forest を見ると精度に差がない．Mutag データセットに関して，使用した実験環境で実行できる範囲で一番よい結果が得られているのは，wildcard を 1 つ許容した時で，minsup=42 の時である．Random Forest では，精度の最大値ではほぼ変わらないと言える．

図 5.2 中央段の CPDB では，いずれの手法も，minsup が 20 以上で良い結果が得られている．minsup を十分に下げるとあまり精度に差がないが，通常の頻出部分グラフパターンによる結果と同等か，それ以上の結果が出ている．さらに minsup=44 の時の結果がよく出ており，minsup を下げずに良い結果を得られていると言える．

図 5.2 中央段の AIDS(CAvsCM) では，wildcard を許容することにより求められる範囲が狭くなる上，精度が落ちていることがわかる．通常の頻出部分グラフパターンに対して見ると，minsup を下げることで精度が落ちているため，この 2 値分類問題に対しては

頻出部分グラフパターン自体があまり良い結果につながらない可能性がある．

以上の実験結果より，wildcard が有効であるデータベースとそうでないデータベースが存在するが，minsup を下げると精度が上がるような問題に対して，wildcard を許容することで，精度が下がることはなく，状況により精度が上がる．これは，データベースに対して，それらをうまく表現するような特徴としての候補を得ることができるためだと考えられる．つまり，頻出パターンによる機械学習の傾向が一般的な（Mutag や CPDB のように minsup を下げると基本的に精度が良くなる）場合には，wildcard を入れるのは有効であると言えるような実験結果となった．

