JavaによるSATソルバHerrsatの実装と学習節の削減法について

大森, 晋作
九州大学大学院システム情報科学府知能システム学専攻 : 修士課程

松下, 幸之助
九州大学大学院システム情報科学府知能システム学専攻 : 修士課程

長谷川, 隆三
九州大学大学院システム情報科学研究院知能システム学部門

藤田, 博
九州大学大学院システム情報科学研究院知能システム学部門

他

https://doi.org/10.15017/1517816

出版情報：九州大学大学院システム情報科学紀要. 12 (1), pp.33-39, 2007-03-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第12巻　第1号　平成19年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.12, No.1, March 2007

JavaによるSATソルバHerrsatの実装と学習節の削減法について

大森晋作* ・ 松下幸之助* ・ 長谷川隆三** ・ 藤田博** ・ 越村三幸**

Implementation of a SAT Solver Herrsat in Java and a Method for Eliminating Learned Clauses

Shinsaku OMORI, Konosuke MATSUSHITA,
Ryuzo HASEGAWA, Hiroshi FUJITA, and Miyuki KOSHIMURA

(Received December 15, 2006, Revised January 30, 2007)

Abstract: We have developed a SAT solver Herrsat written in Java. Herrsat is based on MiniSat, a state-of-the-art SAT solver written in C++. Most SAT solvers generate 'learned clauses' during solving process in order to prune search spaces. We built a simplification function, which eliminates unnecessary clauses, into Herrsat. We compare the solving speed of Herrsat with that of MiniSat using some typical problems. We also measure the effect of the simplification function.

Keywords: SAT, Minisat, Davis-Putnam, Subsumption, Self-subsuming resolution, Multithread

1. はじめに

SAT(Boolean satisfiability problem)は，与えられたブール式を真とするような変数の値の割当てがあるかどうかを判定する問題である．最初にNP-完全であることがCookにより証明された問題として知られている．計算論や人工知能，ハードウェア設計・検証など，計算機科学の様々な分野で重要視されている問題でもある．また，モデル検査やスケジューリング問題といった実用上重要な問題がSATに還元できることが知られている．

我々は，これまでSAT問題を含むAIを指向した自動推論システムMGTPをJavaにより開発してきた[10),11)]．JavaはC/C++に比べ低速であるが，プラットホーム独立，GUIの作りやすさ，マルチスレッド化の容易さや，動的メモリ管理が不要等の利点を持つ．MGTPは一種のSATソルバであり，これまでに開発したソフトウェア資産を共有/活用できることや，Java-MGTPとの公平な比較ができることから，JavaによるSATソルバHerrsatの開発を進めている．

Herrsatは，世界最高性能を持つとされるSATソルバMiniSatを基に製作されている．Herrsatは，C++で記述されたMiniSatと比べると，おおよそ2.5倍低速である．この差は，主に両者の実装言語であるJavaとC++の性能に起因するものである．当初，この差は5〜6倍であったが，基本データ構造の工夫により，差を2倍以上縮めることに成功した．この工夫は本文にて詳述する．

多くのSATソルバは，CNF(Conjunctive Normal Form)で記述されたSATに適用される．HerrsatもCNFのSATを解く．CNFは節の論理積で，節はリテラルの論理和である．ここでリテラルは命題変数もしくはその否定である．任意のSATは，充足可能性が等しいCNFのSATに変換可能である．そこで，本論文ではSATを節集合とみなすことにする．

最近のSATソルバは，解探索中に学習節（learned clause）を生成し，これを利用した探索空間の刈り込みを行っている．この刈り込み効果は絶大で，近年のSATソルバの開発競争の隆盛は，これによるところが大きい．生成された学習節は，元からある節と同等に扱われるが，その数は探索が進むにつれて膨大になるので，節集合に対するアクセス時間の推論時間への影響が無視できなくなる．

