2011 年度 修士論文

基数制約の概念を持つ
拡張 SAT ソルバの設計と実装

提出日： 2012 年 1 月 27 日
指導　： 上田　和紀 教授

早稲田大学大学院 基幹理工学研究科
情報理工学専攻

学籍番号： 5110B130-0
山根　裕二

概要

山根 裕二

SAT (boolean SATisfiability problem, 充足可能性問題) は，与えられた CNF (Conjunctive Normal Form, 乗法標準形) 形式の論理式を満たすような命題変数の割当てを見つけるか，あるいはそのような割当てが存在しないことを示す問題である．SAT は代表的な NP 完全問題であり，多項式時間以内での解法はまだ発見されていない．

近年，メモリや演算装置の高機能化や効率の良いヒューリスティックの発見により SAT ソルバの発展が目覚ましい．特に現在は CDCL (Conflict-Driven Clause Learning) ソルバが定理の自動証明や回路の検証，ソフトウェアの検証などの分野で実際に利用されている．しかし，そのような実世界の問題から生成される SAT 問題に含まれる基数制約はしばしば SAT ソルバの処理のボトルネックとなる．

本論文では，実問題のより高速な求解を目的として，基数制約を表現する特殊節を導入した拡張 SAT を提案する．代表的な CDCL ソルバである MiniSat2.2 をベースとして拡張 SAT ソルバ ex-sat を実装し，基数制約の占める比率が大きい時間割作成問題をベンチマークとして MiniSat2.2 との比較実験を行い，ex-sat の性能評価を行った．その結果，MiniSat2.2 と比較して逐次で算術平均で 6.7 倍，幾何平均で 1.8 倍の速度向上が得られた．

An Extended SAT Solver for Cardinality Constraints

Yuji YAMANE

SAT (boolean SATisfiability problem) is the problem of determining if the variables of a given Boolean formula in CNF (Conjunctive Normal Form) can be assigned in such a way as to make the formula evaluate to true, or determining whether no such assignments exist. SAT is the first known example of an NP-complete problem. That briefly means that there is no known algorithm that efficiently solves all instances of SAT.

Now, modern CDCL (Conflict-Driven Clause Learning) solvers can efficiently solve a large enough subset of SAT instances to be useful in various practical areas such as circuit design and automatic theorem proving and software verification, by solving SAT instances made by transforming problems that arise in those areas. However, Cardinality constraints generated from real-world problems are often a bottleneck of SAT solving.

This paper proposes an extended CNF that can express cardinality constraints and allows high-performance unit propagation. We have implemented it on MiniSat2.2, the newest version of MiniSat which is one of the most popular SAT solvers, by improving deduction and propagation to work with the extended CNF.

To evaluate the performance improvement over the original MiniSat2.2, we made an experiment on this extented solver, named ex-sat, with the ITC2007 benchmark set that contains cardinality constraints problems. Our experimental results show that ex-sat outperforms MiniSat2.2 by a factor of 1.8.

目次

図目次

表目次

第 1 章

はじめに

1.1 研究の背景と目的

命題論理の充足可能性問題 (Boolean Satisfiability Problem: SAT) は，代表的な NP 完全問題として計算機科学の分野で研究対象とされてきた．近年のハードウェアの性能向上とアルゴリズムの改善にともない SAT ソルバは数百万変数の大規模な問題も解けるようになり，現在ではソフトウェア検証やハードウェア検証のエンジンとしての利用が期待されている．しかし，実問題には基数制約 (Cardinality Constraints) が多く含まれる場合がある．そのような問題を SAT に変換して解かせようとすると，真偽値のみを扱う SAT ソルバでは制約の表現に多くの節を必要とし，実行中にメモリ不足に陥ることがある．本研究では，基数制約を含む実問題に対してよりスケーラブルかつ高速な求解を目的に，SAT の入力形式である CNF (Conjunctive Normal Form) に基数制約を表す特殊な節を加えた拡張 SAT を提案する．SAT ソルバの探索アルゴリズムを拡張 SAT に対し適用できるように実装し，基数制約を含む問題として代表的な時間割作成問題を拡張前と拡張後両方のソルバにそれぞれ解かせて比較することで，評価を行った．

1.2 論文構成

本論文の構成は以下の通りである.

- 第 2 章では，SAT とその探索アルゴリズムについて解説する.
- 第 3 章では，基数制約について解説する.
- 第 4 章では，提案となる拡張 SAT とそのソルバである ex-sat の探索アルゴリズムについて解説する.
- 第 5 章では，関連研究について述べる.
- 第 6 章では，提案手法の実験の内容と実験結果を記載する.
- 第 7 章では，まとめと考察，今後の課題について述べる.

1.3 基本用語

本論文において多様される基本的な用語を以下に記述する.

- 変数 (variable)
  論理式を構成する 2 値論理の変数．真か偽のいずれかを取る.
- リテラル (literal)
  変数の正か負の出現を表す．正の出現とは否定記号を伴わない変数を指し，負の出現とは否定記号を伴う変数のことを指す.
- 節 (clause)
  リテラルが論理和 (or) により結合したもの.
- 単位節
  本来はリテラルが 1 つの節 (unit clause) のことを指すが，本論文では k 個のリテラルを含む at-least-k 節 (後述) のことを指す.
- CNF (Conjunctive Normal Form，乗法標準形)
  節が論理積 (and) により結合したもの．SAT の入力の形式.
- decision (変数割当)
  解の探索において，未知の変数に値を割当てる処理.
- propagation (変数伝播)
  ある変数に値を割り振ることで，同時に他の変数にも自動的に値を決定すること.
- backtrack (バックトラック)
  現時点での探索を中止して，decision をやり直すこと.
- conflict (衝突)
  decision と propagation により変数の割当てを行った結果，節が偽になること.
- lemma (learnd clause，学習節)
  conflict を解析した結果，探索空間の剪定のために学習した節のこと.
- 決定レベル (decision level)
  ある変数が真偽値を割当てられたときの探索木における深度.
- 基数制約 (cardinality constraints)
  リテラルの真偽の個数に対し，上限 (または下限) を設定する制約.
- at-least-k 制約 (at-least-k constraints)
  基数制約の 1 つで，あるリテラルが k 個以上真であることを設定する制約.

- at-most-k 制約 (at-most-k constraints)
  基数制約の 1 つで，あるリテラルが k 個以下真であることを設定する制約.
- 特殊節 (special clause)
  本研究の根幹を成す，基数制約を表す節.
- at-least-k 節 (at-least-k clause)
  at-least-k 制約を表す節．この節に含まれるリテラルのうち k 個以上が真でなければならない．リテラルの正負を反転させることで, at-most-k 制約を at-least-(n-k) 節で表現できる.
- 通常節 (normal clause)
  特殊節に対して，従来の節をこのように呼ぶこととする．通常節は at-least-1 節と見なせる.
- 拡張 SAT (extended SAT)
  特殊節の導入により拡張された CNF を入力とする SAT.

第 2 章

SAT

本章では，SAT とその探索アルゴリズムについて解説を行う．この章の内容は主に参考文献 [2] [13] [15] および MiniSat2.2 [11] のソースコードに基づいている．MiniSAT は標準的な SAT ソルバであり，MiniSat2.2 は現時点 (2012 年 1 月) では最新版である．

2.1 概要

充足可能性問題 (Boolean Satisfiability problem: SAT) は与えられた命題論理式に対して，式全体を充足するような変数の割当てが存在するか，あるいはそのような変数の割当てが存在しないということを示す問題である．SAT は NP 完全な問題であることが証明されており，多項式時間での解法は存在しない．近年では SAT ソルバの発展が目覚ましく，ソフトウェア検証，ハードウェア検証などの分野での応用が期待されている．

2.1.1 入力例

SAT は節を論理積で繋げた **CNF** (Conjunctive Normal Form，乗法標準形) という形式で表現される．SAT ソルバは DIMACS という形式で与えられる CNF を入力とする．入力例を以下に記す．

```
p   cnf   4   4
c   4 variables, 4 clauses
1   2   0
1   -2   3   0
1   -3   0
3   4   0
```