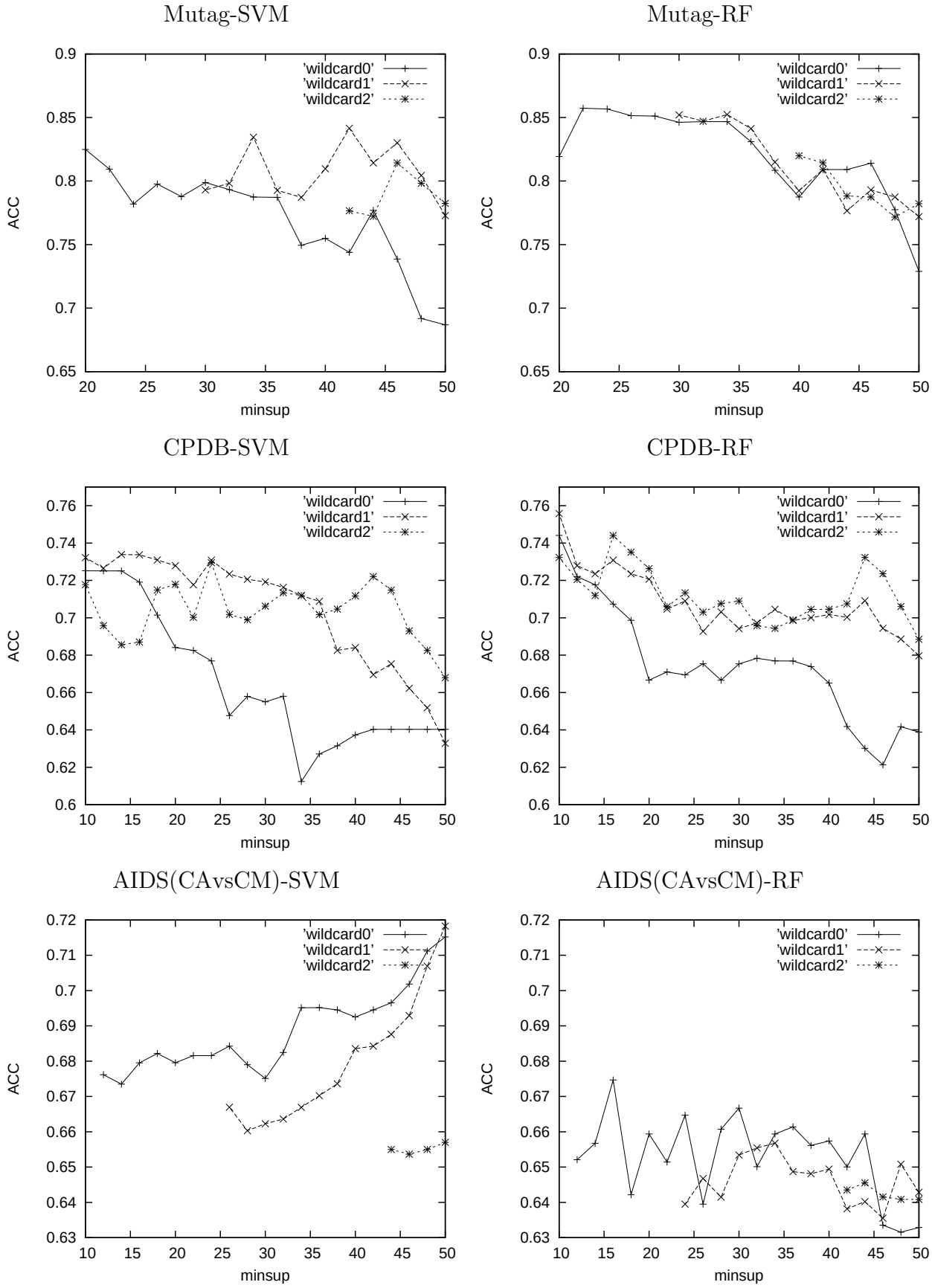


図 5.2: Mutag(上)CPDB(中央)AIDS(CAvsCM)(下) の minsup の変化と正解率の実験結果：左から順に SVM，Random Forest

5.4.3 実験 2:結果と考察

図 5.3 に wildcard 許容に対して wildcard pruning による枝刈りによる出力と出力時間の実験結果を示す．それぞれ左に Mutag，右に CPDB の実験結果で，上段が出力数，下段が出力時間に対する実験結果である．

図 5.4，図 5.5 に wildcard 許容に対して wildcard pruning による枝刈りによる特徴削減の実験結果を示す．図 5.4 が Mutag，図 5.5 が CPDB に対する実験結果である．それぞれ左に SVM，右に Random Forest の実験結果である．

図 5.3 左上より，Mutag では wildcard pruning による枝刈りによりパターン数が抑えられており，wildcard 許容頻出部分グラフに対して余計な処理を行った時間に対して，minsup を下げた場合に出力時間が抑えられていることが図 5.3 左下からわかる．