そこで本研究では，不要な節の除去や節の縮退を行い，膨張する節集合の簡約化を行う包摂（subsumption）検査と自己包摂導出（self-subsuming resolution）[4),5)]を導入する．文献4) では推論開始直前に入力節集合に対して簡約化操作を適用するのみであるが，ここでは，学習節の爆発を抑止するため，推論途中で複数回適用する．但し，簡約化アルゴリズムのオーバーヘッドを軽減するために，一定サイズ以下の学習節のみに簡約化アルゴリズムを適用するという基準を新たに設けた．

2. SATソルバHerrsat

HerrsatはNiklas EénとNiklas Sörensson(スウェーデン，チャーマーズ大学)によって製作されたSATソルバMinisat[2)]のバージョン1.14をベースとしている．

Minisatの最大の特徴はソースコードの短かさであり，コメントと空行を除いてわずか600行であるにもかかわら

平成18年12月15日受付，平成19年1月30日 再受付
* 知能システム学専攻修士課程
** 知能システム学部門

ず効率を犠牲にすることなく動作する．ソースコード及び関連の文書はMinisatのWebページ[3]から入手できる．

現版のHerrsatは，2006年現在でのほぼ最新SATアルゴリズムを実装している．オリジナルのMinisatがC++言語で記述されているのに対して，HerrsatはJavaによる実装のため，オリジナルのC++言語に比べると2〜3倍程度実行速度は低下するが，開発と可読性はオリジナルMinisatのC++コードに比べるとかなり容易になっている．

2.1 Herrsatの動作

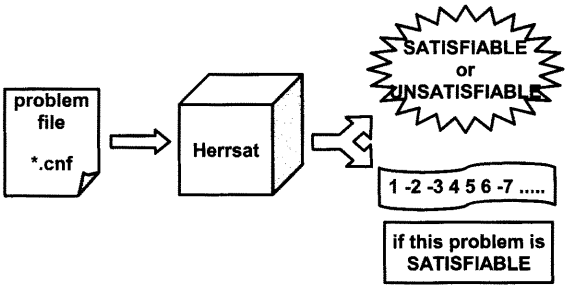

Fig. 1　Operations of Herrsat.

HerrsatはFig. 1のように，問題ファイルを与えられるとその問題がSATISFIABLE(充足している，解が存在する)かUNSATISFIABLE(充足していない，解が存在しない)のいずれかを判定し，SATISFIABLEの場合はその解を出力する．

2.2 CNF式によるSAT問題の表現

Herrsatはブール式の充足可能性問題を解く．ブール式を線形時間／線形サイズでCNF形式に変換するアルゴリズムが知られており[8]，CNFに対する効率の良い決定手続が存在する[9]ので，ここではCNF形式を採用する．尚，CNF形式は殆んどのSATソルバで標準の形式として採用されている．

CNF形式ではブール式を次のようなリテラルの選言の連言で表記する．

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2})$

変数(**Variable**)　x_1, x_2, x_3 などを $1, 2, 3 \cdots$ のように表す．

リテラル(**Literal**)　x_1 か $\bar{x_1}$ のように変数の肯定と否定で現れる項をリテラルと呼び，それぞれ1，-1のように表す．

節(**Clause**)　リテラルの選言．SAT問題は節の連言で表す．

HerrsatではCNF式をFig. 2のような配列で表現している．

$(x_1 \vee x_2 \vee x_3) \wedge$
$(x_1 \vee x_2 \vee \bar{x_3}) \wedge$
$(x_1 \vee \bar{x_2}) \wedge$
$(\bar{x_1} \vee \bar{x_2}) \wedge$
x_3

1	2	3
1	2	-3
1	-2	
-1	-2	
3		

Fig. 2　Internal representation of CNF.

2.3 SATソルバの基本的動作

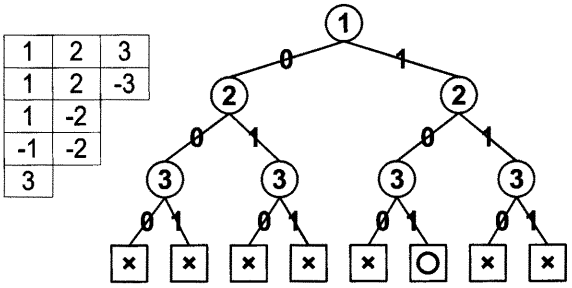

Fig. 3　Naive search of SAT solver.

