2009年度 修士論文

クラスタ向けハイブリッドSATソルバの設計と実装

提出日 : 2010年1月28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 : 5108B085-3

露崎 浩太

概 要

システム LSI を中心としたハードウェアの検証や AI プランニングなどといったソフトウェア検証のエンジンとして近年非常に高速化の進んでいる研究として充足可能性問題 (propositional boolean SATisfiability: SAT) があげられる．

SAT はある命題論理式が与えられた際に，式が真となるような解の割り当てを示すか，解が存在しないことを示す問題である．SAT の基本アルゴリズムが考案された 1960 年代には扱える変数の数がせいぜい 100 程度の命題論理式であったが，ソルバの性能が飛躍的に向上した現在では 1000 万程度の変数まで扱うことができ，上記のような検証問題が多項式時間に SAT 形式に還元可能なことから，実用的な検証アプリケーションとして SAT を高速に解く意義は大きい．

しかし，近年多く研究されている逐次 SAT ソルバのほとんどは初期に考案された DPLL アルゴリズムに基づいており，このアルゴリズムは探索木の生成が一様でなく，単純なワークシェアによる探索が困難なため，並列 SAT ソルバの研究はまだ少ない．また，主要な並列 SAT ソルバはベースとなる SAT ソルバをマルチコア向けに並列化したものが多く，大幅な性能向上は望みにくいのが現状である．

本研究では，大規模クラスタ向けの SAT ソルバとして開発された c-sat をベースに SAT Copmetition2009 の Application 部門で最も優秀な成績を収めた逐次 SAT ソルバ PrecoSAT[3] の並列化である c-preco の実装，および Precosat と Minisat を並行に実行し，探索域の剪定に有益である学習節を共有するハイブリッド並列 SAT ソルバ c-satw の設計と実装を行った．c-sat は MPI で実装され，クラスタ上での大規模な運用に耐えられるよう実際に解を探索するワーカとそれを管理するマスターのマスターカ構造を採用している．この時，通信のボトルネックを抑えるため，探索域削減に有益である学習節の共有量の制限を動的に変化させるといった手法で通信オーバヘッド削減の工夫がなされている．

この c-sat のワーカ部分を PrecoSAT に変更し，新たな学習節の共有指針として学習節の世代情報を利用した並列 SAT ソルバ c-preco を実装し，ソルバ全体の速度向上を図った．また，通信に MPI を利用していることから実装やヒューリスティックの全く異なるソルバ同士をつなぐことによって，従来以上の学習節共有効果と，解探索速度の向上を目指した c-satw を実装した．

以上の実装により SAT Competition 2009 の問題 292 題を親マスタ 1PE-子マスタ 5PEs-ワーカ (Minisat)5PEs-ワーカ (PrecoSAT)20PEs の計 31PEs で解かせたところ SAT Competition の制限時間である 1200 秒以内に解けた問題は 216 題と逐次 Precosat より約 50 題多く解くことができた．また、Precosat で解くのに 60 秒以上かかった問題，SAT(充足可能な問題)44 題，UNSAT(充足不可能な問題)62 題に対して 31PEs 実行時に幾何平均でそれぞれ 4.1 倍，2.6 倍という性能向上を得ることができた．

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

近年，ハードウェア検証やソフトウェア検証に応用可能な技術として命題論理式の充足可能性問題 (propositional boolean SATisfiability: SAT) はその求解手法において非常に高速化が進んでいる．SAT は 1971 年に S.Cook において NP 完全問題として証明されているが，多くの研究者によって発展を遂げた解探索のアルゴリズムにより，100 万程度の変数を持つインスタンスを数十分という比較的現実的な時間で求解できる程度まで進歩してきている．SAT の求解ソフトウェアである SAT ソルバの基礎となっている DPLL アルゴリズムを Davis らが考案した当時は 100 程度の変数しか扱えなかったことから比較すると，今日の SAT ソルバの技術発展は著しく，さらなる高速化は巨大なアプリケーションの検証技術として期待されている．

しかし，CPU や計算技術の発展がマルチコア，クラウドなど並列技術の発展へと進んでいく中，近年の優秀な SAT ソルバの多くは逐次ソルバであり，並列 SAT ソルバは全体のごく一部しか存在しない．この理由として，SAT は制約伝播における，枝狩りの効果が非常に高く，探索域を静的に，かつ一様に分割することが難しく大きな並列効果が望みにくいことがあげられる．

本研究では，古典的な DPLL 逐次 SAT ソルバである MiniSat をベースに学習節の共有と探索域の動的分割により効果の高い並列化を行った並列 SAT ソルバである c-sat の解探索部分であるワーカを SAT Competition2009 のアプリケーション分野において最も優秀な成績を修めた逐次 SAT ソルバ PrecoSAT を組込み，解探索性能の向上を図った．また，異なるヒューリスティックから得られる学習節の有効性を示すため，MiniSat と PrecoSAT という全く実装の異なる SAT ソルバをつないだ世界初のハイブリッド SAT ソルバを実装し，より高速な解の探索を目指した．

1.2　論文構成

本論文の構成は下記の通りである．第 2 章では SAT の概要と基本アルゴリズム，近年のソルバに利用されている高速求解テクニックにについて述べる．第 3 章では本研究の研究対象とされた逐次 SAT ソルバ MiniSat，PrecoSAT，及び，並列 SAT ソルバである c-sat の構成，実装などについて述べる．第 4 章では並列 SAT ソルバ

c-preco，ハイブリッド SAT ソルバ c-satw の設計と実装について述べる．第 5 章では実装を行った c-ppreco と c-satw の性能評価実験の方法と結果について述べ，第 6 章にて考察する．第 6 章ではまとめと今後の課題について述べ，第 7 章に関連研究をまとめる．

第 2 章　SAT ソルバの基本的な概念と構成，高速 SAT ソルバに利用されている技術について述べる

第 3 章　本研究に研究対象とされた逐次 SAT ソルバである MiniSat，PrecoSAT，及び並列 SAT ソルバである c-sat について述べる

第 4 章　本研究で実装された並列 SAT ソルバ c-preco，ハイブリッド SAT ソルバ c-satw について詳細に述べる

第 5 章　c-preco，c-satw を用いた評価実験とその結果について述べる

第 6 章　実験結果のまとめと考察，今後の課題について述べる

第 7 章　関連研究について述べる

第2章　SAT

本章では，本研究が対象とする充足可能性問題 (Satisfiblity Problem:SAT) と，SAT の探索アルゴリズムについて説明する．この章の内容は主に参考文献 [17][19][20][18] に基づいて再構成したものである．

2.1　SAT とは

SAT とは乗法標準形で与えられた論理式において全ての節が真となる解の割り当てを求めるか，全て真にならないことを示す充足可能性問題である．SAT は計算機科学の分野において古典的な NP 完全問題として知られており，解を求めるのに最悪指数オーダーの時間がかかってしまうが，実際のソフトウェア検証や人工知能，回路テストなどの分野から得られるインスタンスに対しては，探索木の剪定や変数の伝播により，非常によい性能を示すことがある．また，情報標準形に帰着できる問題であれば求解が可能なことと 10^6 以上の変数を含む大規模な検証問題に対応できることから CAD やモデル検査機のエンジンとして多くのソフトウェアに利用されてきた．

近年ではアルゴリズム，テクニックの発達が著しく，MiniSat[13] をはじめとするほとんどの逐次 SAT ソルバに利用されている DPLL アルゴリズムや変数選択ヒューリスティック，restart など数々の工夫がこらされてきており，その性能はこの 10 年で飛躍的に向上した．また，求解テクニックの応用として入力形式である CNF の単純化 (subsumption) アルゴリズムを基に SAT ソルバに有効な CNF 生成の研究もさかんに行われており SatELite[6] など，CNF 簡略化に特化したアプリケーションも産み出された．

SAT は多くの分野に適用可能であるとともに，毎年，その求解速度を競うコンペティション [1] が開かれており，より高速に SAT を解くためのアルゴリズムの研究とソルバの開発が盛んに行われてきた．現在はまだ逐次 SAT ソルバがその主流であるが，cpu のクロック限界から来るマルチコア化，並列処理の必要性により並列 SAT ソルバも研究が求められて始めている．しかし，SAT は探索木の幅，深さなどが一様でなく単純な並列化が難しいため，動的な探索域の分割や学習節の共有などに着眼し，より効率的な並列化の研究が行われている．

2.1.1 基本用語

SAT および本論文では専門的な用語を多用するため本節において基本的な用語を解説する.

- Variable
  変数，True もしくは False が割り当てられる.

- Literal
  Variable に真偽値を割り当て正負記号をつけたものを総じて Literal と呼ぶ.

- Clause
  Literal を論理和でつなげたもの.

- Unit Clause
  Clause の中でも特に Literal を 1 つしか持たないものを Unit Clause と呼ぶ. Unit Clause に含まれる Literal は恒真でなければならない.

- Binary Clause
  Clause の中で Literal を 2 つだけ持つものを binary Clause と呼ぶ.

- Decision
  Variable に対して真偽値を割り当てることを言う.

- Implication
  Binary Constraint Propagation(BCP) とも呼ばれ，Decision によって Variable に真偽値が割り当てられたことによって，連鎖的に他の Clause の Variable にも真偽値が割り当てられることをこう呼ぶ.

- Conflict(衝突)
  Decision によって割り当てられた真偽値によって Clause が偽となってしまうこと.

- Learned Clause(Lemma)
  Conflict を解析し探索空間の剪定のため学習を行う時に追加される Clause のこと.

- Backtrack
  Conflict が生じた際に，Decision をやり直すこと．

- Flip
  変数の割り当てを反転させる動作．真に割り当てられた変数を flip すると偽に，偽に割り当てられた変数を flip すると真となる．

2.1.2 SAT の例題

SAT で扱われるインスタンスは Clause を論理積でつなげた CNF 形式 (情報標準形：ConjunctiveNormalForm) で表される．

例題

式 1=(a+b+c)(a'+b)(b'+c'+d)(c+d')
式 2=(a+b')(a+b+c)(a+c+d)

式 1 において，a=True，b=True，c=False，d=False と真偽値を割り当てると式全体が充足可能となるなので SATISFIABLE となる．また，最初に a=T と割り当てる場合，Clause(a'+b) が真になるために b=T が自動的に割り当てられる．これが Implication である．式 2 の場合，a=False，B=True と割り当てた場合，一つ目の Clause(a+b') が False となるため Conflict が生じる．

2.2 アルゴリズム

前述したように SAT は人工知能分野やソフトウェア検証などに適用可能であることから，それを解くためのアルゴリズムの研究が数多く研究されてきた．SAT ソルバは利用されているアルゴリズムごとに大別して系統的 SAT ソルバ (systematic SAT solver) と確率的 SAT ソルバ (stochastic SAT solver) とに分けることができ，系統的 SAT ソルバは完全な (complete) アルゴリズムに基づいている．

完全アルゴリズムは与えられた SAT の問題に対して，解を示すか，あるいは充足不可能 (UNSAT) を示す．これに対して確率的 SAT ソルバでは確率的局所探索を行うため UNSAT を示すことのできない不完全な (incomplete) アルゴリズムとなっている．確立的 SAT ソルバではランダムに解の割り当てを設定し，それが充足していれば充足可能であるということを示すアルゴリズムで UNSAT を示すことができないが，問題によっては完全アルゴリズムより高速に解くことができる．しかし，多くの問題では充足不可能であることを示す必要があるため，完全アルゴリズムによる系統的 SAT ソルバの研究が多く進められている．

この節では近年の complete SAT ソルバのほとんどに利用されている DPLL アルゴリズムを中心に，DPLL の基となった DP アルゴリズム，VSIDS ヒューリス

ティック，restart，subsumption など，高速に SAT を解くためのアルゴリズムとテクニックを紹介する．

2.2.1 DP アルゴリズム

DPLL アルゴリズムの基となった完全アルゴリズムで Davis,Putnam によって考案され resolution とも呼ばれる．resolution では異なる clause に同時に含まれる変数を削除し，clause の融合によって解を得ることができる． 例えば図?? のように二つの Clause を一つの Clause へ簡約化しこれを連鎖的に行うことで解を探索する．しかし，DP アルゴリズムはメモリ爆発を起こすという問題がある．