入力の 1 行目には，慣習として「p cnf 変数の数 節の数」のように記述する．また，「c」から始まる行はコメントと見なされる．数字は変数，ハイフン記号の有無は変数の正負の出現を表す．各行が節とそれに含まれるリテラルを表しており，0 は終端記号である．よって上の入力は，4 変数，6 リテラル，4 節から成る CNF であり，次の論理式を表している．

入力の例が表す論理式

$\psi = (a \vee b) \wedge (a \vee \neg b \vee c) \wedge (a \vee \neg c) \wedge (c \vee d)$

CNF ψ に含まれる全ての節を充足させる変数の割当ては，例えば $a = True$，$b = False$，$c = True$，$d = False$ という割当は ψ を充足可能である．求める変数の割当が存在したため，ψ は**充足可能 (SAT) な問題**と分類される．一方，すべての節を充足する変数の割当が存在しない問題は**充足不可能 (UNSAT) な問題**と分類される．

2.1.2 SAT ソルバの種類

SAT ソルバは以下の 2 つに大別される．

- 系統的ソルバ (systematic solver)
  CNF を充足するような変数の割当が存在するか否かを判定する．

- 確率的ソルバ (stochastic solver)
  CNF を充足するような変数の割当のみを示す．一般に，その問題の充足不可能性は判定出来ない．

後者の確率的ソルバは変数の割当てをランダムに行うことで充足例を素早く解答することができるため，もともと充足可能でありかつ充足不可能性の証明が必要ない問題に適し

ている．本研究では前者の系統的ソルバを利用し，以後は SAT ソルバといえば系統的ソルバを指すこととする．

2.2 探索アルゴリズム

近年の SAT ソルバの標準である **CDCL** (Conflict Driven Clause Learning) ソルバの探索アルゴリズムの概要を以下に記す．本論文では後述する **DPLL** アルゴリズムとの混同を避けるため，CDCL ソルバの探索アルゴリズムを CDCL アルゴリズムと呼ぶことにする．CDCL アルゴリズムの基礎となっている DPLL アルゴリズムは，1960〜62 年に Davis, Putnam, Logemann, Loveland によって開発された．DPLL アルゴリズムは，Davis と Putnam によって考案されたすべての変数に resolution を適用する DP アルゴリズムに対して，Davis，Logemann，Loveland が必要メモリを縮小し，より効率的な探索を行う改良を加えたものを指す．

DPLL アルゴリズムに矛盾からの節学習の機能を加えたものが CDCL アルゴリズムである．CDCL アルゴリズムの処理の流れを参考文献 [15] より引用し，図 2.1 に記す．このアルゴリズムはループ版 DPLL とも呼ばれる．

dl は現在の決定レベルを表し，propagate 関数は後述する単位伝播を行って変数割当てを返す，analyze 関数は矛盾の原因を解析して後述する学習節およびバックトラックすべき決定レベルを C と bl にそれぞれ返す．lv 関数は変数値の決定レベルを返す．次に，CDCL 以下の主要な技術について述べていく．

- decision
- deduction
- conflict analysis

2.2.1 decision

未知の変数に対する値の割当てを行うフェーズである．その時点の変数の値の割当てだけでは他の変数への割当てが定まらない場合に行われる．適当なヒューリスティックに従って，値が割当てられていない変数の中から 1 つ変数を選び, 正または負の値を割当てる．問題によっては最初に割当てを行った変数が異なるだけで，数秒で終わる処理に何時間も掛かってしまうことがある．そのため，なるべく分岐の回数やバックトラックの回数を減らすことが出来る変数選択ヒューリスティックを採用することが重要である．

CDCL(ψ, ν)
ψ: a CNF formula, ν: a variable assignment
begin
  $dl := 0$
  loop
    ν = propagate(ψ, ν)
    if $\nu(\psi) = 1$ **then return** 1
    if $\nu(\psi) = 0$ **then**
      if $dl = 0$ **then return** 0
      (C, bl) := analyze(ψ, ν)
      $\psi := \psi \cup \{C\}$
      $\nu := \{(x_i, v_i) \in \nu \mid lv(x_i) \leq bl \}$
    $dl := bl$
    else
      $dl := dl + 1$
      choose an unassigned variable x and its
      value v
      $\nu := \nu \cup \{(x, v)\}$
end

図 2.1 CDCL アルゴリズム

以降に代表的なヒューリスティックを記載する.

- MOM (Maximum Occurrences on Minimum sized clauses)
  MOM は初期のヒューリスティックであり，最小の大きさの節の中で最も多く出現している変数を選択する．貪欲アルゴリズムのひとつで，推論の際に最も多くの変数が連鎖的に割り当てられるように考えられている．Random 分野など特定の問題にのみ効果が高いヒューリスティックである.
- DLIS (Dynamic Largest Individual Sum)
  DLIS は GRASP [5] で実装されたヒューリスティックであり，各変数の出現回数を保持するカウンタを用意し，最も多く出現した未割当の変数を選択する．カウンタの情報は状態依存であり，新たに割当てやバックトラックが発生する度にカウン

タの更新が必要という欠点が有る．

- VSIDS (Variable State Independent Decaying Sum)
  VSIDS は Chaff [6] で実装されたヒューリスティックであり，現在の CDCL ソルバで採用されている変数選択ヒューリスティックの原型となっている．DLIS のように各変数の出現頻度を保持するカウンタを用意するが，カウンタの更新が比較的少ない上に変数選択の効率が良い．

2.2.2 deduction

その時点での変数の値の割当てで連鎖的に値が決まる変数を探し，真理値を割当てていくフェーズである．deduction を実現するために様々な手法が考案されたが，その中で最も効果が高いのが unit clause rule である．ある節において 1 つのリテラルを除いたすべてのリテラルが False になる割当てをした場合，その節を True にするためには，残る 1 つのリテラルが True になるような割当てを行わなければならない．このような状態の節を**単位節** (unit clause)，リテラルを**単位リテラル** (unit literal) と呼ぶ．また，全ての単位リテラルを True になるような割当てを行う一連の作業を**単位伝播** (unit propagation) もしくは **BCP** (Boolean Constraint Propagation) という．BCP の仕組みは単位節の検知と値の割当てという単純なものだが SAT ソルバの処理時間の 80% 程度を占めると言われており，BCP の高速化は SAT ソルバの課題である．

図 2.2 に，単位伝播を行う propagate 関数のアルゴリズムを記載する．

propagate(ψ, ν)
ψ: a CNF formula, ν: a variable assignment
begin
  while $\exists\ C_{\geq 1} \in \psi$ ($C_{\geq 1}$ is a unit clause) **and**
  $\nu(\psi) \neq 0$ **do**
    $l :=$ the unassigned literal in $C_{\geq 1}$
    $\nu := \nu \cup \{(var(l), sign(l))\}$
  return ν
end

図 2.2 BCP を行う propagate() 関数のアルゴリズム

関数 propagate は入力として CNF 形式の論理式 ψ と値の割当ての組み合わせ ν を受

け取り，ν に新たな値の割当ての組み合わせを追加して返す．$\nu(\psi)$ は 1 のときは充足可能，0 のときは充足不能を意味する．未定リテラルを 1 つだけ含む節を単位節として，リテラルを真にするような変数割当を繰り返す．

以降に単位伝播の代表的な手法を記載する．

- カウンタ方式 (counter-based BCP)
  GRASP [5] などの初期の SAT ソルバに採用された手法で，全ての節にカウンタを用意する手法．具体的には，各節に 2 つのカウンタを設けて，それぞれ True あるいは False なリテラルの数をカウントし，変数割当ての度に節のカウンタを更新していく．更新後，節の False カウンタがリテラル数と等しくなった場合は衝突が発生していると判断する．節の False カウンタがリテラルの数より 1 つ少なくなっており，かつ True カウンタが 0 のとき，その節は単位節であると判断できる．昨今の SAT ソルバが利用している衝突の原因解析から生じる学習節は比較的長さが大きいため，カウンタの更新回数に節の長さが比例するこの手法はコストが高いものとなる．
- ヘッドテイル法 (Head/Tail List)
  節の中の 2 つのリテラルを監視する手法である．具体的には，各節に 2 つのポインタを持たせて，それぞれ Head ポインタと Tail ポインタとして，節の先頭のリテラルと末尾のリテラルを指させる．監視中のリテラルだけを意識すれば良いので，カウンタ方式よりも冗長な処理が少なく高速に動作する．しかし，バックトラックの際にポインタの移動に時間を要する欠点がある．
- 2 リテラル監視法 (2-literal watching)
  ヘッドテイル法と同じく節の中の 2 つのリテラルを監視する仕組みだが，昨今の CDCL ソルバで採用されている手法である．具体的には，各変数に 2 つのリストを設けてそれぞれ True あるいは False のリテラルが監視リテラルとして出現する節を保持する．ヘッドテイル法と同じように高速に動作する上に，一定の時間でバックトラックできるという特徴を持つ．これは，2 つの監視リテラルはその節で False になった最後のリテラルとなっており，バックトラック後には未割当てか True になることが保証されていることによる．