SATソルバはFig. 3のように，各変数に0(FALSE)か1(TRUE)を割り当ててブール式がTRUEになるかを検査する．可能な全ての割り当てで，ブール式がFALSEならUNSATISFIABLE，一つでもTRUEとなるような割り当てがあればSATISFIABLEである．この方式だとn変数に対して全ての割り当てを確かめてみることになるので，探索空間が深さnの完全2分木になってしまう．この探索空間を削減するために次節以降に示すようなアルゴリズムが存在する．

2.4 Davis-Putnam法

Davis-Putnam法[1]はSATを解く標準的なアルゴリズムで，Fig. 4にそれを基にしたソースコードを示す．

```
while (1) {
 if (decide_next_branch()) //Branching
 {
  while (deduce() == conflict) //Deducing
  {
   blevel = analyze_conflicts();
   if (blevel < 0)
    return UNSATISFIABLE;
   else back_track(blevel); //Backtracking
  }
 }
 else //no branch means all variables got assingned.
  return SATISFIABLE;
}
```

Fig. 4　Davis-Putnam code.

大まかな動作の流れは次のようになる．尚，以下では，

一つのリテラルを除く全てのリテラルが0に割り当てられている節を，Unit Clauseと呼ぶ．

1. まだ値が割り当てられていない変数をdecide_next_branch()で探し，0か1を割り当てる要求を出す．(例. 変数14番に1(True)を割り当てる)
2. deduce()によりdecide_next_branchから出された要求を実際に割り当て推論する．それにより偽になっている節がないか，Unit Clauseがないかを探す．
   - 節が一つでも偽になっている場合はconflictを返し，3.へ．
   - Unit Clauseがあれば，それに現れるリテラルに1を割り当て要求を出し2.に戻る．
   - Unit Clauseが無ければ1.に戻り新しい割り当て要求を出す．
3. analyze_conflict()により偽になってしまった理由を調べ，どこまで推論を戻せばよいかをblevelに格納する．
4. back_track(int blevel)により推論を後戻りする．戻った後は1.からやり直す．また，既に推論の終わった枝にはチェックを付ける．

2.5 学習アルゴリズム

学習アルゴリズム[1]はconflictの状態となった原因を解析し，それに関与するリテラルを全て見つけ，そのリテラル全てを否定した節を作ることである，これを学習節という．例えば，

$(a \vee b \vee c \vee d) \wedge (\bar{a} \vee \bar{b} \vee c \vee d) \wedge (a \wedge b \wedge \bar{c}) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{a} \vee \bar{d}) \wedge (\bar{c} \vee d)$

に対する変数の割り当てが，$(a, b, c) = (0, 0, 0)$ ならば，

$(a \vee b \vee c \vee d)$と$(b \vee c \vee \bar{d})$はdが0であっても1であっても充足しない．そこで2つの節から導出された学習節$(a \vee b \vee c)$を節集合に加え，推論を再開する．この節を加えることにより再び同じ変数の割り当てでconflictが起こることを防ぐことが可能になる．

学習アルゴリズムは多くのSATソルバで実装されており，推論を大幅に高速化する．しかし学習節を推論の途中で節集合に追加していくため，これを管理するコストが高くなっていくという問題に対応しなくてはならない．

Herrsatでは通常，新たな学習節の作成に関与した節に高い点数を与えるというヒューリスティクスを用い，定期的に低得点の節を一定量削除するアルゴリズムを使用する．

2.6 restart戦略

最近のSATソルバで使われる戦略の一つとしてrestart戦略[2]がある．これは推論を進めるうちに変数の選択などが悪く，証明に行き詰った推論をしていると判断した場合，一旦現在の変数の割り当てを全て破棄し，パラメタ設定を変更して最初からやり直すという戦略である．restart戦略により証明に寄与しない推論を行う可能性を減らすことができる．再開の基準として，通常，conflict回数に上限値を設ける．この際に学習した節は残されるためやり直された推論は完全に無駄になるわけではない．

