2012 年度 修士論文

Literal Block Distance に基づく学習節の共有を行う並列SAT ソルバ Glucans

提出日　：　2013 年 1 月 26 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：5111B052-8

徐 暁雋

概要

命題論理式の充足可能性問題 (propositional SATisfiability problem: SAT) は，情報標準形で与えられた命題論理式に対して，式が真になるような変数を割り当てるか，そのような割り当てが存在しない事を示す問題である．SAT にはハードウェア検証やソフトウェア検証，人工知能問題などの様々な応用が存在する．近年の SAT ソルバは，以前より高速に SAT 問題を解く事ができるが，その多くは逐次プログラムであり，実際に SAT ソルバの競技会である SAT Competition 2011 で初めて，並列 SAT ソルバが逐次 SAT ソルバを上回った。近年，並列 SAT ソルバの研究は非常に重要となってきている．

本研究では，並列 SAT ソルバ Glucans を作成した．並列 SAT ソルバ Glucans は SAT Competition 2011 で高速だった逐次 SAT ソルバ Glucose, GLUEMINISAT, Contrasat, CIRMiniSat をスレッドで実行し，Conflict-Driven Clause Learning アルゴリズムの推論や衝突解析フェーズ中に，Literal Block Distance で選ばれた学習節の共有を，各スレッドで行う．

Glucans の評価を行うため,2012 年度の SAT の競技会である SAT Challenge 2012 (SC2012) の Parallel Track: Application SAT+UNSAT トラックに参加した．SC2012 は，制限時間 900 秒以内で解けた問題数を競う競技会で，Glucans は全 600 題中 521 題解くことができ 4 位であったが，1 位との差はわずか 10 問であった．また，Glucans は 179 題の問題で最も高速に求解可能で，2 番目に多い SAT ソルバの PeneLoPe の 2 倍以上である．

SC2012 では 900 秒という短い制限時間で結果の差異も小さい事から，追加実験として難しく巨大な問題を解くために制限時間を 5000 秒に延長し，SC2012 の問題に過去の競技会の問題を加えた問題セットで，Glucans と SC2012 で 2 位だった PeneLoPe を比較した．その結果，Glucans は長時間かかる問題ほど制限時間内に解を求める事ができ，推論中に学習節の共有を行うことがパフォーマンスを大きく向上させている事が分かった．

Glucans: A parallel SAT solver with LBD-based sharing of learnt clauses

Xiaojuan XU

SAT (propositional SATisfiability problem) is to determine that the variables of a given boolean formula in CNF (conjuctive normal form) can be assigned to satisfy the formula, or no such assignments exist. SAT has many applications such as hardware verification, software verification, and artificial intelligence problems that can be encoded into CNF. Modern SAT solvers can solve any SAT problems faster than before, but most SAT solvers are single-core programs. Actually, of the competitions of SAT solvers, SAT Competition 2011 was the first competition where a parallel SAT solver won other single-core SAT solvers. It is very important to parallelize SAT solvers nowadays.

In this research, we developed a parallel SAT solver called Glucans which runs threads based on some fast SAT solvers in SAT Competition 2011 such as Glucose, GLUEMINISAT, Contrasat, and CIRMiniSat. In Glucans, each thread shares learnt clauses selected based on Literal Block Distance (LBD) in the conflict analysis and propagation phases of the Conflict-Driven Clause Learning algorithm.

To evaluate its performance, we submitted Glucans to the "Parallel Track: Application SAT+UNSAT" track of SAT Challenge 2012 (SC2012), which was the main SAT competition of 2012. SC2012 was a competition to compare SAT solvers by the number of solved problems in 900 seconds. Glucans solved 521 of a total of 600 problems and ranked 4th at SC2012. The difference of the number of solved number between Glucans and the first-place solver was only 10. Glucans was the fastest at 179 problems, which was more than twice of PeneLoPe that was the second fastest solver.

Because of the small difference of the number of solved problems, we made another experiment to evaluate Glucans and PeneLoPe, by solving more difficult or large problems including the problems of competitions from 2010 to 2012 and by extending the time limit to 5000 seconds. This experimental result shows that Glucans is faster than PeneLoPe in the difficult or large problems, and sharing learnt clauses in propagation phases contributes much to the performance.

目次

定義一覧

図目次

表目次

第 1 章

はじめに

1.1 研究の背景と目的

命題論理式の充足可能性問題 (propositional boolean SATisfiability: SAT) は， ある命題論理式が与えられた際に，式が真になるような変数割り当てを求めるか，どのような割り当てをしても真にならないことを示す問題である．SAT ハードウェア検証，AI プランニングなどの問題は多項式時間で SAT に還元可能であり，SAT を高速に解くことが出来れば，これらの問題も高速に解くことができるようになる．特にソフトウェア検証において SAT ソルバがエンジンとして採用されているものとして，C Bounded Model Checker (CBMC)[21, 22] による ANSI-C プログラムの検証やその他の言語についても Coverity Static Analyzer[19, 20] が挙げられるによる．これらは実際に静的なエラー解析のために実用化されており利用されている．

本研究ではマルチコアを利用する並列 SAT ソルバを研究対象としている．近年 SAT 問題を解く SAT ソルバは高速化が進んでいるが，その多くは逐次プログラムの SAT ソルバで，SAT ソルバの求解速度を競う競技会である SAT Competition や SAT Race において，並列 SAT ソルバが逐次 SAT ソルバを上回る結果を残したのは 2011 年に開催された SAT Competition 2011 の Plingeling が最初である．このため，マルチコアやメニーコア化が進む中で並列 SAT ソルバの研究は非常に重要となってきている．

1.2 論文構成

本論文では，第 2 章で SAT の概要と SAT の基本的な求解アルゴリズムについて述べる．特に，ほとんどの SAT ソルバで採用されている CDCL アルゴリズムについて詳しく

説明する．第 3 章では, 本研究のベースとなる LBD の概念を含めた逐次ソルバのヒューリスティクスについて説明する．第 4 章では本研究で作成した並列 SAT ソルバ Glucans の基本的なアイディアからその実装まで詳しく述べる．第 5 章では関連研究の説明と本研究との違いを説明する．第 6 章では今年度の競技会である SAT Challenge 2012 に参加した結果とより難しい問題を用いた評価実験について述べる．第 7 章では本研究で得られた知見と今後の課題について述べる．

第 2 章

SAT

本章では，本研究において基礎となる内容である SAT について説明する．特に，本研究で作成した並列 SAT ソルバ Glucans に関連する項目について説明する．なお，ここで説明する内容は主に参考文献 [16], [17], [14], [24], [15] をまとめたものである．

2.1 SAT とは

充足可能性問題 (SATisfiability:SAT）とは，与えられた命題論理式全体が真となるような変数の割当が存在するか (SATISFIABLE)，もしくはどのように割当てもそのような割当は存在しない (UNSATISFIABL) ことを示す問題である．一般に NP 完全問題であると知られており，命題論理式が巨大になると現実的な時間内では解を探索できなくなる．しかしながら，人工知能やソフトウェア検証に応用可能な技術でそれらの実問題には，ランダムに生成される問題とは異なり一般的に問題に構造が存在し，それらをうまく利用するアルゴリズムで高速な求解ができるため，これらの実問題を解くエンジンとして SAT ソルバの研究が進んでいる．

2.1.1 基本用語

- 変数 (variable)
  論理式の変数，真偽値が割り当てられる．
- リテラル (literal)
  論理式において変数を否定の有無によって正負を表しているもの．
- 節 (clause)

リテラルを論理和でつなげたもの.

- 乗法標準形 (CNF:Conjunctive Normal Form)
  節が論理積でつながった命題となる論理式の形式.
- 変数割当 (decision)
  未知の変数に適当な真偽値を割り当てること．変数の真偽値の決定.
- 推論 (deduction)
  ある変数に対して変数割当を行った結果が伝播して，他の変数の真偽値が決定されること．Binary Constraint Propagation (BCP) とも呼ばれる.
- 決定レベル (decision level)
  変数割当が行われた回数．探索木を探索中の深さに相当し，推論によって伝播された新しい変数の真偽値も，伝播元の変数割当と同等のレベルである.
- 衝突 (conflict)
  推論の結果，同一の変数に同時に真と偽が伝播されてしまう状態.
- 学習説 (learnt clause, lemma)
  衝突の原因を解析して得られる節．探索域の剪定に使用される.
- 単位節 (unit clause)
  節内にリテラルがひとつしか存在しないもの．このときその変数の割当は恒真となる.
- バイナリ節 (binary clause)
  節が 2 つのリテラルのみで構成されているもの．一方の変数の割り当てが決まると他方の変数の割当が伝播されて決定するため，多くの SAT ソルバでは特別な扱いをしている事が多い.
- アサート節 (asserting clause)
  一般的な SAT ソルバが衝突から学習する学習節の形態.
- バックトラック
  変数割当を特定の段階までやり直すこと.
- リスタート
  局所最適解に陥り，無駄な探索域をひたすら探索する事のないように決定レベル 0 までバックトラックすること.