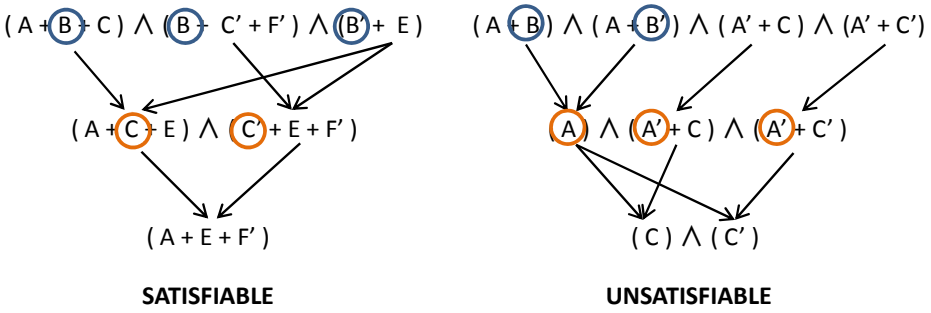

図 2.1: resolution による求解手法 [17]

2.2.2 DPLL アルゴリズム

DPLL は DP アルゴリズムを基に 1960 年代に Davis，Putnum，Logemann，Loveland によって考案されたアルゴリズムである．この 40 年間，多くの SAT ソルバがこの DPLL アルゴリズムをベースに開発されてきた．

ここ 10 年では元々の DPLL アルゴリズムにさらに non-chronological backtracking と conflict driven learning を追加した新しいアルゴリズムの誕生によって大幅に性能が向上した．non-chronological backtracking と conflict driven learning は 1990 年代中頃に，Silva，Sakallah，Bayardo，Schrag によって考案されたものである．

DPLL に基づいた SAT ソルバは数千行程度の比較的小さなソフトウェアであり多くの SAT ソルバは C か C++で記述されている．長年多くの研究者がこの DPLL アルゴリズムをベースにデータ構造，ヒューリスティック，パラメタ最適化などの研究を行ってきた．

2.2.3 DPLL フレームワーク

まず最も初期に Davis や Putnam によって考案された再起版 DPLL アルゴリズムの疑似コードを 2.2.3 に示す.

DPLL 関数 は変数割り当てと式全体 (CNF) を引数に呼び出される．deduction 関数では現在の変数割り当てから推論によって割り当て可能な変数があれば，これを行う．deduction 後，全ての変数割り当てが終了した場合は SAT（充足可能）となる．これ以上割り当てられる変数がなく，conflict を起こしている場合は conflict を起こした変数にバックトラックし変数割り当てのやり直しを行う．変数割り当ては基本的に深さ優先で行われ，バックトラックをする場合は直近の変数割り当てにバックトラックを行う.

再起版 DPLL

```
DPLL(formula, assignment)
    necessary = deduction(formula, assignment);
    new asgnmnt = union(necessary, assignment);
    if (is satisfied(formula, new asgnmnt))
        return SATISFIABLE;
    else if (is conflicting(formula, new asgnmnt))
        return CONFLICT;
    var = choose free variable(formula, new asgnmnt);
    asgn1 = union(new asgnmnt, assign(var, 1));
    if (DPLL(formula, asgn1)==SATISFIABLE)
        return SATISFIABLE;
    else
        asgn2 = union (new asgnmnt, assign(var, 0));
    return DPLL(formula, asgn2);
```

次に現在の主流であるループ版 DPLL を 2.2.3 に示す．まず前処理 (preprocessing) が行われその後，while ループに入る．while ループでは変数選択，推論，衝突の解析と学習，バックトラックを繰り返し解を探索する．preprocess() 関数ではループに入る前に resolution を用いて Clause 同士での矛盾から充足可能か判定する．ここで clause が残り充足可能性が判定できなかった場合に while ループに入る.

ループ内では decide next assignment() 関数により適当なヒューリスティックに従い変数に心理値の割り当てを行う (decision)．次に deduce() 関数を用いて decide next assginment() で決定された変数割り当てが，他の変数に対して連鎖的な割り当てを行えるか推論する (BCP)．変数の割り当ての際には探索木の深さに応じた

decision levelが管理されており，BCP によって割り当てられた変数の decision level は decide next assignment() 関数で値を割り当てられた変数の decision level と同じ値となる．衝突が生じた場合，analyze conflict() では衝突の原因となった変数を解析し，Conflict の起こった変数の decision level を返す．同時に analyze conflict() によって判明した衝突を起こす変数の組み合わせは，そこに解が存在しないことを示すため否定を取った形で新たに clause として追加される (Lemma)．また，この解析によって得られた，decision level が 0 であった場合，これより浅い部分 back track し変数割り当てのやり直し可能な変数が存在しないため UNSATISFIABLE が返される．

新たに変数割り当てのやり直しが可能な場合は back track を行い解探索を続ける．この時 non-chronological backtracking に基づいた back track を行うことで，一度に浅いレベルまで back track することが可能である．このループ版 DPLL は CDCL(conflict-driven clause learning) アルゴリズムと呼ばれることもある．

ループ版 DPLL

```
status = preprocess();
if(status!=UNKNOWN) return status;
while(1)
    decide next assignment();
    while(true)
        status = deduce();
        if(status == CONFLICT)
            blevel = analyze conflict();
            if(blevel == 0)
                return UNSATISFIABLE;
            else backtrack(blevel);

        else if(status == SATISFIABLE)
            return SATISFIABLE;
        else break;
```

2.3 ヒューリスティクス

本節では DPLL アルゴリズムの各フェイズの役割に関して述べる．各フェイズによる動作のヒューリスティクスはソルバによって多少異なるが，近代の SAT ソルバに利用されている最も基本的なヒューリスティクスを紹介する．

2.3.1 変数選択

推論フェーズにてこれ以上変数の割り当てが出来ず，かつ未割当の変数が存在する場合に，decide next assignment() が呼び出され，未割り当ての変数からソルバの変数選択ヒューリスティクスに基づいた変数選択が行われ，変数割り当てが行われる．同じインスタンスを扱う場合にも変数選択ヒューリスティクスが異なると形や大きさの異なる探索木を探索することになる．このことから，分岐の回数やバックトラックの回数を減らすことのできるような探索木を生成する変数選択ヒューリスティクスがよいヒューリスティクスだと言える．

以下に代表的なヒューリスティクスをいくつか紹介する．

- Maximum Occurrences on Minimum sized clauses(MOM)
  初期のヒューリスティックで，最小の大きさの Clause 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．

- Dynamic Largest Combined Sum(DLIS)
  GRASP[11] で実装されたヒューリスティックで各変数の出現回数を記録するカウンタを実装し，最も多く出現し，かつ，未割り当である変数に割り当てを行う．実装方法が単純な割に効果が高いヒューリスティックである．ただし，カウンタの情報は状態依存であるため新たに割り当てやバックトラックが発生するたびにカウンタの更新を行う必要がある．

- Variable State Independent Decaying Sum(VSIDS)
  Chaff[12] で用いられたヒューリスティックで各変数のアクティビティに応じて変数を選択する方法．DLIS 同様，カウンタを用い，初期値は出現回数を用いるが，学習した Learn Clause によって変数が追加された場合，その Clause に出現する変数分カウンタを更新する．また，出現回数の少ない変数のカウンタは減少させる．VSIDS は変数選択の効率が良く，他のヒューリスティックより高速なため，現在多くの SAT ソルバが採用している．

現在の多くの DPLL ソルバは VSIDS を応用した変数選択ヒューリスティクスを採用している．

2.3.2 推論

推論フェイズでは deduce() 関数を呼び decide next assignment() 関数によって行われた変数割り当てを基に他の変数に連鎖的な割り当てを行う．推論フェイズで全ての変数に真偽値の割り当てが行われ，充足していれば SATISFIABLE を返し探索が終了する．衝突が発生している際は conflicat を返し衝突の解析後，バック

トラックを行う．推論を行う方法はいくつか存在し，その中でも最も効果の高いものが unit clause rule である．unit clause とは一つの Literal を除いた全ての Literal が偽に割り当てられた Clause を言い，この場合，自動的に残る一つの Literal が真に割り当てられなければならない．この時，真に割り当てられるリテラルを unit literal と呼ぶ．また全ての unit literal を真に割り当てる作業を unit propagation もしくは Boolean Constraint Propagation(BCP) と呼ぶ．

2.3.3 BCP

ここではを unit clause rule の実装に使用される BCP のメカニズムを紹介する．BCP は SAT ソルバにおいて unit clause や変数割り当てによる変数同士の Conflict を見つけることのできるものであり，ほとんどの SAT ソルバで最も時間のかかる作業であるため，これを高速に行う様々な工夫がなされている．

BCP の直感的な実装には各 Clause にカウンタを用いる方法がある．この実装は GRASP[11]，satz [?] といった多くの SAT ソルバで利用された．例えば GRASP では Clause 毎に二つのカウンタを持つ．変数に真が割り当てられた場合はカウンタ 1 を更新し，偽に割り当てられた場合はカウンタ 0 を更新する．カウンタ 0 の値が Clause 中の変数と等しくなった場合，その Clause 中の全ての変数が偽に割り当てられたことになるので，その Clause は Conflict clause となる．またカウンタ 0 の値がリテラルの個数-1 であり，かつカウンタ 1 が 0 である場合，その Clause は unit clause となる．この方式では Clause が m 個，変数が n 個，Literal が 1 個の場合，新たな変数割り当ての際に，カウンタは平均 1*m/n 回更新される．しかし，最近の SAT ソルバに実装されている衝突の解析と学習による Learn Clause は比較的サイズの長い Clause を保有するため，更新のコストが比較的高くなる．

こうしたカウンタでの実装方法よりも高速な手法として head tail lists と 2-literal watching がある．どちらも各 Clause の二つのリテラルをポインタにより監視する手法である．この方式に利用されるポインタ，リテラルを，それぞれ監視ポインタ，監視リテラルと呼ぶ．監視ポインタの監視する変数に割り当てが行なわれた場合，監視リテラルを更新する．これにより未割り当てのリテラルが常に監視され，割り当てが行われた場合，unit clause rule が適用される．この実装では Clause が m 個，変数が n 個の場合，新たな変数割り当て時に平均して m/n 回の更新しか必要としないためカウンタでの実装より高速に動作する．また，この 2 つの手法の違いはバックトラックのしやすさにあり，2-literal watching の方が比較的高速である．

2.3.4 衝突の解析と学習

SAT ソルバでは Conflict が起こった場合，衝突の解析を行った後，バックトラックをして変数割り当てをやり直す．衝突の解析とは衝突の起こった変数の組み合

わせを発見し clause として学習することである．

衝突の学習結果は，ある探索空間に解がないことを示し SAT ソルバの探索空間を削減する効果をもたらす．

初期の DPLL アルゴリズムである再帰版 DPLL では各割り当て済み変数に真偽値のどちらの割り当てを試したかどうかのフラグを立て，衝突が起こった際，このフラグを基に，まだ両方の割り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を行い，割り当てていないほうの真偽値割り当てを行なっていた．この手法はまだ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ので chronological backtracking(年代順バックトラック) と呼ばれてる．chronological backtracking は SAT のランダム分野で効果的であり satz などのいくつかの SAT ソルバに利用されている．

しかし，矛盾の原因が浅い decision level にあった場合にそこに戻るまでに無駄な探索を繰り返してしまう．一般的に実アプリケーションから生成されたような構造化された SAT 問題に対してはこうした現象が起こりやすく chronological backtracking はあまり効果的ではないことが知られている [?]．そこでもっと改良された衝突解析方法としてループ版 DPLL アルゴリズムである CDCL アルゴリズムでは non-chronological backtracking(非年代順バックトラック) が採用された．non-chronological bactracking では conflict が発見された時，比較的浅い decision level にバックトラックする．この手法では衝突解析のプロセス中において Conflict の原因を解析した結果を Clause として学習する．この時に追加された Clause は SAT の解に影響することはなく，探索空間の削減に利用される．このメカニズムを conflict-directed learning と呼ぶ．このように学習された Clause は発見された衝突である conflicting clause に対し conflict clause あるいは Learn Clause(Lemma) と呼ばれる．

