卒業論文

系列二分決定グラフに基づく
文からのパターン抽出に関する研究

電気情報工学科

1TE11191R 平野大貴

指導教員 田中久美子 教授

平成 27 年 2 月

概 要

インターネット技術の発達により，最近ではある単語の意味を調べる際に紙の辞書ではなく，オンライン辞書で検索を行う人も少なくない．オンライン辞書で単語の意味を調べる利点として，紙の辞書よりも手軽に調べられることはもちろん，ネットワーク上に存在する様々なテキストデータを利用することで，その単語の意味と同時に用例を調べることができる点がある．しかし，膨大な量でかつ未整理なテキストデータから，利用者自身が文章の内容を吟味し，必要な用例のみを取捨選択することは到底不可能である．そこで，有用な手段として，それらのテキストに対して必要な用例のパターンを機械的に抽出するという方法が挙げられる．本研究では，利用者が検索する単語に対して，テキスト群から，その単語の前後に出現する可能性の高い単語をパターンとして抽出し，それらを利用者に単語の用例として提示することを目的としている．

論理関数を効率よく表現できるデータ構造の一つに二分決定グラフ (*Binary Decision Diagram: BDD*) がある．近年では，PC の主記憶量の増加に伴って，BDD がデータマイニングや知識発見といった分野でも効果的に活用できることが分かってきた．その中でも，系列二分決定グラフ (*Sequence BDD: SeqBDD*) と呼ばれる，大規模な系列データを効率よく演算処理することができるデータ構造がある．本論文では，既存の SeqBDD の構築方法を応用することで，文章を入力とする SeqBDD の構築を行い，そのデータ構造からパターンを抽出する手法を提案する．本稿におけるパターンの定義とは「複数の単語からなり，ワイルドカード部

を含み得る一連の頻出表現」である．

文章をデータ構造に変換してパターンを抽出しようとする際に，既存の手法では Trie 構造などの末端部分がデータ数に応じて増加し，データ構造の縮退を行うことが難しかった．しかし，SeqBDD を用いて文章を構造化することによって，各文章の同一部分の共有や余分な表現の削除を行うことが可能となり，文章群のよりコンパクトな表現を得る．本稿では，構築した SeqBDD の画像出力を行うことでグラフの可視化を行い，データ構造を実際に目で見て確認することを可能にした．さらに，構築した SeqBDD からパターンの抽出を行うために，グラフの枝のランク付けやそのランクを用いた枝切り，節点となる単語表現の圧縮や枝表現の変更といった，データ構造の改変などの工夫を行った．これらの処理によって，パターン抽出を単純な工程で行うことが可能となる．

今回，目的とするパターンのうち，ある単語の後ろに頻出するパターンを獲得する問題に限定し，評価実験を行った．実験データは「British National Corpus」(略称，BNC) という電子データベースから英語の例文を，検索語および評価パターンは「TheFreeDictionary.com」というサイトから英語のイディオムを収集した．検索語であるイディオムを含む例文を BNC より抜き出し，イディオムの前で区切った例文を入力として SeqBDD を構築することで，検索語とその後ろに続く単語の頻出パターンを抽出し，正解パターンと照合を行う．単純適合率 (L)，平均逆順位 (MRR)，平均適合率 (AP) という指標を用いて評価を行い，$L = 0.84,\ MRR = 0.394,\ AP = 0.838$ という結果を得た．ワイルドカード部を持つ穴あきのパターンの抽出にも成功しており，作成したモジュールの改良やパターンの抽出方法の再考などの問題を残しているが，本研究の可能性を示せたのではないだろうか．本研究の発展に向けて，更なる実験，および評価を進めていきたい．

目 次

第1章

はじめに

1.1 研究背景

近年のインターネットやWeb技術の発達により，ネットワーク上には膨大な量の系列データが存在しており，それらを対象として有益な情報を抽出するデータマイニングの研究が盛んに行われている．その中でも，テキストは語の並びで構成された系列データの一種であり，文字列テキストに適用するアプローチは，特にテキストマイニングと呼ばれている．テキストマイニングの主な特徴は，頻度や共起性にあり，頻度の高いパターン（語の並びやフレーズ）を抽出し，文章の要約や重要な事象のラベル付けを行うことができる．今後テキストデータは更に増え続けると予想され，有益な情報を取り出す頻出パターンの抽出に関する研究が必要となる．

また，最近ではある単語の意味を調べる際に，オンライン辞書を用いて検索を行う人も少なくない．オンライン辞書を利用する利点としては，ある単語の意味を調べる際に，その単語の意味と同時に，その単語の用例をネットワーク上のテキスト情報から検索できる点が挙げられる．その際に，ネットワーク上に存在する莫大な量のテキストデータからその単語に関する用例情報を取捨選択する必要があるが，利用者がそれを行うのは到底不可能である．そこで，有用な手段としてそれらのテキストに対して機械的にマイニングを行ってパターン

を抽出するという方法がある．しかし，膨大なテキスト量ゆえに探索空間が巨大となるため，テキストをよりコンパクトなデータ構造で表現する必要がある．そこで本研究の目指す，「大量のテキスト群をよりコンパクトなデータ構造で表現し，そこから頻出パターンを抽出すること」という目的が生まれる．

ここで，本稿におけるパターンの定義を「複数の単語からなり，ワイルドカード部を含み得る一連の頻出表現」とする．このパターンを抽出する目的で，本稿ではテキストデータを，二分決定グラフ (Binary Decision Diagram: BDD) と呼ばれるデータ構造に変換する．近年では，家庭用の PC でも相当量の主記憶を備えているため，従来では解くことが不可能だった大規模な問題（データマイニングや知識発見といった分野）を BDD によって扱うことができるようになった．特に，BDD のデータ構造を応用した系列二分決定グラフ (Sequence BDD: SeqBDD) を用いることで，系列データを論理関数や集合データとして考えることができ，大規模な系列データ集合を，効率よく演算処理することが可能になった．既存手法では、図 1.1 の左の図に示すように、文章群を Trie 構造などのデータ構造に変換し，同一部分を共有しようとする際に，裾となる構造の末端部分がデータ数に応じて増加し，うまく縮退させることが出来なかった．しかし、SeqBDD を用いることで図 1.1 の右図のようにデータ構造の縮退が可能になるのではないか，また，同時に SeqBDD によってコンパクトに表現された文章群からのパターンの抽出も可能となるのではないかと考えた．このような考えに基づき，任意の語句が与えられた際に，SeqBDD を用いることで教師データを必要とせず入力となる文章群のみから，その語句の前後に出現しやすいパターンを抽出する，という手法を提案する．これによって，ある単語の使用頻度の高い用例を，膨大な量のテキスト群からパターンとして抽出することにより，検索を行う利用者に提示することができる．テキストからの精密なパターン抽出は，既存の手法では係り受け解析と言った構文解析が必要となっていたが，提案手法では教師なし手法によって実現することができる．また同時に，木構造に変換した構造テキストデータの末端部分の共有を実現することで，有用な頻出部分構造抽出も可能となる．

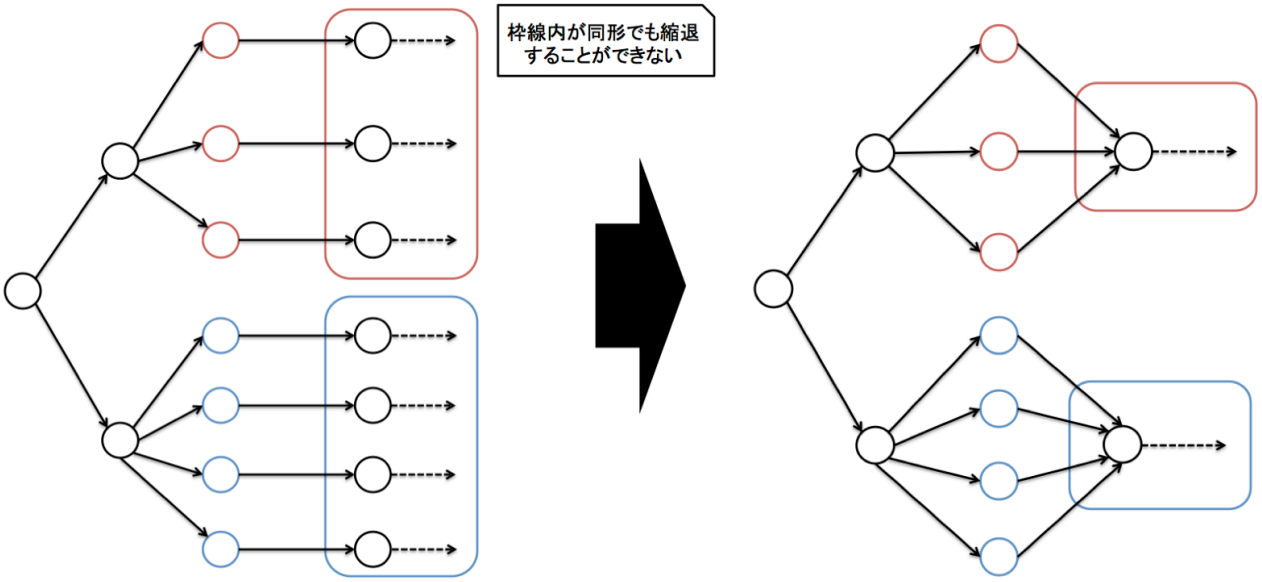


図 1.1: SeqBDD を用いることで実現するデータ構造の縮退

1.2 目的

本研究では，SeqBDD を用いることで，既存手法では不可能だった文章群をデータ構造に変換した際の末端部分の縮退を実現し，入力となる文章群をよりコンパクトな構造で表現し，可視化することでその構造の有用性を確認すること，また，構築した SeqBDD からパターンとなる語句群を抽出し評価することを目的としている．

大きく分けて２つの処理を通して，文章からのパターン抽出を行う．１つは，系列データであるテキスト文章を入力とした SeqBDD を構築することである．この実現を基に，様々な節点の作成方法の検討や，節点ラベルの圧縮，余分な枝切りなどを行いデータ構造を縮退させ，よりコンパクトなデータ構造を目指す．また，SeqBDD を可視化するために，pydot と呼ばれる，Graphviz を Python に対応させたパッケージを用いて，画像ファイルとして出力をする．もう１つは，構築した SeqBDD からパターンを抽出する処理である．SeqBDD の構築を行う際に，よりパターンを抽出しやすいデータ構造にするために手を加えるため，比較的単純な処理によって実現することが可能となっている．

これらの作業をすべて機械的に行うため，Python を用いてプログラムを実装する．実験に

は，英語の例文とイディオムを用いて頻出パターンを抽出し，それらを正解となるパターンと照合する．これらによって得られた実験結果を単純適合率，平均逆順位，平均適合率という 3 つの指標を用いることで評価を行う．

1.3　本稿の構成

本稿はこれ以降，4 つの章によって構成されている．第 2 章では，本研究と深く関係する研究や論文を紹介する．第 3 章では，本研究で提案する SeqBDD を用いた文からのパターン抽出手法を，基本となるデータ構造の説明や検討した様々な手法と共に記載している，第 4 章では，第 3 章で紹介した手法を用いて実際にパターンの抽出を行い，それによって得られた結果，および考察を示す．第 5 章では，それらを踏まえた本研究の結論と今後の課題について述べる．

第2章

関連研究

本研究と大きく関連する研究に，以下に示すパターン抽出，教師なし構文解析，用例検索が考えられる．また提案手法の準備 (第 3.1 節) のデータ構造の説明にも関連研究を記載している．

2.1 パターンの抽出に関する研究

現在，データマイニング手法を用いたパターン抽出は，コンピュータ関連の研究分野において非常に活発に行われている．その中でも，90 年代前半に Agrawal らが提案した頻出集合列挙問題 [1] と，その効率的な解決方法である Apriori アルゴリズム [2] はデータマイニングの主要技術の一つとして挙げられる．Apriori は「長さ k の頻出でないパターンを含む長さ k+1 のパターンは頻出ではない」という仮説に基いて，ボトムアップに頻出するパターンを抽出するアルゴリズムである．また，Apriori のアルゴリズムを基に提案された GSP[3] と呼ばれるアルゴリズムがある．これは，Apriori の考えをそのまま系列データに対して応用し，頻出系列を抽出するという手法である．しかし，Apriori，GSP のアルゴリズムでは候補となるパターンが膨大な数になってしまうという欠点がある．この欠点を改善するために，SPADE[4] が提案された．SPADE は束 (Lattice) という概念を用いて，候補となる系列をグループに分