2.1.2 例題

SAT ソルバで求解する命題は節が論理積でつながった CNF で表される．以下にその一例を示す．

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee b \vee c)$$

この例では変数 a を真と割り当てたときや，変数 b と変数 c を真とした時に，論理式全体が充足される．このように充足可能な問題は複数の解が存在する可能性がある．次に別の例を示す．

$$(\neg a \vee b) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee c)$$

この例では変数 a に真を割り当てた場合に, 最初と 2 つ目の節より $b = true$, $c = false$ という割り当てが得られる．しかしこの割り当てでは最後の節が偽と成ってしまうため衝突が発生することがわかる．

2.2 SAT ソルバとは

SAT ソルバには系統的に解を探索することで，充足可能であれば変数の割当の組み合わせを，充足不能であればその判定ができる系統的 SAT ソルバ (systematic solver) と，確率的に充足可能な解のみを探索する確率的 SAT ソルバ (stochastic solver) が存在する．本研究では前者の系統的 SAT ソルバを扱う．また，系統的 SAT ソルバには様々な種類のものが存在するが，本研究の並列 SAT ソルバ Glucans に組み込まれている SAT ソルバである glucose や glueminisat のベースとなった MiniSat で採用されている基本的なアルゴリズムやヒューリスティクスについて説明する．

2.2.1 融合

融合 (resolution) とは Davis と Putnam によって考案された DP アルゴリズムで使用される手法で，近年の SAT ソルバの探索手法の基礎となった．複数の節に含まれる変数を削除し，節を融合することによって解が得られる．例えば，$(\neg a \vee b) \wedge (\neg a \vee \neg b)$ とい

う CNF が与えられた場合に変数 a は削除して 2 つの節を融合することができる．最終的に得られた $(b) \wedge (\neg b)$ は，b にどのような割当を行っても命題論理式全体が偽となってしまうため，UNSATISFIABLE という解が得られる．

2.3 CDCL アルゴリズム

現在ほとんどの SAT ソルバの基本的なアルゴリズムは，Davis，Putnam，Logemann，Loveland によって提案された DPLL アルゴリズムを基にし，衝突時の原因を解析して学習節を生成する機能がある Conflict Driven Clause Learning (CDCL) アルゴリズムで実装されている．CDCL は主に以下の 3 フェーズから成り，それぞれ後に CDCL アルゴリズムの疑似コードの図 2.1 とともに詳しく説明する．なお疑似コードにおける ν は変数の割当とする．

CDCL アルゴリズムではまず前処理が行われ，その後，変数割当によってヒューリスティックに従って，適当な変数に真偽値の割当を行う．このとき問題中の節によって他の変数の割当が決まることがあるので，推論によって連鎖的に割当を行う．誤った変数の割当を行った場合，伝播中に割当の衝突が発生するので，その原因の解析を行って，適度な決定レベルまで割当を取り消すという操作であるバックトラックを行う．このとき同時に後述する学習 (clause learning) が行われ，取り消して戻った決定レベルからは同じ衝突が繰り返し発生しないようになっている．最終的に推論の途中で充足するような変数の割当の組み合わせが見つかる充足可能 (SATISFIABLE) な状態になるか，最初の割当までバックトラックを行ってもまだ衝突が発生する充足不可能 (UNSATISFIABLE) な状態になったら，探索が完了したことになる．

2.3.1 変数割当

推論で得られる変数の割当がない場合に，変数選択が行われて未割当の変数からヒューリスティックに基づいて，変数に真偽が割り当てられる．ヒューリスティックによっては，衝突の回数を減らすことができるため，探索木を小さくできることが期待される．具体的には以下のようなヒューリスティックが現在までに考案されてきた．

- Maximum Occurrences on Minimum sized clauses (MOM)
  リテラルの数が少ない節の中で出現頻度の高いリテラルに値を割り当てを行う手法である．この結果，推論を行う際に高い頻度で連鎖的な割当が発生するが，一般にランダムに作成された SAT 問題にのみ効果がある．

```
CDCL(ψ, ν)
begin
  if Propagate(ψ, ν) == CONFLICT
    return UNSATISFIABLE
  dl := 0
  while (not AllVariablesAssigned(ψ, ν))
    if Propagate(ψ, ν) == CONFLICT
      if bl == 0
        return UNSATISFIABLE
      (C, bl) := Analyze(ψ, ν)
      else
        Backtrack(ψ, ν,bl)
        dl := bl
    else
      (x, v) := Decision(ψ, ν)
      dl := dl + 1
      ν := ν ∪ {(x, v)}
  return SATISFIABLE
end
```

図 2.1 CDCL アルゴリズムの疑似コード

- Dynamic Largest Combined Sum(DLCS)
  充足していない節中の未割当で最もよく出現するリテラルに割り当てを行う手法である．MOM に比べて一定の構造を有する SAT 問題に効率的な変数選択が期待できるものの，変数毎にカウンタを用意する必要があり，変数割当やバックトラックによって値が異なるためその度に更新する必要がある．このため，探索に要するコストが比較的高い手法である．逐次の SAT ソルバである GRASP で最初に実装された．
- Variable State Independent Decaying Sum(VSIDS)
  DLCS と同様に各変数に変数の出現頻度であるアクティビティ (DLCS のカウンタに相当) を用意して，アクティビティに基づいて変数選択を行う方法である．アク

ティビティは学習節として衝突の解析の結果が得られたときに，その節中に変数が出現する頻度である．このため，アクティビティの更新頻度が DLCS のカウンタ方式に比べて変数割当に要するコストが少ない．逐次の SAT ソルバである Chaff において

多くの SAT ソルバでは高速化のため VSIDS を採用している．また，何回目の割当かを示す決定レベルを変数割当が行われるたびに記録しておいている．

2.3.2 推論

SAT ソルバは直前の変数割当に基づいて問題の充足性を判断する．多くの場合，変数割当フェーズでの割当によって連鎖的に他の変数の割当が決定される．このときすべての変数の割当が完了した場合は SATISFIABLE となり解が見つかったことになる．割当に矛盾が生じた場合には，衝突が発生したものと判断し，後に説明する衝突の解析が行われる．推論を行う方法のうち，高速なアルゴリズムとして単位節法 (unit clause rule) が一般的である．単位節は，未割当のリテラルが 1 つとそれ以外の部分がすべて偽となるように割り当てられている節のことを指す．このとき，未割当のリテラルを単位リテラルと呼び，真となるように新たな変数の割当の伝播である単位伝播 (unit propagation) が行われる．

Boolean Constraint Propagation (BCP)

単位伝播の実装方式である BCP は，単位リテラルに対する割当で発生する衝突を検査するものであり，多くの SAT ソルバではその探索アルゴリズム全体で大半を BCP の時間が占める．BCP は，近年のソルバでは head-tail-list もしくは，2-literal-watching と呼ばれる方法で実装される．特に多くのソルバで採用されている 2-literal-watching は，すべての節についてそれぞれ 2 つのリテラルをポインタで監視することによって高速な BCP を実現している．これらの手法では，未割当のリテラルを常に監視し，割り当てられた場合に次のリテラルへポインタを移していく．この実装によって，単純に各変数に割当を表すカウンタを用意する場合に比べて，高速化がはかられている．

2.3.3 衝突解析

推論において変数の割当に矛盾が発生した場合，衝突の解析を行い，衝突の原因である割当の組み合わせから節を学習する．その後，衝突が発生した直接の原因である変数まで

割当をバックトラッキングを行う．多くの SAT ソルバでは後に説明するアサート節を学習節として学習する．しかし矛盾が比較的浅い決定レベルで発生していても，学習の結果，同じような衝突は発生しなくなる．ただし，学習節は元の問題が含有する冗長な節で，元の問題と同等に扱われるが，それだけメモリ空間がふくれあがってしまうため，単純に学習節を学習するだけで探索が高速化されるわけではない．

含意グラフ

変数割当や推論が探索中に進んでいく様子をグラフ形式で表すことができ，このグラフを含意グラフ (implication graph)[12] と呼ぶ．以下の図 2.2 に例を示す．ここでは，*abc...* が変数を表しており，-記号の有無で真偽の割当を表現している．またその割当がどの変数割当に基づくものであるかを表す決定レベルのが@に続いている数字である．

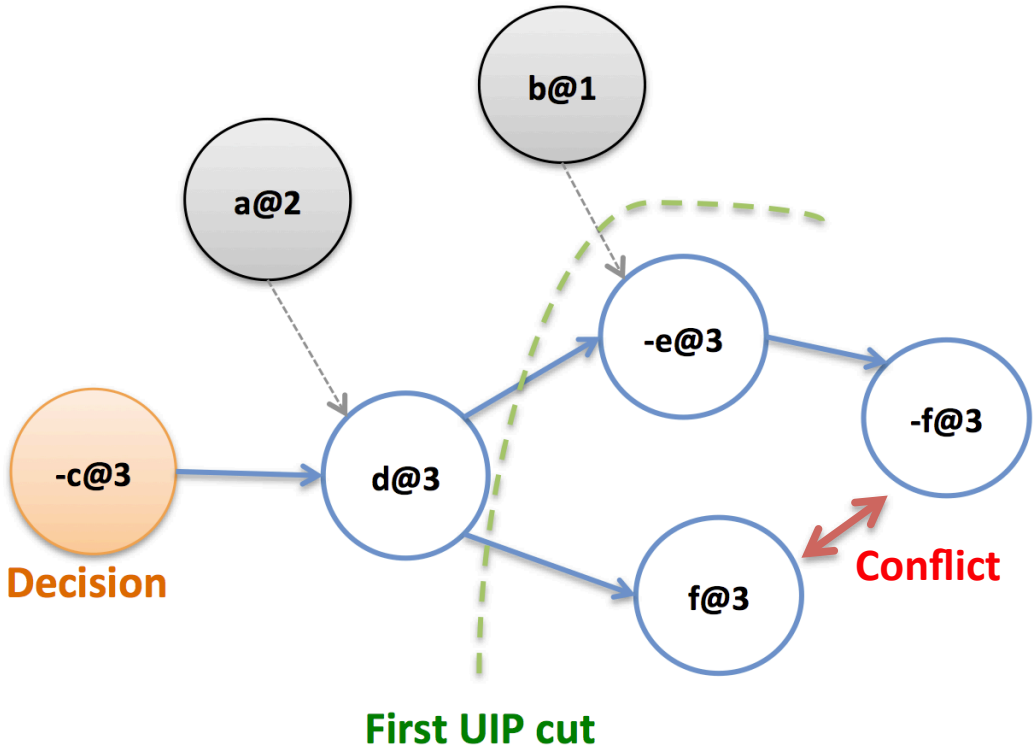

図 2.2 **Implication Graph の例**

グラフのノードからのびている矢印が推論を表しており，この例では最終的に変数 f に真と偽が同時に伝播されるため，衝突が発生している．ここで，推論によって変数の割り当てが衝突が発生している変数まで伝搬されていく際に，かならず通る割当のノードを

UIP(Unique Implication Point) と呼ぶ．特に，衝突した変数のノードに最も近い UIP を first UIP と呼び，図 2.2 中の first UIP cut のようにグラフカットを行うことで学習節を生成する．図 2.2 の例では，b, d がこれに相当するので，$(\neg b \vee \neg d)$ という節が学習節として得られる．

アサート節

First UIP でカットした際に得られる学習節のことである．first UIP のことを特にアサートリテラル (asserting literal)[15] と呼ぶ．バックトラッキングにおいて，アサートリテラルに衝突時の真偽の割当とは反対の割り当てを行えば直近の衝突を回避でき，またこの節は非常に有用である [12] ことから，一般的な SAT ソルバでは，主にこの節を学習節として学ぶ．