2.7 Herrsatの実装における特徴

HerrsatはJava言語とC++言語の性能差のため，単純な移植の場合5〜6倍程度低速になった．そのためJava言語でボトルネックとなっている要素を解析し，以下の改良を施した．

可変長配列の実装 SATソルバでは節の管理や推論状況の監視のために動的な長さを確保できる可変長配列が必要になる．しかしJava言語の可変長配列ArrayListは非常に低速であるためこれを独自な可変長配列Vecの実装によって高速化している．この可変長配列VecはJavaの標準の可変長配列とは違い内部に固定長配列を保持している．Vecの大きさは自由に変更することができるが，大きさの変更は明示的に行わなければならない．Vecは現在確保している固定長配列を越える要素数を指定された場合，現在の固定長配列を破棄し必要分の固定長配列を確保しなおす．配列はObject[]型で定義してあるため，どのような型でも扱うことができる。

プリミティブ型の利用 Java言語ではint型に対してInteger型を使うと値を参照する際にアドレス参照を行うため低速になってしまう．上記の可変長配列などではint型よりもInteger型のほうがObject型の継承型であるため利用がしやすいのだが，低速になることを避けるためになるべくint型を利用している．同様にByte型はbyte型，Long型はlong型，Double型はdouble型，Float型はfloat型を利用する．前述の可変長配列Vecではオブジェクトの配列しか扱えないため，そのままではint型のオーバーラップ型であるInteger型しかVecで扱うことができない．そのためVec型を継承し，VecInt型やVecFloat型などプリミティブ型に対応する可変長配列クラスを用意してある．コードサイズは増すが，これによって1.25倍程度の速度向上が得られた．

メモリの確保 JavaのJVMはデフォルトで2MBしかメモリ領域を確保しない．この容量ではたびたびメモリ領域が足りなくなり，ガベージコレクションを頻発してしまう．JVMはコマンドラインの引数で確保するメモリの大きさを指定できる．本研究では1GBのメモリ領域を確保して実行している．

コードの分割 高速化には特に寄与しないが，方式比較の融通性のためオリジナルのMinisatのコードを意味単位で分割しクラスとして定義している．Minisat

もある程度クラスで分割してはいるが，Javaの特徴であるオブジェクト指向に強いという要素を活かし，さらに拡張しやすく可読性の高いコードに改良している．

2.8 Minisatとの比較

我々が開発した最新版のHerrsat1.01と，オリジナルのMinisat1.14の性能比較を**Table 1**に示す．

Herrsat1.01とMinisat1.14の実行時間の差は平均しておおよそ2.5倍程度に抑えていることが分かる．一般的にJava言語はC++言語に対して5倍以上遅いと言われることを考慮すると優れた結果といえる．

2.9 Herrsatの速度向上比較

初期のバージョンであるHerrsat1.00と最新版であるHerrsat1.01の性能比較を同じく**Table 1**に示す．

旧版のHerrsat1.00から1.01までの大きな改良点は可変長配列Vecの改良と，プリミティブ型をなるべく利用するようコードを書き換えた点である．この結果速度は大幅に向上し，実行時間1分未満程度の問題の場合は約2倍弱の速度向上がみられた．clauses-4.renamedの実行時間を見ても分かるとおり，実行時間を要する大きな問題ほど速度向上は大きくなるとみられる．

一部のSatisfiable問題では改良後の方が低速な場合があるが，これは変数選択の良し悪しが性能に大きく影響するためだと考えられる．

3. 簡約化アルゴリズム

3.1 subsumption

subsumptionは他の節にsubsumeされている節を取り除く．節Cのリテラルの集合を$l(C)$とする．節C_1とC_2が与えられ，もし$l(C_1) \subseteq l(C_2)$が成り立つなら，C_1はC_2をsubsumeし，C_2はC_1にsubsumeされる．subsumeされた節は冗長であるので取り除くことができる．例えば，2つの節