割し，ID-List というデータ構造を用いることで計算コストの削減を行った．さらに SPADE を改良した SPAM[5] というアルゴリズムが提案された．SPAM では ID-List をビットマップを用いて表現することで，さらなる計算速度の向上を図った．

一方で，候補となる系列を生成せずにデータベースを射影することによって頻出系列を抽出する PrefixSpan[6] と呼ばれる手法が提案された．PrefixSpan アルゴリズムを用いた簡単なパターン抽出の動作例を図 2.1 に示す．この例では，4 つの系列 [“abc”, “aab”, “acd”, “cba”] から長さが 1 以上，出現回数が 2 回以上のパターンを抽出している．まず，系列を成す各文字を長さ 1 のパターンであると考え，それぞれの系列における出現回数を数え上げる．ここで d は出現回数の条件に反するため，パターン候補から除外され，残りの “a”,“b”,“c” について，それらの後続の文字列だけを取り出す「射影」を行う．その後，射影されたデータに対して再び出現回数を数え同様の処理を行う．“a” の射影データでは “b”,“c” がそれぞれ 2 回ずつ出現しているため，これらの直前の文字である “a” と結合して長さ 2 の “ab”,“ac” というパターンを抽出する．パターンの探索を繰り返し，新たなパターンが発見されなくなると終了する．

しかし，PrefixSpan を用いてテキストからパターンを抽出しようとする場合，抽出する過程で入力となるデータ毎に射影データを作成する必要があるため，多大な計算時間を必要としたり，無意味なパターンを数多く抽出してしまうという問題が存在する．また，ワイルドカード部を含むものとそうでないものを区別することが出来ないという問題もある．図 2.1 のパターン抽出例で，結果の “ac” は，入力データ “acd” では $< ac >$，“abc” では $< a * c >$[*1] という異なる出現の仕方をしているのがこれに当たる．そこで，Modified PrefixSpan[7] では，頻出パターン中に存在する複数のワイルドカード領域 “∗” の長さを予め設定することで PrefixSpan アルゴリズムの改良を行った．

[*1] 本稿では “∗” はワイルドカード部を表す

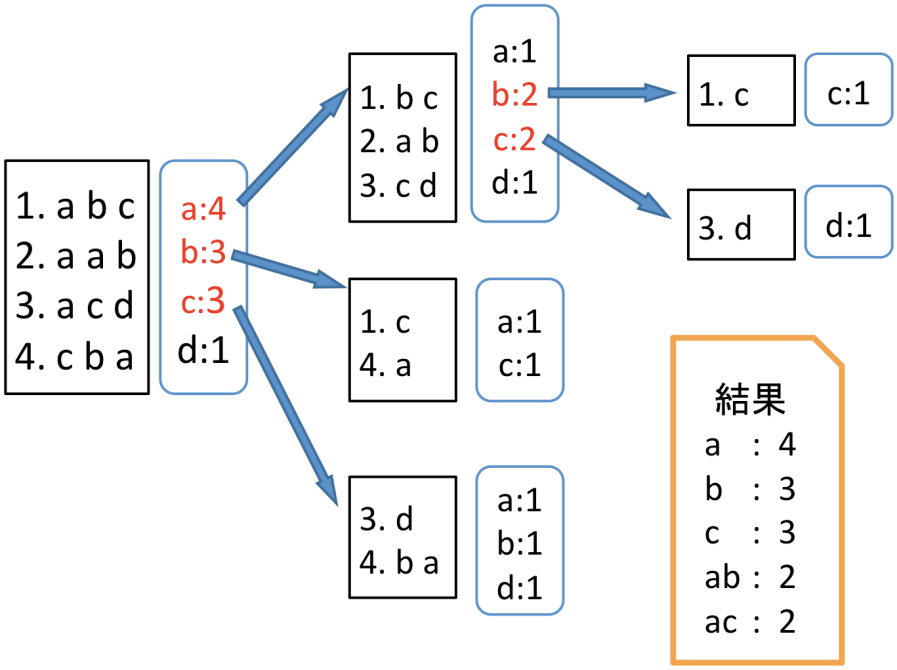


図 2.1: PrefixSpan を用いたパターン抽出例

テキストを対象としたパターン抽出では，CRYSTAL[8] という，注釈付きデータを含むトレーニングコーパスを用いた機械学習によって概念辞書を作成し，これを用いて生の文章からパターンの抽出を行うというシステムの研究も行われた．しかし，この手法では正確な構文規則に基づく文章からしかパターンの抽出を行うことが出来ず，柔軟性が高いとは言えない．また注釈付きデータの作成などの人的コストも高くなる．近年のネットワークや Web 技術の発達により，ネットワーク上には膨大な量のテキスト表現が増え，それに伴って煩雑な構文を含むテキストも増えた．それらに対応するために行われた研究に WHISK[9] というものがある．これは，教師あり機械学習によって正規表現によるパターン抽出を行うことでより柔軟なパターンの抽出を可能とした．

既存のテキストからのパターン抽出方法では，教師なし手法では正確なパターンの抽出を行うことは難しく，またパターンの抽出に重点を置くと，文章の係り受け解析などの構文解析が必要となる．本研究では，教師データを必要としないパターンの抽出を，後述する SeqBDD[10] と呼ばれるデータ構造を用いて行う手法を提案する．

2.2　教師なし構文解析に関する研究

教師なし構文解析の基本的なアプローチとして，PCFG（確率的文脈自由文法）に文章を変換した上で，手を加えて行く手法がある．代表的なものに Klein，Manning ら [11] の手法がある．しかし，自然言語を厳密に解析するために必要となる文法は，通常，非常に複雑となってしまうため人の手によって文法を書いて与える必要がある．また，PCFG の文法カテゴリの加工処理を自動的に行う方法として，Goldwater らの階層ディリクレ過程による単語のバイグラムモデルを用いた方法 [12] がある．また MDL 原理を用いて，文字のチャンキングを繰り返す方法 [13] による，統計的機械学習と考えた研究も行われている．

Pattern Grammar[14] では，頻出する単語に関して，前置詞や熟語，節などの言い回しをパターンとして扱い，言葉の意味を理解しようとするパターン文法が説明されている．ここでは，単語のパターンとは，単語とその意味が必ず関係づけられている構造だと定義できると述べられている．そのため，同時にかつ頻繁に出現したり，特定の言い回しであったり，明らかな意味関係が存在するような単語に関してはパターンを特定することができる．この考えを基に，本研究では教師なしの構文解析を，SeqBDD というデータ構造を用いることでパターンの抽出に拡張し，ある単語の前後の頻出単語のパターンを抽出する方法を提案する．

2.3　用例検索に関する研究

用例検索とは, ある単語を与えたときに, その単語の用例を頻度の高いものから列挙するという検索技術であり, 系列マイニングの一種である．用例検索に用いられる技術に KWIC(keyword in context) システムと呼ばれるものがある．これは，テキストからある単語を検索する際に，その前後の文脈も同時に取り出すことで検索効率を高めるという手法である．KWIC を用いることで，ある単語とその単語の前後の単語を，出現頻度を用いてパターン抽出することが，単純な数え上げによって行うことができる．しかし，パターンとなりうる形や同じ単語でも異なるパターンであることを計算機自身に判別させることが難しく，ある単語の前後のパターンを抽出する技術は自動化にまで至っていない．

また，用例検索を行うことのできる技術の一つに，田中らが提案した Tonguen[15] と呼ばれるツールがある．これは利用者がある単語群パターンをワイルドカード部を含むパターンの枠と穴を用いて与え，そのパターンの用例のうち，穴の部分に出現する可能性の高い単語をインターネット上からリアルタイムで取得し，利用者に提示するというものである．例として，利用者が「how to ＊ jet lag」と検索すると，ワイルドカード部である＊に出現する可能性の高い単語 (avoid, beat, prevent…) を得ることができる．

本研究では，ある単語を与えた際に，その前後に頻出する単語のパターン抽出の自動化を SeqBDD というデータ構造を用いて実現する手法を提案する．

第3章

提案手法

文章群からパターン抽出を行うに当たって，まず文章の構造化を行う必要がある．本章では，系列データマイニングなどに用いられている系列二分決定グラフ（Sequence BDD: SeqBDD）を用いて文章群の構造化を行い，パターン抽出を行えるようデータ構造の調整，改変を行う手法を論じる．

3.1 準備

本研究で用いたデータ構造である系列二分決定グラフ（SeqBDD），およびその基礎となっているデータ構造である二分決定グラフ（BDD），ゼロサプレス型二分決定グラフ（ZDD）の説明を以下で行う．

3.1.1 BDD（二分決定グラフ）

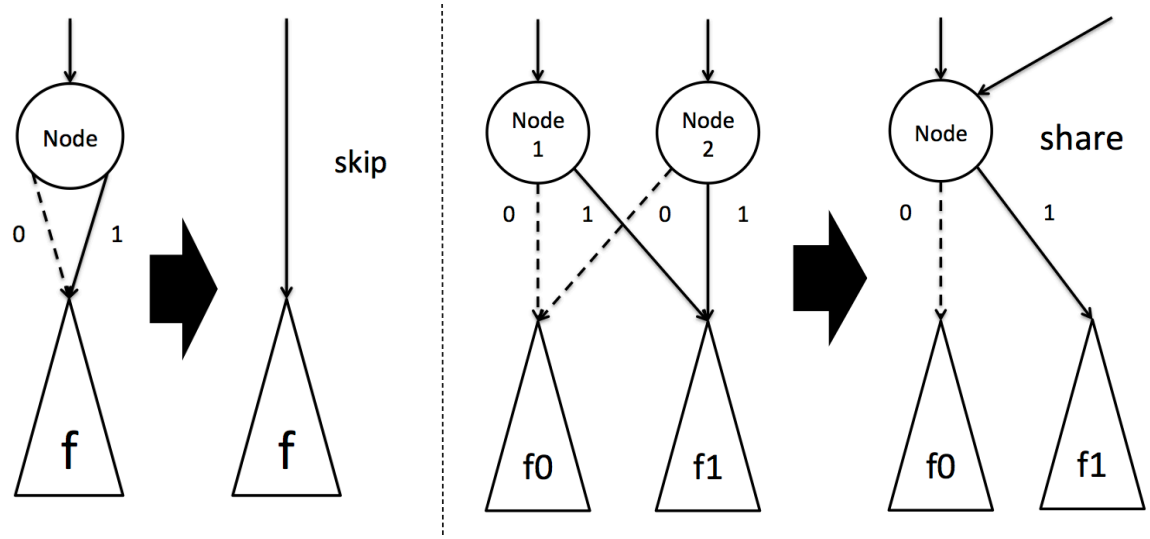
二分決定グラフ (Binary Decition Diagram: BDD)[16] は，グラフ構造を用いて論理関数を表現する方法の一つである．ある論理関数のそれぞれの変数値について，入力（0, 1）に対する場合分けをそれぞれ 0-枝，1-枝で表す．これによって，すべての変数値についての場合分

けを二分決定木を用いて表すことができる．さらに，この二分決定木を縮約した形にするために以下の

- 冗長な節点の削除
- 同等な節点の共有

という 2 つの縮約規則を用いる．

1 つ目の「冗長な節点の削除」について，ある節点から伸びる 0-枝と 1-枝の両方が同じサブグラフを指している場合，その節点と両枝を，1 つの枝に置き換えても表現は同一であるため，それらを省略することができる．(図 3.1(a) を参照 [*1]) 2 つ目の「同等な節点の共有」について，ある 2 つの節点から伸びる 0-枝と 1-枝がそれぞれ同じサブグラフを指している場合，その節点を同等な節点とみなして，共有することでデータ構造を縮約することができる．(図 3.1(b) を参照) これら 2 つの規則を，二分決定木に対して可能な限り適用することで縮約された形の木を得ることができる．