リスタート

SAT ソルバでは節の並び順によっても探索時間が大きく変わるが，これを防ぐために SAT ソルバでは短い間隔でリスタートと呼ばれる決定レベルが 0 になるまですべての割当をやり直す方法が一般的である．主に衝突回数が一定に達するたびにリスタートを行う方式が一般的だが，既に学習節として学習した結果がデータベースに追加されているため，決定レベルが 0 からの再探索でも再び同じ衝突が起きる事がなく，効率的な探索が期待できる．このリスタートの間隔は，様々なソルバで異なるヒューリスティクスに基づいて決められている．

2.4 その他のアルゴリズム

2.4.1 Hyper Binary Resolution

バイナリ節が得られる特殊な場合の節の融合から，Hyper Binary Resolution (HBR) を定義できる．この概念は PrecoSAT[23] や 5 章で説明する PrecoSAT を元に作成された代表的な並列 SAT ソルバである Plingeling で主に使用されている．

HBR は単位伝播の速度を低下させる学習節のサイズを縮小できる観点から有効的なアルゴリズムと考えられるが，以下に説明する一般例のように大量のバイナリ節を利用する必要があり，単純な実装では明らかにメモリアクセスコストが高いため，PrecoSAT では中間変数である dominator を用いて HBR を実現している．これを特に Lazy Hyper Binary Resolution (LHBR) と呼ぶ．

一般例

以下に HBR が適用される一般例を示す.

$$(a \vee b_1 \vee \ldots \vee b_n) \wedge (\neg b_1 \vee c) \ldots (\neg b_n \vee c) \Rightarrow (a \vee c)$$

左端の a と b_1から b_n までの $n+1$ 個のリテラルが含まれる節に対して，共通のリテラル c を含むバイナリ節が，既にすべての literal $b_i (1 \leq i \leq n)$ に対して存在している場合は $(n+1)(n)$ 個のリテラルが含まれる論理式が，2 つのリテラルからなるバイナリ節に融合される.

dominator

LHBR において dominator がどのように割り当てられるかを示す．なお，実際に dominator がどのように利用されるかについては後で述べる.

1. 既に真偽の割り当てられているリテラル l の dominator を bindom(l) と定義する.
2. literal l に対して変数割当が行われた時に bindom(l) = l を初期値として与える.
3. $(a_1 \vee b)$ というバイナリ節に対して，$a_1 = false$, $b = true$ が割り当てられた場合は，bindom(b) = bindom($\neg a_1$) として dominator を更新する.

上述の手順を踏まえると，dominator はバイナリ節においてその変数についてどのような割当を行った場合でもインプリケーショングラフ上必ず通る変数，すなわち first UIP となる.

dominator による節の融合

以下のような同一の dominator を持つリテラル a_i とそれ以外のリテラル b から成る節が与えられた場合，LHBR によってバイナリ節が生成される.

$$(a_1 \vee \ldots \vee a_n \vee b) \quad \text{ただし } 2 < n$$

$$bindom(\neg a_1) = \ldots = bindom(\neg a_n)$$

このとき，任意の dominator : $bindom(a_i)$ を d とすれば，LHBR によって節 $(\neg d \vee b)$ が元のより冗長な節 $(a_1 \vee \ldots \vee a_n \vee b)$ の代わりに得られる.

Hyper Binary Resolution の効果

文献 [24] によれば，HBR が最初に実装された SAT ソルバである PrecoSAT では SAT'09 Competition の問題において，LHBR を利用することによって学習する学習節のうち約 19% が LHBR から得られるバイナリ節となる．

PrecoSAT では，LHBR によって，SAT'09 Competition で出題された問題 292 題のうち制限時間内に解ける問題が，LHBR を利用しない場合に比べて 6 題多く解けるような結果が得られるとのことである．

第 3 章

SAT ソルバとそのヒューリスティクス

本研究で作成した並列 SAT ソルバ Glucans は，逐次の SAT ソルバ Glucose[5] やその派生である GLUEMINISAT[7] をベースに，様々な手法を用いてその並列化を行ったものである．本研究で利用したすべての SAT ソルバは MiniSat[2] の設計や実装を踏襲しており，本章では特に MiniSat のアルゴリズムと Glucose や GLUEMINISAT のヒューリスティクスについて説明する．

3.1 MiniSat

MiniSat は CDCL アルゴリズムを実装した標準的な SAT ソルバであり，SAT Competition においてそのヒューリスティクスのチューニングを競う MiniSat Hack 部門が開催されるばかりでなく，多くの SAT ソルバ [8][5][6][9] は MiniSat をベースに独自に改良を加えたものである．本研究で利用した Glucose は，MiniSat を基に様々な改良を行った SAT ソルバであり，特に本研究で作成した Glucans と直接関係のある MiniSat のヒューリスティクスについて理解する必要がある．以下に文献 [2] からそれらを列挙する．

3.1.1 活性度

MiniSat では衝突時に生成される学習節に含まれる変数群の使用頻度を表す活性度 (変数の活性度) と呼ばれる指標を増加させ，変数割当時の選択基準とする事で衝突に関連する変数から割当を行うようにしている．また，新規の学習節生成にあたって探索に利用し

た関連する既存の学習節の使用頻度を表す活性度 (学習節の活性度) を増加させる事で，後述する学習節の削除によって，あまり探索に利用されていないものの削除基準としている．

3.1.2 学習節の削除

探索空間の剪定に必要な学習節だが，学習節の増加は推論時に使用する節の増大を招き，探索が進むに従って伝播がなかなか発生しない学習節が伝播の発生頻度を減らすことなる．学習節は元の命題に対して補題にあたるため，MiniSat では任意のタイミングで学習節を削除する．図 3.1 に MiniSat において実際に学習節を削除する関数 ReduceDB()[2] の実装を示す．

```
void reduceDB()
  int i, j
  int limit = learnts.size()/2
  double lim = cla_inc/learnts.size()
  sortOnActivity(learnts)
  for(i=j=0; i < learnts.size(); i++)
    if(limit > 0 & learnts[i].activity < lim)
      Remove(learnts[i])
      limit=limit-1
    else
      learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 3.1 MiniSat における reduceDB() の実装

3.1.3 Phase Saving

Phase Saving は RSat[13] で最初に実装された Progress Saving と呼ばれる手法を MiniSat で実装したものである．Phase Saving では変数割当がキャンセルされる場合にその割当を記録して，次回の割当で利用する．これは，SAT 問題ではしばしば複数の独

立した問題 (コンポーネント) が 1 つの問題中に含まれていることがあるからであり，具体的には図 3.2 のような探索状態が挙げられる．

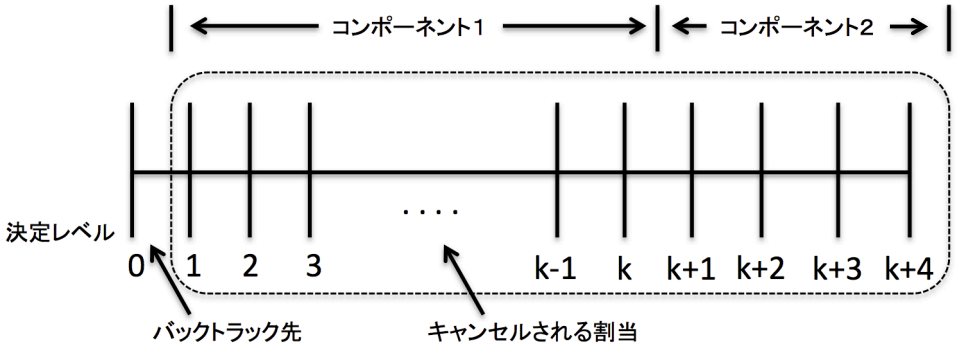

図 3.2 **バックトラックの例**

ここではコンポーネント 1 を解いた後にコンポーネント 2 を解いている場合にリスタートすると，コンポーネント 1 を解き直さなくてはならない事がわかる．そこでバックトラックやリスタートで割当がキャンセルされる場合にはそれを記録する事で，時間の変数割当で利用してコンポーネント 1 を何度も解き直さないようにすることができる．

3.2 Glucose

3.2.1 Literals Blocks Distance

Literals Blocks Distance (LBD) は，実問題からエンコードされた SAT 問題においてしばしば関連性のある変数群が存在することを利用した学習節の評価尺度であり，次の定義 3.2.1 のように定義される．

定義 3.2.1 (LBD (Literal Block Distance)) [4] ある節 C が与えられたときに，節 C 中に含まれるすべてのリテラルを，その現在の割当の決定レベルが同じものを 1 ブロックとして，n 個のブロックに分割できる．この時，節 C の LBD は n とする．

特に LBD の値が 2 であるものは，Glue Clause と呼ぶ．Glue Clause は，片方のブロックが充足されない状態となった場合に，残りの 1 ブロックの割当に伝播するので，推論フェーズで特に有用な学習節と考えらる．また，LBD は，学習節が生成される衝突解析時だけでなく，変数の割当に依存するため，推論フェーズでも動的に変化する．Glucose では推論フェーズ中にも LBD の更新を行う．

3.2.2 学習節の削除

LBD がまず用いられるのが冗長な学習節の削除である．Glucose では学習節の数を衝突 4000 回を初期値として 1000 回毎に半分に縮小する．その結果約半分の学習節を削除する事になる．また，LBD が 2 以下の学習節は決して削除しないようにしている．

```
void reduceDB()
  int i, j
  int limit = learnts.size()/2
  sortOnLBD(learnts)
  for(i=j=0; i < learnts.size(); i++)
    if(limit > 0 & learnts[i].lbd > 2)
      Remove(learnts[i])
      limit=limit-1
    else
      learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 3.3 Glucose における reduceDB() の実装

特に学習節の LBD 平均値が良い場合には，次回の学習節の削除タイミングを遅らせる事で，学習節の保持量動的に増大させる．具体的には，LBD 順にソートしたときに中央値が 3 の場合に 300 回，最大値が 5 の場合に更に 300 回タイミングを遅らせる．

3.2.3 リスタート

Glucose では LBD の良い学習節の獲得を目的に，LBD を基準にリスタートを行う．具体的には，直近 50 回の衝突で生成された平均値が，すべての学習節の LBD の合計値を衝突回数で割った LBD の平均値の約 1.43 倍以上になった時にリスタートを行う．