この Lemma の生成手法と学習メカニズムを以下に示す．

Lemma の学習には SAT の探索における含意グラフ (implication graph) を基に行われる．

- implication graph
  学習の基となるアルゴリズムは non-chronological backtracking は implication graph によって実現された．implication Graph は割り当てられた変数の相互関係を示す．implication graph の例を図 2.2 に示す．図 2.2 で丸は各ノード (変数)，矢印はエッジを表し，ノードに添えられた括弧に書かれた数字はそのノードの decision level を表している．BCP によって連鎖的に割り当てられた場合，decision level は等しくなるので V11 の割り当てによって割り当てられた変数は全て decision level が 5 となっている．初期のソルバでは Conflict が生じるまでに割り当てを行なったすべての変数を学習していたため，Conflict に直接関係のない変数まで Clause に追加し冗長な Lemma を生成してしまっていた．implication graph を利用することで decisio level を用いた効率的な Lemma の学習が可能となる．

- UIP
  implication graph 内で conflict したリテラルから decision フェーズにて最後に割り当てた変数までに必ず通る変数の事． 図 2.2 では V11，V2，V10 が UIP である．

- FirstUIP
  UIP の中でも特に最も conflict に近い UIP を FirstUIP という．図 2.2 において FirstUIP でカットする場合-V10，V8，-V17，V19 となるのでその否定を取った (V10+V8'+V17+V19') が Lemma して学習される．UIP での implication graph のカットは FirstUIP でのカット以外に，全ての UIP でカットするなどの方法もあるが，一般的には FirstUIP でのカットによる Lemma の追加が最も良い性能であるとされている．

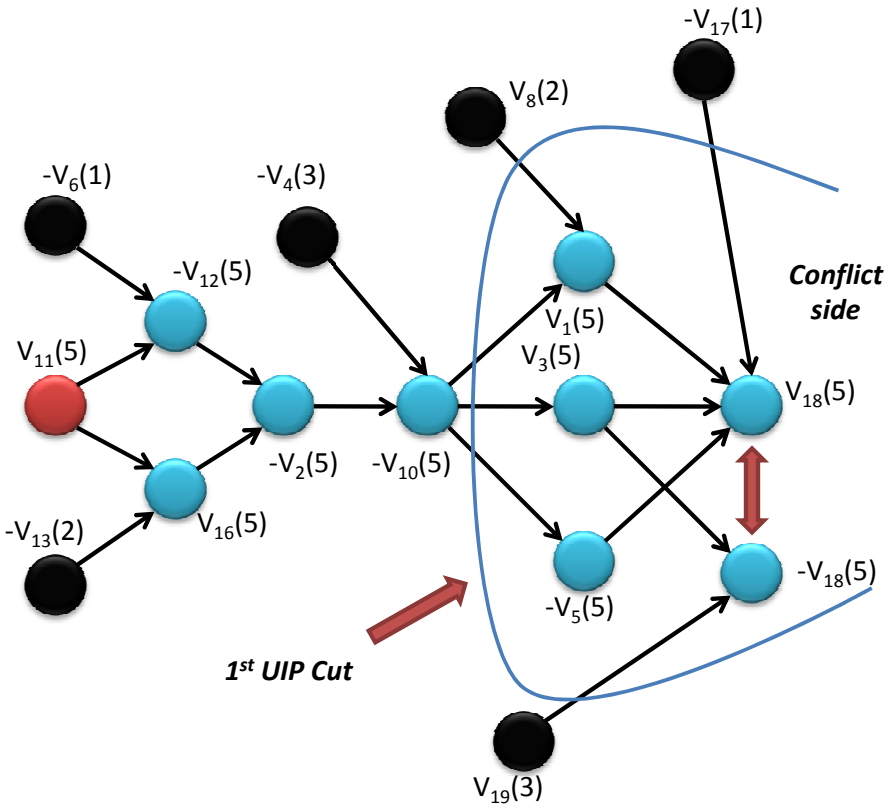

図 2.2: Implication graph の例 [17]

このような衝突学習は conflict driven clause learning と呼ばれ，学習される Lemma は元の問題の Clause の resolution によって生成される．このことから Lemma が追加されても元の問題の充足性に影響はない．しかし Lemma は大抵冗長な Clause

となる為，Lemma による探索空間の削減で得られる処理時間の削減と Clause の増加による処理時間の増加はトレードオフとなる．以上のことから効率的な Lemma を学習するようにすることが重要であり，一般的には長さが短く，浅い探索域で発見された lemma が有益な lemma とされている．[2]

Resolution による Lemma の生成アルゴリズム

```
analyze conflict()
    cl = find conflicting clause();
    while (!stop criterion met(cl))
        lit = choose literal(cl);
        var = variable of literal( lit );
        ante = antecedent( var );
        cl = resolve(cl, ante, var);

    add clause to database(cl);

    back dl = clause asserting level(cl);
    return back dl;
```

2.3.5 Restart

DPLL アルゴリズムでは，chronological-backtracking から局所解に陥ってしまう可能性があった．CDCL アルゴリズムで比較的浅いレベルにバックトラックを行うようになったが，SAT では変数選択次第では何時間もかかる問題が一瞬で解ける場合も少なくない．

そこで，近年のほとんどの SAT ソルバにはリスタート (restart) と呼ばれる，探索を比較的短い間隔で最初から探索をやりなおすという手法が使われている．リスタート戦略には大抵の場合，conflict の発生回数が利用され，この値がある閾値を超えるとリスタートを行う．リスタートを行っても衝突解析によりある探索域に解がないことを示す Lemma を保持することが可能なため，今までの探索が無駄になることはなく，リスタートにより SAT ソルバの振舞いの不安定さをある程度取り除くことができる．

近年では，実行時間における実行時間の期待値を最小化する Restart の間隔が研究されており [10]，同じソルバでもこの間隔の調整によって比較的大きな速度向上が見込めることが知られている [9].

2.3.6 Subsumption

DP アルゴリズムに基づいた clause の簡略化は clause データベースのコンパクト化が行えるため，非常に大きな問題に対して非常に効果が高く，多くの SAT ソルバの preprocessing で実装されている．これを Subsumption(clause simplification) と呼ぶ．近年の SAT インスタンスにおいては clause データベースにおける Lemma の占める割合が肥大化し BCP の効率を下げている場合があり，これを解決するために MiniSAT2.0[14] など比較的新しいソルバにおいては定期的に Clause データベースで activity に応じた Lemma の削除や Subsumption を行うことが主流となった．

第3章　SATソルバ

この章では逐次SATソルバであるMiniSat，PrecoSAT，並列SATソルバであるc-satについて紹介する．

ここで述べたもの以外の近年のSATソルバは関連研究として7章にまとめる．

3.1　逐次SATソルバ

現在は研究されている多くのSATソルバがこの逐次SATソルバであり，毎年SAT Competitionに約40個もの逐次SATソルバが投稿されている．投稿されるSATソルバは実装の言語や方針においては様々であるが，多くのSATソルバが前章で述べたCDCLアルゴリズムに基づいている．しかし，CDCLアルゴリズムにおける変数選択，推論，学習，restartなど各フェイズのストラテジにおいて大きく異なり，それが各ソルバの性能に大きく影響するため，ストラテジに関する小さな差異がソルバの問題に対する得手不得手を決定づけることも少なくない．こうした性能に影響を及ぼすSATソルバのストラテジをMiniSatについて次節で，Preconisatについて次々節にて詳細に説明する．

3.2　MiniSat

MiniSatはSAT Race2006，SAT Competition 2007，SAT Race 2008において非常に優秀な成績を収めたCDCL SATソルバである．

現在公開中のものに，MiniSat1.14とminisat2.0とがあるが，この章では3.5節で述べるc-satで利用されているMiniSat1.14のC版であるMiniSat-C_v1.14.1について述べる．

以下，MiniSat-C_v1.14.1をMiniSatと表記する．

3.2.1　概要

MiniSatはDPLLアルゴリズムに基づいたCDCL SATソルバである．

MiniSatにおいてDPLLの各フェイズは以下の関数に対応している．

MiniSat で実現される DPLL アルゴリズム

変数選択 = order_select()
推論 = solver_propagate()
衝突の解析と学習 = solver_analyze(), solver_record()
バックトラック = solver_canceluntil()

MiniSat では図**??**に示すように solver_saerch() 関数で上記の各フェイズを行うループに入り繰り返し探索を行う．MiniSat では各 clause や変数割り当てといった探索に必要な情報の多くは solver 構造体に格納されている．以下にソルバ構造体に格納されている情報の一部を示す.

- *int size*
  問題で扱われる変数の数
- *int cap*
  問題サイズに対して割り当てるメモリ量
- *int qhead*
  割り当て済み変数キューの head を示す
- *int qtail*
  割り当て済み変数キューの tail を示す
- *vecp clauses*
  問題の clause を格納した vector．探索の途中で不要となった clause は適宜ここから削除される
- *vecp learnts*
  探索中に得られた学習節 (lemma) を格納した vector
- *vecp* wlists*
  2-literal watching で監視されている clause へのポインタを格納した vector
- *lbool* assigns*
  各変数に割り当てられた真偽値
- *clause** resasons*
  BCP で変数割り当ての原因となった clause．変数ごとに持っている
- *int* levles*
  各変数の decision level
- *lit* trail*
  割り当て済み変数キュー

- *veci trail_lim*
  各 decision level での割り当て済み変数の数

3.2.2 変数選択

MiniSat では VSIDS をもとにした変数選択ヒューリスティックを採用している．変数選択の際にリテラルの出現回数である activity を参照し，activity の高い変数に優先的に割り当てを行っていく．MiniSat では各変数の activity の初期値は 0 に設定されており，定期的に減少していくが，衝突に関係した際に増加される．activity の高い変数は衝突に関係しやすく，BCP 効率の高い変数と考えられるためこれに割り当てを行うことで BCP 効果の向上を図っている．

3.2.3 推論

MiniSat では 2-literal watching を採用しており，各 clause に含まれるリテラルのうち先頭の 2 つのリテラルが BCP の対象として監視されている．これをソルバ構造体に含まれる wlists vector で保持している．solver_propagate() 関数が呼ばれた際には割り当て済みのリテラルを trail キューから取りだし，wlists で管理されている clause にリテラルが含まれていれば，clause の充足性をチェックする．

以下に solver_propagate() の大まかな動作を示す．

solver_propagate() 関数

```
clause* solver_propagate(solver* s){
    while ( trail is not empty ){
        lit p = s->trail[s->qhead++];
        vecp *wlists = solver_read_wlist( s , p );
        for each clause wlists{
            if ( clause is already satisfied ) {
                continue;
            } else if ( clause is unit clause ) {
                enqueue();
            } else if ( clause is conflicting clause ) {
                return clause;
            }
        }
    }
    return confl;
}
```

solver_propagate() によって新たに unit clause が生じた場合，unit clause に含まれる変数の割り当てをリテラルとして trail キューに積み，リテラルが全て取りだされるか，clause に衝突が起こるまでこれを繰り返す．wlists に保持されている変数が割り当てを受けた clause では clause 中のリテラルの順序を変更し，常に先頭 2 つに監視リテラルが来るように設定する．

割り当てによって衝突が生じた場合には，その clause を返し衝突学習フェイズに移行する．

3.2.4 衝突の解析と学習

solver_propagate() によって衝突が発生した場合，衝突が起きたた clause が得られるので，これを基に solver_analyze() 関数に手衝突の解析を行う．衝突が起きるまでの変数割り当ては，各変数ごとに割り当ての原因となった reason clause，割り当てられた decision level をそれぞれ reason，level に格納しているため，そこから衝突の原因となった clause を解析する．この時，衝突の原因となった変数と clause の activity が増加される．