2.2.3 conflict analysis

ある変数の割当てが True かつ False と推論されることを**衝突**という．衝突が発生したとき，SAT ソルバは衝突の原因を解析し，適当な決定レベルまでバックトラックして変数割当てをキャンセルする必要がある．

以降に衝突の解析とバックトラックに関する手法を記載する．

- 時間順バックトラック (chronological backtracking)
  DPLL アルゴリズムでは，True と False 両方の割り当てを試していない直近の変数にバックトラックする時間順バックトラックという手法を採用していた．具体的には，値が割当てられた変数に対してその変数が True と False のどちらかが割当てられたかを示すフラグを用意し，衝突が起こった際，そのフラグを基に，まだ両方の割当てを試していない変数を解析し，その中で最も深いレベルの変数までバックトラックし，True か False の割り当てていない方の値を割当てを行なっていた．この手法はランダムに生成された SAT インスタンスに対して有効であると言われている．その一方で衝突を引き起こした原因の変数の決定レベルが浅い場合には，そこに戻るまでに無駄な探索を繰り返すことになる．
- 非時間順バックトラック (nonchronological backtracking)
  非時間順バックトラックは CDCL アルゴリズムにおいて採用されており，昨今の SAT ソルバには欠かせない手法となっている．衝突を解析することにより，衝突の直接の原因となったコアとなる割当てを**学習節** (learnt clause, lemma) として問題に追加し，更に，適切な決定レベルにバックトラックすることが可能となる．学習節は実行の面からは冗長な節だが探索木の剪定に有用であり，探索の高速化を促すことが期待できる．そこで，探索木の削減と学習節の追加による処理時間の増加のトレードオフとなり，効率の良い学習節を生成することが重要となる．
  以降に学習節の生成手法とそれに関連する用語を記載する．
  - 含意グラフ (implication graph)
    衝突が発生すると，まず含意グラフを作成する．含意グラフは割当てられた変数の関係を表す非循環有向グラフである．含意グラフの例を [15] より引用し，図 2.3 に示す．ノードに付記された文字列は変数及びその真偽値と決定レベルを表し，矢印は含意を表している．例えば，変数 x_5 が決定レベル 5 において True を割当てられたことで $\neg x_7@5$ は含意により False を割当てられたとい

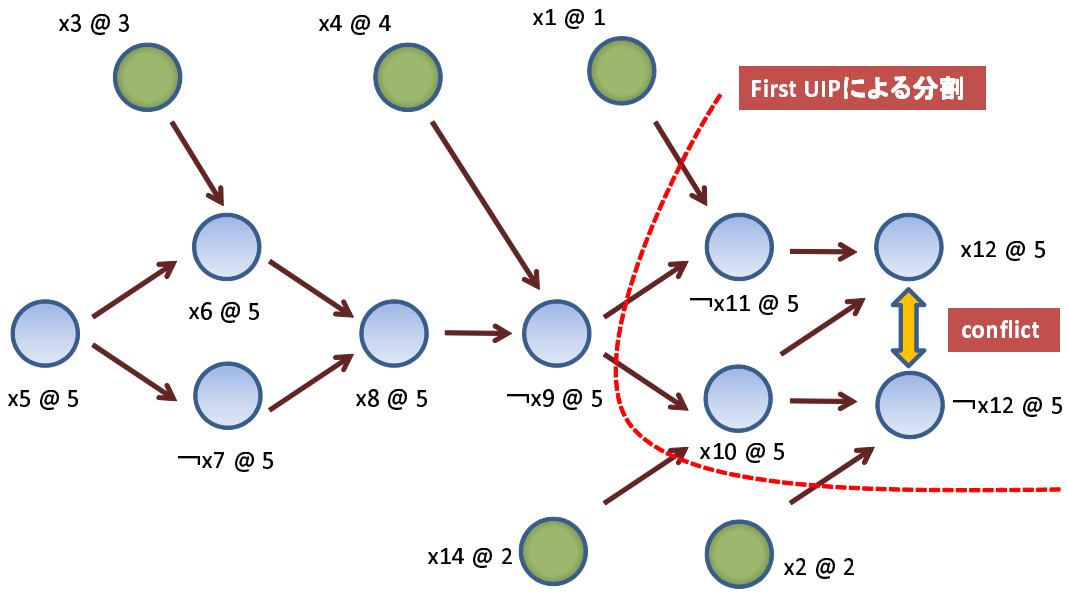

図 2.3　含意グラフ

うことを意味している．単位伝播によって連鎖的に割り当てられた場合，決定レベルは等しくなるので x_5 の割当てによって割当てられた変数は全て決定レベルが 5 となっている．

– UIP (Unique Implication Point)
含意グラフ内で衝突を起こした頂点から，現在の決定レベルにおいて decision によって値を割当てた変数までに必ず通る頂点を UIP と呼ぶ．図 2.3 では x_5，x_8，$\neg x_9$ が UIP である．

– first UIP
最も衝突を起こした頂点に近い UIP を first UIP と呼ぶ．図 2.3 では $\neg x_9$ が first UIP である．一般的に, first UIP により含意グラフを分割することで最良の学習節を生むと言われており，この場合は学習節 $C' = \{\neg x_1, \neg x_2, x_9, \neg x_{14}\}$ を元の問題に追加することになる．しかし，図 2.3 では省略しているが，実は頂点 x_1 と x_2 は x_{14} への矢印を持つので，C' を簡略化することが可能となり，学習節 $C'' = \{\neg x_1, \neg x_2, x_9\}$ を元の問題に追加すれば良い．ここではその全容を記載しないが，衝突を起こした節とその伝播の原因となった節とで resolution を取り，その結果生成された節と更にその伝播の原因となった節とで resolution を取るという処理を繰り返していくと C'' が生成できる．

学習節を追加すると，次にバックトラックすべき決定レベルを決める．その値は，

学習した節に含まれるリテラルの中で 2 番目に深い決定レベルとなる．例えば先ほどの学習節 C'' を例に考えると，各変数の決定レベルは $\{ 1, 2, 5 \}$ であり，本来 x_9 は決定レベル 2 の時点でその値を決定することが出来たと考えられる．
学習節の追加は処理速度やメモリ空間とのトレードオフであり，利用価値の低い学習節を持つのは好ましくない．そこで，MiniSat に代表される昨今の CDCL ソルバは学習節の価値の指標として活性度 (activity) に着目し，衝突の原因となるような学習節を優先的に残すようにして，逆に衝突に関わらない学習節を不要なものとしてデータベースから消去していく．

2.3 MiniSat2.2 における高速化手法

これまでは CDCL ソルバの基本となるアルゴリズムについて解説してきたが，以降は MiniSat2.2 の実装に着目し，高速化に繋がったと考えられる技法について解説していく．

- VSIDS-dvo (VSIDS dynamic variable order)
- 2 リテラル監視法
- リスタート (restart)
- phase saving (progress saving)
- blocking literals

2.3.1 VSIDS-dvo (VSIDS dynamic variable order)

MiniSat では，VSIDS の派生と言える VSIDS-dvo を採用している [2]．VSIDS-dvo のアルゴリズムを以降に示す．

1. 全ての変数に対しカウンタを用意し初期値 0 を与える．
2. 学習節がデータベースに追加されるとき，学習節を作るための導出過程に出現する節に含まれる変数のカウンタを増やす
3. 変数選択の際には，未定変数のうちカウンタの値が最大のものを選ぶ．
4. 定期的に，全てのカウンタの値を減らす．
5. 2 から 4 のステップを繰り返す．