3.3 GLUEMINISAT

GLUEMINISAT は Glucose から派生した SAT ソルバであり, 非常に積極的なリスタートを行うことが特長である．また，SAT Competition 2011 では Application UNSAT 分野で優勝するなど，UNSAT が解である問題が得意**??**であることで知られている．

3.3.1 strict LBD

Strict LBD は LBD を拡張した評価尺度でありよりも細かく学習節を区別できる．次のように定義される．

定義 3.3.1 (strict LBD) *[7] C が LBD が n の節であるとする．もし節 C がブロック中にリテラルがひとつのみしか存在しないユニットリテラルブロックを１つでも含むなら，節 C の strict LBD を n とする．１つも含まない場合は strict LBD は未定義とする．*

例えば，LBD が 2 の学習節では，片方のブロックのリテラル群の割当が決まれば，もう一方の割当が決まる (=伝播する) ことが期待されるが，この時ユニットブロックでない場合には，この伝播が発生するかは自明ではない．

3.3.2 学習節の削除

GLUEMINISAT では学習節の削除を Glucose と同様に Strict LBD 順に行う．ただし，Strict LBD を評価尺度として使用しているため，Strict LBD が 3 以下の学習節を o の k のこ s 巣

3.3.3 リスタート

GLUEMINISAT は Glucose と同様にリスタートを行う．相違点としては，GLUEMINISAT では推論中の LBD も考慮したリスタートを行う．これにより LBD の平均値が Glucose のそれより良くなるため，よりリスタートが頻繁になる．

第 4 章

並列 SAT ソルバ Glucans の設計と実装

本章では，本研究において作成した Glucans の設計と実装について述べる．Glucans では実行時のパラメータセットの異なるバリエーションがあり，それらについても合わせて説明する．

4.1 Glucans の概要

Glucans は LBD に基づいた学習節の共有を行う並列 SAT ソルバであり，逐次の SAT ソルバである Glucose を基に実装した．具体的には，Pthreads を用いて Glucose と Glucose から派生した SAT ソルバである GLUEMINISAT から成るスレッドを任意の数実行し，スレッド間で学習節を LBD に基づいて共有する．更に，SAT Competition 2011 の MiniSat-hack 部門において充足可能な問題で解けた問題数の多かった Contrasat や CIRMiniSat を取り入れており，オプションでそれらと同様のアルゴリズムを使用する事も可能にした．MiniSat-hack 部門に出場する SAT ソルバは全て MiniSat をベースに実装されている事から，Glucans に取り入れる事が比較的容易であり，上位のこの 2 つの SAT ソルバを選んだ．

また，最新の SAT Challenge 2012 へ，Glucans の特定のヒューリスティクスでチューニングしたバージョンで参加している．

4.1.1 基本的な並列化

Glucose と GLUEMINISAT の 2 つの SAT ソルバをベースに並列 SAT ソルバを作成するにあたって，最も基本的な形態にベースソルバをそれぞれ異なるスレッドで実行することが考えられる．これには次の 3 つの機能を実装した．

- 問題ファイル読み込みながらメモリにコピーするメインスレッド
- メインスレッドが終了したらメモリから問題を読み込む Glucose または GLUEMINISAT のスレッド
- 解を発見したときに自身が一番最初に発見した事がわかるようなロック機構

以上の機能で，最も単純な逐次 SAT ソルバを複数回実行したのと同等な結果が得られる基本的な並列 SAT ソルバができる．

4.1.2 学習節の共有

Glucans では各スレッドで衝突の発生時に学習した学習節を自スレッド以外のスレッドと共有する．この時 LBD が 5 以下の節についてのみ共有し，これらを非同期的に共有が出来るようにリンクリストで実装されたキューへ，学習節とその LBD 値を入れる．キューはスレッドで 1 つずつ持つ．共有のタイミングは学習節が生成される度であり，学習節の重複のチェックは行わない．学習節共有のキューとその実装の概要図を以下に示す．

共有時に学習節をほかのスレッドへ書き出すには各スレッドが非同期的に以下の手順に従って行う．

1. 衝突が発生し，衝突解析で得られた学習節が Sharing Limit 以下なら共有するためにコピーを作成する．
2. コピーとその使用回数を表すカウンタを束ねる構造体 C を作成する．
3. 新しいリストの要素 E を作成し，E に C へのポインタを持たせる．
4. まだ共有していないスレッドの共有リストの Tail とその要素 T をロックする．リストが空なら Head も同時にロックする．
5. Tail の次の要素を終端を表す NULL から E に書き換える．
6. E の次の要素を NULL とし，Tail を E に書き換えて Tail と元の要素 T をアンロックする．

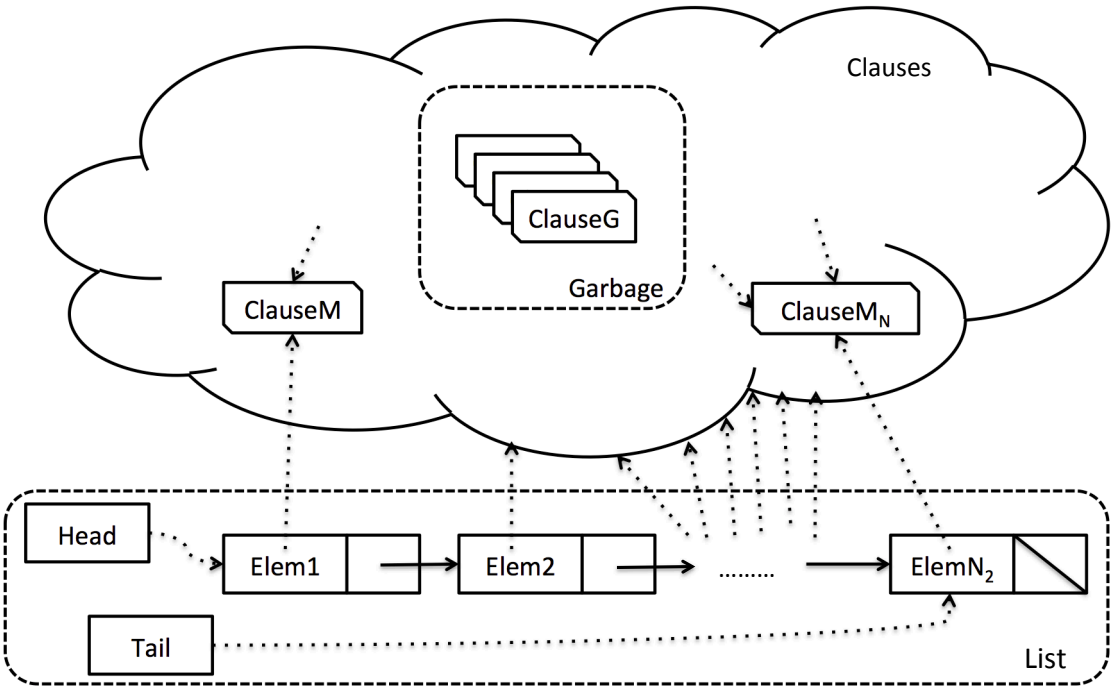

図 4.1 **学習節共有用キューの概要**

7. 3 から 6 を残りのスレッドの数だけ繰り返す.

また反対に他のスレッドからの学習節を取り込むために以下の手順を書き出し後に行う.

1. 自分のリストの Head をロックする．空を表す NULL なら以下の手順を行わずに終了する.
2. Head から要素 E を取り出す.
3. E をロックして E に含まれるポインタから学習節の構造体 C を参照する.
4. C から学習節をコピーし，自スレッドのデータベースへ追加する.
5. C の使用回数を表すカウンタをインクリメントし，スレッドの数-1 と等しくなれば C を削除する.
6. E の次の要素 N を読み出して E を削除する.
7. N が NULL でなければ次の要素を E として 3 へ戻る.
8. N が NULL なら Tail をロックして Tail を NULL へ書き換えて Tail をアンロックする.

推論フェーズ中での共有

探索域が異なるなどの理由で, 推論中に学習節の LBD が変化することがある．Glucans では推論中の LBD 更新によって LBD の小さくなった学習節を共有することでより多くの効果的な学習節の獲得を可能にした．特に有用と考えられる Glue Clause を共有するようにしたが推論中に共有するとそのままでは各スレッドで多数の節が受信したものと自スレッドで生成されたものとで重複する恐れがある．Glucans では，重複率を減らすため推論中の学習節共有について次の 2 つの制約を設けた.

- 学習節に共有されたかどうかのフラグをつけ，衝突解析によって生成された時に共有済みのものを再度共有しない
- Strict LBD が 2 の学習節に限定し，推論フェーズでの共有量自体を抑制する

4.1.3 CIRMiniSat と Contrasat

CIRMiniSat と Contrasat はどちらも MiniSat をベースとした SAT ソルバであり，差分を Glucose や GLUEMINISAT のスレッドへ実装した.

4.2 Glucans のヒューリスティクス

Glucans は逐次の SAT ソルバをベースに機能を加えたものであるため，各種ヒューリスティクスについて調査し並列向けの変更を行った．特に，学習節の削除については学習節共有をすることを前提に変更を行った.

4.2.1 学習節の削除

Glucans では直近の探索に必要な学習節は活性度で，比較的遠い未来に必要となると考えられる節は LBD で評価し，その結果に基づいて学習節を削除する LBD-Activity 方式と，glucose と同じ LBD に基づいた学習節の削除を行う．これらはオプションで振る舞いを切り替える事が可能である．特に，glucose と同じ基準で学習節の削除を行う場合には，学習節の共有基準を削除されない学習節の LBD 上限から算出することが可能であり，後で説明するように最適値を推測して必要に応じて Sharing Limit を変更する．なお，ここでソートを行う場合には降順でソートされるものとする.