この解析によって得られた lemma は solver_record 関数によって lemma データベースに追加される．

3.2.5 restart

MiniSat では衝突の回数がある閾値以上になった場合，restart をかけるという戦略を取っている．この閾値は初期値で 100 が設定されており，restart が行われると閾値の 1.5 倍に増加していく．

3.3 PrecoSAT

PrecoSAT は SAT Competition2009 の Application 部門で最も優秀な成績を収めた CDCL SAT ソルバである．

PrecoSAT は名前氏によって製作された SAT ソルバであり，C 言語で書かれた Picosat[3] をベースに C++言語でで新たに書き起こされ，2 項制約推論等の新しい機能が追加された SAT ソルバである．

この節では PrecoSAT の公開最新版である PrecoSAT version236 について解説する．

3.3.1 概要

PrecoSAT は MiniSat 同様，DPLL アルゴリズムに基づいた CDCL SAT ソルバである．C++言語で記述されているため，多くの関数はクラスの Solver クラスのメンバ関数として定義されている．よく使われるクラス，構造体を以下に示す．

- Solver クラス
  SAT の探索に必要な情報，関数がまとめられたクラス

- Opts 構造体
  Solver クラスに属し，変数選択ストラテジなど初期設定がまとめられたクラス

- Stats 構造体
  Solver クラスに属し，restart や conflict の回数など動作の経過情報が記録される変数を多数保持する

- Cls 構造体
  cnf の 1clause をこの形で保持する．Literal 情報以外に，学習節か否か，binary clause であるか否かなどいくつかのフラグ情報を保持している

- Var 構造体
  各変数の decision level や reason clause の情報等を管理する．各変数が binary clause に属しているかもここで管理される．

- Anchor テンプレートクラス
  precosat の clause リストの先頭と終端のポインタを持つテンプレートクラス

PrecoSAT は clause 用の構造体，変数用の構造体をそれぞれ持つが，clause の管理は MiniSat のベクター方式と異なり双方向のリンクリスト構造を取っている．各変数に対して decision level や reason clause の情報を持つのは MiniSat 同様である．
また，PrecoSAT において DPLL アルゴリズムは以下の関数に対応している．

PrecoSAT で実現される DPLL アルゴリズム

変数選択 = Solver::decide()
推論 = Solver::bcp()
衝突の解析と学習 = Solver::analyze()
バックトラック = Solver::undo(int newlevel)

以降は Solver クラスのメンバ関数に対して,「Solver::」表記を省略する.
この他に Solver 構造体で利用される主な変数に，以下のようなものがある

- *int size*
  問題で扱われる変数の数

- *stack trail*
  問題で割り当て済みの変数が積まれるスタック

- *stack lits*
  問題を clause として格納する際に clause に含まれるリテラルを積むスタック.
  Lemma を作成する際にも利用され，解析された変数がここに積まれる

- *stack units*
  BCP() を行った結果，unit clause になったものが積まれるスタック

- *Anchor<Cls> original*
  問題の clause リストの先頭と終端のポインタを持つアンカ

- *Anchor<Cls> binary*
  binary clause のみが連結される clause リストのアンカ．binary clause であった場合は original clause や lemma であってもこのリストに連結される

- *Anchor<Cls> learned*
  lemma が連結されるアンカ．他と同様に先頭と終端のポインタを持つ

- *int agility*
  変数割り当てが行われると減少するが割り当てが flip だった場合増加される

- *Cls∗ conflict*
  衝突が起こった clause を格納する変数．初期値は 0

3.3.2 変数選択

PrecoSAT では変数選択は schedule スタックによって決定される．schedule スタックはにバックトラックである undo() 関数で割り当てを撤回された変数が優先的に積まれていく．これは MiniSat の衝突に関係した変数の activity を増加させる変数選択の優先度と，ほぼ同様である．さらに PrecoSAT でも，ある一定の条件でのランダムな変数選択を行われている．PrecoSAT でランダムな変数割り当てが行われる確率もランダムであるが，仮にランダムな割り当てが行われる状態であったとしても，flip が発生すると増加する agility が閾値より高い場合，スケジュールされた変数選択が優先される．

3.3.3 推論

precosat の推論は bcp() 関数で呼ばれる prop2() 関数と propl() 関数によって行われる．以下に bcp() 関数の大まかな動作を示す．

bcp() 関数

```
bool bcp (){
if ( units is not empty ) flush();
    while ( trail is not empty ){
        if ( queue2 < trail ){
            int lit = trail[ queue2++ ];
            prop2( lit )
        } else if ( queue < trail ){
            if ( clause is conflicting ) break;
            int lit = trail[ queue++ ];
            propl ( lit );
            if ( clause is conflicting ) break;
        } else break;
    }
    return not conflict;
}
```

bcp() 関数を呼ぶと，trail に積まれた変数を取り出し prop2() 関数によって binary clause として特別に binary リンクリストに積まれている clause に対して propagation を行う．prop2() 関数では binary clause しか扱わないため propagateion した結果，リテラルを含む clause が見つかった場合，その clause を unit clause と断定することができるため，すぐに unit propagation をかけることができる．precosat では，この propagation は imply() 関数によって実現されている．

次に prop2() 関数で衝突が起こっていなかった場合，propl() 関数によって 2 以上の長さを持った clause に対して propagation を行う．propl() 関数では 2-literal watching が採用されており，先頭二つの監視リテラルを含む Cnf に対して propagation が行われる．割り当てが行われた結果，binary clause となった clause は binary リンクリストに積まれ，それ以外は MiniSat 同様監視リテラルの更新が行われる．

衝突が起きた clause はメンバ関数である conflict に格納され，ループが break される．

3.3.4 衝突の解析と学習

bcp() 関数によって衝突が発生した場合 analyze() 関数を呼びだし，衝突の解析と学習を行う．precosat では 1st UIP による Lemma の学習を行っている．analyze() 関数が呼び出されると今まで割り当てが行われた変数のうち UIP が計算され学習される．学習された Lemma は最も衝突に近い trail 変数の flip を dominator とし，管理される．さらに，バックトラックを行う前に，この dominator を真とした propagation が行われる．その後，学習されたリテラルのうち，最も浅いレベルで決定された変数をバックトラックレベルと設定しバックトラックを行う．

3.3.5 restart

PrecoSAT では衝突の回数に基づいた restart 戦略を取っている．衝突と探索レベルが一定数を超えるとと変数割り当てを全てリセットする restart が行われる．衝突の閾値は Luby シーケンス [10] に基づいて計算され，探索レベルは固定値である．

3.4 並列 SAT ソルバ

SAT ソルバは長年，逐次アプリケーションとしてアルゴリズムや実装テクニックの研究がされてきた．しかし，元々，バランスの良いワークシェアリングを静的に行うことの難しい SAT ソルバでは，効率的な propagation アルゴリズムなどが研究され，逐次的なアクセスに特化することで，並列化時のデータ共有が重大なボトルネックを招くことも多い．

特に，探索域削減に有効であるとされる Lemma の共有は並列 SAT においてスタンダードな技術であるが，大量の Lemma はデータ共有にかかる時間の増加だけでなく，propagation 自体も増加させ探索効率を低下させる可能性があり，並列化における有効なデータの効率的な共有は大きな課題となっている．

3.5 c-sat

この節では研究対象であり，Lemma の制限や制限付き探索域分割により通信コストを抑え，かつ，スケーラビリティを実現したクラスタ向け並列SAT ソルバ c-sat の構成，探索手法などについて紹介する．

3.5.1 概要

c-sat は MiniSat V1.14 をベースに C 言語で実装されたソルバであり，Message Passing Interface(MPI) を用いてクラスタ向けに並列化されている．図 3.1 に示すように c-sat ではマスタープロセスとワーカプロセスに分かれ，探索を行うワーカプロセスをマスタープロセスが管理を行うマスターワーカモデルを採用している．各ワーカは問題用，Lemma 用などの clause データベースを独自に持つが，Lemma や探索域の情報はマスターを介して別のワーカプロセスと共有される．ワーカ間では通信を行わず全てのデータはマスターを介して共有される．また，一つのマスタに通信が集中しレイテンシが起こるのを避けるため図 3.2 に示すようないくつかのワーカを管理する子マスタ，子マスタを管理する親マスタからなる 3 層マスターワーカ構造をとっており，ディフォルトでは親マスタ 1:，子マスタ:n，ワーカ:n*5 という構成になっている．

3.5.2 探索手法

各ワーカで動作しているソルバは MiniSat であるが，それぞれが異なる探索域を探索するよう，ランダムシード，ヒューリスティックをワーカ毎に少しづつ変更するように設定されている．この工夫により，極力，探索域の重複を避け，異なる探索域の Lemma が得られるような設計になっている．

以下に c-sat で行われている工夫を示す．

ランダムシードによる変数選択順序変更手法 c-sat 上の各ワーカは 98%の割合でそれぞれ適当な変数選択ヒューリスティックに従い，探索を進めるが，2%の割合で起こるランダムシードを種とする変数選択 (Random Decision) を行っている．この Decision の基となるランダムシードを各ワーカ毎に異なる値を割り当てることによって，Random Decision が起こった場合の探索域の重複をなるべく回避するように調整されている．

ヒューリスティックによる変数選択順序変更手法 c-sat では decision の際の変数選択ヒューリスティックを 2 種類用意している．一つは MiniSat のデフォルトである VSIDS に基づくヒューリスティックである．もう一つは VSIDS のバリエーションとして，変数でなくリテラルに基づく変数選択 (Literal State Decaying Sum:LSIDS)

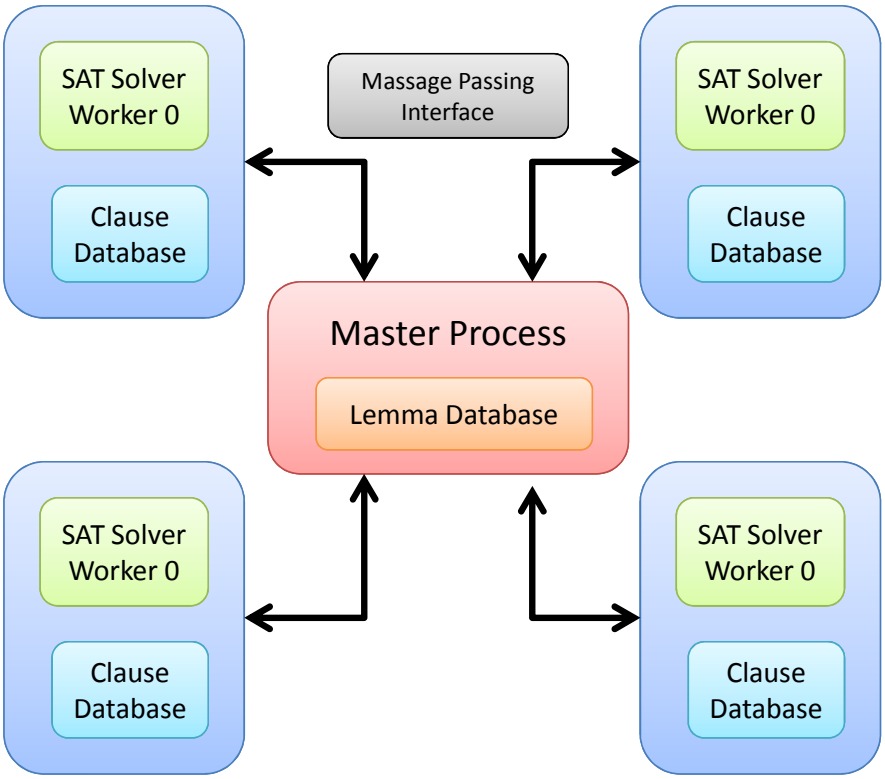

図 3.1: c-sat の構成 [15]

である．従来のヒューリスティックでは変数選択は各変数の activity に基づいており，選択した変数に偽になる割り当てを行うようになっている．しかし，Clause 上では変数は符号付きのリテラルとして登場するため，その変数を持つ Clause 上での真偽値は一様にならない．これを回避するため，LSIDS ではリテラルに activity を持たせ，activity の高いリテラルに優先的に割り当てを行う．これにより任意のリテラルを多く真または偽にすることができる．