(a) 冗長な節点の削除　　(b) 同等な節点の共有

図 3.1: BDD の 2 つの縮約規則

[*1]図において，丸が節点，矢印がそれぞれの枝，三角形がサブグラフを表している

このようにして作られる木で，変数順序が固定されているものを本論では BDD と呼ぶ．ここで，変数順序について説明を加える．BDD における変数順序とは，入力となる論理関数の各変数について，予め定めておく順序関係のことを指す．この順序関係を用いて，変数順序の小さいものから節点を作成していくことになる．例えば変数順序を

$$a \prec b \prec c$$

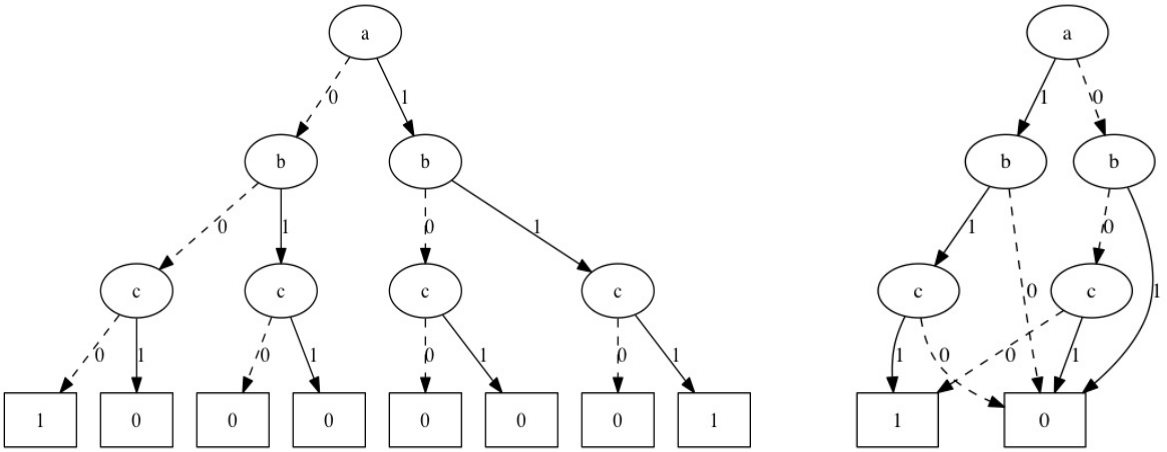
と定めた場合，まず変数 a に関する節点を作成し，続いて b, c の節点，と作成していく．

BDD は次のような性質を持つデータ構造である．基となる二分決定木のどの部分から縮約を行っても，同一の論理関数であれば出力される形は同じとなる．つまり，BDD を用いることで論理関数に対して一意に表現することができるため，等価性検証を容易に行うことができる．また，異なる BDD を入力とし，それらの BDD についての二項論理演算の計算結果である BDD を出力する「Apply 演算 [17]」と呼ばれるアルゴリズムが考案されている．このアルゴリズムによって，データサイズが主記憶の大きさに収まる限り，論理関数同士の演算をグラフの大きさに比例する計算時間で実行することができる．[18]

BDD の具体例として，論理関数 F について

$$F(a,b,c) \;=\; abc \vee \bar{a}\bar{b}\bar{c}$$

とした場合の，二分決定木と BDD の例を図 3.2 に示す．この関数に対して，各入力変数に固定値を代入した際の関数の値を得たい場合，節点のラベルとなっている各変数に対して，真の場合は 1-枝に，偽の場合には 0-枝に進んでいき，辿り着いた終端節点の値を確認すればよい．例として，F(1,1,0)[a=1,b=1,c=0] という関数の値を得たい場合には，節点 a の 1-枝→節点 b の 1-枝→節点 c の 0-枝，と辿り結果として 0-終端節点に行き着くため，F(1,1,0)=0 を得ることができる．



(a) 二分決定木　　(b)BDD

図 3.2: 木による論理関数の表現例

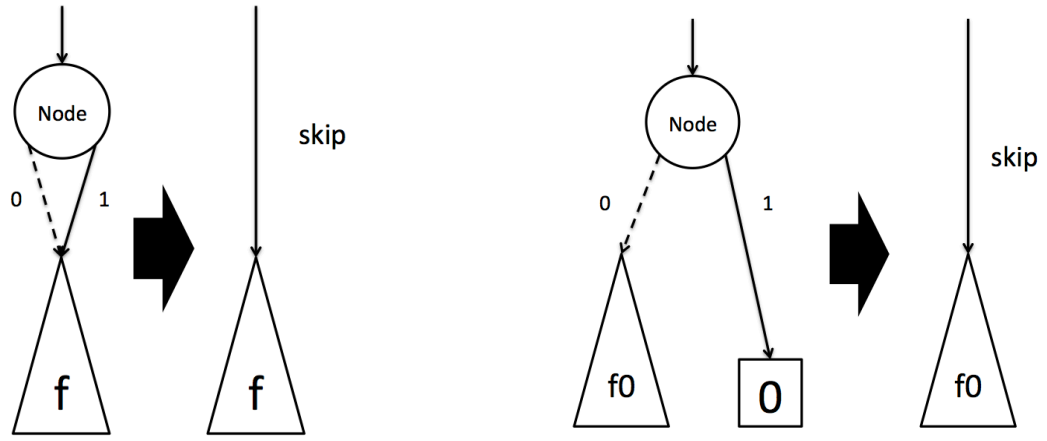
3.1.2　ZDD（ゼロサプレス型 BDD）

ZDD（Zero-suppressed Binary Decition Diagram：ゼロサプレス型二分決定グラフ）[19] は，組み合わせ集合を効率よく表現することが出来るデータ構造である．ZDD では，冗長な節点の削除をするための縮約規則が BDD とは異なる．BDD では，0,1-枝が同じ節点を指している際に節点を取り除いていたが，ZDD では 1-枝が 0-終端節点を指しているときのみに，その節点を冗長として取り除く．図 3.3 に BDD と ZDD の冗長な節点の削除に対する縮約規則の比較を示す．同等な節点の共有に関する規則は BDD と同じである．

よって，ZDD の 2 つの縮約規則は以下のようになる．

- 1 -枝が 0 -終端節点を直接指している節点の削除
- 同等な節点の共有

ZDD における根から終端節点への 1 つの経路は，1 つの組み合わせを表している．1-終端節点に辿り着く経路の 1-枝を持っている節点のラベル（変数）の集合を，1 つの組み合わせ集合として扱うことができる．図 3.4 に，図 3.2 と同じ論理関数を表した ZDD を示す．



(a)BDD の節点削除規則　　(b)ZDD の節点削除規則

図 3.3: BDD と ZDD の縮約規則の比較

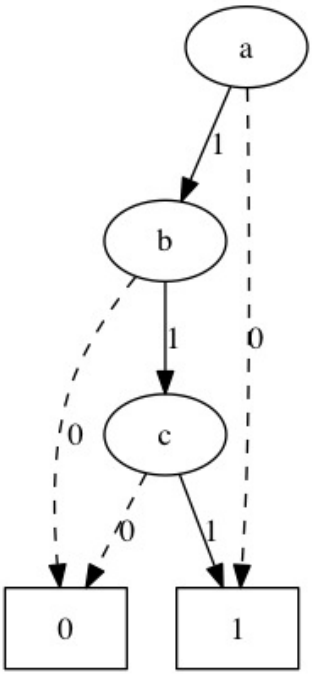


図 3.4: 論理関数 abc$\vee\bar{a}\bar{b}\bar{c}$ の ZDD

論理関数 F(a,b,c)=abc$\vee\bar{a}\bar{b}\bar{c}$ を組み合わせ集合と考えた場合，集合 S とすると

$$S = \{\{\varepsilon\},\ \{abc\}\}$$

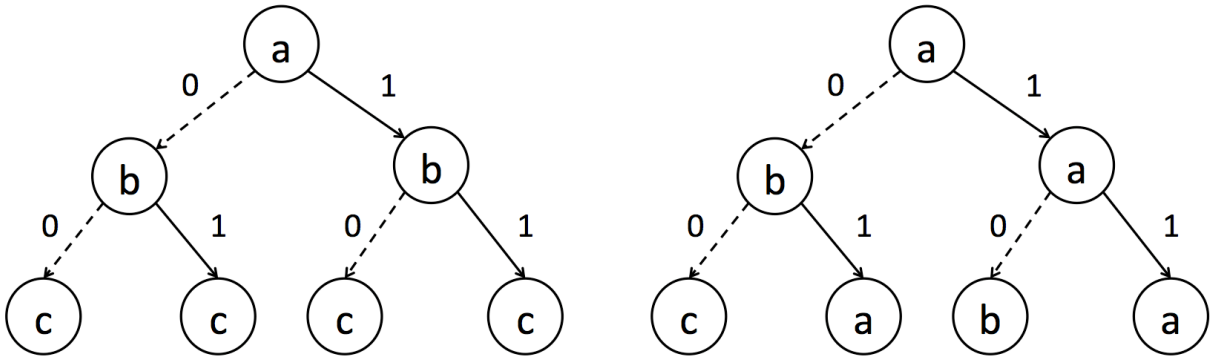
と表すことができ，それぞれの組み合わせについて真ならば集合に含まれ，偽ならば集合には含まれないことが分かる．変数に値を代入して関数値を得たい場合は，BDD と同様に節点（変数）について各枝を辿っていき，終端節点の値を調べる．また図 3.4 の例では，経路によって出現しない変数が存在する（例として，a=1,b=0 の場合,c の節点が出現しない）が，これは例に沿うと，a が集合に含まれ，b が集合に含まれない組み合わせには，c は必ず出現しない，ということを表している．よって，組み合わせ集合に出現しない変数については節点が削除されるため，BDD と比べると，組み合わせ集合を表現することに特化していると言える．また ZDD も，BDD と同様に表現の一意性を持っており，Apply 演算 [17] を用いることによって，２つの ZDD 同士の集合演算を行うことも可能である．

しかし，ZDD は組み合わせ集合に特化しているデータ構造であるため，変数が複数出現 ({aa,aab,abb}) したり，順序違い ({ab,ba}) などは扱うことが出来ない．変数毎に符号化を行うことで，ZDD で系列集合を表現する方法 [20] も考案されている．この方法であれば，同じ変数でも出現位置によって符号を変える（$a_1a_2a_3$など）ことで異なる変数として扱うことができるため，組み合わせ集合によって系列集合を表現することは可能になる．しかし，系列の最大長とアルファベットのサイズの積に比例して変数の数が増加したり，出現位置の微差でも部分系列の共有ができなくなる [21] という欠点があるため，あまり効率のよい方法とは言えない．

3.1.3 SeqBDD（シーケンス型 BDD）

SeqBDD[10] は，ZDD では効率よく表現出来なかった，系列集合に対して最適化されたデータ構造であり，伝住ら [22] によって “Suffix-DD” と呼ばれる SeqBDD を用いた部分文字列索引の構築などの系列データマイニングに利用されている．

BDD，ZDD では，各親の節点と子の節点に対して変数順序と呼ぶ，各変数間に定めた順序付けが存在した．これに対して SeqBDD は，変数の順序付けを 0-枝のみに適用し，1-枝側についてはその順序付けの制約を外した．この制約緩和によって，各変数について 1 つの経路で複数回の出現や変数の逆順を許すことになり，系列集合を効率よく表現することが出来る．図 3.5 に変数 $\Sigma = (a, b, c)$ に対して，変数順序を $a \prec b \prec c$ とした場合の ZDD と SeqBDD の変数順序の例を示す．ZDD ではすべての節点について変数順序が定められているため，すべての節点の変数はその親の節点の変数よりも大きい変数順序の変数をラベルとして持っている．一方，SeqBDD では前述した通り，1-枝側での変数順序の制約が存在しないため，子の節点の変数が親の節点の変数より小さいことを許している．例えば，b の変数の節点の 1-枝側には b よりも変数順序が小さく，すでに節点が存在する a について再び節点が作成されている．