LBD-Activity 方式

LBD-Activity 方式では学習節の削除基準として LBD と活性度の 2 つをハイブリッドに使用する．主に，LBD の値が高いが活性度も高い学習節の確保を目的する．事前に設定した上限である UBLBD より小さな LBD の節は残し，それ以外は活性度の低い順に削除する．LBD-Activity 方式の初期の実装を図 4.2 に示す．

```
void reduceDB()
  int limit = learnts.size()/2
  sortOnActivity(learnts)
  for(i=j=0; i < learnts.size(); i++)
    if(limit > 0 & UBLBD < learnts.lbd())
      Remove(learnts[i])
      limit=limit-1
    else
      learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 4.2 上限以上の学習節は活性度で削除する reduceDB()

学習節を活性度に従ってソートを行ってから，LBD が UBLBD より高い学習節を削除する．この実装では，UBLBD という固定値を用いて活性度と LBD を使い分けているが，実際の問題によっては生成される学習節の LBD に偏りがある．例えば UBLBD 以下の学習節のみしかデータベースに存在しない場合，活性度の高い学習節も削除されてしまう．特に LBD が 3 以下の LBD の良い学習節が多く得られる問題の場合，活性度が高く LBD が UBLBD を超える学習節を削除せずに残すためには図 4.3 のような実装が考えられる．

この実装方法では，活性度順のソートと LBD 順のソートを 2 回を行うためソートが 1 回で済む他の学習節削除方法と比べるとコストが高いものの，活性度の高い学習節が一定量保持されることが保証できる．

```
void reduceDB()
  int limit = learnts.size() / LockRatio
  sortOnActivity(learnts)
  for(i=j=0; limit > 0; i++)
    Lock(learnts[i])
    limit=limit-1
  limit = learnts.size()/2
  sortOnLBD(learnts)
  for(i=j=0; i < learnts.size(); i++)
    if(limit > 0 & isNotLocked(learnts[i]) & 2 < learnts.lbd())
      Remove(learnts[i])
      limit=limit-1
    else
      learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 4.3 活性度の高い学習節を一定量残す reduceDB()

Dynamic Sharing Limit

Dynamic Sharing Limit は，すべてのスレッドで共通の Bounded Queue に推定した最適値を代入し，その平均値を新しい学習節の共有リミットとする方式である．LBD の小さい順に長く学習節を保持する場合には一定数保持するので，削除されずに残った学習節の中で，LBD 最大値を求める事でそれを学習節の共有リミットの最適値と推定することができる．図 4.4 にその実装を示す．

探索が進むに従って LBD の小さな学習節が生成されなくなることが予想され，それと同様に共有される学習節も減少すると考えられる，ここでは保持される学習節に着目し，その量を増やす事によって，学習節のデータベース全体の平均 LBD を改善するのを目的とする，また，上限以上の学習節は活性度で削除する LBD-Activity 方式と併用する場合は困難であるので，活性度の高い学習節を一定量残す方法で併用する，図 4.5 に Dynamic Sharing Limit と LBD-Activity 方式を併用した実装を示す，

```
void reduceDB()
  int limit = learnts.size()/2
  sortOnLBD(learnts)
  pushToBoundedQueue(learnts[limit].lbd())
  sharing_limit = getAverageFromBoundedQueue()
  for(i=j=0; i < learnts.size(); i++)
    if(i < limit & 2 < learnts.lbd()) Remove(learnts[i])
    else learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 4.4 Dynamic Sharing Limit を決定する reduceDB()