VSIDS と VSIDS-dvo の違いは，変数のカウンタに対しリテラル単位の区別の有無である．VSIDS-dvo では変数に対して後述する phase saving が有効にならない限りは常

に False を割当てるため，変数の正負の出現を問わないと考えられる．VSIDS-dvo が選択する変数は衝突に関係しやすく，単位伝播を促しやすいと考えられている．更に，効率よく優先度の高い変数を選択させるために，変数をカウンタの値の順にヒープ木に格納して値の更新に柔軟に対応させている．

以前のバージョンである MiniSat2.0 では，変数選択の際に 2% の確率でランダムな選択を行っていたが，MiniSat2.2 では，デフォルト設定ではランダムな選択を一切行わず，完全に優先度のみを基準とした選択を行っている．

2.3.2 2 リテラル監視法

MiniSat では 2-literal watching を採用しており，各節に含まれるリテラルのうち先頭の 2 つのリテラルが単位伝播の対象として監視されている．ソルバ構造体に含まれる節のリスト watches[p] (ws) には，リテラル p を正負について反転させた p を配列の 0 番目か 1 番目に持つような節が格納されている．Solver::propagate() 関数が呼ばれた際には，割り当て済みのリテラル p を trail キューから取りだし，watches[p] で管理されている節の充足性をチェックする．以下に MiniSat2.2 の Solver::propagate() の大まかな動作を記載する．後の説明のために，blocking literals の機能は意図的に外している．

blocking literals を外した Solver::propagate() 関数

```
CRef Solver::propagate(){
    while ( trail is not empty ){
        Lit p = trail[qhead++]
        vec <Watcher>& ws = watches[p]
        make sure the false literal is c[1]
        for each clause of ws |c|{
            if ( c[0] is true ) {
                clause is already satisfied, continue
            } else
                for ( for i in 2..c.size()-1 )
                  if ( c[i] is not false )
                    swap(c[1], c[i]), looking for new watch
            if ( c[0] is false ) {
                conflict, return c
            } else
                unit_propagation()
        }
    }
    return CRef_Undef;
}
```

ws には p を 0 番目か 1 番目に持つような節が格納されていて，更に，p を True にするような割当てが行われていることから，ws の要素である節 c は必ず 0 番目か 1 番目に False になったリテラルを含むことになる．そこで，MiniSat では必ず 1 番目の要素に False になったリテラルを置くようにしている．図 2.4 に概略図を示す．

Solver::propagate() によって新たに単位節が生じた場合，未定変数に unit clause rule を適用した後，その変数の割り当てをリテラルとして trail キューに積み，trail のリテラルが全て取りだされるか，衝突が起こるまで処理を繰り返す．c[2] から c[c.size()-1] までを巡回して False でないリテラルを探し c[1] と交換することで，常に節の先頭 2 つに監視リテラルが来るように設定する．衝突が生じた場合には，その節を返し値として返し衝突の解析フェーズに移行する．

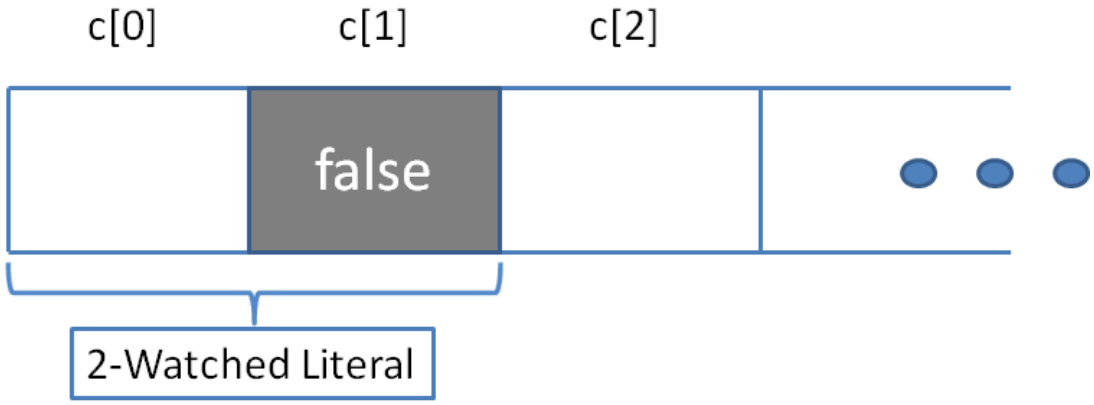

図 2.4 2 リテラル監視法

2.3.3 リスタート (restart)

CDCL ソルバにおいては, 類似した問題でも処理時間が大きく変化する場合が多々ある．例えば, CNF を構成する節やそのリテラルの並び順だけが違う (論理的には等価) だけでも，処理時間が大きく異なる場合がある．この現象の対処法として，random restart の使用が提案されている．これは，ランダムな時間間隔でそれまでの探索域を全て捨て去り最初から探索をやり直す方法である．一見無駄な処理だが，それまでに得た学習節は保存しているため，それまで調査した探索域を再び探索することは無く，局所解に陥ることを防いでいる．また，探索の初期に行うことで有効な学習節を得られ，ソルバの頑健性の向上にも繋がる．近年ではリスタート戦略の研究も盛んに行われており, luby strategy[3] や dinamic restart などのリスタート戦略が考案されている．

2.3.4 phase saving (progress saving)

phase saving (progress saving とも呼ばれる) は，RSat[8] により実装された，decision とバックトラックやリスタートに関連する技術である．具体的には，バックトラックやリスタートなどで割当てをキャンセルする際に，各変数の割当てを保持しておく．一見無駄な処理だが，この技術は従来の非時間順バックトラックにおける問題点から着想を得たと言われている．文献 [8] の例を挙げると，あるインスタンスがそれぞれがほぼ独立した 2 つの大きな制約のコンポーネントを含んでいて，更に，コンポーネント 1 を解いてからコンポーネント 2 を解くというようなケースを考える．図 2.5 に概略図を記載する．

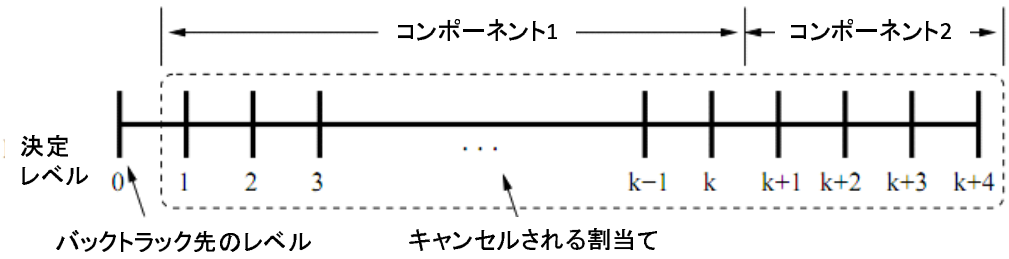

図 2.5 バックトラックやリスタートにより求解が困難になる例

コンポーネント 2 に取り掛かっているときに非時間順バックトラックやリスタートによって浅い階層まで一気に戻ってしまうという事態が想定され，最悪の場合はコンポーネント 1 を繰り返し何度も解くことになる．このような問題が従来の CDCL ソルバでは見られたが，phase saving を導入して割当てを記憶しておけば，decision をし直すことになっても 1 度解いたコンポーネントであれば難なく解くことが可能となる．phase saving は現在は多くの CDCL ソルバで利用され，MiniSat2.2 でも活用されている．

2.3.5 blocking literals

blocking literals は CMUSAT[4] により実装された，2 リテラル監視法による単位伝播を更に高速化する技術である．節 2.3.2 の Solver::propagate() に対して解析をかけると，最も時間を要するのは以下の箇所である．

Solver::propagate() 関数のボトルネック

```
CRef Solver::propagate(){
        ...
         make sure the false literal is c[1]
        ...
}
```