(a)ZDD (b)SeqBDD

図 3.5: ZDD と SeqBDD の変数順序について

SeqBDD では，1 つの経路が 1 つの系列を表しており，最終的に 1-終端節点に到達した場合には，その系列が集合に含まれるということになる．例として，系列集合

$$f = \{aaa,\ aca,\ ba,\ bc\}$$

に対する SeqBDD を図 3.6 に示す．変数順序は $a \prec b \prec c$ とする．本論ではこれ以降提示する図において，0-終端節点の表示を省略し，1-枝を実線，0-枝を破線で表す．

ここで Loekito ら [10] によって提案された SeqBDD を構築するアルゴリズムの擬似コードを **Algorithm1**，および **Algorithm2** に示す．SeqBDD の構築は，まず頂点となる節点からそれぞれの系列の先頭文字が節点の変数で始まるかどうか，を場合分けして枝別れを行う．図 3.6 を参考にすると，まず変数順序が最小である変数 a について，系列集合 f の各系列を a で始まるものとそうでないものに分けて節点 $N = node(a, N_a, N_{\overline{a}})$ を作成する．ここで $node(a, N_a, N_{\overline{a}})$ は，文字 **a** を変数として持ち，1-枝が N_a，0-枝が $N_{\overline{a}}$ を指す節点を示す．N_a に含まれる系列集合は，先頭文字を削除した系列を追加する．図 3.6 の場合，

$$N_a = \{ca, aa\},\ N_{\overline{a}} = \{ba, bc\}$$

となる．この処理を Algorithm1 では，7 行目で変数 x に変数順序が最小の文字を入れ，8 行目からの for ループを用いて各系列を x で始まる系列とそうでない系列に分けて，Algorithm2 に示す Getnode という関数を用いて節点の作成を行っている．続く文字に対しても同様の処理を再帰的に行い，最終的に場合分けをした際に $\{\varepsilon\}$ となるとき 1-終端節点を，Φ となるときには 0-終端節点を指すように枝を作成する．Algorithm1 では，2 行目および 4 行目によって終端節点かどうかの判定を行っている．図 3.6 の例では，左下に位置する c の節点に対して場合分けをした場合，作成される節点 $node(c, N_c, N_{\overline{c}})$ では，

$$N_c = \{\varepsilon\},\ N_{\overline{c}} = \Phi$$

となるため，1-枝は 1-終端節点を，0-枝は省略しているが 0-終端節点を指している．

また，Algorithm2 に示した *Getnode* のアルゴリズムは，SeqBDD(ZDD と同様) の縮約規則である

- 1-枝が 0-終端節点を指している場合節点を削除
- 同一な節点の共有

を行いながら，入力として与えられる変数と 2 つの枝先となるサブグラフを指す節点の作成を行う．節点の共有を実現するために，ハッシュテーブルを用いて演算結果を管理しておき，同一な節点が既に作成済みである場合には新たに節点を作らず，既存の同一な節点を返すようにしている．

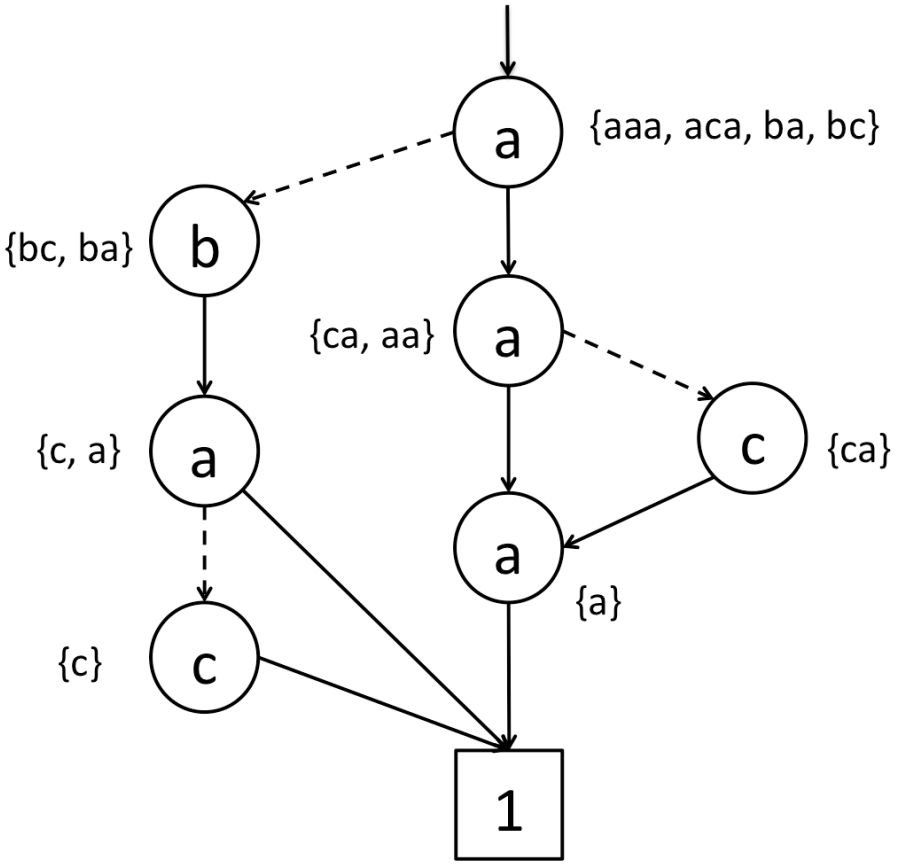


図 3.6: SeqBDD の例

Algorithm 1 Build SeqBDD

Require: f(系列集合)

Ensure: $SeqBDD$

1: sort(f)
2: **if** $f = \{\lambda\}$ **then** { 空列 (1-終端節点) の場合 }
3:　**return** **true**
4: **else if** $f = \Phi$ **then** { 空集合 (0-終端節点) の場合 }
5:　**return** **false**
6: **end if**
7: $x \leftarrow f[0][0]$ { 系列集合のうち変数順序が最小の先頭文字を x とする }
8: **for** $i = 0$ to $|f|$ **do**
9:　**if** $f[i] = x$ **then**
10:　f_x.append($f[i][0]$)
11:　**else**
12:　$f_{\overline{x}}$.append($f[i][0]$)
13:　**end if**
14: **end for**
15: $R \leftarrow$ Getnode($x,\ BuildSeqBDD(f_x),\ BuildSeqBDD(f_{\overline{x}})$)
16: **return** R

Algorithm 2 Getnode
Require: top, P_0, P_1

Ensure: $tree$ {node with two subgraph P_0 and P_1}

1: **if** $P_1 = \Phi$ **then**
2:　**return** P_0 {1-枝が0-終端節点を指していたら節点を削除}
3: **end if**
4: $P \leftarrow$ search a node with $< top, P_0, P_1 >$ in $uniqtable$
5: **if** P $exist$ **then**
6:　**return** P {既に同形のサブグラフを作成済みの場合共有}
7: **end if**
8: $P \leftarrow$ newnode(top, P_0, P_1)
9: $uniqtable < top, P_0, P_1 > \leftarrow P$
10: **return** P

3.2　文を入力としたSeqBDDを構築するための工夫

3.1.3で説明した既存のSeqBDDの構築法を基に，文章を入力としたSeqBDDの構築を行う．言語はPythonを用いて実装し，出力にはPydotというモジュールを用いて画像出力を行っている．

3.2.1　文字，単語による節点の作成

SeqBDDを用いて文を構造化しようとする際に，まず節点のラベルとなる各変数について定めなければならない．本節では，まず文の各文字を変数として考えた場合のSeqBDDの構築を行う．

例として，入力fとなる文を，

$f =$ [私は中学生です, 私は大学生です, 私は小学生です,
私は学生です, 私は学生ではない]

とした場合に構築される SeqBDD を図 3.8(a) に示す [*2]．ここで同一節点間に，枝が複数現れているサブグラフが見受けられるが，これは SeqBDD の縮約規則である「同等な節点の共有」を分かり易く示したものである．

節点を文字として SeqBDD を構築した場合，入力となる系列集合の各変数間の変数順序が問題として挙がった．変数順序は SeqBDD を構築する際に，各節点の作成順序を定めるものとなるが，この順序によって木が理想形とならずパターンの発見が難化した．簡単な具体例を図 3.7 に示す．入力は $f = [$小学生, 学生$]$ の 2 文である．この場合，理想形は根となる節点のラベルが「“小”」で，「“学”,“生”」という節点を共有して，「“学生”」というパターンを得ることが理想である．しかし，Python の内部関数を用いてソートを行うと「“小”」よりも「“学”」のほうが変数順序が小さいため，結果として「“生”」の節点しか共有されない．前に示した 3.8(a) の小学生，学生の枝分かれの部分でも確認することができる．

これを踏まえて次に，入力となる文に形態素解析を行って単語毎に区切り，節点を単語とした SeqBDD を構築した．例を図 3.8(b) に示す，入力は図 3.8(a) と同じである．

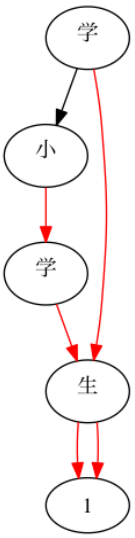
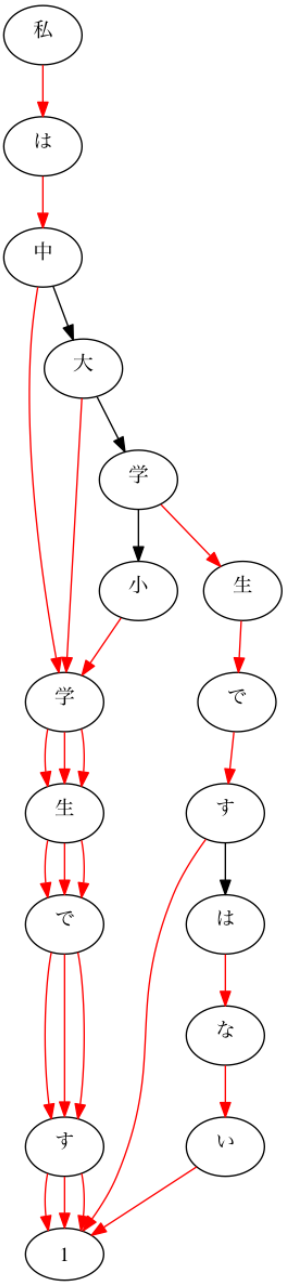
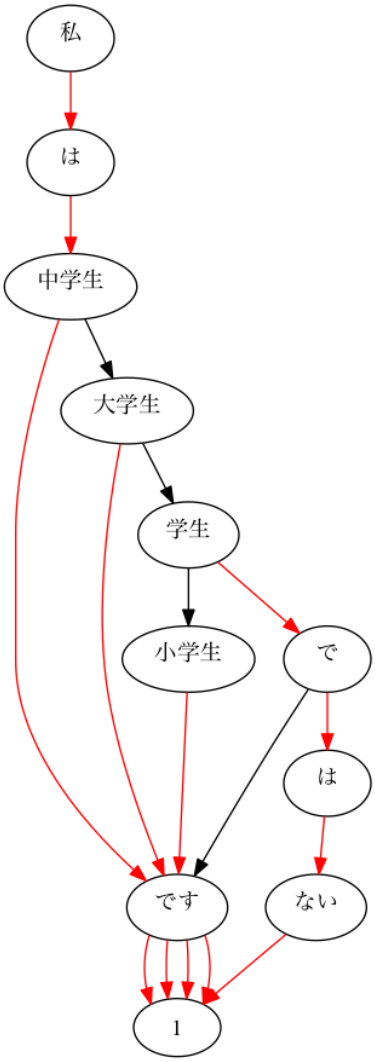