```
void reduceDB()
  int limit = learnts.size() / LockRatio
  sortOnActivity(learnts)
  for(i=j=0; limit > 0; i++,  limit–)
    Lock(learnts[i])
  limit = learnts.size()/2
  sortOnLBD(learnts)
  pushToBoundedQueue(learnts[limit].lbd())
  sharing limit = getAverageFromBoundedQueue()
  for(i=j=0; i < learnts.size(); i++)
    if(limit > 0 & isNotLocked(learnts[i]) & 2 < learnts.lbd())
      Remove(learnts[i])
      limit=limit-1
    else learnts[j++] = learnts[i]
  learnts.shrink(i - j)
```

図 4.5 Dynamic Sharing Limit と LBD-Activity 方式を併用する reduceDB()

4.2.2 変数割当の分散

複数のスレッドが同じ探索をするのは回避するため，一定の割合でスレッド間で同じ変数でも違う割当を行うようにする，Glucans では次のような静的な変数割当の分散を行う，

phase saving のレベル

MiniSat と同様に Glucans のベースとして採用した Glucose などは phase saving を採用している，phase saving を行うと充足可能なコンポーネントを記録してリスタート後にすぐに同じ深さまで探索を再開できるが，充足不能な問題では同じ探索域に入る可能性がある，そこで Glucans では一部のスレッドで phase saving をしない，または phase saving を直近一回の変数割当とその伝播に限る，

ランダムな割当

CDCL に基づく SAT ソルバは，変数割当で基本的に False を割り当てる，これは節のリテラルが 1 つだけ未定でその他がすべて充足されないようにしむける事で，単位伝播が置きやすくなるためである，しかしながら全ての問題で変数の正負の出現が正であるとは限らないので，Glucans ではスレッドによって一定の割合で True もランダムに割り当てるようにすることで変数割当の分散化を図ることができるようにした，

4.3 バリエーション

Glucans は複数のヒューリスティクスやパラメータの組み合わせで振る舞いが異なる．競技会への参加やどの組み合わせがより高速か評価を行うため，次の 3 つのバリエーションを使用する．LBD で制限された学習節共有を行う事で探索の並列化をはかり，LBD の大きい順に削除する Basic 版と学習節の削除方針について活性度の小さい順に削除する Activity 版の 2 つを作成した．また，GLUEMINISAT で用いられている概念である StrictLBD を利用して共有される学習節の量を抑制しながら，推論中に獲得した新たな Glue Clause を共有する Strict 版を作成した．

第 5 章

関連研究

本章では関連研究について, 特に他の並列 SAT ソルバについて説明する.

5.1 Penelope

PeneLoPe は Glucans と同様に学習節の共有を行う並列 SAT ソルバである. 基本的には Glucans の LBD 順に学習節の削除を行うバージョンと似た実装になっているが, 学習節の共有で受け取り方法が異なる. PeneLoPe では学習節を他のスレッドから受け取る際に, 現在の探索状態における割当から, その節の単位伝播の発生頻度を充足しやすさで推測し, 一定の学習節を自身のデータベースに追加しない.

5.2 c-sat

c-sat[1] はクラスタ向けの並列 SAT ソルバであり, MPI で実装されている. 本研究と同様に学習節を共有するが長さに基づいて行う. 本研究と違い c-sat では使用できるコア数に理論的に制限が存在しないが, 分散メモリ型プログラムであるため通信にコストがかかる.

5.3 Plingeling

Plingeling は本研究の Glucans を作成した時点で最も高速な並列 SAT ソルバであり, SAT Competition 2011 で優勝している. Pthreads で実装されたマルチスレッドソルバで, スレッド間では長さ 1 の学習節である自明な変数の割当行い, Lingeling で実装されて

いる equivalence(同値, 等価) な変数の共有を Boss-Worker モデルを採用して union find data structure(素集合データ構造) を用いて行う.

第 6 章

評価実験

本章では，本研究で作成した並列 SAT ソルバで参加した競技会である SAT Challenge 2012 の結果や追加の評価実験について述べる．なお，追加の評価実験ではバイナリを入手する事ができる PeneLoPe と比較した．

6.1 実験内容

評価実験では，Glucans の他のソルバに対する優位性を中心に，学習節の保有戦略の違いによる差異やスケーラビリティーについて実験を行う．またそれぞれの評価実験では，時間とメモリ制限を設けている．また，メモリ使用量が制限を超過した場合には制限時間を超えたものとして扱う．

6.1.1 学習節共有量と保有戦略

冗長な学習節の削除について，活性度に基づいて削除する Activity 版と LBD 版の 2 つのバージョンで評価を行った．更に LBD 版と同じ保有戦略だが StrictLBD を評価尺度として使用し，推論の LBD 更新によって新たに Glue Clause となった学習節を送出する Strict 版も作成して併せて評価する．なお，評価に使用する問題は 2010 年度の競技会から今年度の SAT Challenge 2012 の問題約 750 題で行う．

学習節削除戦略による違い

予備実験として，問題ごとに LBD と活性度の相関関係を調査した．図 6.1 と図 6.2 にその結果を示す．LBD が小さい順に活性度が高いという問題もあれば，双方に相関関係

がない問題がある事も分かる.

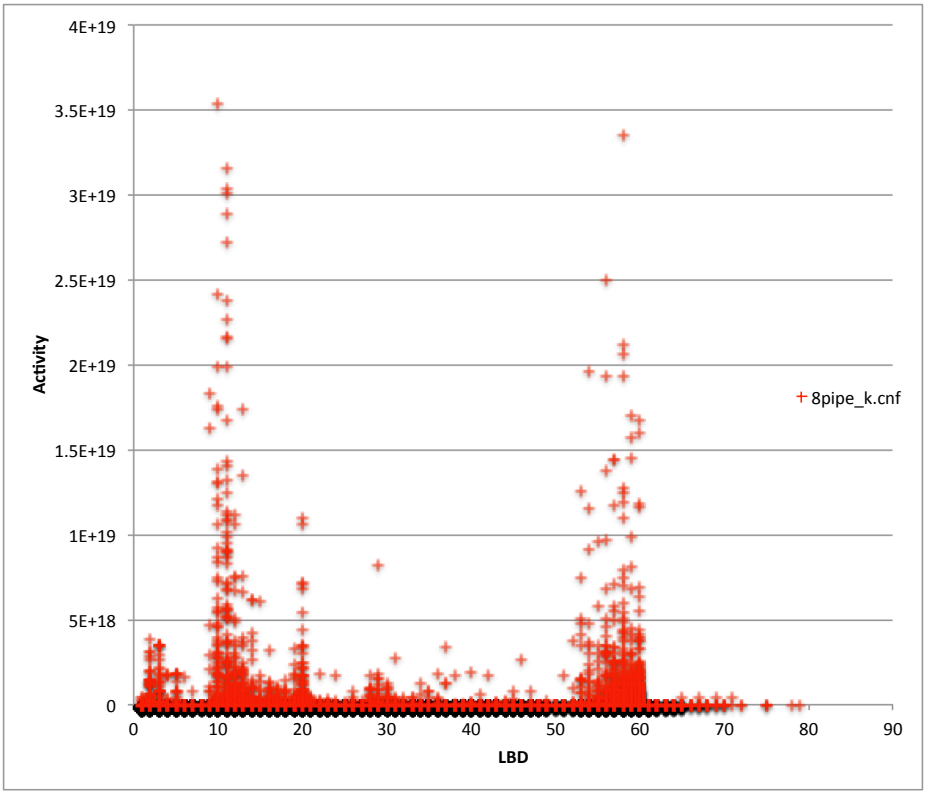

図 6.1 **8pipe_k.cnf 問題における LBD と活性度関係**

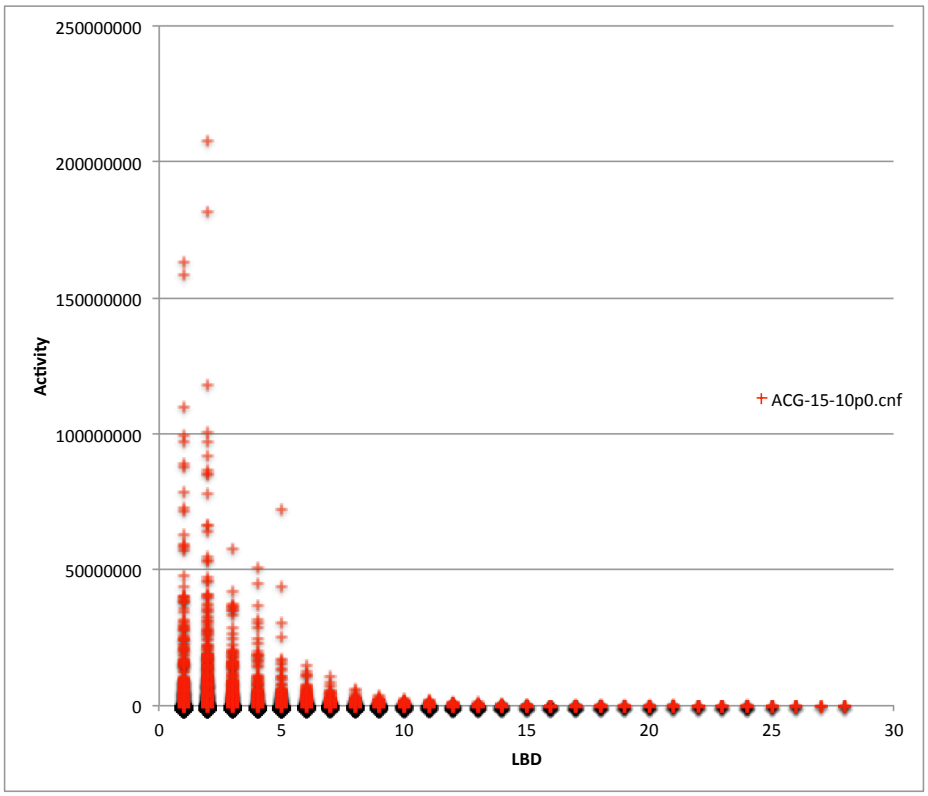

図 6.2 **ACG-15-10p0.cnf 問題における LBD と活性度関係**

6.2 実験環境

SAT Challenge 2012 を除いた実験は表 6.1 と表 6.2 の 2 通りの環境で行った．どのサーバーを使用したかは実験結果の項で併せて記載する．なお，実行可能ファイルはすべて表 6.1 の環境で静的にリンクするように O3 オプションでコンパイルしたものを使用した．

表 6.1 Xeon 16Cores 32GB per node

OS	CentOS 6.3 (kernel 2.6.32)
CPU	2x Eight-Core Intel Xeon E5-2660 2.20GHz
メモリ容量	32GB
gcc version	4.4.6

SAT Challenge 2012 は，ドイツの the State of Baden-Württemberg の bwGRiD

表 6.2 Xeon 8Cores 24GB per node

OS	CentOS 5.3 (kernel 2.6.18)
CPU	2x Quad-Core Intel Xeon E5-2660 2.20GHz
メモリ容量	24GB
gcc version	4.1.2

cluster 上で行われた．クラスターを構成するサーバーのスペックは表 6.3 の通りである．

表 6.3 bwGRiD Node

OS	Scientific Linux (kernel 2.6.18)
CPU	2x Quad-Core Intel Xeon E5440，2.83 GHz
メモリ容量	16 GB per node
gcc version	GCC 4.1.2，javac 1.6.0

6.3 実験結果

6.3.1 SAT Challenge 2012

表 6.4 に SAT Challenge 2012 の結果を解けた問題数の総数と SAT，UNSAT 別を集計したものを示す．

表 6.4 SAT Challenge 2012 の結果

ソルバ名	SAT	UNSAT	SAT+UNSAT	順位
pfolioUZK	254	277	531	1
PeneLoPe	240	290	530	2
ppfolio2012	242	283	525	3
Glucans (Activity)	234	287	521	4

図 6.3 に SAT Challenge 2012 の結果を上位 4 つのソルバについて示す．このグラフは縦軸が探索時間で横軸は解けた問題を探索時間について昇順にソートした問題数となっている．系列が右によっているほど解ける問題数が多く，下によっているほど高速であると考えられる．なお，横軸は ppfolio2012 が約 100 秒かかる問題以上を拡大したものとした．