ワーカはこの二つのヒューリスティックのいずれかに従って変数選択を行っている．

3.5.3 探索木の分割

これまでにも述べたように SAT は均等なワークシェアリングを静的に行うことが難しいため，c-sat では制限付き動的分割を行っている．c-sat の動的分割アルゴリズムは以下のように実装されている．

ワーカはマスターに探索域を問い合わせ，順序づけられた割り当てを受け取り，それを基に，探索をトレースする．探索のトレースをし，動的分割されるべき変

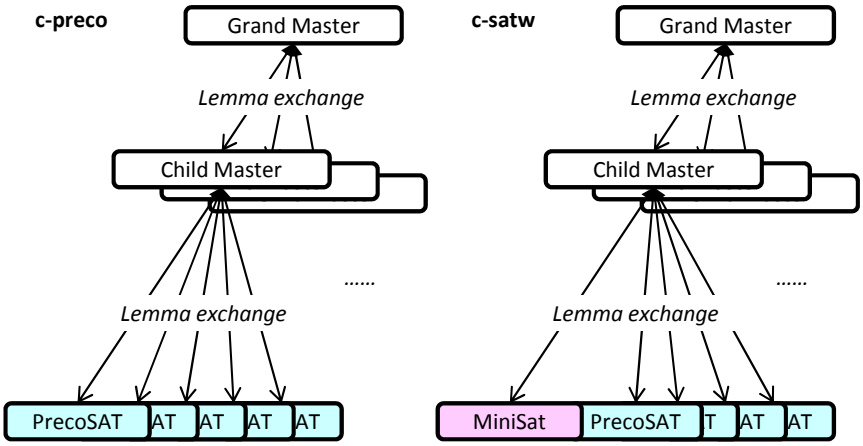

図 3.2: c-sat の 3 層マスタワーカ構造 [15]

数まで辿り着いた後，その変数に flip した値を割り当てて解を探索する．探索の続きを行う，あるいは分割された探索木がない場合，自分の割り当てをマスターに送ることで，以後のワーカは，分割された探索域を探索することになる．

この方法は，順序づけられた決定変数を共有するだけでよいため通信コストが削減されるが探索をトレースによる無駄な propagation が増大するため，c-sat では分岐レベルを制限して浅いレベルのみで行うように設定されている．

3.5.4 lemma 共有

lemma は探索域の剪定効果があるためワーカ間で共有することで探索域の大幅な削減が期待できるが，頻繁な lemma の交換は膨大な通信コストがかかってしまうため，c-sat では通信のタイミングと交換する lemma の量に制限をかけている．

一般的に clause は長さが短い方が propagation の効率がよい．そのため，長さを基準とした lemma 制限が一般的であるが [4] 学習する lemma は探索が進むにつれ，長くなっていくことが多いため [15]，静的に制限値を決めることが難しい．このため，c-sat では未交換の lemma の平均長に基づいて動的に制限値を変更する手法を取っている．

また，lemma 生成の度に通信を行うとオーバーヘッドが非常に高くなるため，ある程度，まとまった量が貯まってから通信を行うように工夫がされている．

第4章　c-satwの設計と実装

c-satw は c-sat をベースに実装された並列 SAT ソルバである．本章では本研究の成果物である c-satw と，そのベースとなった c-preco の設計と実装について述べる．

4.1　c-precoとは

c-preco は c-sat のワーカ部分を MiniSat を PrecoSAT に変更し，全体的な速度向上を図ったソルバである．

4.2　c-precoの構成

c-preco は c-sat と同様に直接探索を行うワーカとそれを管理するマスタからなるマスターワーカ構造をとっている．ワーカは探索によって得られた lemma をマスターに送り，マスターがそれを配布する．c-preco では c-sat と同じ 3 層構造を取り，親マスタ:子マスタ:ワーカの比も同様に 1:n:n∗5 に設定しているが，各ワーカにあたる SAT ソルバの探索速度が向上する分，c-sat 以上の通信が発生することが考えられるため，探索域の動的分割機能を排除し，負荷の低減を図った．また，通信待機時間の削減のため，通信間隔が c-sat より大きくなるように通信タイミングの調整も行った．

4.3　c-precoにおける探索域分割

c-preco では子マスタワーカ間の通信の削減を図るため，c-sat に実装されていた探索域の動的分割機能は停止されている．また，変数選択ヒューリスティックは全てのワーカがデフォルトの VSIDS を利用しており，探索域の分散にはランダムシードの変更による手法のみを用いている．

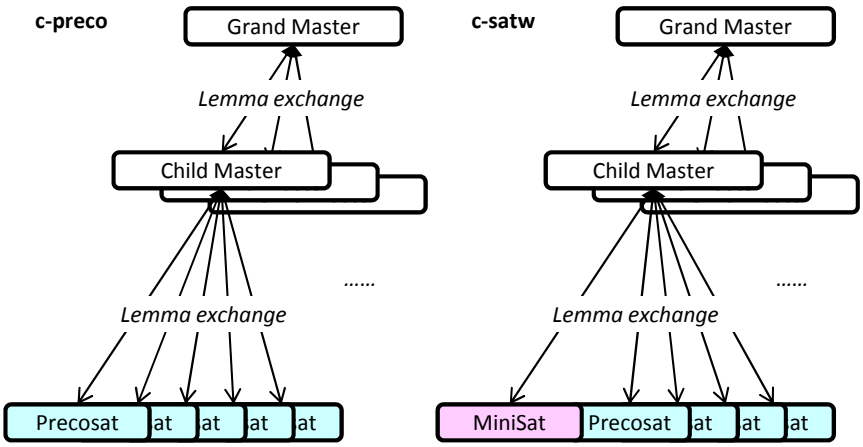

図 4.1: c-preco と c-satw の構成

4.4 lemma の共有

lemma となる clause は resolution によって生成されるため，元の問題の充足可能性に影響はない．しかし c-sat のエンジンであった MiniSat と PrecoSAT では clause の保持の形態，propagation の実装に違いがあるため，共有される lemma の形態に工夫を加える必要があった．以下にその工夫を紹介する．

4.4.1 dominator の共有

PrecoSAT の lemma は全て uip の flip された dominator の reasoning clause として学習されていることは PrecoSAT の項で述べた．PrecoSAT では探索時の衝突解析に，この変数情報とそれに紐づく clause 情報を利用しているため，共有された clause を learned リンクリストにつなぐだけでなく，それに関係する変数情報も同時に共有必要があった．これを解決するため，c-preco では図 4.2 に示すように従来の lemma 共有に加えて dominator とその決定レベル (decision level:dlevel) の共有を行っている．

全てのワーカは lemma 学習時に保持していた dominator の情報を通信時にマスターに送る．マスターはこの lemma と dominator の情報を全てのワーカ間で共有するが，ワーカ側で得られた dominator 情報に基づいて，その lemma を学習するかどうか決定する．

dominator を用いた lemma 制限については次節で述べる．

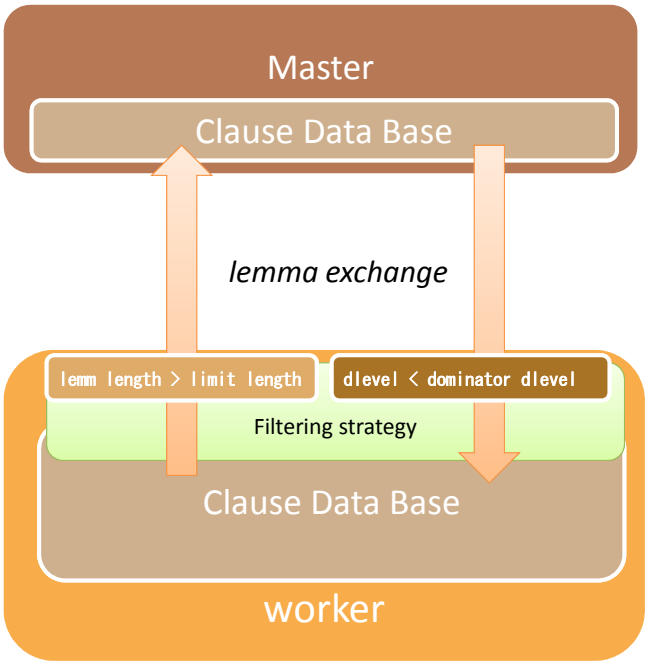

図 4.2: c-preco，c-satw における lemma 制限

4.4.2 dominator に基づく lemma 制限

c-preco では dominator の dlevel に基づく lemma の制限を行っている．一般的に lemma は浅い探索レベルで得られる lemma の方が探索域の刈り取りに有効であるため，ワーカが受け取った lemma の dlevel が現在 lemma を保持する dominator の dlevel より浅かった場合のみ lemma として学習し，新たな dominator として設定する．これにより，より浅いレベルでの lemma による探索域剪定が期待でき，lemma の無駄な増大も抑えられる．

4.4.3 長さによる制限

この他に c-preco では，c-sat 同様，Lemma の平均長に基づく動的な Lemma 制限も行っている．この制限は lemma の送信を行うワーカ上で実装されており，lemma の送信制限と言える．c-preco は lemma の長さによる送信制限によって lemma の通信量を抑え，dominator に基づく受信制限によって，さらに各ソルバが使うメモリ量の増大を抑えるという二重の制限が実装されている．

4.5 c-satw とは

c-satw は c-sat の通信が MPI であることを利用して，従来の c-sat のエンジンであった，MiniSat と現在世界最速である PrecoSAT を並列に実行し，かつ別々のストラテジから生じる lemma を共有することのできる世界初のハイブリッド SAT ソルバーである．c-satw では lemma の全く異なるストラテジから得られる lemma の共有効果を期待するとともに，ストラテジの違いによる問題の得手，不得手を吸収し，より多くの問題を解けることを目指して設計されている．

4.6 c-satw の構成

c-satw でも従来の c-sat と同様のマスターワーカ構造を採用している．基本的な構成は c-preco とほぼ同様だが各子マスタに 1 つずつ MiniSat がつながった親マスタ 1：，子マスタ:n，MiniSat:n，PrecoSAT:n*4 という構成になっている．

また，逐次性能のよい PrecoSAT を多く配置することから通信量は c-preco とほぼ同等と考えられるため通信タイミングは c-preco のストラテジに合わせた．

4.7 lemma の共有

c-satw では通信の実装が MPI であることを利用して MiniSat と PrecoSAT という実装もストラテジも異なる lemma をお互いに受け入れられる形にキャストできるよう工夫されている．

4.7.1 lemma のキャスト

lemma は MPI で通信が行われる際に 1 つの clause をリテラルの配列の形で表現しているため，データのキャスト自体は比較的容易である．しかし，前述した通り PrecoSAT では dominator に基づく学習を行っているため，MiniSat から得られる lemma は PrecoSAT 上で不完全な lemma となってしまう．これを解決するために，今回の実装では c-preco で利用する dominator，dlevel に MiniSat の識別子を乗せることで，MiniSat から得られる lemma であった場合は，受け取ったワーカ側で dominator を生成し，設定するという工夫を行った．

4.7.2 lemma の制限

c-satw では c-preco と同様の lemma 長による動的送信制限と dominator に基づく受信制限をかけている．ただし，MiniSat には元々lemma を dominator に紐づけて管理する機構がないため，受信した全ての lemma を学習する．

MiniSat の台数割合は PrecoSAT より少ないが，上記の理由により全てのワーカが PrecoSAT である c-preco よりメモリを多く消費する可能性がある．

第5章　評価実験

本章では，並列SATソルバc-precoおよびハイブリッドSATソルバc-satwの評価実験について述べる．

並列実行時の表記は7PEとある場合は親マスタ:1，子マスタ:1，ワーカ：5を示しており，31PEとある場合は親マスタ:1，子マスタ5:，ワーカ:25であることを示している．