図 3.7: 変数順序によるパターン発見の難化

[*2] これ以降の SeqBDD の出力画像は，1-枝を赤線，0-枝を黒線で示し，また 0-終端節点は省略する．



(a) 文字を節点



(b) 単語を節点

図 3.8: 構築したSeqBDDの例

文を単語列に区切るために，「MeCab[*3]」という形態素解析エンジンを用いた．単語毎に節点を作成することで，文をソートする際の参照する変数（文字）が増えるため，前述した変数順序によるパターン発見の問題を大幅に改善することができた．3.8(b) を見ると，(a) よりもコンパクトな表現になっているのが分かる．本来は，(小, 中, 大) といった文字で枝分かれし，[“学生”,“です”] という節点を共有することが理想である．これはより詳細な形態素解析データを用いて節点の作成を行うことで可能となるが，今回は単語による節点の作成を主としていないため省略した．

単語で節点の作成を行ってパターンを取得しようとする場合，単語が完全に一致していないと節点の共有が出来ないため，容易にパターンを得ることが出来ない．そこで，節点の共有判定の緩和を目的とした新たな節点の作成方法として次節 3.2.2 で述べる，単語に品詞情報を付与した，単語の品詞による節点の作成を行う．

3.2.2　品詞による節点の作成

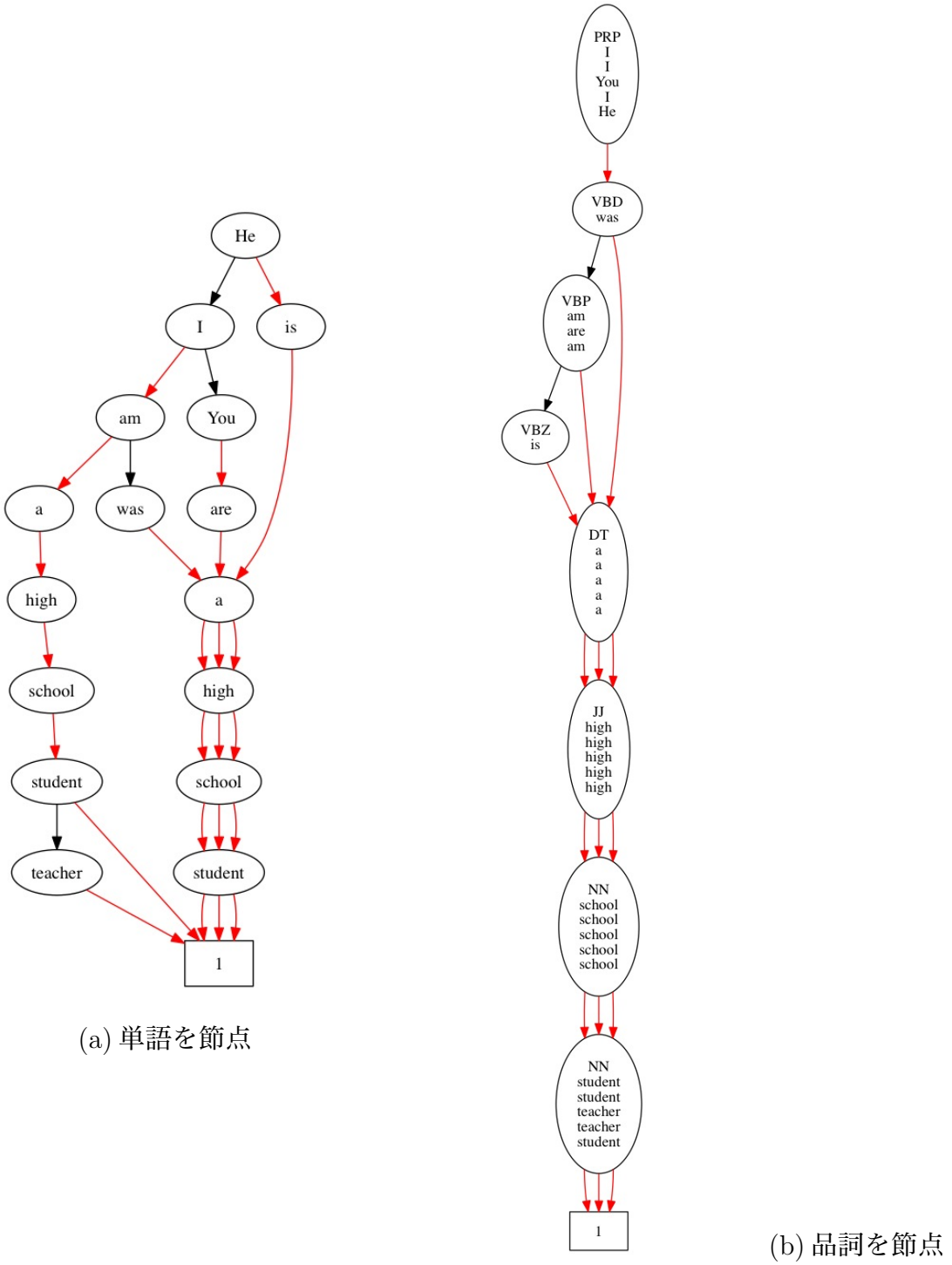
入力となる文を単語列に区切り，品詞情報を単語ごとに付与して，単語の品詞を節点の変数として SeqBDD の構築を行う．また，入力となる文章は英語を対象とした．

入力となる文を，

$$f = [\ I\ am\ a\ high\ school\ student\ \ ,\ \ I\ was\ a\ high\ school\ student,$$
$$You\ are\ a\ high\ school\ teacher\ \ ,\ \ I\ am\ a\ high\ school\ teacher,$$
$$He\ is\ a\ high\ school\ student\ ]$$

とした場合の，節点を単語および品詞で作成した SeqBDD の比較を図 3.9 に示す．文の単語列への分割，品詞情報の付与は「NLTK[*4]」という形態素解析エンジンを用いた．

[*3] http://mecab.googlecode.com/svn/trunk/mecab/doc/index.html
[*4] http://www.nltk.org



(a) 単語を節点

(b) 品詞を節点

図 3.9: 節点を単語/品詞で作成したSeqBDDの比較例

品詞毎に節点を作成する際に，節点の変数は節点のラベル 1 行目に表示されている品詞（例：PRP，VBP など）とし，その品詞が入力ではどの単語に当たるかを節点のラベルの 2 行目以降に例として表示している．節点を品詞毎に作成することで，節点の変数となり得る数が単語に比べて非常に少なくなるため，サブグラフの共有判定の緩和に繋がり，またパターン抽出の易化に繋がる．図 3.9(a) の単語で節点を作成した場合には共有されない部分が，(b) の品詞の場合では改善されていることが見て取れる．

3.2.3　枝のランク付け

作成した SeqBDD に対して枝にランク (重み) を付けることで，各経路，および各枝の出現頻度を調べることができるようになる．ランク付けの方法は

- 入力となる各文章に対して，その文章に対応する経路の枝のランクをすべて＋ 1 する

という単純なアルゴリズムを採用している．この方法によってランク付けされた SeqBDD を更に第 3.3 節で説明する方法によって，パターンを抽出し易い形へと調整する．

3.3 SeqBDD の調整とパターンの抽出

前節までに，品詞毎に節点を作成しランク付けを行った SeqBDD に対して，更にパターンを抽出し易い構造になるようにデータ構造の改良を行った．

3.3.1 親を共有する 0-枝の削除

SeqBDD の構造上 0-枝は必然的に出現するが，0-枝は可能な限り数が少ない構造の方が，視覚的にわかりやすく，またパターンの抽出を行い易い．このとき，**根以外の節点を枝元とする 0-枝**は，その 0-枝を根の方に辿って行き，初めに通る 1-枝の枝元の節点から 1-枝を伸ばすことで，枝の置換が可能である．図 3.9(b) と同様の入力に対してランク付け，および 0-枝の削除を行ったものを図 3.10 に示す [*5]．この処理を行うことで，根を枝元とする 0-枝以外はすべて 1-枝に置き換えることができるため，文章を SeqBDD で表す際に，より簡潔なデータ表現として扱うことが可能となる．

3.3.2 ランクによる枝の削除

SeqBDD からパターンを抽出する前処理として，次に「枝切り」の処理を行う．ここで行う枝切りのアルゴリズムを **Algorithm3** に示す．

ランクが 1 の枝をすべて削除し，その後どの枝の枝先にもなっていない節点を始点とするサブグラフの削除という方法で枝切りを行っている．枝切りを行うことで出現頻度の少ない枝，節点を削除し，出現頻度の高い節点のみにフィルタリングすることができ，データサイズの圧縮にも繋がることでパターン抽出をより高速に行うことが可能となる．

[*5] 品詞の後ろの括弧内の数字は根となる節点からの距離を示す

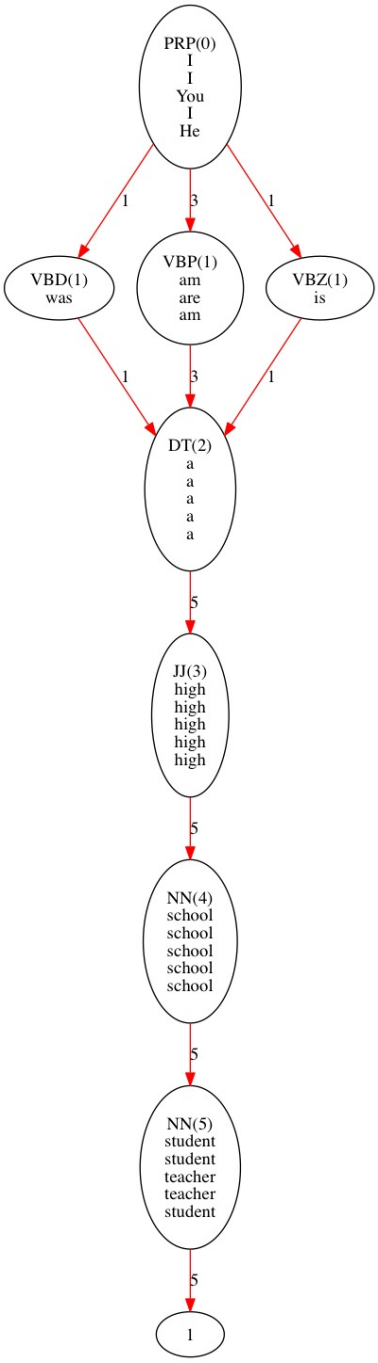

PRP(0)
I
I
You
I
He
1
3
1
VBD(1)
was
VBP(1)
am
are
am
VBZ(1)
is
1
3
1
DT(2)
a
a
a
a
a
5
JJ(3)
high
high
high
high
high
5
NN(4)
school
school
school
school
school
5
NN(5)
student
student
teacher
teacher
student
5
1

図 3.10: 0-枝の削除

Algorithm 3 Remove Rank1-edge

Require: $SeqBDD$

Ensure: $SeqBDD\ removed\ rank-1 edge$

1: **for all** $edge \in SeqBDD$ **do**
2: **if** $edge.label = 1$ **then**
3: **remove** $edge$ **from** $SeqBDD$ { ランクが “1” の枝をすべて削除 }
4: **end if**
5: **end for**
6: $RemoveNodes \leftarrow Node\ (which\ is\ not\ used\ as\ the\ destination\ of\ edge) \in SeqBDD$
7: **for all** $n \in RemoveNodes$ **do**
8: **remove** $Subgraph\ that\ starts\ with\ n$ **from** $SeqBDD$
{ どの枝の枝先にもなっていない節点以下のサブグラフをすべて削除 }
9: **end for**
10: **return** $SeqBDD$

3.3.3　節点ラベルの圧縮

品詞付きの単語で SeqBDD を構築した際に，節点となっている品詞をパターンとして抽出すると，品詞列のパターンしか取ることが出来ない．そこで，節点のラベルに含まれていた単語例を用いてパターンを抽出することを考える．節点のラベルとなっている品詞＋単語列のうち，単語列の各単語について，次の式

$$\text{閾値 } x \leqq (\text{単語の出現回数}) / (\text{節点の単語例の総数}) \tag{3.1}$$

を満たす単語 W が存在する場合には，その節点のラベルを品詞＋ W という表示にする．図 3.11 に閾値を $x = 0.6$ としたときの圧縮例を示す．IN で表される前置詞の節点について，“about” という単語が閾値を超えているため，節点のラベルの表示を変更する．

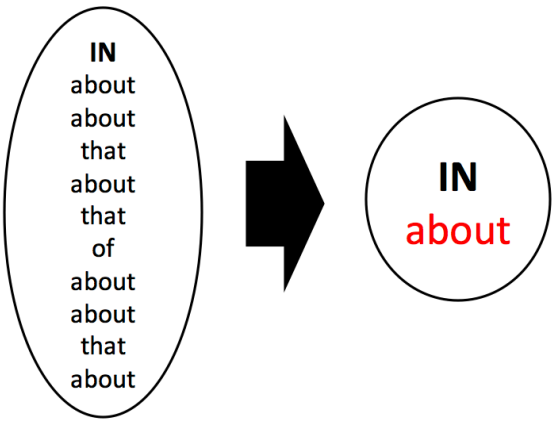


図 3.11: 節点ラベルの圧縮例

3.3.4 パターンの抽出方法

前節までの方法で構築されたSeqBDDからパターンを抽出する方法を説明する．例として，図3.9と同じ入力からのパターン抽出を考える．まず，前節までに説明した処理を行ったあとのSeqBDDで，式(3.1)の閾値を$x = 0.6,\ 0.65$としたものを，それぞれ図3.12 (a), (b)に示す.