該当箇所のソースコードは次のようになっている．

Solver::propagate() 関数のボトルネック部分のコード

```
CRef Solver::propagate(){
        ...
        // make sure the false literal is c[1]
        ...
         Lit false_lit =  p;

         if (c[0] == false_lit)
            c[0] = c[1], c[1] = false_lit;
        ...
}
```

この入れ替え作業は，$c[1]$ に確実に False なリテラル (上のコードで言う $false\_lit$) を置くという内容だが，もしリテラル $c[0]$ が True の場合は，無駄な作業となる．無駄なメモリアクセスを防ぐために，各節に $false\_lit$ でない方の監視リテラルを参照するポインタを与え，上の入れ替え作業を行う前に $false\_lit$ でない方の監視リテラルが True かどうかを確認するようにした．これが blocking literals であり，無駄なメモリアクセスを減らして高速化することが出来た．MiniSat2.2 では実装済みの機能であり，多くのソルバに利用されている．

以下に blocking literals を取り入れた MiniSat2.2 の Solver::propagate() の大まかな動作を記載する．

MiniSat2.2 本来の Solver::propagate() 関数

```
CRef Solver::propagate(){
    while ( trail is not empty ){
        Lit p = trail[qhead++]
        vec <Watcher>& ws = watches[p]
        if ( blocker is true)              clause is already satisfied, continue
        make sure the false literal is c[1]
        for each clause of ws |c|{
            if ( c[0] is true ) {
                clause is already satisfied, continue
            } else
                for ( for i in 2..c.size()-1 )
                  if ( c[i] is not false )
                    swap(c[1], c[i]), looking for new watch
            if ( c[0] is false ) {
                conflict, return c
            } else
                unit_propagation()
        }
    }
    return CRef_Undef;
}
```

第 3 章

基数制約

本章では，基数制約について解説を行う．

3.1 概要

基数制約とは，0 か 1 を取る 2 値の整数変数の有限集合 $x_1, x_2, \ldots$ と正の定数 k が存在するとし，

$$\sum_{i=1}^{n} x_i \leq k \quad (\text{または} \sum_{i=1}^{n} l_i \geq k)$$

と記述される制約である [13]．一般に時間割問題や配置問題などの実問題で整数を取り扱う場面は多く，実問題と基数制約には密接な関係があると言える．SAT では真偽値のみを扱うので，単純な変換方法を用いると節の数は指数的に増加する．

3.2 擬似ブール制約

擬似ブール制約はブール制約の一般形である．0 か 1 を取る 2 値の整数変数の有限集合 $x_1, x_2, \ldots$ と整数 $a_1, a_2, \ldots$，k が存在するとし，

$$\sum_{i=1}^{n} a_i x_i \leq k$$

と記述される制約である [13]．基数制約は擬似ブール制約の特別な場合である．

第 4 章

拡張 SAT と拡張 SAT ソルバ ex-sat の設計

本章では，本論文において提唱する拡張 SAT と拡張 SAT ソルバ ex-sat の探索アルゴリズムについて解説を行う．

4.1 概要

拡張 SAT (extended SAT) は，従来の SAT に基数制約を表す特殊節を追加した問題である．実問題を SAT に変換する際に基数制約が登場することが多く，特殊節を用いて基数制約を簡潔に表現可能になることのメリットは大きい．また，SAT の探索アルゴリズムを特殊節に適応出来るように改造することで，拡張 SAT を高速に解くことが期待できる．

4.1.1 特殊節

基数制約を表現する**特殊節**とは，その節のリテラルの k 個以上が真であることを意味する at-least-k 節とその節のリテラルの k 個以下が真であることを意味する at-most-k 節を指す．

4.1.2 特殊節の記法 (リテラルの重複が無い場合)

リテラルの集合 C を含む at-least-k 節を $C_{\geq k}$ と表記することとする．通常節は $C_{\geq 1}$ と表記できる．そして at-most-k 節は $C_{\leq k}$ と表記することにする．ここで，特殊節の性

質を以下に記す.

$$C_{\geq k} = \overline{C}_{\leq n-k} \tag{4.1}$$

$$\{A \cup (\bigcup_{i=1}^{k} l_i)\}_{\geq x} \otimes \{B \cup (\bigcup_{i=1}^{k} \neg l_i)\}_{\geq y} = \{A \cup B\}_{\geq x+y-k} \tag{4.2}$$
$$(A \cap B = \emptyset \quad x, y, k \in N \quad k \leq min(x, y))$$

式 (4.1) の性質は，n 個のリテラルを持つ at-least-k 節はそのリテラルを全て正負について反 転させた at-most-$(n-k)$ 節と等しいことを意味する．つまり，at-least-k 節または at-most-k 節のどちらか片方を導入するだけで，基数制約をコンパクトに表現することが出来る．本研究では at-least-k 節だけを採用し，以降は特殊節の性質を述べる際は at-least-k 節の場合だけを考えることとする．式 (4.2) は以降の章で述べる単位伝播や矛盾解析において重要な性質を示している．$\otimes$ は 2 つの節の resolution を意味している．あるリテラルの有限集合 A を含む at-least-x 節とあるリテラルの有限集合 B を含む at-least-y 節がそれぞれ互いに正負の反転したリテラルを k 個持つ場合，resolution すると A と B の和集合の全要素をリテラルとして含む at-least-$(x+y-k)$ 節となる．$\{A \cup (\bigcup_{i=1}^{k} l_i)\}_{\geq x}$ と $\{B \cup (\bigcup_{i=1}^{k} \neg l_i)\}_{\geq y}$ を両方真にすることを考えたとき，$\{A \cup B \cup (\bigcup_{i=1}^{k} l_i) \cup (\bigcup_{i=1}^{k} \neg l_i)\}_{\geq x+y}$ を満たす必要があるが，この式は k 個の正負のペアを含むため式 (4.2) のように変形できる．

4.1.3 特殊節の記法 (リテラルの重複が有る場合)

リテラルの重複がある場合，resolution を行った結果リテラルの係数の絶対値が 1 を超えることになる．係数が 1 のときのみを考える基数制約で扱うために，cardinality の値と共に係数を調整する操作が必要となる [1].

4.1.4 入力例

拡張 SAT ソルバの入力形式は DIMACS 形式を踏襲しており，特殊節を認識するために「Ln l_1 l_2 ...0」 $(n \in \mathbb{N})$ という記法を追加している．入力例を以下に記す.

```
p    cnf    4    5
c    4 variables, 5 clauses
1    2    0
1    -2    3    0
1    -3    0
3    4    0
L2    1    2    3    4    0
```

ほとんどの記述は SAT の入力例と同一だが，最後の行の節は，変数 1 から変数 4 のうち 2 つ以上が真となる制約を表している．上の式を SAT で表現する場合は次のようになる．

```
p    cnf    4    8
c    4 variables, 5 clauses
1    2    0
1    -2    3    0
1    -3    0
3    4    0
1    2    3    0
1    2    4    0
1    3    4    0
2    3    4    0
```

このように，特殊節を導入することで制約を簡潔に記述することが出来るようになる．

4.2 拡張 SAT ソルバ ex-sat の実装に向けて

ex-sat は MiniSat2.2 をベースとしている．MiniSat2.2 は問題を読み込むと，各節をリテラルの配列などを含む Clause 構造体として保持する．ex-sat の特殊節は MiniSat2.2 の Clause 構造体を改造することで実現している．以降に特殊節の構造を示す．

特殊節の構造

```
class Clause {
    struct {
        unsigned mark       : 1; // before: mark = 2
        unsigned exclause   : 1; // added
        unsigned learnt     : 1;
        unsigned has_extra  : 1;
        unsigned reloced    : 1;
        unsigned size       : 27;
    } header;

    union  Lit lit; int type; float act; uint32_t abs; CRef rel;  data[0];
   ...
}
```

MiniSat2.2 では mark というフラグを 2 ビットで持っているが，ex-sat では 1 ビット分の領域を特殊節であることを示す exclause フラグに充てている．また，自身の cardinality (at-least-k における k) を表す整数を保持する必要があるので，int 型の type という変数を追加している．