5.1　評価環境

本実験で利用した評価実験環境を表5.2に示す．

表 5.1: 評価実験環境

マシン名	satsuki
CPU	AMD Opteron^{tm}
CPU周波数	3.0GHz
コア数 (PE数)	Dual-core × 8 (16PE)
メモリ	128GB / node
ノード数	28
通信	Giga bit ethernet(10Gps)
MPI	MPICH2

5.2　ベンチマーク

本実験では最新の大会であるSAT Competition 2009の本戦で利用されたApplication部門の問題292問を利用した．また，大会同様，解探索の制限時間を1200秒に制限し，1200秒かかって解けなかった問題は1200秒で解けたものとして，速度向上比を計算している．

5.3 実行結果

以下に c-preco と c-satw の実行結果，及び，同環境における MiniSat，PrecoSAT の実行結果を示す．表中で SAT とは充足可能である問題を示し，UNSAT とは充足不可能である問題を示す．また UNK とは UNKNOWN を示し，予め問題に対する答えが示唆されておらず，かつ，未だにどのソルバも解くことのできていない問題である．c-sat，c-preco，c-satw は並列実行のため実行結果に幅が生じるため実行時間は 3 回計測した平均値を取り，3 回のうち一回でも解けた問題は解けた問題として換算した．

5.3.1 各ソルバ同士の比較

表 5.2 に minisat，precosat，c-sat，c-preco，c-satw の実行結果を示す．c-sat，c-preco，c-satw はそれぞれ 31PE で実行させた結果を掲載する．

表 5.2: c-satw の実行結果比較

	minisat	precosat	c-sat	c-preco	c-satw
solved (SAT99 題)	45	66	83	86	91
solved (UNSAT178 題)	64	100	112	125	125
solved (UNK15 題)	0	0	0	0	0
total	109	166	195	211	216
総実行時間 (SAT)	71010.1s	50457.4s	34210.8s	35185.1s	26927.9s
総実行時間 (UNSAT)	153332.9s	115877.6s	98711.1s	97660.7s	89882.7s
総実行時間 (UNK)	18000.0s	18000.0s	18000.0s	18000.0s	18000.0s
total	242342.7s	184335.0s	150921.9s	152155.4s	134810.6s

表 5.2 において solved は各ソルバが 1200 秒以内に解けた問題数を示す．総実行時間は各問題の実行時間を足し合わせたものである．satsuki 上では c-sat の並列効果もよく得ることができ，逐次性能で世界最速の precosat より 50 問近く多く解けていることがわかる．

次に c-preco(31PE) の性能向上比をプロットしたものを図 5.1，図 5.2 に，c-satw(31PE) の性能向上比をプロットしたものを図 5.3，図 5.4 に示す．プロットは下記の条件の全てを満たした問題 106 問を対象に行った．

プロット条件

- c-satw(31PE) で 3 回とも 1200 秒以内に解けた問題
- c-preco(31PE) で 3 回とも 1200 秒以内に解けた問題
- PrecoSAT で解くのに 60 秒以上かかった問題

図 5.1，図 5.3 では precosat に対する性能向上比を，図 5.4，図 5.2 では c-sat(31PE) に対する性能向上比を示している．

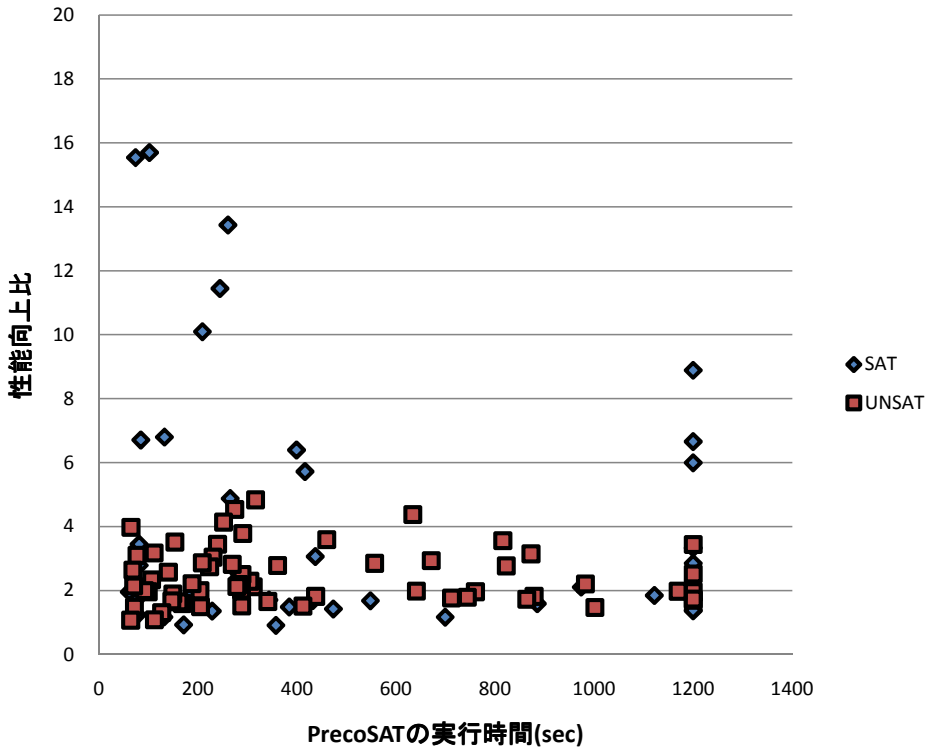

図 5.1: c-preco(31PE) の PrecoSAT に対する性能向上比

全ての性能比較図で比較元のソルバの実行時間を横軸にとり，それに対する性能向上比を縦軸に取った．

図 5.1 より，c-preco でも大抵の問題で PrecoSAT に対して性能向上していることがわかるが，c-sat が比較的早く解けた問題に関しては c-preco の方が遅い場合があることがわかる．

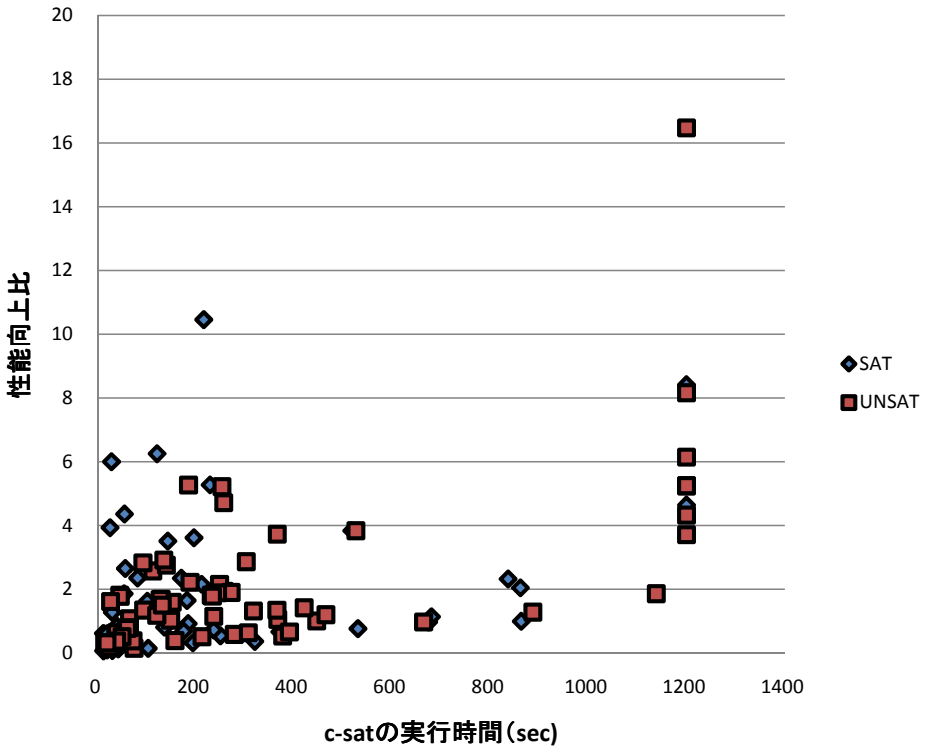

図 5.2: c-preco(31PE) の c-sat(31PE) に対する性能向上比

このような問題の得手，不得手に当たる部分が c-satw では解消され PrecoSAT が比較的早く解けた問題に対しても性能向上していることがわかる．

また，同様の 106 問に対して precosat に対する性能向上比の幾何平均と最大を表 5.3，c-sat に対する性能向上比の幾何平均と最大を表 5.4 に示す．SAT，UNSAT の各項目の数字が幾何平均値，() 内の数字が最大値を示し，SAT + UNSAT では平均値のみを示す．

表 5.3: PrecoSAT に対する性能向上比

	c-preco(31PE)	c-satw(31PE)
SAT(44 題)	2.9 (29.4)	4.1 (62.0)
UNSAT(62 題)	2.3 (4.4)	2.6 (8.7)
SAT + UNSAT(106 題)	2.5	3.1

c-satw では SAT 問題の最大性能向上が 60 倍以上得ることのできる問題もあったが，対象 106 問全体で幾何平均を取ると 3 倍程度に落ち着いた．しかし逐次性能

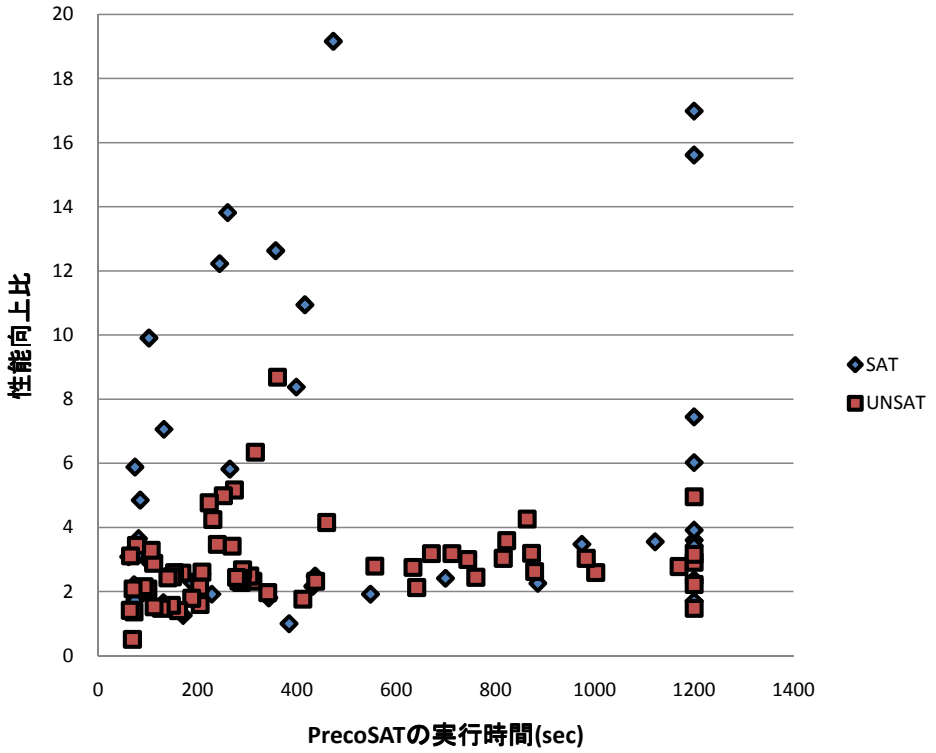

図 5.3: c-satw(31PE) の PrecoSAT に対する性能向上比

で PrecoSAT に劣る MiniSat が 5 台導入されている 31PE 同士の並列 SAT ソルバであるにも関わらず，c-satw は c-preco より全体的に性能が向上していることがわかる．

また，c-satw では c-preco，c-sat のどちらでも解けなかった問題を SAT 問題で 6 題，UNSAT 問題で 2 題多く解くことができた．

5.3.2 ソルバの並列効果

次に，satsuki 上の c-preco の並列効果を表 5.5 に，c-satw の並列効果を表 5.6 示す．ここで，precosat31pe とは precosat をランダムシードだけ変え，lemma を共有せずに 31PE で実行させた結果を示す．