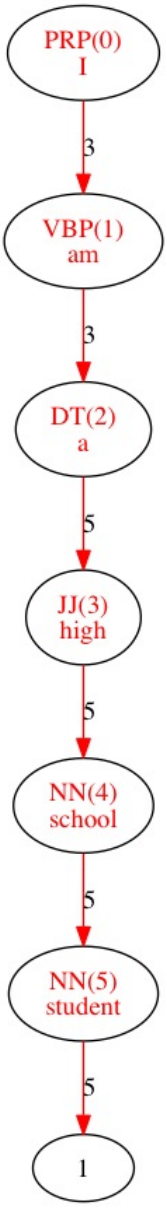
(a)の場合，すべての節点にラベルの圧縮処理がされているため，パターン**P**は

$$P\ =\ \textbf{I am a high school student}$$

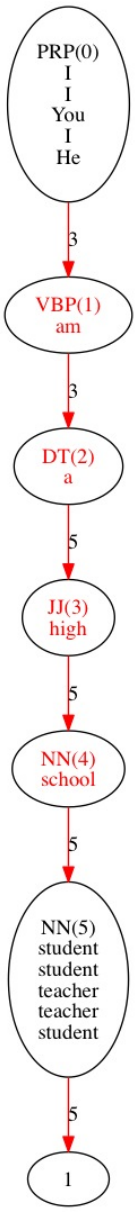
と抽出する．一方，(b)のように圧縮されていない節点を含む場合には，

$$P\ =\ 【*/PRP】\ \textbf{am a high school}\ 【*/NN】$$

というように圧縮されていない節点を，*を用いて【*/(品詞)】として表現することで**ワイルドカード**部として扱ってパターンとする．



(a) 閾値 $x = 0.6$



(b) 閾値 $x = 0.65$

図 3.12: パターン抽出を行う SeqBDD の例

第4章

実験と評価

4.1 実験概要

構築したSeqBDDからパターンを抽出するに当たり，本研究において目的とするパターンを「ある単語が与えられたときに，その単語の前後に現れる単語列の中で頻度の高い単語（品詞）列」であると定める．

4.1.1 実験方法

今回は，目的とするパターンのうち特に「与えられる単語の後ろに出現しやすい単語列」に注目をした．与えられる単語(以後“Query”と呼ぶ）が出現する文章をコーパスから抜き出し，文章をQueryの前で区切り，それ以降の単語列を入力としてSeqBDDを構築する．これにより，根となる節点のラベルは常にQueryとなり，Queryに続く可能性の高いパターンを抽出することが可能となる．

本実験では，以下に示す実験工程を行って最終的に得られるパターンとそのパターンに対応する正解データを比較することによりパターンの照合を行う．SeqBDDの構築方法については第3.2章にて論じている提案手法を用いる．正解データは英語のイディオム，Queryはイディオムの先頭語，としてSeqBDDの構築を行い，得られるイディオムのパターンと正解

パターンが一致するかどうかを検証する．

実験工程

1. “Query” となるイディオムを決める
2. Query 全体を含む文章をコーパスより抜き出し，Query 前で文章を区切る
3. 一定数の文章が集まったイディオムについてのみ次の工程に移る
4. 工程 (2) までに出来た文章を入力として SeqBDD を構築
5. パターンとして抽出できる単語列をランク順に列挙し，パターン候補とする
6. パターン候補と正解データを比較し，評価を行う．

上記の実験工程について，いくつか説明を付け加える．まず実験工程（２）の「Query 全体を含む文章」について，コーパスから文章を抜き出す際に，**イディオムの各単語間が２単語以内**という制約を設けた．これによって，文章にイディオムの各単語は含むものの，イディオムとしては表れていない文章 [*1] をかなり取り除くことができる．

実験工程（５）の「ラベル順のパターン列挙」を実現する関数のアルゴリズムを **Algorithm4** に示す．根となる節点であるイディオムの先頭語を Query として関数を呼び出すことで，根を始点とする各経路をランクの高い順にトップダウン方式で再帰的に 1-終端節点まで辿って行き，得られるパターンをパターンリストに加える．9 行目で “Query” を枝元に持つ全ての 1-枝を取り出し，10 行目でそれらをランクを用いてソートする．これによって，11 行目からの for ループ処理で枝のランク毎に再帰的に経路を探索してパターンを発見する．flag を用いることで，直前の節点がパターンとなっている節点であったかどうかを記憶し，パターンとなっていない節点の品詞（ワイルドカード部）の複数の列挙を防止している．1 行目から 8 行目にかけては，1-終端節点まで辿り着いた際にその経路のパターンを “Pattern List” に追加

[*1] 例:「see in: (家などに) 招き入れる／新年を迎える」というイディオムに対して，「We see a good example in him.：彼は我々の良い手本だ.」という文章

する処理を記述している．この関数を呼び出して作成されるグローバル変数 “Pattern List” に，ランクの高い順にパターンが列挙されることになる．

4.1.2　実験データ

本実験を行う上で必要になるデータは，入力となる文章を含んだコーパスである．本実験では，「British National Corpus[*2]」（略称，BNC）というイギリスの学術機関や出版社など多数の企業によって管理される大規模電子データベースを用いた．例文の量が豊富に備わっており，様々な文法パターンや用法を引き出すことが可能となっている．第 4.1.3 節にて述べる評価用正解データに含まれる，検索語となる英語イディオムを BNC にて検索し，実験工程（3）の一定数の文章数を今回は，**20 以上の例文が存在するイディオム**と設定した．

4.1.3　評価用データ

実験を行う際に用いる検索語となるイディオム，およびそのイディオムの正解となるパターンを今回は，「TheFreeDictionary.com[*3]」という Farlex 社によって運営されているアメリカの様々なオンライン辞書や百科事典と相互参照を行っているサイトを用いて評価用データの収集を行った．具体的には，サイトの “idioms” というイディオム検索専用ページの Top に “Try it out” として表示されるランダムイディオム (図 4.1) と，その各イディオムの詳細ページ (図 4.2) の小見出しで記載されているイディオムの用例を (検索語，評価データ) のペアとして収集した．図 4.2 の場合には (“paint in”, “paint * in”) というペアが作成される．ここで “some∼” で表されているパターンのワイルドカード部については “**∗**” を用いて表現する．

今回の実験では，まず TheFreeDictionary.com の idioms のページからイディオムを自動的に収集するモジュールを作成し，約 400 個ほどのイディオムを収集した．次に収集したイディオムの各詳細ページに表示される正解パターンとなるイディオムの用例に関して，“some∼” を含まない，つまり目的語などの何らかの単語が続かない名詞句となるイディオム (例:bad times/不景気) を取り除いた．こうして得られたイディオムと用例のペアのうち，そのイディオムの例文が 20 以上存在する，計 50 個のペアに対して実験を行った．

[*2] http://www.natcorp.ox.ac.uk
[*3] http://www.thefreedictionary.com

Algorithm 4 Get Patterns

Require: *Arranged SeqBDD*, *Query*, *Pattern*, *WildcardList*, *flag*

Ensure: *PatternList* (*Global Variable*)

if *Query* = *node* "1" **then**
  if *flag* = 0 **then**
    Pattern.*append*(*WildcardList*[0])
  end if
  Pattern.*append*(*WildcardList*)
  PattenList.*append*(*Pattern*)
  return *None*
end if
RedEdges ← *All edges that source* = "*query*"
sort RedEdges used edge's rank
for $i = 0$ **to** $|RedEdges|$ **do**
  node ← *RedEdge*[*i*] *destination node*
  if *node* = *compressed node* **and** $flag = 1$ **then**
    Pattern.*append*(*node label*)
    Get Patterns(*SeqBDD*, *node label*, *Pattern*, *WildcardList*, 1)
  else if *node* = *compressed node* **and** $flag = 0$ **then**
    Pattern.*append*(*WildcardList*)
    Pattern.*append*(*node label*)
    Get Patterns(*SeqBDD*, *node label*, *Pattern*, ϕ, 1)
  else
    WildcardList.*append*(【 ∗ / (*parts of speech*)】)
    Get Patterns(*SeqBDD*, *node label*, *Pattern*, *WildcardList*, 0)
  end if
  WildcardList ← ϕ
end for

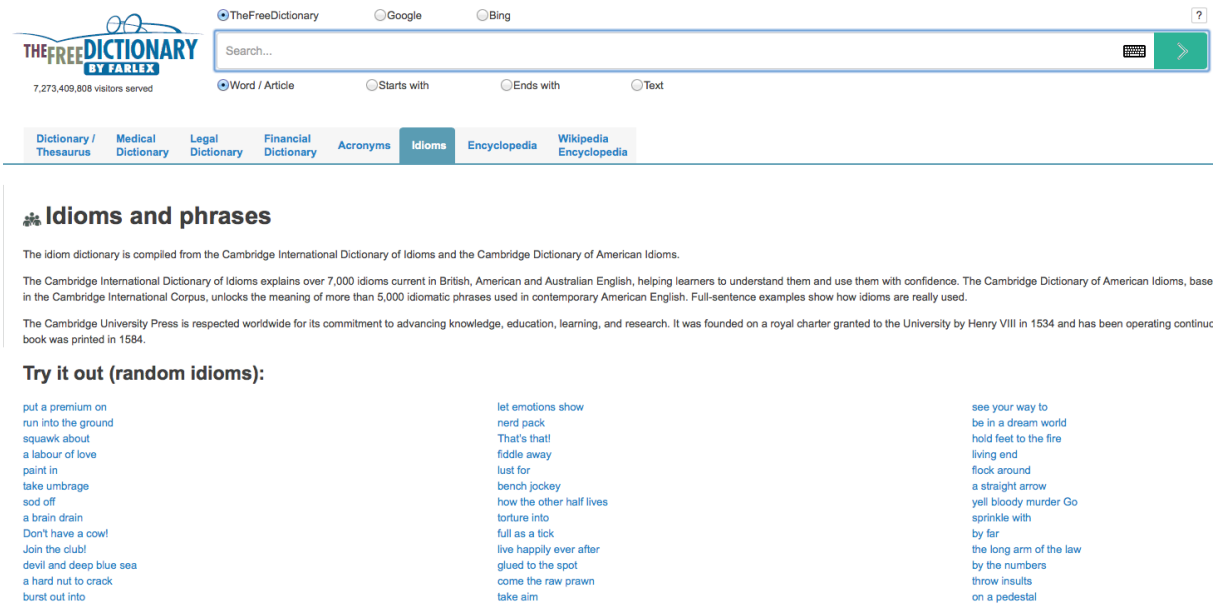


図 4.1: Try it out のページ

図 4.2: idiom の詳細ページ

4.2 評価指標

本実験の評価を行った指標を以下に示す．ここで，実験の評価用データである 50 ペアのデータ集合を T, その要素となる各ペア $t \in T$ について，

$$\begin{cases} S^* & : \text{正解パターンの集合} \\ S & : \text{出力パターンの集合} \\ A\ (\subseteq S) & : S \text{のうち正解パターンと一致したパターンの集合} \\ c_i\ (\in S) & : S \text{のランク順に列挙されたパターン候補} i \text{番目が正解かどうか} \\ |P_i|\ (=\ \sum_{j=1}^{i} c_i) & : i \text{番目までの正解と一致したパターン数} \end{cases}$$