4.3 ex-sat の探索アルゴリズム

拡張 SAT の探索アルゴリズムは，SAT の探索アルゴリズムである CDCL を拡張させたものである．具体的には，deduction フェーズに特殊節を解く処理を加えて，conflict analyze フェーズの resolution 処理を従来のものから式 (4.2) のものに変更すれば良い．

4.3.1 deduction の拡張

節 2.2.2 では通常節 (at-least-1) の単位伝播について解説した．at-least-1 節の場合は未定リテラルが 1 個の節を単位節と見なしていたが，at-least-k 節の場合は未定リテラルが k 個の節を単位節と見なす．図 4.1 に，拡張した propagate() 関数 (拡張前は節 2.2.2 参照) を記載する．

特殊節の単位伝播は，通常節の単位伝播の後に衝突が無ければ行われる．図 4.2 に，特殊節の単位伝播を行う propagate_LC() 関数を記載する．

propagate(ψ, ν)
ψ: a CNF formula, ν: a variable assignment
begin
  while $\exists\, C_{\geq 1} \in \psi$ ($C_{\geq 1}$ is a unit clause) **and** $\nu(\psi) \neq 0$ **do**
    $l :=$ the unassigned literal in $C_{\geq 1}$
    $\nu := \nu \cup \{(var(l), sign(l))\}$
  if $\nu(\psi) = 1$ **then**
    $\nu =$ propagate_LC(ψ, ν)
  return ν
end

図 4.1 ex-sat の propagate() 関数のアルゴリズム

propagate_LC(ψ, ν)
begin
  while $\exists\, C_{\geq k} \in \psi$ ($C_{\geq k}$ is a k-ary clause) **and** $\nu(\psi) \neq 0$ **do**
    while $\exists\, l \in C_{\geq k}$ (l is an unassigned literal)
      $\nu := \nu \cup \{(var(l), sign(l))\}$
  return ν
end

図 4.2 ex-sat の propagate_LC() 関数のアルゴリズム

4.4 従来の高速化手法の適用

4.4.1 VSIDS-dvo

特殊節は通常節よりも情報量が多く，特殊節に含まれる変数に対して優先的に割当てを行えば効果的な伝播が期待できると考えられる．現在の実装では，cardinality の個数に比例して優先度を増やしている．

4.4.2 リテラル監視法

at-least-1 節の場合は 2 個のリテラルを監視すれば十分だったが，at-least-k 節の場合は k+1 個のリテラルを監視することになる．特に，長さ n の節で at-least-(n-1) 即ち at-most-1 節の場合は，節に含まれるリテラルを全て監視する必要がある．

propagate_LC() 関数が呼び出されたとき，必ず節の先頭から k-1 番目の監視リテラルのうち 1 つは False になっており，監視リストは更新する必要がある．しかし，at-most-1 節の場合は全てのリテラルを監視するのでリストの更新の必要がない．そこで，ex-sat では at-least-1 節と at-least-k 節と at-most-1 節を区別して保持し，それぞれ異なる関数で処理している．

4.4.3 リスタート

現時点では評価実験を重ねていないので，MiniSat2.2 の実装をそのまま引き継いでいる．

4.4.4 phase saving

こちらも MiniSat2.2 の実装をそのまま引き継いでいる．

4.4.5 blocking literals

blocking literals は，2 リテラルを監視する場合には無駄なアクセスを省き有効だったが，k+1 個を監視することになる特殊節の場合は参照している 1 つのリテラルが True になっていることを確認する意味が薄く，逆に余分なアクセスを招き効率が低いと考えられる．ex-sat では通常節のみ blocking literals を有効にしている．

第 5 章

関連研究

本章では関連研究について述べる．具体的には，基数制約への取組みに関する研究や特殊な節を導入して CNF を拡張させることで高速化を図った研究を挙げる．

5.1 基数制約に関する先行研究

基数制約を扱う先行研究としては，擬似ブール制約 (Pseudo-Boolean Constraints) が挙げられる．擬似ブール制約は変数がブール値を取る制約充足問題であり，$\sum_{i=1}^{N} a_i x_i \geq b$ の形の線形制約を主な対象とする．a_i と b は整数，x_i はリテラルであり，真のとき 1，偽のとき 0 と解釈する．$a_i = 1, i \in (1, 2, ..., N)$ のとき，基数制約とみなせることからも明らかなように，擬似ブール制約ソルバでは基数制約をそのまま記述可能である．擬似ブール制約ソルバとしては SAT4J などが代表的であるが，本稿での実験で利用したベンチマーク問題を解かせたところ，いずれの問題においても求解に膨大な時間を要し，SAT ソルバと比較すると速度に大きな差があった．そこで，本研究では疑似ブール制約ソルバを用いず，SAT ソルバをベースとしての基数制約へのアプローチを試みた．

一方，疑似ブール制約ソルバに倣うべき機能として，基数制約を学習するための resolution が挙げられる．節 4.1.2 の式 (4.2) はリテラルが重複する場合に基数制約ではなく疑似ブール制約を生成してしまう．文献 [1] では，生成した疑似ブール制約を極力情報を削ぐことなく基数制約として学習するための手法が記述されている．

5.2 特殊節を導入した拡張 CNF に関する先行研究

特殊な節の導入に関しての先行研究では，基本対称関数に基づく節 (ES 節と呼ばれる) の導入による CNF の拡張が挙げられる [14]．ES 節とは「n 個の変数のうち，ちょうど k 個が真」という制約を意味する節を表す．この研究に対する本研究の新規性としては，at-least-k 節と at-most-k 節をそれぞれ分割しているため ES 節よりも柔軟に制約の記述が可能な点や，特殊節に対して監視リテラルを設けている点，既存の SAT ソルバである MiniSat に対して拡張，実装している点などが挙げられる．

特殊な節の導入に関する先行研究としては，他にも SAT-Race 2010 の Main Track 部門で優勝した CryptoMiniSat [10] が挙げられる．CryptoMiniSat は排他的論理和を表す特殊節を導入することで暗号問題などにおいて高速な処理を可能にしている．

第 6 章

評価実験

本章では，ex-sat とその原型となる MiniSat2.2 の比較実験の結果を記載する．比較のために，基数制約がボトルネックになるような問題を対象とした．

6.1 時間割作成問題

本論文では，基数制約を多く含む代表的なベンチマーク問題として時間割作成問題を選択した．

6.1.1 ITC2007

時間割作成問題の競技会である International Timetabling Conpetiton 2007 (ITC 2007) [3] の大学の講義の時間割作成問題である Post Enrolment Course Timetabling 部門で提供されるベンチマーク 24 題を実験の対象として選んだ．

6.1.2 Post Enrolment Course Timetabling

本部門の問題は数百人の生徒が特定の教室で特定の講義を履修する状況を想定し，数百もの講義に対して 45 個のタイムスロット (5 日制で 9 時限) で定められる日程と教室を適切に割り当てることが求められている．各問題において以下の要素が含まれる．

- 授業数，教室数，教室の特徴の数，生徒数
- 各教室の収容人数
- 各生徒が履修する授業の集合

- 各教室が有する特徴の集合
- 各授業に求められる教室の特徴の集合
- 各授業の割当てられないタイムスロットの集合
- 各授業のタイムスロットの前後関係の制約の集合

各授業に対して適切なタイムスロットと教室を割当てる必要がある．

6.1.3 最適解の探索

ベンチマーク問題には必ず満たす必要のある Hard Constraints (H1 ∼ H5) と満たしていないとペナルティとなる Soft Constraints (S1 ∼ S3) が存在する．以降に，Hard Constraints と Soft Constraints を記す．

- H1: どの学生も同時刻に 2 つ以上の授業を履修することはない．
- H2: 教室は常に履修人数以上のサイズであり，授業が教室に求める特徴を全て満たしている．
- H3: 同時刻に同じ教室で 2 つ以上の授業は行われない．
- H4: 各授業が実施可能なタイムスロットに割当てられている．
- H5: 各授業が指定された正しい順序で割当てられている．
- S1: 生徒は，1 日の最後の時限に授業を取らない．
- S2: 生徒は，3 つ以上連続して授業を取らない．
- S3: 生徒は，1 日に 1 つの授業だけを取ることはない．