しかし，図 5.3 右側より，CPDB ではほぼ出力数は変化していないため，出力時間では元より少し時間がかかる結果となった．

一方で，機械学習に対する変化であるが，精度に対して，変動こそあるものの一方的に悪くなったり良くなったりしているわけではなく，誤差の範囲と言えるだろう．

また，Mutag に関しては，図 5.4 より，出力が抑えられるため，元より minsup を下げたところまで求めることができ，良い結果が出ている部分もある．

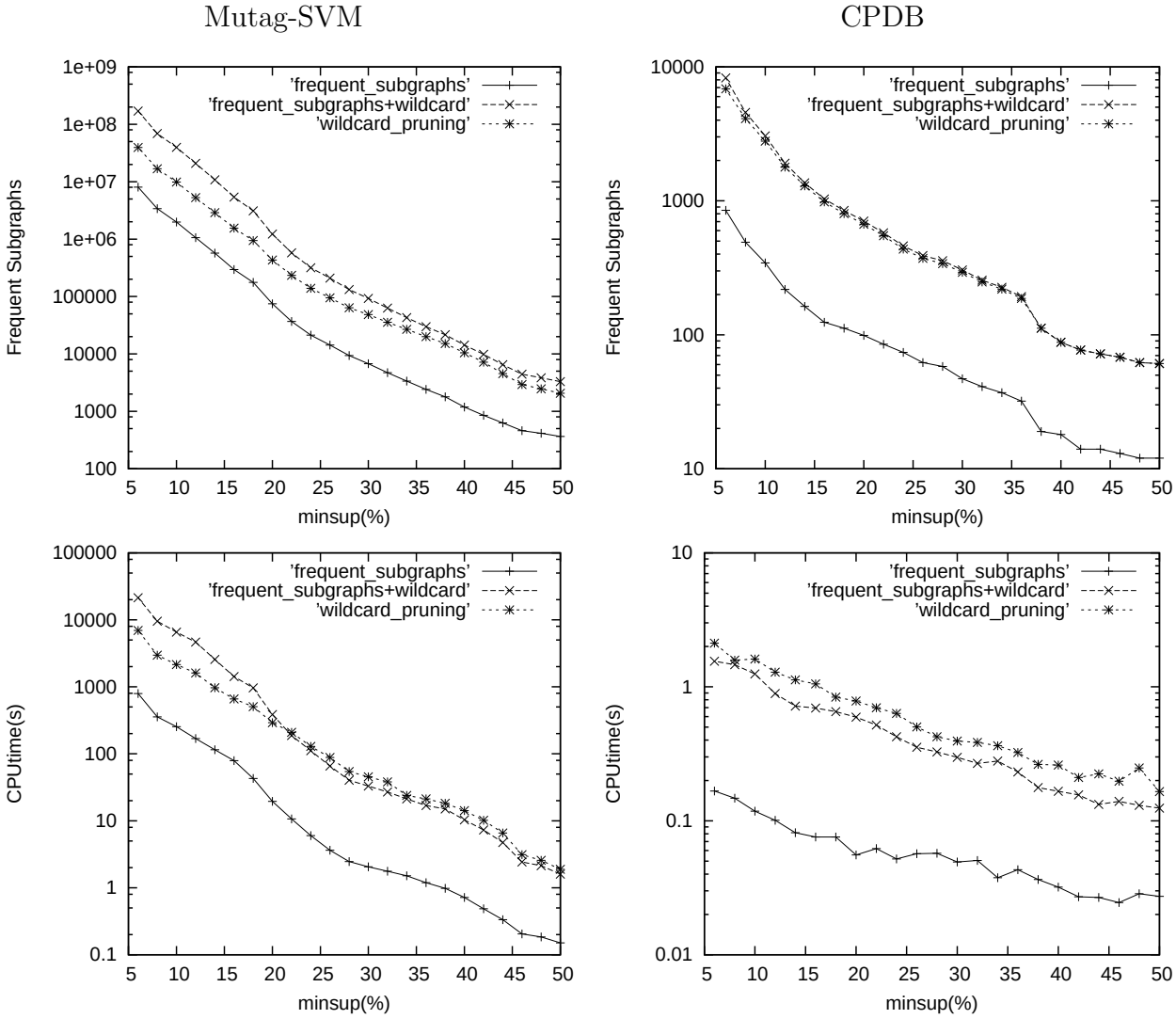


図 5.3: Mutag(左)CPDB(右) の wildcard pruning による出力と出力時間の実験結果：上から順に出力数，出力時間