とする．今回の実験では，イディオムの正解パターンを 1 つのみとしたため，すべての t について $|S^*| = 1$ となる．また，$c_i = 1$ でパターン候補 i 番目が正解パターンと一致，$c_i = 0$ で不一致とする．

4.2.1 単純適合率

単純な正解率として，各 t について，$c_i = 1$ となる c が存在するか，すなわち出力パターン集合に正解パターンが含まれているか否か，を L を用いて

$$L = \begin{cases} 0 & (|A| = 0) \\ 1 & (|A| > 1) \end{cases}$$

と表す．各 t について L を算出し，T に関して平均する．

4.2.2　平均逆順位

平均逆順位は “MRR” と呼ばれ，本実験ではランクごとに列挙されたパターンのうち，最後に正解パターンと一致した順位までのすべての順位の逆数の和をとる．

$$MRR = \frac{1}{|A|}\sum_{i=1}^{n}\frac{p_i}{i}$$

ここで，「n：最後に正解パターンと一致したパターン候補の順位」とする．仮にパターン候補の 1 位，2 位，5 位が正解パターンと一致していた場合，

$$MRR = \frac{1}{3}\,(\frac{1}{1}+\frac{1}{2}+\frac{1}{5}) \cong 0.567$$

と計算することができる．この MRR を各ペア t について算出し，T に関して平均することで評価を行う．

4.2.3　平均適合率

平均適合率は AP(Average Precision) と呼ばれ，S のパターン候補 1 位から n 位までの中に含まれる正解パターンと一致した数の平均値で与えられる．

$$AP = \frac{1}{|A|}\sum_{i=1}^{n} p_i \times \frac{|P_i|}{i}$$

ここで，n は第 4.2.2 節と同様である．パターン候補の 1 位，2 位，5 位が正解パターンと一致していた場合，

$$AP = \frac{1}{3}(\frac{1}{1}+\frac{2}{2}+\frac{3}{5}) \cong 0.867$$

と計算できる．これも，各ペア t について算出し，T に関して平均する．

4.3 実験結果

以上の実験方法，および評価指標を用いて抽出できたパターンがどれほど正しいかを計算し，集合Tに関して平均をとったところ

$$L_T = 0.84 \quad MRR_T = 0.394 \quad AP_T = 0.838$$

という結果を得ることが出来た．適合率である，LとAPに関してはかなり高い評価を得ることが出来た．これは，文章をコーパスから抜き出す際の検索語がイディオム全体となっているため，Queryとなるイディオムと用例が同じ文章を数多くコーパスから抜き出せたことが要因となっている．また，MRRについては，高い順位から低い順位まですべて正解パターンと一致していた場合，数値が低くなってしまうため，平均すると比較的低い結果となっている．

4.4 パターン抽出例

入力となるコーパスから抜き出した文章, およびそこから抽出したパターン候補の例を図4.3, および図4.4に示す．この例では, $Query =$ "*bother with*", 正解パターンは"*bother with* ～"(～のことを思い悩む) である．

出力されたパターン候補のうち1位から5位までは “bother with～” という正解と一致するパターンを抽出することに成功している．6,7位のパターン候補に関しては，入力となった文章例にも存在する “bother” と “with” の間に別の単語を含んだ例文がノイズとなって抽出されていると考えられる．

・We didn't bother with lights.

・We've found ourselves in a spot of bother with the police over at the office.

・I am not going to bother with a certain group.

・We didn't bother with that in the end.

・But we can't bother with him any more now.

・Well perhaps you wouldn't bother with that.

・He would not bother himself with it now.

・Well that lady don't bother with her next door down there.

図 4.3: bother with の入力文章例

1–bother, with, 【*/DT】

2–bother, with, 【*/NNS】

3–bother, with, 【*/PRP】

4–bother, with, the, 【*/NN】

5–bother, with, 【*/NN】

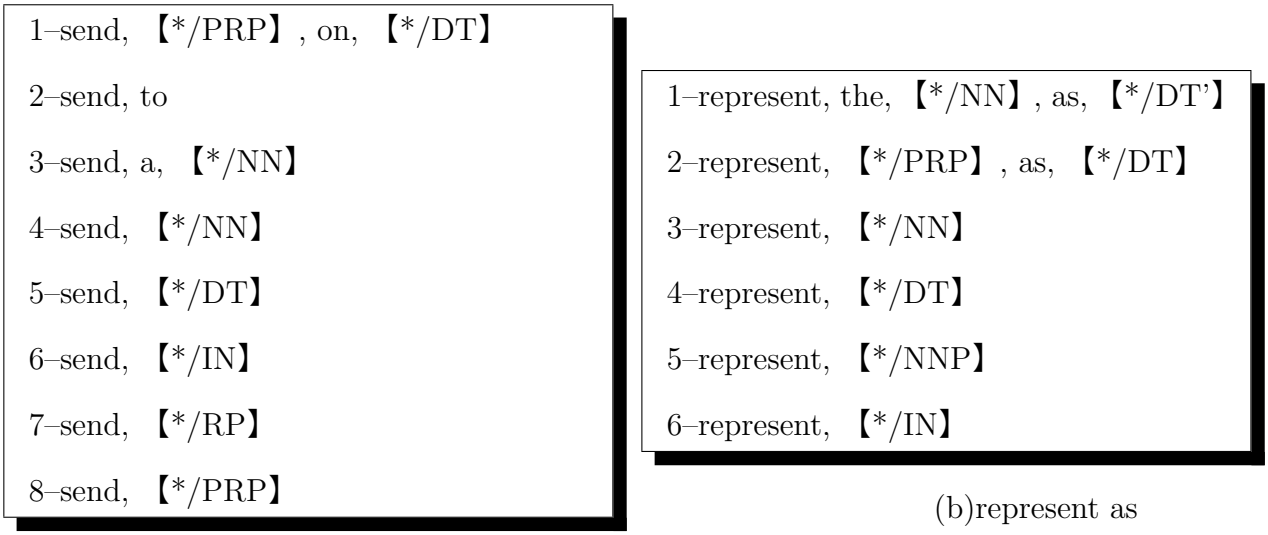
6–bother, 【*/PRP】

7–bother, 【*/IN】

図 4.4: bother with の抽出パターン候補

抽出できたパターンの中には，「send A on B：A を B に送る，行かせる」，「represent A as B：A を B と描く，述べる」と言ったイディオムの単語間に穴あきの部分が存在するようなパターンについても抽出を行うことが出来た．図 4.5 にそれぞれの抽出したパターン候補を示す．

“send on” では 1 位のパターン候補のみ，“represent as” では 1 位と 2 位のパターン候補，が正解と一致している．数は少ないものの，穴あきのパターンという抽出することが難しいパターンを出力パターン候補のうちの上位で抽出することが出来ている．



(a)send on

(b)represent as

図 4.5: 穴あきパターンの例

逆にパターンを抽出できなかった例としては，「appear for〜：〜（休んだ人の）代わりに現れる」などがあった．“appear for” に関して，入力となった文章の例を図 4.6，抽出したパターン候補を図 4.7，および作成された SeqBDD の一部を図 4.8 に示す．appear for の場合，コーパスから入力となる例文を抜き出す際にイディオムの単語間の距離を 2 単語まで許すという制約によって，appear と for の間に単語を含んだ文章を多く抜き出してしまったため，うまくパターンを取り出せていないことが分かった．これは，パターン抽出モジュールよりもコーパスから文章を抜き出すモジュールの改良が求められる．

・Told me that one youth was in custody and another due to appear in court for the break − ins.

・Wings appear externally for the first time and the insect takes on the appearance of an adult.

・Even its two windows appear thatched, for they are fitted with wads of straw.

・Their clicks are not FM, nor do they appear suitable for Doppler-shift speed metering.

・They didn't appear to care for each other greatly.

図 4.6: appear for の入力文章例

1–appear, 【*/IN】, the, 【*/NN】

2–appear, 【*/IN】, the, 【*/JJ】

3–appear, 【*/JJ】

4–appear, 【*/NN】, for

5–appear, 【*/JJ】, for

6–appear, to, 【*/VB】

7–appear, 【*/RB】

8–appear, 【*/TO】

図 4.7: appear for の抽出パターン候補

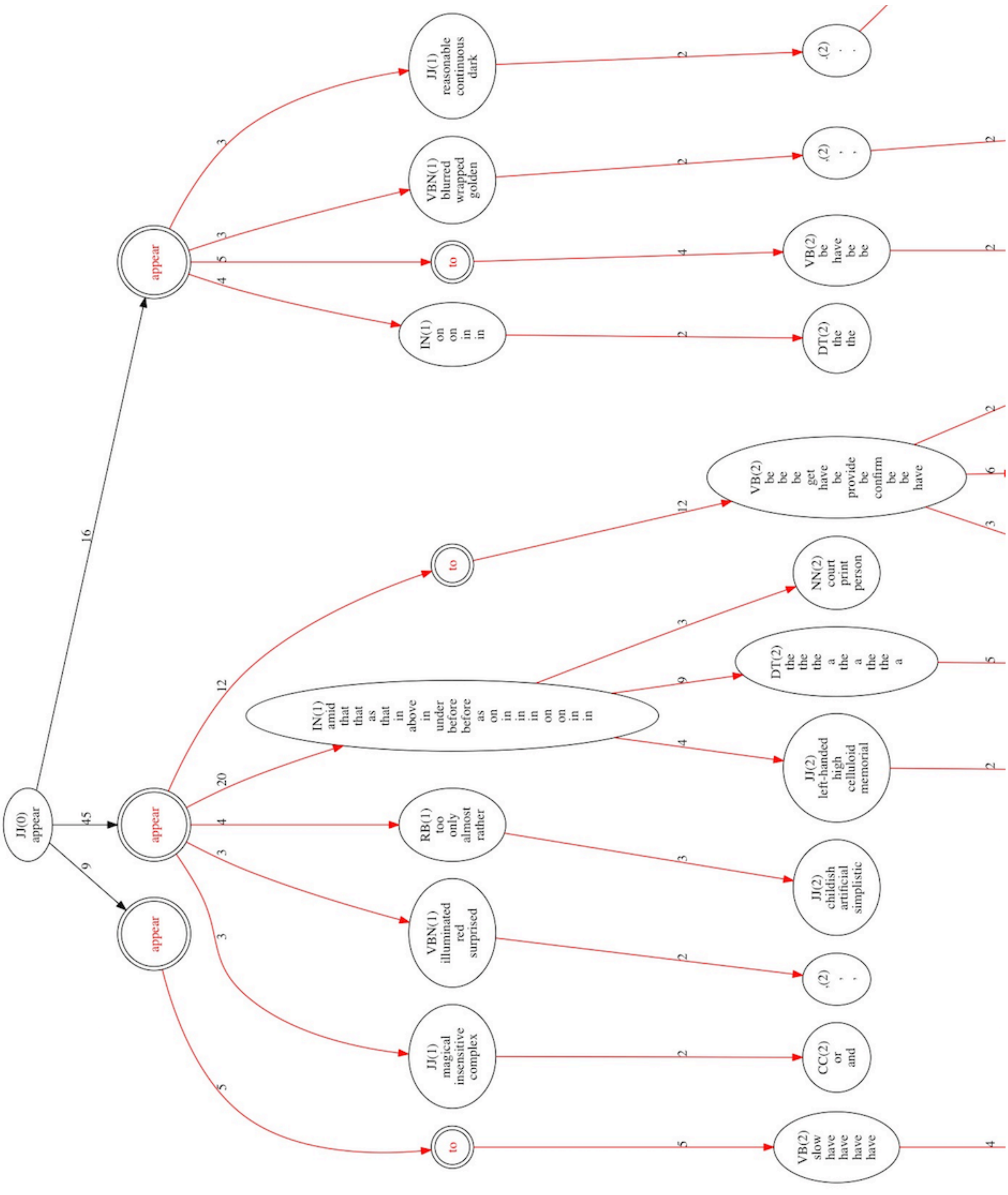


図 4.8: appear for で作成された SeqBDD の一部

4.5　考察

4.5.1　品詞タグ付の精度

現在は，文章から SeqBDD を構築する際に，まず入力となる文章それぞれについて構文解析エンジンを用いて各単語に品詞付を行っている．しかし，パターンを抽出できなかった “appear for” の例 (図 4.8) を見ると，まず根となる “appear” の節点が複数に別れてしまっている．これは，構文解析の段階で，文章毎に “appear” が異なる品詞である，とタグ付けされているために生じている．そのため，品詞によって節点を作成する際に，本来同一であるべき根となる節点が複数個存在してしまい，その分枝が細かく別れてパターンの抽出を妨げていると考えられる．この場合，根となる節点のみに関しては，無条件で同一の節点であるとするという規約の元で SeqBDD を構築することで，無駄な枝分かれを防ぐことができ，パターン抽出精度の向上に繋がると考えられる．