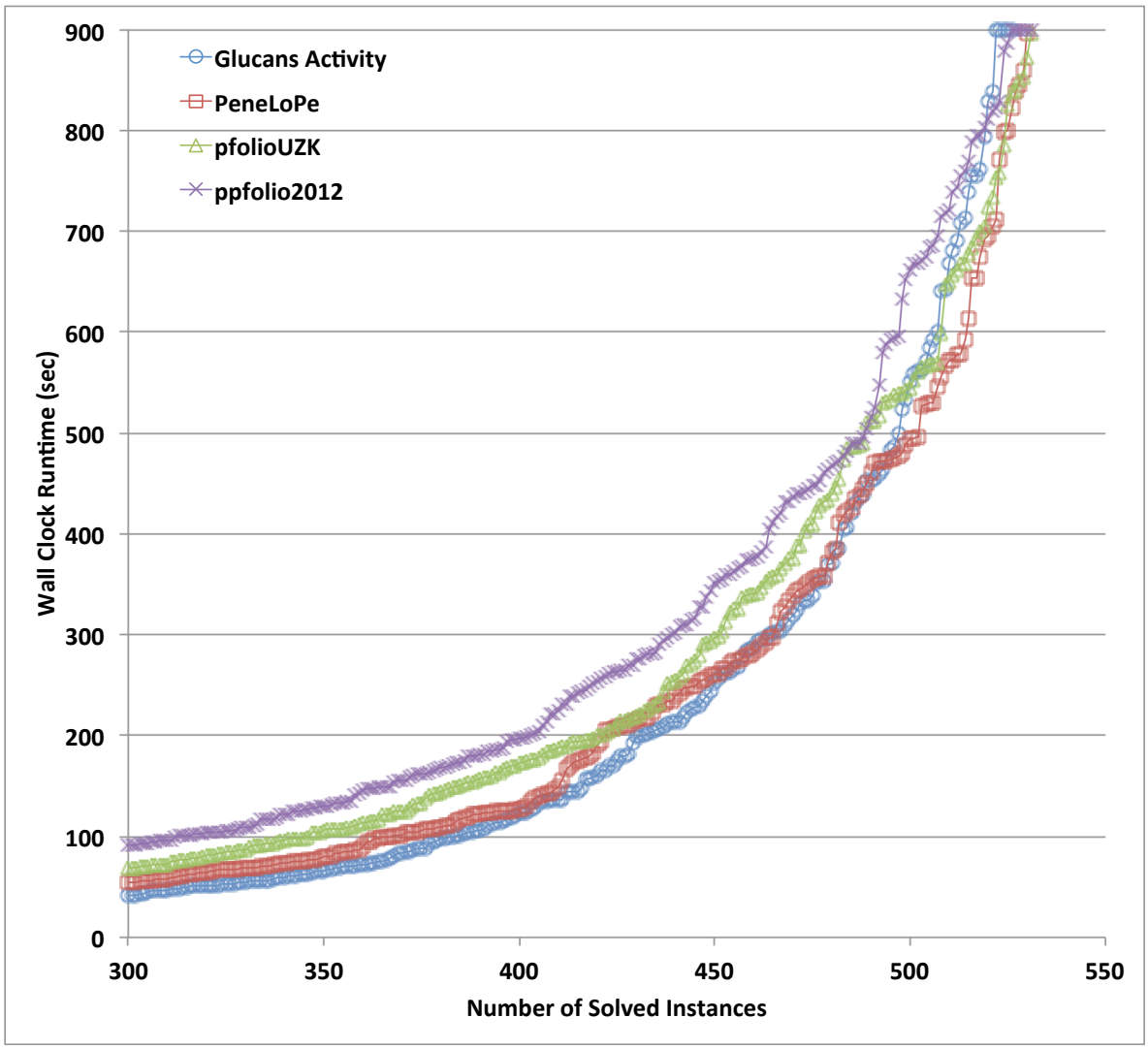

図 6.3　**SAT Challenge 2012 の Cactus プロット**

SAT Challenge 2012 ではグラフから分かるように突出して速いソルバがあるという結果ではない．しかしながら途中まで Glucans が最も下によっているように，全問題の約 30% (179 題) において Glucans が最も高速である．これは 2 番目に多い PeneLoPe の 78

題 (約 13%) の 2 倍以上であり，特徴的な結果となっている．

6.3.2 SAT Competition ルールでの実験結果

難しい問題も含めた評価を行うため，長い制限時間を設けた SAT Competition 2011 のルールで評価実験を行った結果を表 6.5 に示す．

表 6.5 制限時間 5000 秒での実験結果

ソルバ名	SAT	UNSAT	SAT+UNSAT	順位
Glucans Strict	253	377	630	1
Glucans Basic	249	279	628	2
Glucans Activity	249	276	625	3
PeneLoPe	250	266	616	4

この結果から制限時間を延ばし，問題数を増やした場合には Glucans のどのバージョンでも PeneLoPe より多くの問題を解く事が出来ることがわかる．特に最も多くの問題を解く事が出来た Glucans の Strict 版と PeneLoPe を比較すると Glucans の方が 14 題も多い．

次に図 6.3 に実験結果をグラフで示す．このグラフは縦軸が探索時間で横軸は解けた問題を探索時間について昇順にソートした問題数となっている．系列が右によっているほど解ける問題数が多く，下によっているほど高速であると考えられる．なお，横軸は実行時間の短いものを除いて拡大したものとした．

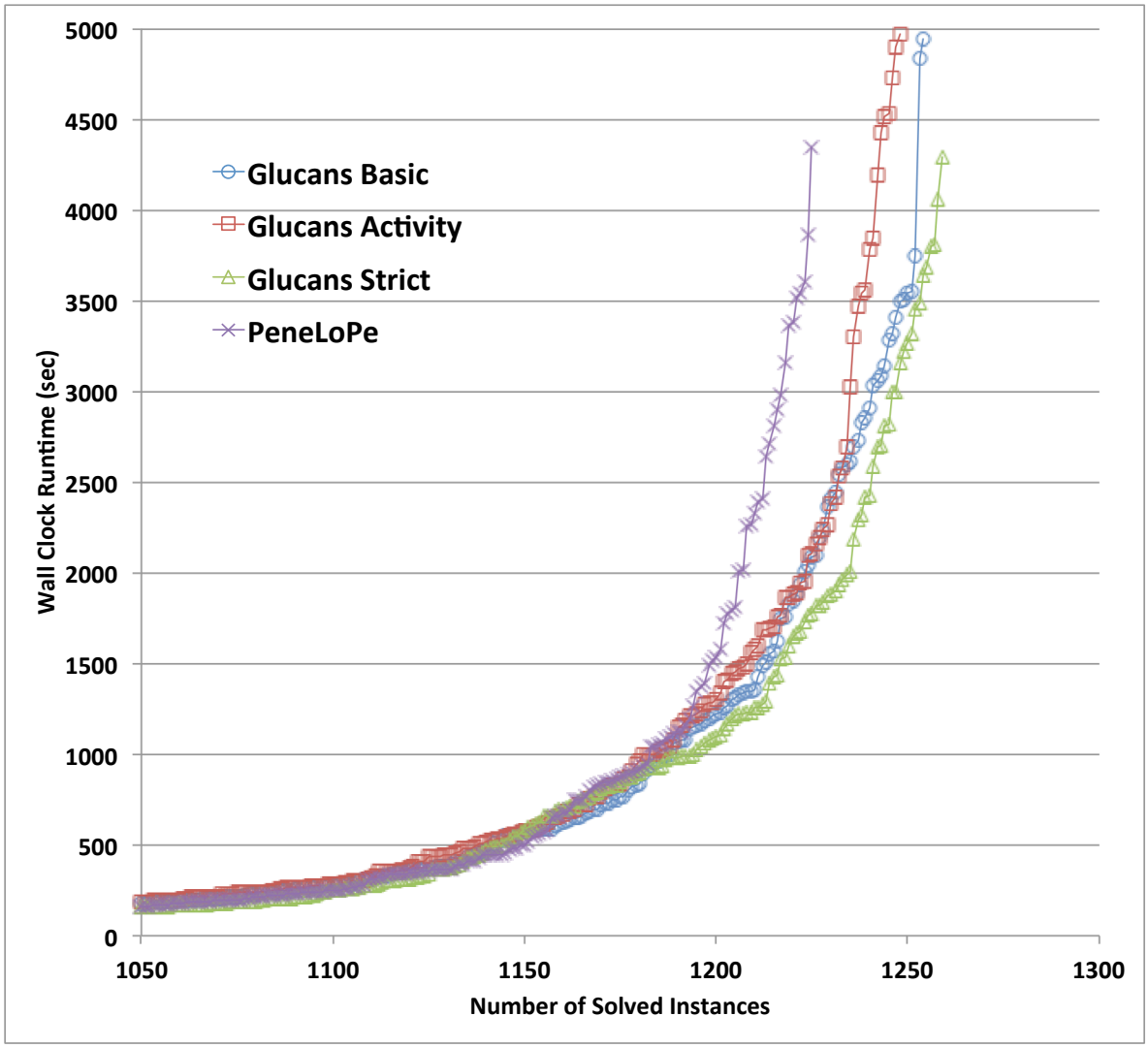

図 6.4　**SAT Competition ルールでの Cactus プロット**

このグラフを見ると Glucans の 3 バージョンが PeneLoPe よりも右側によっており，このことから実行時間が長くなるにつれて，PeneLoPe よりも Glucans の方が問題を多く解ける事が分かる．更に Glucans の 3 つのバージョンを比べると推論中に学習節共有を行い StrictLBD を使用する Strict 版が最も右よりかつ下よりで高速である事が分かる．

次に同じ問題について PeneLoPe と Glucans の Strict 版のより詳しい比較を行う．図 6.5 に Glucans を横軸に，PeneLoPe を縦軸に同じ問題の実行時間プロットして示す．

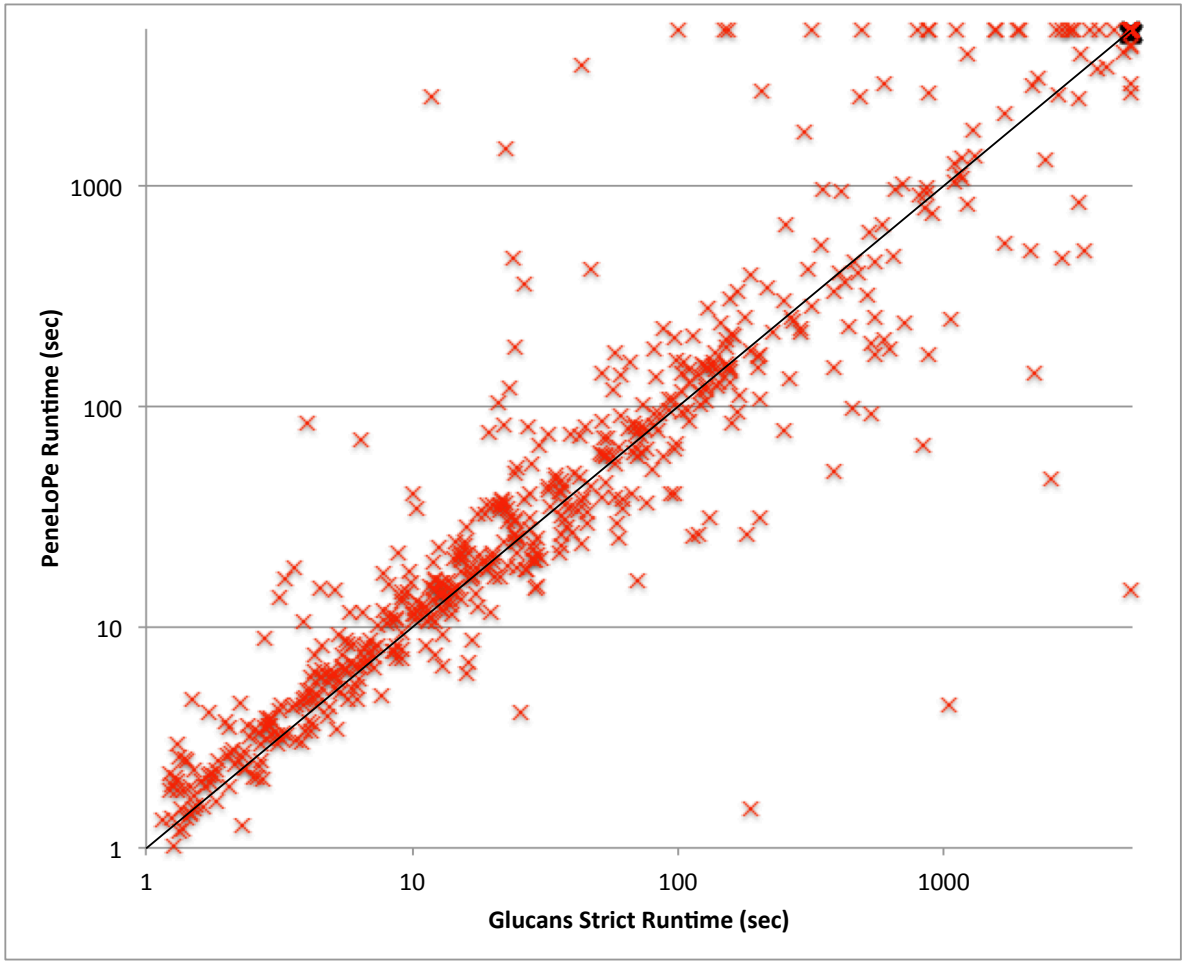

図 6.5 **Glucans Strict と PeneLoPe の比較 (両対数)**

この結果 Glucans Strict の探索時間が PeneLoPe より短い右上のプロットは 467 題でこれは全体の約 63% の問題について Glucans Strict の方が高速である事を示す．このグラフでは双方のソルバ共に分散しているためその性能差が明らかではない．そこで図 6.6 で横軸を PeneLoPe の実行時間に，縦軸を Glucans が PeneLoPe よりどれくらい高速かについて PeneLoPe の実行時間を Glucans の実行時間で割ったもので示す．

次に同じ問題について PeneLoPe と Glucans の Strict 版のより詳しい比較を行う．図 6.6 に Glucans を横軸に，PeneLoPe を縦軸に同じ問題の実行時間プロットして示す．

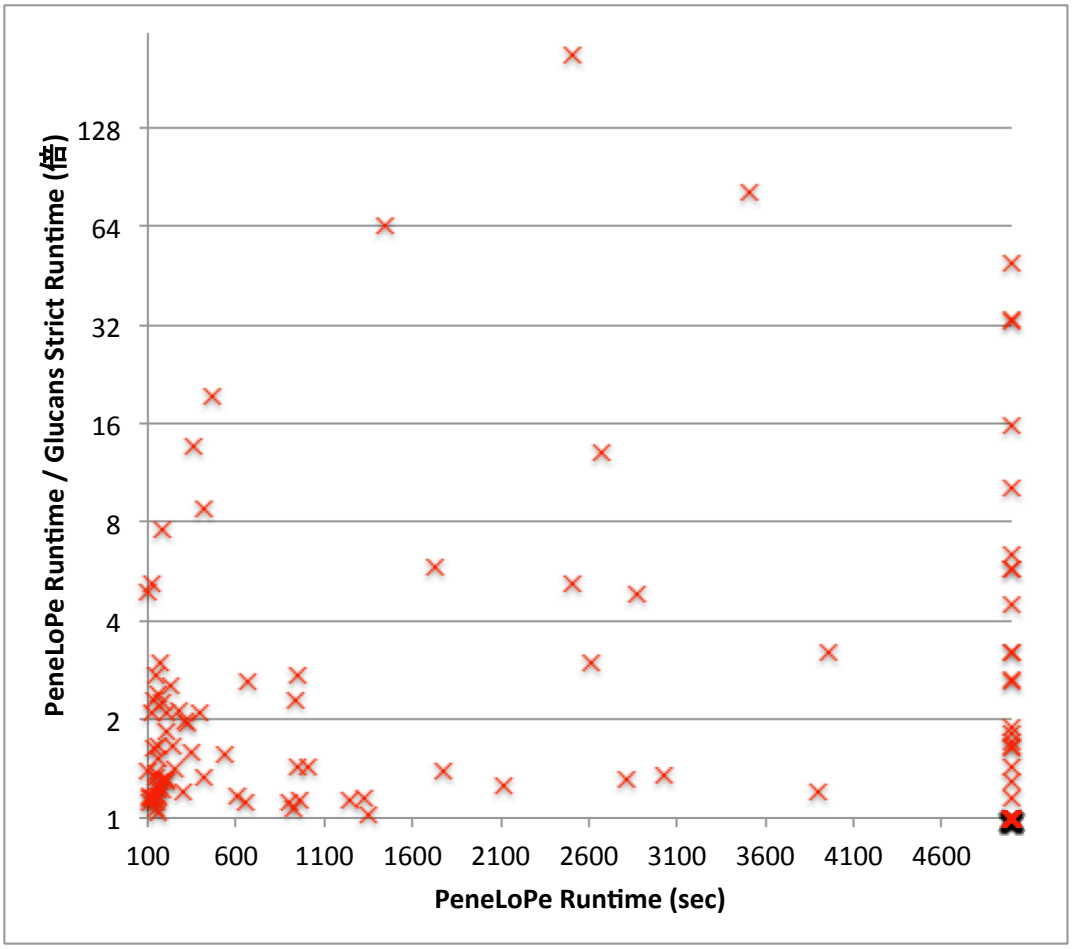

図 6.6 **Glucans Strict と PeneLoPe の比較 (倍率)**

PeneLoPe がタイムアウトしている右側の 5000 秒の縦に続くプロットを注目すると，Glucans の Strict 版が制限時間内に解けている問題が多く存在する事が分かる．

第 7 章

まとめと今後の課題

本章で，本研究で行った実験やその結果についてまとめ，今後の課題を提起する．

7.1 まとめと考察

本研究で SAT Competition 2011 で高速であった逐次 SAT ソルバ Glucose, GLUEMINISAT, CIRMiniSat, Contrasat をベースに並列 SAT ソルバ Glucans を作成した．

Glucans は LBD で制限された学習節共有を行う事で探索の並列化をはかり, LBD の大きい順に削除する Basic 版と学習節の削除方針について活性度の小さい順に削除する Activity 版の 2 つを作成した．また, GLUEMINISAT で用いられている概念である StrictLBD を利用して共有される学習節の量を抑制しながら, 推論中に獲得した新たな Glue Clause を共有する Strict 版を作成した．

このうち Activity 版で SAT Challenge 2012 の制限時間 900 秒以内で解けた問題数を競うルールで, Parallel Track: Application SAT+UNSAT に投稿した Glucans はの最終的な結果は，全 600 題中 521 題解き 4 位であったが，1 位の pfolioUZK との差は 10 問であった．すべての問題それぞれについて最も速く解けたソルバを集計した場合には 179 題 (全問題の約 30%) の問題で最も高速に求解可能で，2 番目の SAT ソルバ PeneLoPe の 78 題の 2 倍以上多いという特長がある．

追加の評価実験では各種 Glucans と PeneLoPe の比較を行い, Glucans の Strict 版が最も高速であるという結果が得られた．また, PeneLoPe がタイムアウトしている問題でも, Glucans の Strict 版が制限時間内に解けている問題が多く存在する事が分かった．

このような評価実験の結果から LBD が 2 の学習節である Glue Clause が SAT ソルバにとって非常に重要であり, その獲得を促すことが探索の高速化につながる事が分かった．

7.2 今後の課題

Glucans では学習節の共有を LBD が 5 以下の学習節に制限しているがこの値について最適値の算出や動的な調整方法について更に検討する必要がある．また, Strict LBD や学習節保有戦略などの個々の手法について個別に評価し, その寄与度合いを調べることが今後の課題と考えられる．

謝辞

本研究を進めるにあたり，ご指導を賜わった上田 和紀教授に深く感謝致します． また，SAT の基本的なアルゴリズムからソースコードの読解まで様々な助言をいただいた先輩の皆様に感謝いたします．この SAT ソルバを評価するにあたって EDACC[11] を使用しました．EDACC やベースとして利用した SAT ソルバ [5][7][8][25] の作者方々にも感謝いたします．

最後に，どんな時も私を支えてくれた家族に深く感謝いたします．

2013 年 2 月　徐　暁雋

参考文献

[1] Ohmura. K., Ueda. K.: c-sat: A Parallel SAT Solver for Clusters, In Theory and Applications of Satisfiability Testing, pp. 524-537, Springer Verlag, 2009.

[2] Niklas Een, Niklas Sörensson: An Extensible SAT-solver, Theory and Applications of Satisfiability Testing Lecture Notes in Computer Science, Volume 2919, pp. 502-518, 2004

[3] M. Luby, A. Sinclair, and D. Zuckerman: Optimal speedup of Las Vegas algorithms, Information Processing Letters, 47(4):pp. 173-180, 1993.

[4] Gilles Audemard, Laurent Simon: Predicting Learnt Clauses Quality in Modern SAT Solver, In Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09), pp. 399-404, July 2009.

[5] Gilles Audemard and Laurent Simon: GLUCOSE: a solver that predicts learnt clauses quality, SAT Competition 2009 Solver Description, http://www.cril.univ-artois.fr/ SAT09/solvers/booklet.pdf, 2009