$C_1 = (a \vee b \vee c)$

$C_2 = (a \vee b)$

においてC_1はC_2にsubsumeされているのでC_1を取り除くことができる．

3.2 self-subsuming resolution

self-subsuming resolutionとは，ある2つの節にresolutionを行い，そのresolventが片方の節をsubsumeするとき，subsumptionを行うことによって簡約化することである．例えば，2つの節

$C_1 = (a \vee b)$

$C_2 = (a \vee \bar{b} \vee c)$

においてこれらをbについてresolutionを行うとresolvent

$C = (a \vee c)$

が生成され，CはC_2をsubsumeする．そのためC_2を削除し，Cを加えることで，節を簡約化することができる．

4. 実　　験

restartごとに簡約化アルゴリズムを適用するのは，高速化が目的である．しかし，現状では，全ての節に対して簡約化アルゴリズムを適用すると，問題によっては，オーバーヘッドのために膨大な時間を要する．そこで適用対象の節を制限することで，効率改善を図る．

4.1 実験の手順

今回の実験では，入力節集合中の最大の節の長さを基準とし，そのn倍のサイズ未満の学習節に対して簡約化アルゴリズムを適用し，n倍以上の学習節を削除する．

$n = 1,\ 1.5,\ 2,\ 2.5,\ 3,\ 3.5,\ 4$

の各場合を比較した．

SATソルバの評価に用いられるベンチマーク問題は，その作られ方によって通常random，craft，industrialの3種類に分類される．random問題は機械的に作られた問題であり，craft問題はラテン方陣等の人工的に作られた問題であり，industrial問題は回路のルーティングやスケジューリング問題といった実世界の問題である．今回はそれぞれ2問ずつを解いた．

- random問題（uuf250-010，uuf250-020）
- craft問題（qg3-09，qg5-13）
- industrial問題
  （clauses-4.shuffled，clauses-6.shuffled）

Table 2に使用する問題の変数数と節数を示す．

Table 2 Benchmark statistics.

Benchmark	Variables	Clauses
uuf250-010	250	1065
uuf250-020	250	1065
qg3-09	729	16732
qg5-13	2197	125464
clauses-4.shuffled	267767	1002957
clauses-6.shuffled	683996	2623082

実行環境は次の通り．

CPU Intel Core 2 Duo E6600 (2.4GHz Dual Core)

Memory DDR2 SDRAM PC6400 2GB

OS Microsoft Windows XP Service Pack 2

Java VM Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_09-b03)

Table 1 Performance comparison of Herrsat and Minisat.

Benchmark	Herrsat 1.00 CPUtime	Herrsat 1.01 CPUtime	Minisat 1.14 CPUtime	Herrsat1.01 ／ Minisat1.14	Herrsat1.00 ／ Herrsat1.01	Result
clauses-2.renamed	11.030	7.891	2.125	3.713	1.398	SAT
clauses-4.renamed	248.750	86.313	31.312	2.757	2.882	SAT
qg2-08	32.880	16.953	4.578	3.703	1.939	SAT
qg3-09	27.340	12.422	4.062	3.058	2.201	UNSAT
qg5-13	57.590	10.219	3.187	3.206	5.636	UNSAT
uf250-0100	8.690	8.984	3.734	2.406	0.967	SAT
uf250-011	11.590	10.609	4.421	2.400	1.092	SAT
uf250-02	9.950	9.141	3.796	2.408	1.089	SAT
uf250-024	12.660	9.625	3.968	2.426	1.315	SAT
uf250-038	3.050	9.375	3.875	2.419	0.325	SAT
uuf250-020	17.750	14.469	5.937	2.437	1.227	UNSAT
uuf250-021	27.830	19.781	8.031	2.463	1.407	UNSAT
uuf250-033	21.770	16.984	6.953	2.443	1.282	UNSAT
uuf250-046	38.750	15.688	6.453	2.431	2.470	UNSAT
uuf250-053	24.920	15.719	6.515	2.413	1.585	UNSAT
uuf250-066	29.080	19.891	8.109	2.453	1.462	UNSAT
uuf250-070	29.190	16.844	6.812	2.473	1.733	UNSAT
uuf250-077	38.170	18.641	7.687	2.425	2.048	UNSAT
uuf250-087	39.060	25.844	10.546	2.451	1.511	UNSAT
uuf250-09	26.330	18.109	7.421	2.440	1.454	UNSAT
Average	35.819	18.175	6.976	2.605	1.971	