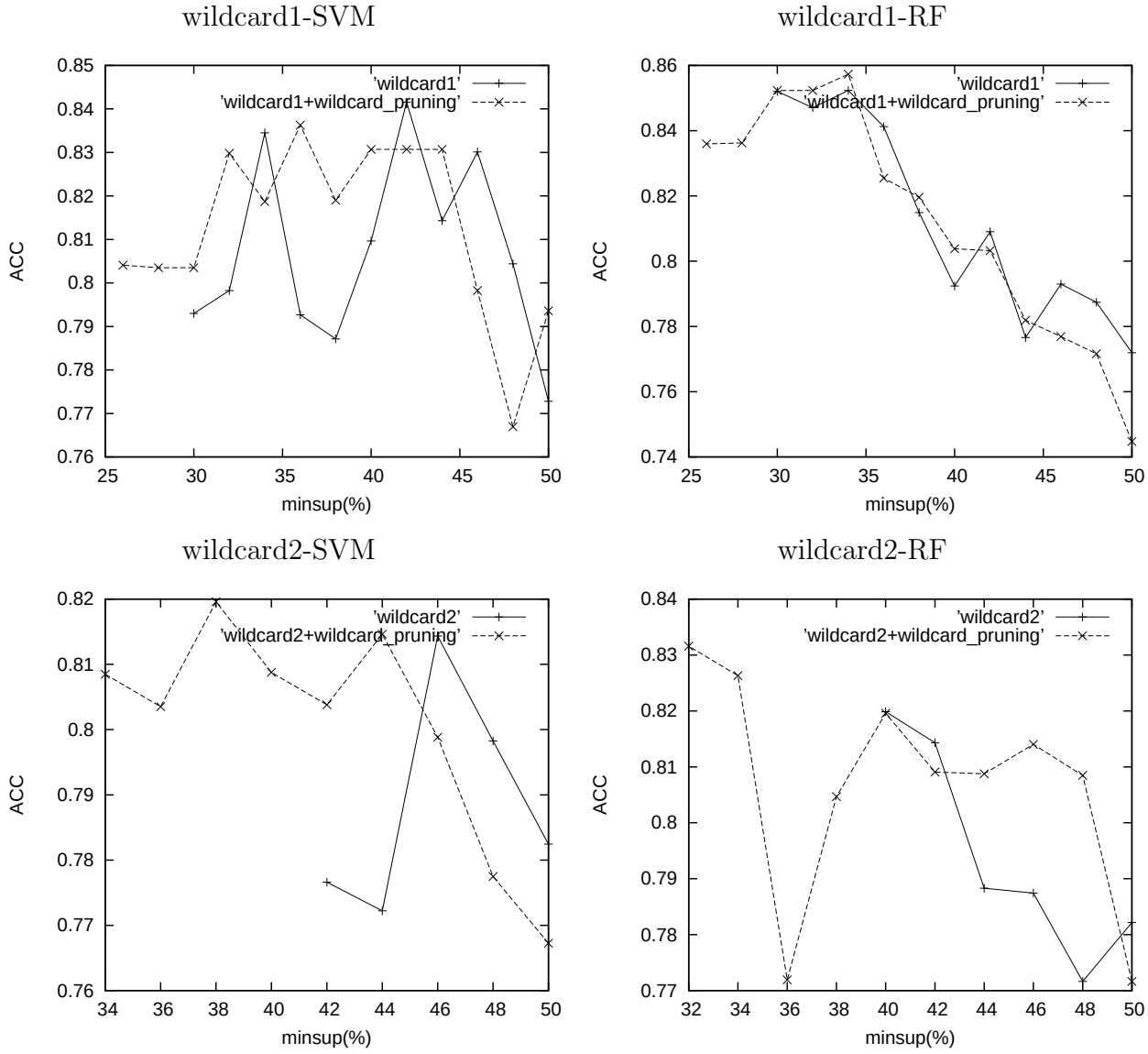


図 5.4: Mutag の wildcard pruning による特徴削減の実験結果:左から順に SVM，Random Forest

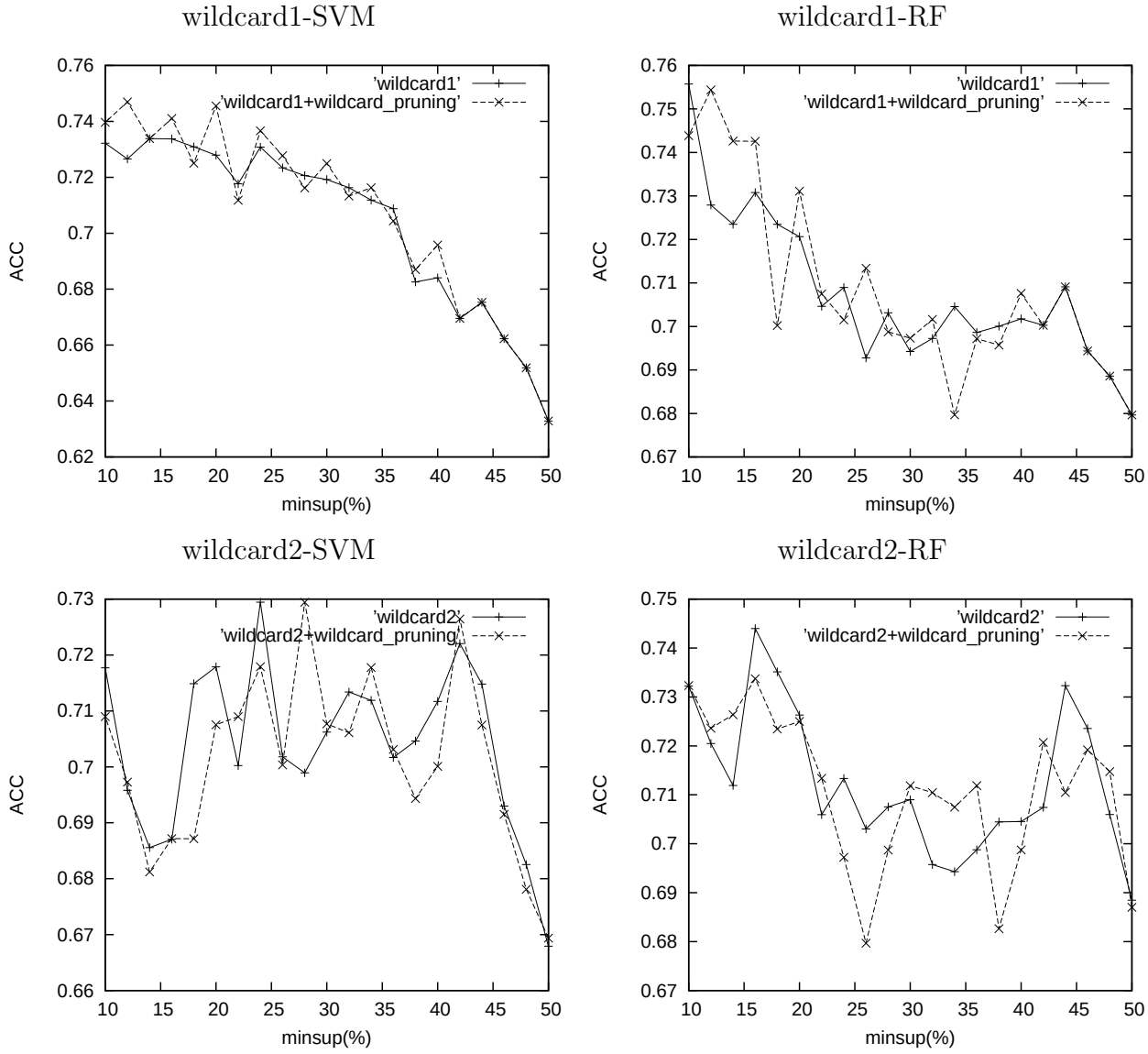


図 5.5: CPDB の wildcard pruning による特徴削減の実験結果:左から順に SVM，Random Forest

第6章

まとめ

6.1 結論

本研究では，頻出部分グラフマイニングのアルゴリズムである gSpan，さらに δ-tolerance closed frequent subgraphs algorithm を拡張し，wildcard を許容した頻出部分グラフパターンの列挙とその飽和パターン集合・極大パターン集合を求める手法を提案した．さらに実験から，wildcard を許容することによる頻出パターン数の増加を示し，飽和パターン・極大パターンによるパターン集合要約の効果を確かめた．さらに，wildcard を許容した頻出部分グラフが実際に有用なパターンであるかを，機械学習に対する特徴として用いることで効果検証した．その結果，グラフデータセットによっては効果的な出力があり，wildcard を許容に対して有効であると言える傾向実験結果を得た．また，冗長なパターンを削減することで，精度を下げずに，よりうまく応用できる．