[6] Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez and Cédric Piette: Revisiting Clause Exchange in Parallel SAT Solving, In Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12), pp. 200–213, May 2012.

[7] 鍋島英知, 岩沼宏治, 井上克巳: GlueMiniSat 2.2.5: 単位伝播を促す学習節の積極的獲得戦略に基づく高速 SAT ソルバー, 日本ソフトウェア科学会第 28 回大会, 6E-1, 2011-9-29.

[8] Allen Van Gelder: Contrasat - A Contrarian Sat Solver, Extended System Description, Journal on Satisfiability, Boolean Modeling and Computation 8, pp. 117-122, 2012.

[9] Youssef Hamadi, Saïd Jabbour, Lakhdar Saïs: ManySAT: a new Parallel SAT

solver, In Journal of Satisfiability Boolean Modeling and Computation (JSAT), special issue on Parallel SAT solving, pp. 245-262, 2009.

[10] Armin Biere: Lingeling and Friends Entering the SAT Challenge 2012, In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions, A. Balint, A. Belov, A. Diepold, S. Gerber, M. J?rvisalo, and C. Sinz (editors), vol. B-2012-2 of Department of Computer Science Series of Publications B, pp. 33-34, University of Helsinki, 2012.

[11] Balint, A., Gall, D., Kapler, G., Retz, R.: Experiment design and administration for computer clusters for SAT-solvers (EDACC), JSAT 7, pp.77-82, 2010.

[12] L. Zhang, C.F. Madigan, M.H. Moskewicz, S. Malik: Efficient conflict driven learning in a Boolean satisfiability solver, In Proceeding of ICCAD 2001, pp. 279-285.

[13] 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.

[14] Niklas Een, Armin Biere: Effective preprocessing in sat through variable and clause elimination, In proc, SAT'05, volume 3569 of LNCS, pp. 61-75, Springer, 2005.

[15] Daniel Kroening, Ofer Strichman: Decision Procedures: An Algorithmic Point of View, Springer, 2008.

[16] 井上克己, 田村直之: SAT ソルバーの基礎, 人工知能学会誌, Vol. 25, No.1, pp. 57-67, 2010.

[17] 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, pp. 68-76, 2010.

[18] Xiaojuan Xu, Yuichi Shimizu, Kazunori Ueda: Glucans System Description, In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions, A. Balint, A. Belov, A. Diepold, S. Gerber, M. J?rvisalo, and C. Sinz (editors), vol. B-2012-2 of Department of Computer Science Series of Publications B, pp. 33-34, University of Helsinki, 2012.

[19] Engler, D., Chelf, B., Chou, A., and Hallem, S.: Checking system rules using system-specific, programmer-written compiler extensions. In Proceedings of the Fourth Conference on Operating System Design & Implementation (San Diego, Oct. 22-25). USENIX Association, Berkeley, CA, 2000.

[20] Flanagan, C., Leino, K.M., Lillibridge, M., Nelson, G., Saxe, J.B., and Stata, R.: Extended static checking for Java. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (Berlin, Germany, June 17?19), pp. 234-245, ACM Press, New York, 2002.

[21] Kroening, D., Clarke, E., Yorav, K.: Behavioral consistency of C and Verilog programs using bounded model checking. In Proceedings of DAC 2003, pp. 368-371. ACM Press, New York, 2003.

[22] Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168-176. Springer, Heidelberg, 2004

[23] Armin Biere: P{re,i}coSAT@SC'09, In SAT 2009 competitive events booklet, pp. 41-43, 2009.

[24] Armin Biere: Lazy Hyper Binary Resolution. Algorithms and Applications for Next Generation SAT Solvers, Dagstuhl Seminar 09461, Dagstuhl, Germany, 2009.

[25] Tomohiro Sonobe: CIRMiniSat, The results of SAT Competition 2011, http://www.satcompetition. org/2011/, 2011.

[26] SAT Challenge 2012, http://baldur.iti.kit.edu/SAT-Challenge-2012/

[27] SAT Competition 2011, http://www.satcompetition. org/2011/

[28] SAT-Race 2010, http://baldur.iti.uka.de/sat-race-2010/

発表論文

[1] Xiaojuan Xu, Yuichi Shimizu, Kazunori Ueda: Glucans System Description, in Proceedings of SAT Challenge 2012 : Solver and Benchmark Descriptions, pp. 23-24, Department of Computer Science Series of Publications B, vol.B-2012-2, University of Helsinki, Helsinki.

[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.

[3] 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.