表 5.5 より，c-preco は 7PE で逐次の PrecoSAT より 20 問以上多く解くことができ，総実行時間，総解答数においては lemma を共有しない PrecoSAT31PE の同時実行よりもよい性能を示すことがわかる．また，この PrecoSAT(7PE) よりも PrecoSAT(31PE) で性能が向上することから，並列効果がうまく出ていることがわかる．

表 5.4: c-sat に対する性能向上比

	c-preco(31PE)	c-satw(31PE)
SAT(44 題)	1.1 (10.4)	1.5 (6.8)
UNSAT(62 題)	1.4 (16.5)	1.6 (14.9)
SAT + UNSAT(106 題)	1.3	1.6

表 5.5: c-preco の並列効果

	precosat	precosat31pe	c-preco(7PE)	c-preco(31PE)
solved (SAT99 題)	66	78	78	86
solved (UNSAT178 題)	100	104	111	125
solved (UNK15 題)	0	0	0	0
total	167	182	189	211
総実行時間 (SAT)	50457.4s	39717.3s	42499.4s	35185.1s
総実行時間 (UNSAT)	115877.6s	113865.5s	105634.5s	97660.7s
総実行時間 (UNK)	18000.0s	18000.0s	18000.0s	18000.0s
total	184335.0s	171582.8s	166133.9s	152155.4s

表 5.6: c-satw の並列効果

	precosat	precosat31pe	c-satw(7PE)	c-satw(31PE)
solved (SAT99 題)	66	78	82	91
solved (UNSAT178 題)	100	104	114	125
solved (UNK15 題)	0	0	0	0
total	167	182	196	216
総実行時間 (SAT)	50457.4s	39717.3s	36387.8	26927.9s
総実行時間 (UNSAT)	115877.6s	113865.5s	102814.5s	89882.7s
総実行時間 (UNK)	18000.0s	18000.0s	18000.0s	18000.0s
total	184335.0s	171582.8s	152155.4s	134810.6s

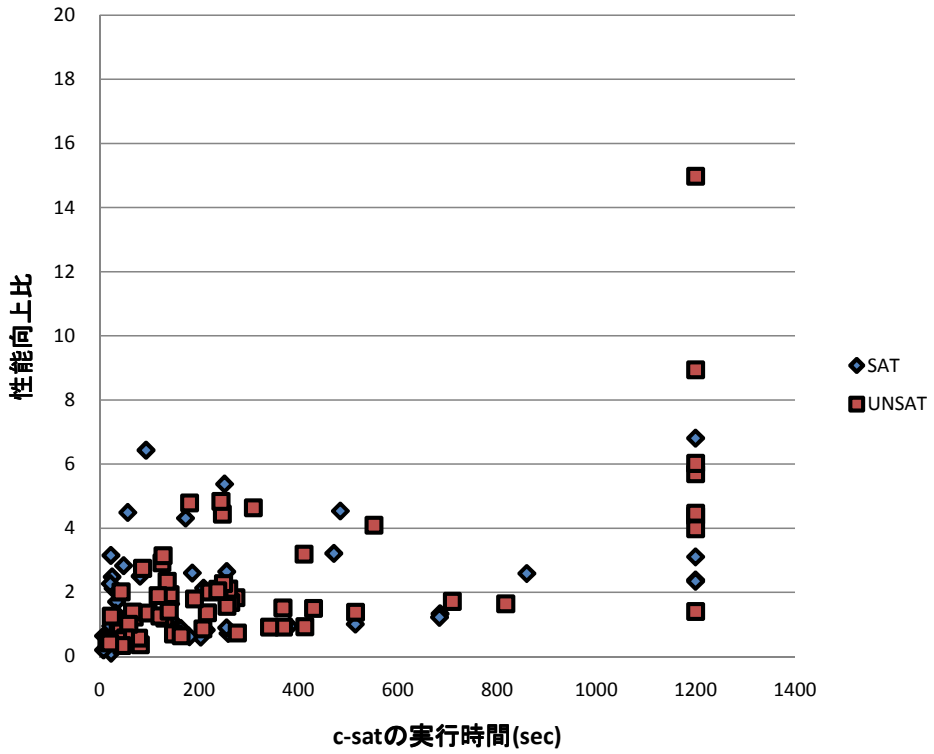

図 5.4: c-satw(31PE) の c-sat(31PE) に対する性能向上比

表 5.6c-satw についても同様で，特に SATISFIABLE の問題で c-preco より問題を解きやすくなっていることがわかる．

5.3.3 propagation 効率

並列化によって lemma 共有のための通信が発生するため，ある程度，本来の解探索の時間が阻害されることが考えられる．PrecoSAT では逐次ソルバで最も時間のかかるとされる BCP をより効率的に行えるよう，構造体の最適化などを行っており，この propagation 回数が，1 秒間に何回行われているか (propagation 効率) で，アルゴリズム以外の面での，ある程度の性能を換算することができる．

この影響度合いを確認するため，1 秒間あたりの propagation 回数を PrecoSAT の解けた問題 166 題に対して，c-preco の PrecoSAT に対する propagation 性能向上比として SAT 問題 66 題を図 5.5 に UNSAT 問題 100 題を図 5.6 に示す．また，具体的な平均値を表 5.7 に示す．

ここでは，縦軸が PrecoSAT に対する c-preco の propagation 回数向上比を示し，横軸において問題が元々の propagation 回数が多い順に左からソートされている．

中央の赤いラインは 1 倍を示し，元の PrecoSAT と同 propagation 毎秒であることを示している．c-satw は PrecoSAT と MiniSat で元々の propagation 効率が違うことから図示，表示の対象外とした．また，それぞれ PE 数が違うため，数値は実際に問題を解いた PE のものを選んだ．

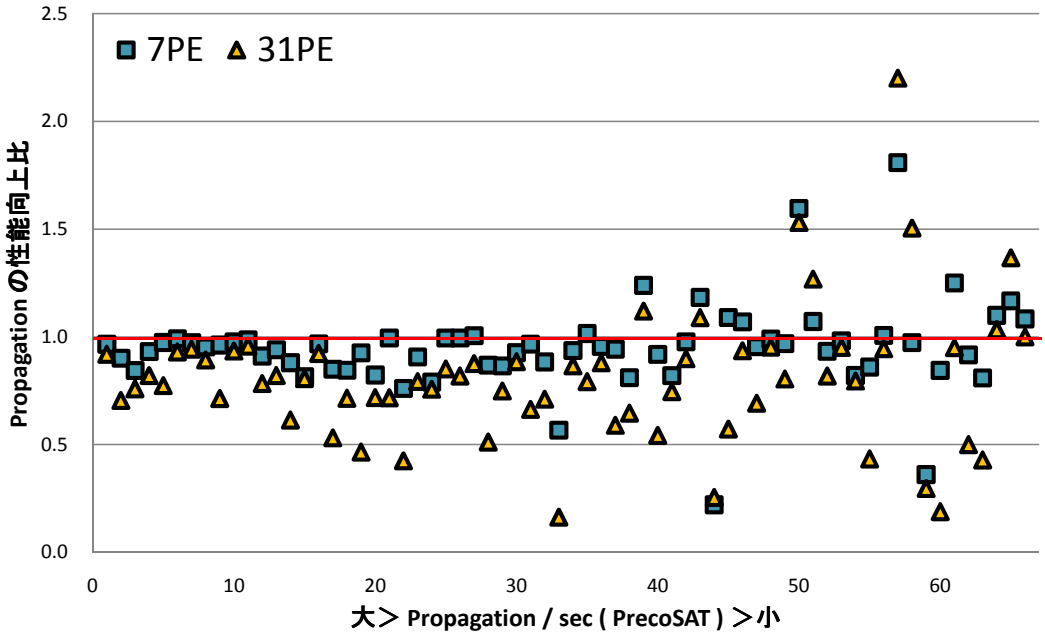

図 5.5: 並列化における propagation 回数向上比 (SAT)

表 5.7: c-preco における propagation 効率

	precosat	c-preco(7PE)	c-preco(31PE)
SAT(66 題)[megaprops/sec]	0.96(2.23)	0.90(2.15)	0.76(2.05)
UNSAT(100 題)[megaprops/sec]	0.93(3.12)	0.82(3.03)	0.76(2.79)
total[megaprops/sec]	0.94	0.85	0.74

表 5.7 において，各項目の数字は単位秒あたりの 100 万 propagation の回数をしめしており () 内は最大値を示している．表 5.7 からは最大値，平均値共に通信によってやや propagation 性能が低下していることがわかる．

図 5.5，5.6 からもおおよそ低下していることがわかるが，UNSAT の問題は比較的全体的に低下していることがわかるが，SAT の問題では，特に元々の propagation 性能がよくない部分に関してはばらつきが大きく，元の propagation の 2 倍以上を示すものもあった．

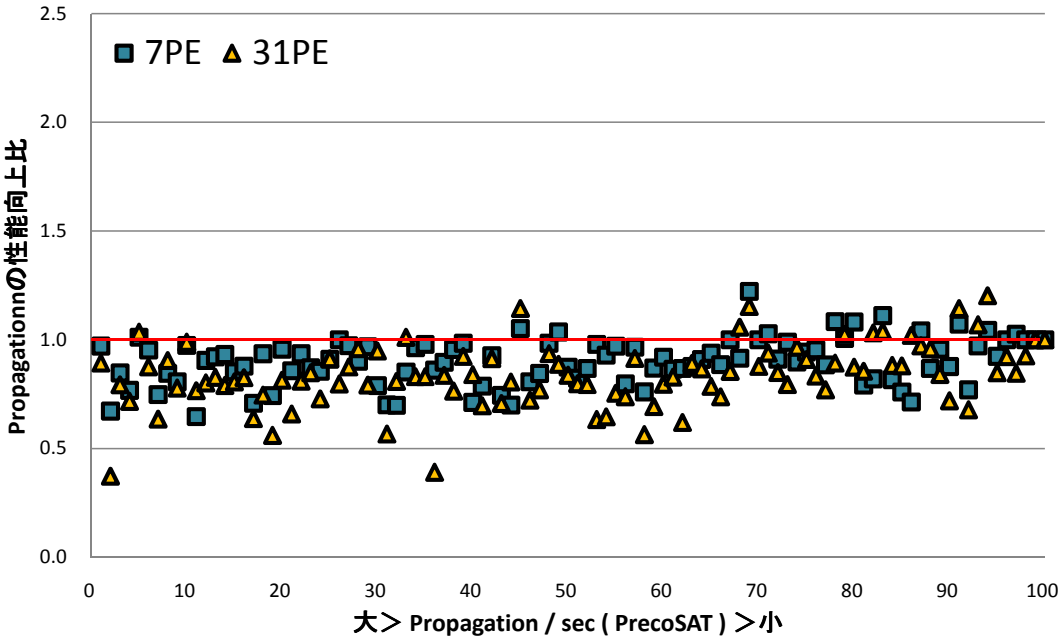

図 5.6: 並列化における propagation 回数向上比 (UNSAT)

特に性能が低下していた問題は，元々10 秒以内程度に解ける高速な問題が多く，並列化によって，全体の実行時間が延びたものが多かった．

逆に，性能が 40%以下に低下した問題において唯一元の PrecoSAT が 100 秒を超えている velev の問題では元の問題より実行時間が 10%ほど増加していた．

第6章　まとめと今後の課題

本章では本研究の成果をまとめ，今後の課題を述べる．

6.1　まとめと考察

c-preco，c-satw の両ソルバに置いて，ベースとなった c-sat，PrecoSAT よりよい性能を示すことができ，PrecoSAT の lemma 共有による並列効果，c-sat に対する PrecoSAT のアルゴリズム導入の効果があったことが確認できた．以下の節ではそれぞれのソルバについて実験結果のまとめと考察を行う．

6.2　c-preco

実験により c-preco は PrecoSAT，c-sat のどちらに対しても性能の向上が確認できた．以下に，実験結果のまとめと考察を述べる．

6.2.1　実験結果のまとめ

本実験によって c-preco はワーカとして動作する PrecoSAT に対して 31PE 実行時に SAT 問題の最大で約 30 倍の性能を示し，幾何平均でも約 3 倍の効果を得ることができた．また，SAT と UNSAT を合わせた値でも幾何平均で 2.5 倍の性能を出しており，SAT，UNSAT どちらの問題に対しても安定して性能を発揮できていることがわかる．