4.2 実験結果

実験結果を**Table 3**〜**Table 7**に示す.

restartはrestartが起こった回数，conflictsはconflictが起こった回数，runtimeは簡約化を含んだ実行時間(秒)である．normalはMinisatをもとにしたSATソルバHerrsatをそのまま用いたものである.

Table 3 uuf250-010(UNSATISFIABLE).

	Restart	Conflicts	Runtime
normal	17	149296	8.219
$n = 1$	16	121195	21.938
$n = 1.5$	16	127769	25.688
$n = 2.0$	16	100357	15.516
$n = 2.5$	16	88250	14.266
$n = 3.0$	15	83293	15.141
$n = 3.5$	15	81040	18.328
$n = 4.0$	16	88534	29.172

Table 4 uuf250-020(UNSATISFIABLE).

	Restart	Conflicts	Runtime
normal	18	274683	15.094
$n = 1$	17	163953	32.125
$n = 1.5$	17	170381	34.047
$n = 2.0$	17	165532	31.625
$n = 2.5$	17	151245	33.125
$n = 3.0$	17	154486	39.688
$n = 3.5$	17	134267	45.906
$n = 4.0$	16	130873	48.484

4.3 考察

Table 3〜**Table 6**はUnsatisfiableな問題に対する結果である.

random問題においては，簡約化効果により，conflict回数が減少しており，uuf250-010，uuf250-020のいずれにおいても，$n = 2.5$の時が最小のconflict 数になっている．但し，簡約化オーバーヘッドのために実行時間は2〜3倍に増加している．これは簡約化の素朴な実装のためである．**Table 3**の$n = 4.0$ではconflictの回数はnormalに

Table 5 qg3-09(UNSATISFIABLE).

	Restart	Conflicts	Runtime
normal	14	50666	10.156
$n = 1$	14	53887	10.109
$n = 1.5$	13	35666	6.438
$n = 2.0$	14	46829	12.344
$n = 2.5$	14	38920	17.797
$n = 3.0$	14	41493	22.823
$n = 3.5$	13	38093	19.844
$n = 4.0$	13	32996	36.266

Table 6 qg5-13(UNSATISFIABLE).

	Restart	Conflicts	Runtime
normal	12	22368	9.031
$n = 1$	13	33315	10.859
$n = 1.5$	13	30740	11.391
$n = 2.0$	13	34197	14.688
$n = 2.5$	13	30933	18.813
$n = 3.0$	13	28259	29.016
$n = 3.5$	13	28338	30.172
$n = 4.0$	13	32496	54.235

Table 7 clauses-4.shuffled(SATISFIABLE).

	Restart	Conflicts	Runtime
normal	13	38174	147.484
$n = 1$	12	24830	146.797
$n = 1.5$	15	67314	284.141
$n = 2.0$	12	24510	146.094
$n = 2.5$	15	62057	276.766
$n = 3.0$	14	44021	197.297
$n = 3.5$	15	74247	345.297
$n = 4.0$	14	55204	252.297

Table 8 clauses-6.shuffled(SATISFIABLE).