4.5.2　節点圧縮の閾値

パターンを抽出する際に参照としている節点は，式 (3.1) を満たした節点内のラベルが圧縮されたものである．ここで，より多くのパターンを抽出するために，式 (3.1) の閾値を低くすることで，圧縮される節点の条件を緩くするという方法がある．これによって，いままでパターンとして表れていなかった新たなパターンも抽出することが可能になり，精度の上昇に繋がるかもしれない．しかし，ただ閾値を低くするだけでは，必要としない無駄なパターンの増加につながったり，節点のラベルに圧縮されるべき単語が２つ以上存在してしまうという新たな問題が生まれる可能性がある [*4]．そのため，パターンの抽出を最も効率よく行うことのできる閾値を調整して，調べる必要がある．

[*4] 例えば節点内の単語が２つのみの場合に，閾値を 0.5 以下にしてしまうと両方の単語ともに条件を満たすことになる

4.5.3 文章抜き出しモジュール

第 4.3 節でも述べたように，パターンの取れなかったイディオムはほとんどがこの文章抜き出しに関する問題のためだった．現在の手法では，コーパスから文章を取ってくる際に，すべてのイディオムについて各単語の間に 2 単語までの距離を許している．そのために，本来のイディオムの用例でない文章も抜き出してしまい，その文章群がノイズとなってパターンの抽出を妨げている．

この問題の一つの解決策としては，文章を抜き出す際のイディオムの単語間の距離の制約を，イディオム毎に変化させることが挙げられる．現在の仕様では，Query となるイディオムは穴あきのパターンとそうでないパターンが混在しており，その両パターンともに対応するために距離の制約を緩めていた．それを，穴あきのパターンの場合にはイディオムの単語間の距離を許し，そうでないパターンの場合には距離の制約を 1 単語も許さないようにすることで，パターンとして抽出できなかったイディオムについては改善が見込め，また既にパターンとして抽出できていたイディオムについては，より精度の高いパターンの抽出が望めるのではないだろうか．

第5章

結論

5.1 まとめ

本研究では，教師データを用いないテキストからのパターン抽出を，SeqBDD というデータ構造を用いて実現した．木構造を持つ SeqBDD の末端部分を，同一な形のサブグラフの共有を行うことで，縮退することに成功し，巨大な量のテキストをよりコンパクトなデータ構造で表現することが可能となった．また，そこからパターンを抽出することで，出現頻度の高い単語の用例を獲得することができた．本稿では，特に構築した SeqBDD からパターンを抽出する手法の提案，および実装，さらにそれらの評価を行った．この提案手法の実現に伴って，大きく２つの処理モジュールを構築した．

１つに，テキスト文章を入力とする SeqBDD の構築があった．節点の作成方法を吟味することで，よりパターンを抽出しやすい方法を導き，それらのプログラムの実装を行った．また，SeqBDD の大規模なデータ構造をコンパクトに表現することができる，という特徴を活かし，かつパターンを抽出しやすいように，節点ラベルの圧縮や枝切と言った，様々な方法によってデータ構造の加工を行った．

もう１つに，構築した SeqBDD からパターンを抽出する処理があった．この処理では，提案手法によってパターンの抽出を行い易いデータ構造へと加工されているため，比較的単純

なアルゴリズムによってパターンを抽出することが可能となった.

また検証として，入力には，様々な文法パターンや用法を含む例文を備えた大規模電子データベース「British National Corpus」から抜き出した例文を，その例文を抜き出すための検索語，および正解データとなるイディオムのパターンを「TheFreeDictionary.com」というオンライン辞書サイトから収集し，上記の提案手法の実験，評価を行った．抽出できたイディオムのパターン候補と，サイトから収集した正解パターンの照合を行い，単純適合率 (L)，平均逆順位 (MRR)，平均適合率 (AP) という指標によって評価を行った．その結果，$L = 0.84,\ MRR = 0.394,\ AP = 0.838$ という値を得ることが出来た．平均順位を示す MRR に関しては，抽出したパターンの上位から下位までのすべてが正解パターンと一致するケースが多かったため，比較的低い数値となってしまったが，適合率を示す L，AP に関しては 8 割を超えるというかなり高い評価を得ることが出来た.

前述の考察 (4.5 節) でも述べたように，まだ様々な問題点，改善点を抱えている本研究ではあるが，SeqBDD を用いた教師なし手法による文からのパターン抽出の有用性を示せたのではないだろうか.

5.2　今後の課題

本研究における今後の課題としては，以下の様なものが挙げられる.

5.2.1　既存のパターン抽出モジュールの改良

第 4.5 節でも述べたが，既存のパターン抽出モジュールでは,

- 品詞タグ付けの精度によって根となる節点が複数に分かれてしまい，それによって経路の細分化を招きパターン発見の難化を招いている
- パターンを抽出する際に，パターンかどうかの判定基準となる節点のラベル圧縮における最適な閾値を調べる必要がある

- 入力となる文書を抜き出すモジュールにが，パターン抽出を妨げるノイズとなる文章も一緒に抜き出してしまっている

といった改良点が挙げられる．これらの問題を改善していくことで，より精度の高いパターンの抽出を見込めるだろう．

5.2.2 パターンの抽出方法

現在のパターンの抽出方法は，節点のラベルが圧縮 (第 3.3.3 節を参照) されているかどうか，という点に注目してその節点のラベルがパターンとなるかを判定している．しかし，考察 (第 4.5 節) でも述べたが，節点のラベルを圧縮するために用いる閾値の問題がある．パターンを抽出するために最適な閾値を，様々なサンプルを通して発見することが望ましいが，発見できなかった場合には，検索語毎に適当な閾値に変更するなどの措置を取るか，抽出方法の変更を余儀なくされる可能性もある．

5.2.3 与えられた語句の前半部分のパターン抽出

今回の実験では，与えられた語句（イディオム）の後半部分のみについて，SeqBDD を構築しパターンの抽出を行った．しかし，与えられた単語の前後に共起しやすい単語，品詞のパターンを抽出することを目的としているため，前半部分についても SeqBDD の構築，およびそこからのパターン抽出を行う必要がある．

前半部分におけるパターン抽出は，後半とすべて同じ方法を用いるということは難しい．なぜなら，後半部分ではイディオムの前で文を区切り，それ以降の文章を入力として SeqBDD を構築したため，根となる節点のラベルは必ずイディオムの先頭語であり，その先頭語の節点から各経路を辿って行き，パターンの判定を行うことが可能であった．しかし，前半部分では，根となる節点は各文章の先頭語であり，もちろん同じ単語，品詞であるとは言えない．そのため，根付近のサブグラフに関して細かい枝分かれが発生することになる．終端節点付

近では，イディオムの直近となるため比較的似た単語や品詞が現れる際には，節点の共有が行われパターンとして抽出することができるが，根の付近の枝分かれにより，経路がより細かく別れるため，パターン発見の難化に繋がるのではないかと考えられる．

5.2.4　検索単語の抽象化と正解パターン集合の拡大

今回の実験は，検索単語はイディオム全体，正解パターンは 1 種のみという限定された範囲内で行った．しかし，現在の目標の一つとして，ある単語（1 つ，または複数）が与えられた際に，その単語の前後に出現しやすい単語や品詞をパターン候補として利用者に提供するというシステムの構築がある．この実現に向けて，まず，行うべきことが，前述した単語の前半部分のパターン抽出，さらに，例えばイディオムの先頭語のみで文章を抜き出してくると言った，より抽象的な表現の検索単語を用いて SeqBDD を構築することである．それに対応して，正解となるパターンの数も増やし，再度実験，評価を行う必要がある．

謝辞

本研究を進めるにあたり，多大なるご指導ご支援を頂きました九州大学大学院システム情報科学研究院の田中久美子教授に深く感謝の意を表します．最後に，本研究において様々なご協力をいただきました田中研究室の皆様方に深く感謝し，お礼を申し上げます．

参考文献

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, Vol. 22, No. 2, pp. 207–216, June 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pp. 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pp. 3–17, London, UK, UK, 1996. Springer-Verlag.

[4] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, Vol. 42, No. 1-2, pp. 31–60, January 2001.

[5] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential PAttern mining using a bitmap representation. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD '02)*, KDD '02, pp. 429–435, New York, NY, USA, 2002. ACM.

[6] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. pp. 215–224, 2001.

[7] Hajime Kitakami, Tomoki Kanbara, Yasuma Mori, Susumu Kuroki, and Yukiko Yamazaki. Modified prefixspan method for motif discovery in sequence databases. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*, PRICAI '02, pp. 482–491, London, UK, UK, 2002. Springer-Verlag.

[8] Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. Crystal inducing a conceptual dictionary. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pp. 1314–1319, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[9] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, Vol. 34, No. 1-3, pp. 233–272, February 1999.

[10] Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowl. Inf. Syst.*, Vol. 24, No. 2, pp. 235–268, August 2010.

[11] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pp. 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[12] Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association*

for Computational Linguistics, ACL-44, pp. 673–680, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[13] 松原勇介, 秋葉友良, 辻井潤一. 最小記述長原理に基づいた日本語話し言葉の単語分割. 言語処理学会第 13 回年次大会発表論文集, 2007.

[14] Susan Hunston and Gill Francis. *Pattern Grammar: A Corpus-driven Approach to the Lexical Grammar of English*. Studies in Corpus Linguistics4. JOHN BENJAMINS PUBLISHING COMPANY, 2000.

[15] Kumiko Tanaka-Ishii and Yuichiro Ishii. Multilingual phrase-based concordance generation in real-time. *Inf. Retr.*, Vol. 10, No. 3, pp. 275–295, 2007.

[16] Fabio Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pp. 303–366. IOS Press, 1999.

[17] Randal Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677–691, August 1986.

[18] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 12th edition, 2009.

[19] Shinichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, pp. 272–277, New York, NY, USA, 1993. ACM.

[20] Ryutaro Kurai, Shinichi Minato, and Thomas Zeugmann. N-gram analysis based on zero-suppressed bdds. In Takashi Washio, Ken Satoh, Hideaki Takeda, and Akihiro

Inokuchi, editors, *JSAI*, Vol. 4384 of *Lecture Notes in Computer Science*, pp. 289–300. Springer, 2006.

[21] 湊真一. BDD/ZDD を基盤とする離散構造と演算処理系の最近の展開. 電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, Vol. 4, No. 3, pp. 224–230, 2011.

[22] Shuhei Denzumi, Hiroki Arimura, and Shinichi Minato. *Suffix-DDs: Substring Indices Based on Sequence BDDs for Constrained Sequence Mining*. 2010.