しかし，最大で約 30 倍を示すものの，SAT 問題で大きく性能向上したものが 500 秒以下に多く 500 秒〜1200 秒の間であまり性能が向上していない．また UNSAT は突出してよい性能を見せるものはないが 500 秒を超える多くの問題で 2 倍以上の性能を見せている．

また，評価実験環境上で MiniSat や PrecoSAT に比べよい性能を出している c-sat に対しては，c-sat 上の解答時間が 500 秒以下の問題では性能低下した問題も多かった．しかし，500 秒を超える多くの問題では SAT，UNSAT のほとんどの問題で性能向上しており，ワーカの解探索の逐次性能向上効果が得られていることが確認でき，幾何平均で 30%程度の性能向上が得られた．

SAT2009 に対する MiniSat と PrecoSAT の性能向上比が約 1.6 倍であることを考えると，c-sat に対する c-preco の性能向上比としてはやや低い結果であったと言える．

6.2.2 考察

ある程度探索時間のかかる問題のほとんどで性能向上を得られたことから c-sat で用いられていた lemma 共有による，並列化は最新である PrecoSAT の上でも十分に効果が得られるということがわかった．

特に，探索域の分割を c-preco と同等にし，lemma を共有せずに 31PE で実行した precosat31pe に対して SAT18 題，UNSAT21 題，総実行時間で 2 万秒程度高速に解けることから，lemma による探索域の削減が非常に効果が高いことが確認できた．

しかし，propagation 効率そのものは元の PrecoSAT より低下しており，低下率の大きかった問題においては，性能向上比があまり得られていないことから，lemma の通信量の動的制限のストラテジなどは見直す必要があると考えられる．

並列効果が得られている 31PE 時の propagation 性能向上比平均が元の 0.7 倍程度であるので，どの問題でもこの程度の低下率に抑えることができる動的制限ストラテジを求められれば，より多くの問題で性能を向上させることができるだろう．

また，使用した問題が SAT2009 の問題であり性能向上比としては伸びにくい問題ではあったが c-sat に対しては性能向上比 1.3 倍程度であった．この理由としては，元々の c-sat の持つ lemma 制限ストラテジが，PrecoSAT に対するチューニングとしては不完全であった可能性があると考えられる．

propagation 効率が低下していることからも通信量は，c-preco の負荷となっていることは十分に考えられるため，propagation 効率向上の点と合わせても，ワーカから子マスタへの lemma 送信制限にも階層情報などを利用するなどの工夫が今後の課題と言えるだろう．

6.3 c-satw

実験により c-satw でも PrecoSAT，c-sat の両ソルバに対する性能の向上が確認できた．以下に，その実験結果のまとめと考察を述べる．

6.3.1 まとめ

c-satw では PrecoSAT，c-sat の両ソルバから性能向上を確認できただけでなく，逐次 SAT ソルバとしては PrecoSAT より性能が低い MiniSat をワーカに導入しているにもかかわらず c-preco 以上の性能向上を見せた．特に PrecoSAT に対する性

能向上比としては幾何平均で約 3 倍，SAT の問題において最大で約 62 倍の性能向上を示しており，元の PrecoSAT より 50 題以上多く解くことができた．また，c-preco 同様 PrecoSAT31pe より 4 万秒近く総実行時間を短縮しており，速度向上が得られていることがわかる．c-sat に対しても全体の幾何平均で 1.6 倍程度得られており c-preco よりも性能が向上していると言える．

6.3.2 考察

ほとんどの問題で c-preco より性能が向上したことから並列 SAT ソルバにおけるハイブリッド化は性能向上に効果が高いと言えることが確認できた．

得手不得手を補うだけでなく，c-preco，c-sat のどちらのソルバでも解けなかった問題を SAT6 題，UNSAT1 題多く解いていることから，単純な問題の得手不得手の裨補だけでなく異なるソルバ間で共有された lemma がそれぞれのソルバで効果の高い探索域の剪定を行ったと考えられる．

しかし，c-preco で解けていた問題でも c-satw で解けなくなった問題もあり，UNSAT 全体では c-preco と c-satw の UNSAT 解答数は同じとなっていることから，解答数に限っては多少のバラつきが存在すると言えるだろう．性能向上比としても c-sat，PrecoSAT の両ソルバに対して c-preco より性能が向上していることから，たとえ，多少，性能のが低いソルバであっても，ハイブリッド化のパーツの一部として導入する意義はあると言える．

また，性能向上が c-preco より高かった理由として，ワーカの一部が MiniSat になったことで，c-sat をベースとする lemma 共有ストラテジの効果が高かった可能性も考えられる．

6.4 今後の課題

今回の実装，評価実験により，PrecoSAT の lemma 共有による並列化の効果，及び，c-satw のハイブリッド化の意義を確認をすることができた．しかし，c-preco，c-satw 共に，lemma の送信制限に関する共有ストラテジは従来の c-sat と同じものを利用しており，それぞれの並列 SAT ソルバとしてそのストラテジが最適かどうかを確認できていないため，さらなるチューニングが課題である．特に，c-preco に関しては，PrecoSAT に対しては非常に性能向上したものの c-sat に対してはそれほど性能向上を得られなかったことから，この課題の重要性が高いと考えられる．

c-satw では，この他にもハイブリッド化の影響を追試することが非常に重要だと考えられる．今回は MiniSat:PrecoSAT の比を 1:4 で実験を行い性能向上を得られたが，実験を繰り返し，より最適なハイブリッド比のチューニングを行うことは，今後の課題である．また，異なるソルバが混在することにより，単体のソル

バとして振舞いがどう変化しているかは，MiniSat，PrecoSAT 逐次性能の見直しと言う点に関しても非常に重要な課題だと思われる．

この他にも，今回使用した MiniSat v1.14 でなく近年の多くの SAT ソルバの基となっている MiniSat2.0 の導入や SAT Competition で上位入賞したソルバの導入によるさらなるハイブリッド化とそのフレームワークの整備も今後の SAT ソルバの発展の上で大変重要になってくると考えられる．

第7章　関連研究

本章では，関連研究である他の並列SATソルバについていくつか述べる．現在公開されている並列SATソルバは全て，同一の逐次SATソルバを並列化したものでありハイブリッドSATソルバの研究は関連研究としては存在しない．

7.1　ManySat

Microsoft ResearchがMiniSat2.0をベースとしてC++で4000行程度で実装したマルチスレッドSATソルバである．並列に探索を進める際に探索域が重ならないようにソルバごとにパラメータと変数選択ヒューリスティックの変更を行うと共に，SAT Competition 2009の投稿版であるManySat1.1では衝突節の共有も加えられている．ManySatはこの大会のマルチスレッド部門で第1位を獲得した．

7.2　MiraXT

zChaffをベースとしているSATソルバMiraをマルチスレッド化したSATソルバである．SAT Race2008に出場したソルバで，C++で3400行程度で実装されており，探索木の動的分割とlammの共有を行っている．clauseデータベースにアクセスする際のロックが多く，4スレッドを超えるスレッド数では性能向上があまり得られない．MiraXTをMPIでつないだPaMiraXTの論文[16]が公開されているが，まだソースは公開されていない．

7.3　pMiniSat

MiniSat2.0をベースにguiding pathを用いた探索域の動的分割を行っているマルチスレッドSATソルバである．work stealingを基にした動的分割の高速化を目指しており，clauseを分割したデータ構造を取る[5]など逐次SATソルバとしても高速に動作するように工夫されている．動的分割だけでなくlemma共有ストラテジにもguiding pathに基づいた手法を用いている．

7.4 PMSat

MiniSat をベースに MPI で実装されたクラスタ向け並列 SAT ソルバである．C++ で実装されており，2 層のマスターワーカ構造をによる lemma 共有を行っている．PMSat[7] は c-sat とパラメータチューニングの点で異なっており [15]，c-sat に比べると性能が環境に依存しやすい設定となっている．

謝辞

本研究を進めるにあたり，ご指導を賜りました上田 和紀教授に深く感謝致します．また、研究その他様々な面において，補助していただいた研究室の皆様に深く感謝致します．研究の一部は，科学研究費補助金（特定 18049015）の補助を得て行いました．本研究で，独立行政法人産業技術総合研究所連携検証施設さつきの検証クラスタを利用させて頂き、深く感謝致します．最後に，入学から現在に至るまで支えてくれた家族に感謝致します．

参考文献

[1] SAT 2009, In *http://www.cs.swan.ac.uk/ csoliver/SAT2009/*, 2009.

[2] Audemard, G., Simon, L., Predicting learnt clauses quality in modern SAT solvers, In *Proceedings of the 21st international jont conference on Artifical intelligence*, pp. 399–404, Pasadena, California, USA, 2009. Morgan Kaufmann Publishers Inc.

[3] Biere, A., Pre,icoSAT@SC'09, In *SAT 2009 competitive events booklet*, pp. 41–43, 2009.

[4] Biere, A., Heule, M., Maaren, H.van , Walsh, T., editors, *HANDBOOK of satisfiability*, IOS Press, 2009.

[5] Chu, G., Harwood, A., Stuckey, P. J., Cache Concious Data Structures for Boolean Satisfiability Solvers, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 6, pp. 99–120, 2009.

[6] Een, N., Biere, A., Effective preprocessing in sat through variable and clause elimination, In proc. SAT '05, volume 3569 of LNCS, pp. 61–75. Springer, 2005.

[7] Gil, L., Flores, P., Silveria, L. M., PMSat: a parallel version of MIniSAT, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 6, pp. 71–98, 2009.

[8] Goldberg, E., Novikov, Y., BerkMin: a Fast and Robust Sat-Solver, pp. 142–149, 2002.

[9] Iser, M., MiniSAT 09z for SAT-Competition 2009, In *SAT 2009 competitive events booklet*, pp. 29–30, 2009.

[10] Luby, M., Sinclair, A., Zuckerman, D., Optimal Speedup of Las Vegas Algorithms, *Information Processing Letters*, Vol. 47, pp. 173–180, 1993.

[11] Marques-Silva, J. P., Sakallah, K. A., GRASP - A New Search Algorithm for Satisfiability, 1996.

[12] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S., Chaff: Engineering an Efficient SAT Solver, pp. 530–535, 2001.

[13] Niklas Een, N. S., MiniSat - A SAT Solver with Conflict-Clause Minimization, In *The International Conference on Theory and Applications of Satisfiability Testing*, p. poster for SAT2005, Chalmers University, Sweden, 2003.

[14] Niklas Een, N. S., MiniSat Page, In *http://minisat.se/*, 2006.

[15] 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.

[16] Schubert, T., Lewis, M., Becher, B., PaMiraXT: Parallel SAT Solving with Threads and Message Passing, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 6, pp. 203–222, 2009.

[17] Zhang, L., Malik, S., The Quest for Efficient Boolean Satisfiability Solvers, pp. 17–36. Springer, 2002.

[18] 英和鍋島, 剛秀宋, 高速SATソルバーの原理, Vol. 25, No. 1, pp. 68–76, 2010.

[19] 圭大村, クラスタ向け並列SATソルバの設計と実装, Master's thesis, 早稲田大学, 日本, March 2009.

[20] 克己井上, 直之田村, SATソルバーの基礎, Vol. 25, No. 1, pp. 57–67, 2010.

発表論文

[1] 露崎浩太, 上田 和紀: 並列SATソルバMiraXTの改良：タスク分割時の待機時間削減. 先進的計算基盤システムシンポジウム SACSIS 2008 論文集, pp. 44–45, 2008.

[2] 鈴木　宏, 露崎　浩太, 上田 和紀: SATソルバMiniSatの並列化とそのチューニング手法. 第n回ディペンダブルシステムワークショップ (DSW2009) 論文集，pp. 134–137, 2009.

[3] 露崎　浩太, 上田 和紀: クラスタ向け並列precosatの開発と性能評価. 第72回情報処理学会全国大会投稿中.