	Restart	Conflicts	Runtime
normal	21	767783	6457.72
$n = 1$	19	370310	2904.33
$n = 1.5$	16	107347	1014.88
$n = 2.0$	18	202389	1777.91
$n = 2.5$	16	93131	1033.8
$n = 3.0$	17	181782	1671.77
$n = 3.5$	18	210383	2374.77
$n = 4.0$	18	290188	2585.94

比べ9/15位に減少しているので，推論の正味の時間は $8.2 * 9/15 \approx 5$秒程度となる．したがって，実装の工夫により簡約化オーバーヘッドを推論正味時間の5割以下に抑えられれば，normalの実行時間の短縮が見込まれる．

一方，qgのようなcraft問題においては，conflict回数の削減効果は顕著でない．一般に，random問題に関しては冗長な節が多く含まれるので，学習節による探索空間の刈り込み効果は大きいが，craft問題は冗長な節を含まないので，学習節の効果はrandom問題よりも薄い．craft問題で簡約化オーバーヘッドに関わらず，実行時間が短くなったのは以下の理由による．

normalでは，入力節集合サイズの1/3までの個数の学習節を残す．$n = 1$〜2の場合，殆んどの学習節の長さが入力節の最大長のn倍以上なのでこれらは削除され，簡約化オーバーヘッドはなくなる．$n > 2$の場合，生き残る学習節が増え，これによりオーバーヘッドが顕在化する．

Table 7と**Table 8** はSatisfiableなindustrial問題(単解探索)に対する結果である．

Satisfiableな問題の実行時間は，最初の成功枝に到達するまでの時間であるので，一般に，学習節の有無による探索空間の変化，即ち，変数選択順序の変化に影響を受けやすい．clauses-4.shuffledでは，nの値とconnflict回数の増減に関しては相関がみられないが，coflict回数が増えると実行時間も増加する傾向にある．一方，clauses-6.shuffledの場合，全ての$n(= 1$〜$4)$に対して簡約化効果(conflict回数と実行時間の減少) が認められる．特に$n = 2.5$では，normalの6.25倍の実行速度を達成している．本例は，変数の数が数十万，節数が数百万規模の大規模な問題になると，簡約化による推論時間の短縮効果が簡約化オーバヘッドの上昇を上回る，という1つの証左を与えている．

5. モジュール化とマルチスレッド化

現在我々は，SATソルバのモジュール化とマルチスレッド化を行っている．これらはJava言語の特徴を活かし，高速化と同時に，コードの可読性と柔軟性を高める．

5.1 モジュール化

現在のHerrsatは単一の設定でしか動かないが，今後は**Fig. 5**のような各種SATアルゴリズムを適宜組み合わせて同時実行可能な複合的なソルバに拡張する．そのためにはコードのモジュール化を進める必要がある．これにより各種SATアルゴリズムの動的な変更が可能になり，問題に適したアルゴリズムを適用することが可能になる．

5.2 マルチスレッド化

現在のHerrsatは単一のCPUで動作することを前提としたシングルスレッドアプリケーションである．今後の

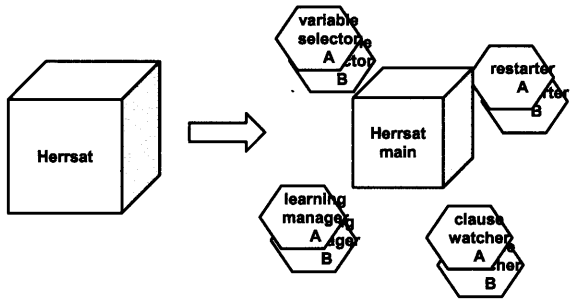

Fig. 5 Herrsat modules.

CPUはIntel Core Duoに代表されるようにマルチコア化が進んでいくものと思われる．2007年には4コアCPUがリリースされる予定でありシングルスレッドアプリケーションでは計算機の力を100%活かしきれない．このため**Fig. 6**のようなマルチスレッドアプリケーション版のHerrsatを開発中である．マルチスレッド版では，複数の推論を同時に進め，各推論を1つのSolverとし，全てのSolverを統括するCoreがこれを管理する．全てのSolverで共有できる情報は全てCoreが管理する．例えば学習節はCoreが管理し，それぞれのSolverは，自己が導いた学習節だけでなく，他のソルバが導いた学習節も利用することができる．これによってより効率的な複数のスレッドによる推論を行うことが出来る．