上述の制約を完全に満たすような最適解が必ず 1 つ存在することが明記されており，本実験では最適解を求めることを目標にしている．そのため，実際には Hard Constraints と Soft Constraints を区別する必要はない．ただし，ベンチマーク問題はもともと大規模な問題なので最適解を求めることは難しく，独自に生徒の数を変数としてその値を変更することで問題の縮小化を行っている．

6.1.4 特殊節の導入による節数の削減効果

通常節のみを用いてベンチマーク問題の制約を単純な変換方法で CNF に変換した場合の節数のオーダーと，特殊節を導入した拡張 CNF に変換した場合の節数のオーダーを表 6.1 に，記す．表中の変数 e は授業の数，s は生徒の数，k は教室の数，$|\overline{C_m}|$ は生徒の授業の履修数の平均値を表す．制約 H1 と S2 と S3 が基数制約であり，H1 は at-most-1

節，S2 は at-most-2 節，S3 は at-least-2 節に対応している．特に，S2 と S3 は従来の CNF 形式に変換したときの節数の 9 割前後を占めており，この 2 制約が大きなボトルネックと言える．生徒の授業の履修数の平均値は各問題では 10 前後であり，特殊節を導入することで実質的に節数を 1% から 10% 前後にまで減らすことが可能である．

表 6.1 特殊節の導入による節数の削減効果

制約	通常節のみ	通常節 + 特殊節
H1	$O(e^2 \cdot s)$	$O(s)$
H2	$O(e)$	$O(e)$
H3	$O(e^2 \cdot k)$	$O(e^2 \cdot k)$
H4	$O(e)$	$O(e)$
H5	$O(1)$	$O(1)$
S1	$O(e)$	$O(e)$
S2	$O(s \cdot \lvert\overline{C_m}\rvert^3)$	$O(s \cdot \lvert\overline{C_m}\rvert)$
S3	$O(s \cdot \lvert\overline{C_m}\rvert^2)$	$O(s \cdot \lvert\overline{C_m}\rvert)$

6.2 実験環境

ITC 2007 のインスタンスを CNF と拡張 CNF に変換して，前者を MiniSat2.2 に，後者を ex-sat にそれぞれ解かせて結果を比較した．拡張 SAT については，特殊節の導入の効果を調べるために以下のような 3 通りの変換方法を試す．

- at-least-2 制約 (S3) だけを特殊節に変換 (以下，L2)
- at-least-2 制約と at-most-1 制約 (H1) を特殊節に変換 (以下，L2&M1)
- at-least-2 制約と at-most-1 制約 と at-most-2 制約 (S2) を特殊節に変換 (以下，L2&M1&M2)

実験環境は表 6.2 に記している．

表 6.2　実験環境

OS	CentOS 5.3
CPU	Intel Xeon(R) X5650 2.67GHz
メモリ容量	48GB
gcc version	4.1.2

6.3　実験結果

それぞれのソルバにおける各問題の実行時間を図 6.1 に載せ，総実行時間を表 6.3 に載せる．

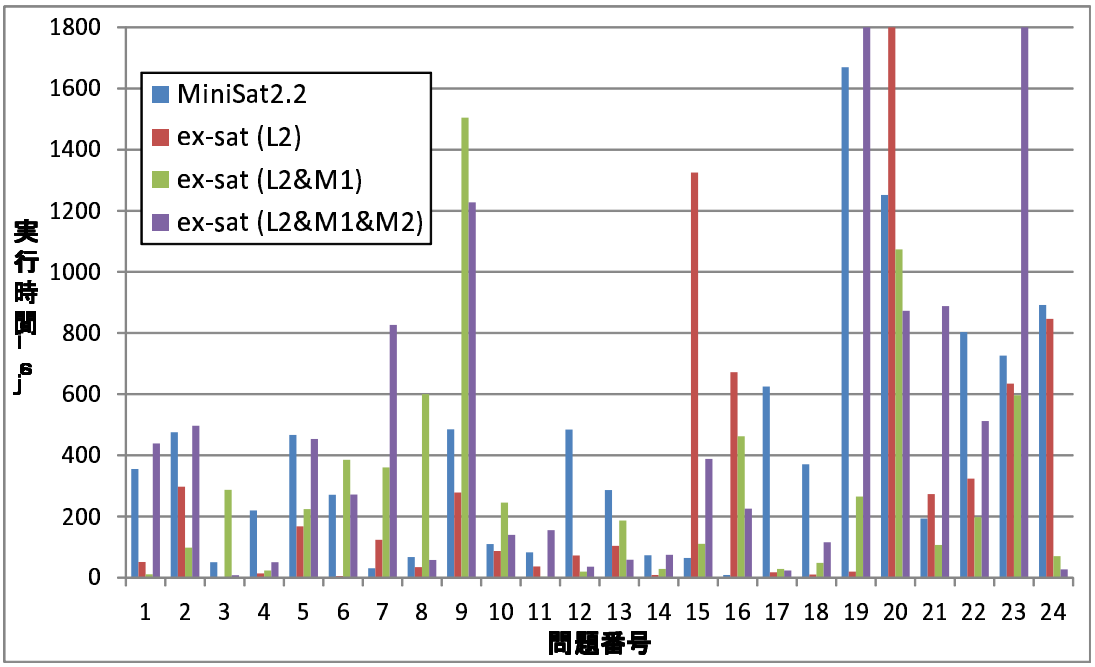

図 6.1　MiniSat2.2 と ex-sat の比較 (実行時間)

24 問のいずれも，ITC2007 で提供されるベンチマーク問題の生徒数を減らしている．生徒数の縮小は，実行時間が 10 秒以上でかつ 1800 秒を超えない間に収まる程度に調整している．3 回の実験のうち、もっとも良い数値を採用したが，各結果にほとんど差は無い．

表 6.3　MiniSat2.2 と ex-sat の比較 (総実行時間)

MiniSat2.2	ex-sat(L2)	ex-sat(L2&M1)	ex-sat(L2&M1&M2)
10068.8 秒	7208.7 秒	6944.8 秒	10953.6 秒

総実行時間に着目して MiniSat2.2 の結果と比較すると, ex-sat (L2) と ex-sat (L2&M1) は短くなっており，ex-sat (L2&M1&M2) は長くなっていた．最も短かった ex-sat (L2&M1) は MiniSat2.2 と比較して，算術平均で 6.7 倍，幾何平均で 1.8 倍の高速化が達成できた．また，最高で 32.0 倍の速度向上が見られた．ex-sat (L2) と ex-sat (L2&M1) と ex-sat (L2&M1&M2) の間では，速く解けた問題はそれぞれ異なっていた．