6.2 今後の展望

今回検証した実験に対して，さらに実験を行うことで今回得た結果に対する考察の信憑性を確かめる必要があるだろう．また，今回は正解率による検証を行ったが，ここからグラフをよりうまく分類できている特徴を見ることで，wildcard を許容した部分グラフがどれほど選ばれているのか調べる必要がある．さらに，通常，minsup で枝刈りをする方法は，データベース中のすべてのグラフをうまく説明できないため，十分に minsup を下げなければ機械学習の特徴としてはあまり精度がでない．minsup では枝刈りをせず，部分グラフのエッジ数で決まるパターンの長さで枝刈りをする方法もあり，これに対し

ても検証を行ったが，うまく説明できるような結果でなかったため，さらに実験をし，傾向を確認したい．

謝辞

本研究を進めるに当たり，本研究に関する専門的な知識やアイデア，研究の方向性，論文の構成から体裁に至るまで，多大な助言・ご指導をいただきました瀧川一学准教授，また技術・知識だけでなく，日常の学生生活において多大なご指導をいただいた湊真一教授に深く感謝いたします．また，本研究に関して議論を交わし，多くの助言をいただいた ERATO 研究員の皆様，研究室の皆様，研究活動を支えてくださいました家族に深く感謝いたします．また，本研究に当たって様々な知識・技術に関するご指導をいただいたすべての皆様に感謝を申し上げます．

参考文献

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pp. 151–158, 1971.

[2] Alfredo Ferro, Rosalba Giugno, Misael Mongiovì, Alfredo Pulvirenti, Dmitry Skripin, and Dennis Shasha. Graphfind: enhancing graph searching by low support data mining techniques. *BMC Bioinformatics*, Vol. 9, No. S-4, 2008.

[3] Rosalba Giugno and Dennis Shasha. Graphgrep: A fast and universal method for querying graphs. In *16th International Conference on Pattern Recognition, ICPR 2002, Quebec, Canada, August 11-15, 2002.*, pp. 112–115, 2002.

[4] Yu He, Xindong Wu, Xingquan Zhu, and Abdullah N. Arslan. Mining frequent patterns with wildcards from biological sequences. In *Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2007, 13-15 August 2007, Las Vegas, Nevada, USA*, pp. 329–334, 2007.

[5] Heiko Hofer, Christian Borgelt, and Michael R. Berthold. Large scale mining of molecular fragments with wildcards. *Intell. Data Anal.*, Vol. 8, No. 5, pp. 495–504, 2004.

[6] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pp. 549–552, 2003.

[7] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm

for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, pp. 13–23, 2000.

[8] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, Vol. 50, No. 3, pp. 321–354, 2003.

[9] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*, pp. 313–320, 2001.

[10] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pp. 647–652, 2004.

[11] Ichigaku Takigawa and Hiroshi Mamitsuka. Efficiently mining δ-tolerance closed frequent subgraphs. *Machine Learning*, Vol. 82, No. 2, pp. 95–121, 2011.

[12] Xindong Wu, Xingquan Zhu, Yu He, and Abdullah N. Arslan. PMBC: pattern mining from biological sequences with wildcard constraints. *Comp. in Bio. and Med.*, Vol. 43, No. 5, pp. 481–492, 2013.

[13] Fei Xie, Xindong Wu, Xuegang Hu, Jun Gao, Dan Guo, Yulian Fei, and Ertian Hua. Sequential pattern mining with wildcards. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1*, pp. 241–247, 2010.

[14] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 721–724, 2002.