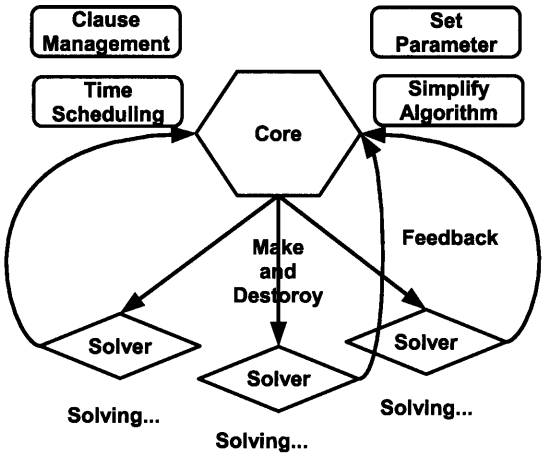

Fig. 6 Multithreaded Herrsat.

6. おわりに

今回製作したJavaによるSATソルバHerrsatは，可変長配列の改良とプリミティブ型を利用した改良により，C++で書かれたオリジナルのSATソルバMinisatの2.5倍程度の実行時間に抑えることができた．

Davis-Putnam法に基づくSATソルバにおいて，学習節は探索空間の削減に多大な効果をもたらす．しかし，その数は推論が進むにつれて増加し，逆に推論速度の低下を招くことも少なくない．

本研究では，包摂検査と自己包摂導出を利用した節の簡約化を導入した．Unsatisfiableなrandom問題では，簡約化による探索空間の削減効果が見られたが，学習節があまり寄与しないcraft問題では，この効果は薄い．Satisfiableなindustrial問題では，実行時間は探索空間の変化に大きく左右される．しかし，大規模な問題では，簡約化効果が支配的になり，実行時間の大幅な短縮が確認された．

以上の結果を踏まえると，SATソルバの総合的な性能向上のためには，問題毎に異なる戦略を適切に選択したり，並列実行するのが効果的である．このため，Herrsatのモジュール化ならびにマルチスレッド化の拡張を現在進めている．

参考文献

1) Lintao Zhang. SAT-Solving: From Davis-Putnam to Zchaff and Beyond. Day1-Day3(スライド), Microsoft Research. http://research.microsoft.com/users/lintaoz/SATSolving/satsolving.htm
2) Eiklas Eén, Niklas Söresson. An Extensible SAT-solver [extended version 1.2], SAT2003.
3) Minisat Page. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/
4) Niklas Een and Armin Biere. Effective Preprocessing in SAT through Variable and Clause Elimination, SAT 2005, LNCS 3569, pp.61-75, 2005.
5) Lintao Zhang. On Subsumption Removal and On-the-Fly CNF Simplication, F.Bacchus and T.Walsh(Eds.): SAT2005, LNCS 3569, pp.482-489, 2005.
6) SATLIB - The Satisfiability Library. http://www.intellektik.informatik/tu-darmstadt.de/SATLIB/
7) SAT Competitions. http://www.satcompetition.org/
8) D.A. Plaisted and S.Greenbaum. A stucture-preserving clause from translation. Journal of Symbolic Computation, 2:293-304, 1986.
9) Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. Communications of the ACM, 5(7):394-397, July 1962.
10) Ryuzo Hasegawa, Hiroshi Fujita, and Miyuki Koshimura. Efficient Minimal Model Generation Using Branching Lemmas, Proc. of 17th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence 1831, pp.184-199, Springer, June, 2000.
11) 長谷川 隆三，藤田 博．Javaによるモデル生成型定理証明系MGTPの開発．情報処理学会論文誌，Vol.41，No.6，pp.1791-1798，2000年6月．
12) 松下 幸之助．JavaによるSATソルバHerrsat-Iの実装，九州大学工学部電気情報工学科 卒業論文，2005.
13) 大森 晋作．JavaによるSATソルバHerrsat-IIの実装，九州大学工学部電気情報工学科 卒業論文，2005.