次に，それぞれのソルバにおける各問題の消費メモリを図 **??** に載せ，総消費メモリ数を表 6.4 に載せる．

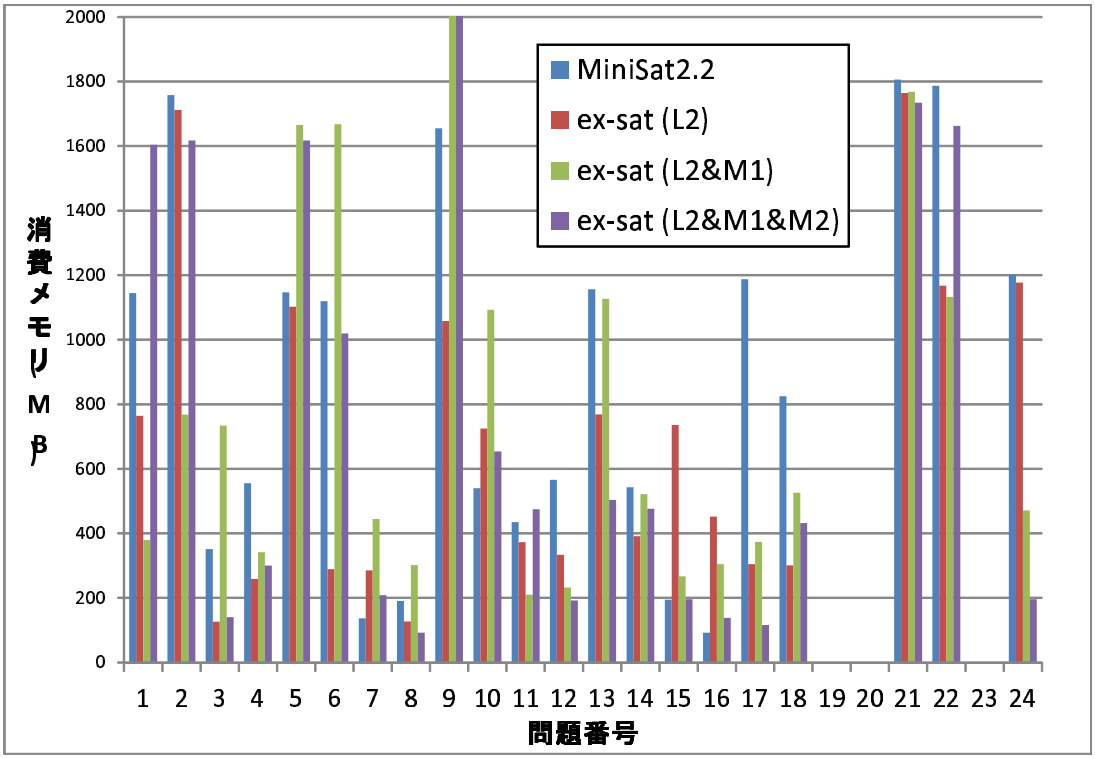

図 6.2　MiniSat2.2 と ex-sat の比較 (消費メモリ)

SAT ソルバにおいて実行時間と消費メモリの量には相関がある．入力として与えられる節を特殊節に変換することで消費メモリを減らす効果があることと矛盾するように思わ

表 6.4 MiniSat2.2 と ex-sat の比較 (総消費メモリ)

MiniSat2.2	ex-sat(L2)	ex-sat(L2&M1)	ex-sat(L2&M1&M2)
18392 MB	14128 MB	18444 MB	15953 MB

れるが，これは，SAT ソルバが実行中に学習節の保持する限界量を徐々に増やしていくことや配列の入れ替えを繰り返していくうちにメモリリークが消費することによる．

第 7 章

まとめと今後の課題

本章では，まとめと考察，今後の課題を述べる．

7.1 まとめと考察

基数制約を表す特殊節を含む拡張 CNF を解く拡張 SAT ソルバ ex-sat を設計，実装した．ex-sat (L2) と ex-sat (L2&M1) については，特殊節の導入の有用性を確認できた．ex-sat (L2&M1&M2) は予想に反して実行に時間がかかっていた．

7.1.1 実験結果のまとめ

基数制約を多く含む時間割作成問題をベンチマークとして MiniSat2.2 と ex-sat を比較した．

総実行時間で比較したところ，ex-sat (L2&M1) が最も短く，ex-sat (L2) が次に短く，ex-sat (L2&M1&M2) は最も長かった．

一方，各問題の MiniSat2.2 の実行時間とそれぞれの ex-sat の実行時間との比を取ると，ex-sat (L2) は算術平均で 12.9 倍，幾何平均で 2.8 倍，最も速く解けた問題では 83.6 倍の速度向上が得られた．ex-sat (L2&M1) は算術平均で 6.7 倍，幾何平均で 1.8 倍，最も速く解けた問題では 32.0 倍の速度向上が得られた．ex-sat (L2&M1&M2) は算術平均で 4.3 倍，幾何平均で 1.2 倍，最も早く解けた問題では 32.6 倍の速度向上が得られた．

7.1.2 考察

基数制約は SAT においては節数の増大を招くことが想定され，SAT を拡張し基数制約を 1 つの節の構造体にまとめることで速度向上が確認できた．

今回のベンチマーク問題は全て SAT 問題であり，偶然良い選択が行われた結果，素早く解を発見することも考慮する必要がある．

どの制約を特殊節に変換するかによって問題ごとに速度の差が出るのは，通常節を特殊節に変換することで VSIDS による変数選択の順序に影響を及ぼし適切でない選択を行ってしまうことが原因として考えられる．

7.2 今後の課題

現段階の実装では特殊節を学習する機能が無いが, 文献 [1] などを元に適切な resolution を行うようにして特殊節の学習機能を追加することで，更なる高速化が実現できると考えられる．さらに，特殊節の導入による速度向上を確認できたので，より大規模な実問題の求解を目指してクラスタ環境向けの並列 SAT ソルバである c-sat [7] のアイデアを元に，特殊節の学習節を授受する並列 SAT ソルバの実装を今後の課題とする．

謝辞

本研究を進めるにあたって, 数々のご指導を賜わりました上田和紀教授に深く感謝致します. また，仲良くして下さった研究室の同期や後輩の方々，友人達に感謝致します．最後に，現在に至るまで，あらゆる面で支えてくれた家族に深く感謝致します.

2011 年 2 月 山根 裕二

参考文献

[1] Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver, in Proc of DAC-03, pp. 830–835, 2003.

[2] Eén, N. and Sörensson, N.: An Extensible SAT-solver, in Proc of SAT 2003, LNCS 2919, Springer, pp. 502–518, 2004.

[3] International Timetabling Competition 2007, 2007.
http://www.cs.qub.ac.uk/itc2007/

[4] Jain, H. and Clarke, E. : Sat solver descriptions: Cmusat-base and cmusat. System description for the SAT competition 2007.

[5] Marques-Silva, J. P. and Sakallah, K. A. : GRASP: A New Search Algorithm for Satisfiability, in Proc of ICCAD 1996, pp. 220-227, 1996.

[6] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. : Chaff: Engineering an Efficient SAT Solver, in Proc of DAC-01, pp. 530-535, 2001.

[7] Ohmura, K. and Ueda, K.: c-sat: A Parallel SAT Solver for Clusters, in *Proc. SAT 2009*, LNCS 5584, Springer, pp. 524–537, 2009.

[8] Pipatsrisawat, K. and Darwiche, A. : A Lightweight Component Caching Scheme for Satisfiability Solvers, in Proc of SAT 2007, LNCS 4501, Springer, pp. 294–299, 2007.

[9] Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints, in Proc of CP 2005, pp. 827–831, 2005.

[10] Soos, M.: CryptoMiniSat 2.5.0, 2007.
http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_13.pdf

[11] Sörrenson, N.: MINISAT 2.2 and MINISAT++ 1.1, 2010.
http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_25+26.pdf

[12] Zhang, L. and Malik, S.: The Quest for Efficient Boolean Satisfiability Solvers,

in Proc of CAV 2002, LNCS2404, Springer, pp. 17–36, 2002.

[13] 井上克巳, 田村直之: SAT ソルバーの基礎, 人工知能学会誌, Vol. 25, No. 1 (2010), pp. 57–67, 2010.

[14] 馬野洋平, 酒井正彦, 西田直樹, 坂部俊樹, 草刈圭一朗: 基本対称関数を付加した CNF 論理式の充足可能性判定, IEICE Transactions on Information and Systems, Vol. J93–D, No. 1, pp. 1–9, 2010.

[15] 鍋島英知, 宋剛秀: 高速 SAT ソルバーの原理, 人工知能学会誌, Vol. 25, No. 1(2010), pp. 68–76, 2010.

[16] 山根裕二, 徐暁雋, 上田和紀: 基数制約の概念を持つ SAT ソルバの設計と評価, 人工知能学会第 25 回全国大会, 3J1-OS7-4, 2011.

発表論文

[1] 山根裕二, 徐暁雋, 上田和紀: 基数制約の概念を持つ SAT ソルバの設計と評価, 人工知能学会第 25 回全国大会, 3J1-OS7-4, 2011.

[2] 徐暁雋, 山根裕二, 上田和紀: 基数制約に対応するクラスタ向け並列 SAT ソルバとその評価, 2011 年 並列/分散/協調処理に関する『鹿児島』サマー・ワークショップ, 電子情報通信学会技術報告, Vol. 111, No. 164, DC2011-17, pp. 13–18, 2011.

受賞論文

[1] 2011, 人工知能学会全国大会優秀賞
山根裕二, 徐暁雋, 上田和紀: 基数制約の概念を持つ SAT ソルバの設計と評価, 人工知能学会第 25 回全国大会, 3J1-OS7-4, 2011.