iSugar : インクリメンタル SAT 解法が利用可能な SAT 型制約ソルバー

迫 龍哉　宋 剛秀　番原 睦則　田村 直之　鍋島 英知　井上 克巳

近年，命題論理の充足可能性判定 (SAT) 問題を解く SAT ソルバーの飛躍的な性能向上を背景に，制約充足問題を SAT に変換し，SAT ソルバーを用いて求解する SAT 型制約ソルバーが成功を収めている．しかしながら，制約最適化問題，解列挙問題などに対しては，SAT ソルバーを複数回起動する必要があり，求解性能が大きく低下することがある．この問題を解決する方法として，インクリメンタル SAT 解法の利用が挙げられる．SAT Race 2015 で，このような解法を容易に実現するためのインクリメンタル SAT API が提案された．本稿では，インクリメンタル SAT API の拡張と，これに対応したインクリメンタル SAT 型制約ソルバーを提案する．提案システムの実現として，iSugar を開発した．iSugar の特長は，上記の求解困難な問題に対して，ただ一度だけ SAT ソルバーを起動し，かつ学習節を再利用することにより，求解性能を大きく向上できる点である．実験の結果，ショップスケジューリング問題，N-クイーン問題，ハミルトン閉路問題に対して，iSugar の有効性を確認した．

Remarkable improvements on SAT solvers over the last decade have led the success of SAT-based constraint solvers. However, existing SAT-based constraint solvers cannot efficiently solve constraint optimization and enumeration problems since those solvers invoke SAT solvers multiple times from scratch and cannot reuse search information such as learnt clauses. Recently, incremental SAT method has been proposed to efficiently solve these problems. In addition, SAT Race 2015 provides its API which enables us to implement general purpose system using incremental SAT method. In this paper, we extend the incremental SAT API and develop the incremental SAT-based constraint solver iSugar. By using the incremental SAT method, iSugar can reduce the invocation overhead and allows us to reuse learnt clauses. To evaluate the efficiency of iSugar, we carried out experiments on shop-scheduling, N-queens, and Hamiltonian cycle problems. In the result, iSugar succeeded in improving computation time and the number of solved instances compared with the existing constraint solver Sugar.

1　はじめに

制約充足問題 (CSP; Constraint Satisfaction Problem) は与えられた制約条件を満たす値の割当てを決定する問題である．人工知能の分野における多くの組合せ問題は CSP として定式化可能であり，CSP を効率よく解くことは重要なテーマとなっている [23] [1] [4] [14]．近年，命題論理の充足可能性判定 (SAT; Boolean Satisfiability Testing) 問題を解く技術が進歩しており，非常に高速に解を求めることのできる SAT ソルバーが実現されている [12] [3]．これによって CSP を一度 SAT 問題に符号化してから高速 SAT ソルバーで解を求め，その解を復号化して元の解を求める SAT 型制約ソルバーと呼ばれるシステムが注目を集めており [26] [17] [24] [10] [18]，そのひとつである Sugar は多

iSugar : A SAT-based Constraint Solver Utilizing Incremental SAT Method

Tatsuya Sako, 神戸大学大学院システム情報学研究科, Graduate School of System Informatics, Kobe University.

Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, 神戸大学情報基盤センター, Information Science and Technology Center, Kobe University.

Hidetomo Nabeshima, 山梨大学大学院医学工学総合研究部, Department of Research Interdisciplinary Graduate School of Medicine and Engineering, University of Yamanashi.

Katsumi Inoue, 国立情報学研究所情報学プリンシプル研究系, Principles of Informatics Research Division, National Institute of Informatics.

くの問題で高い性能を示している [20].

CSP に関連する問題として，制約最適化問題，解列挙問題がある．これらは CSP を繰り返し解く必要があり，一般に CSP よりも求解が困難である．既存の SAT 型制約ソルバーは，これらの問題を解く際には CSP を解く回数と同じだけ SAT ソルバーを起動する．しかしながら，ひとつの問題から生成される各 SAT 問題の大部分は共通の制約であるため，SAT ソルバーは同一の探索空間を何度も調べることになり，求解効率が低下するという問題点がある．

上記のように SAT ソルバーを繰り返し起動することなく，制約最適化問題や解列挙問題を効率的に解くために，**インクリメンタル SAT 解法** [6] の利用が有効であることが知られている．インクリメンタル SAT 解法は有界モデル検査 [2] などに利用されている手法であり，SAT ソルバーが同様の探索失敗を避けるために獲得した学習節を保持することで，無駄な探索を行うことなく，節を追加した SAT 問題を連続的に解くことができる [13] [11]．このため，CSP を繰り返し解く問題群に対して求解性能の向上が期待される．

また最近では，SAT Race 2015[†1] においてインクリメンタル SAT を利用する汎用的な API が公開された．しかし，この API には提供されている機能が低レベルで記述が冗長になる，Java アプリケーションからの利用を考慮していないという問題点がある．そこで，本稿では以下の 2 点を提案する．

1. SAT Race 2015 のインクリメンタル SAT API を高機能化し，また Java アプリケーションからも利用可能にする拡張された API
2. 1. の API に対応し，インクリメンタル SAT 解法が利用可能な SAT 型制約ソルバー

提案システムの実現として，インクリメンタル SAT 型制約ソルバー iSugar を開発した．iSugar の特長は，次の通りである．

- インクリメンタル SAT 解法が利用可能である．
- SAT Race 2015 のインクリメンタル SAT API を実装している高速 SAT ソルバーとの結合が可能である．

iSugar で既存のシステムとの比較実験を行った結果，オープンショップスケジューリング問題では平均 16 倍の高速化に成功した．また N-クイーン問題の解列挙では解の総数が多いほど性能が向上し，ハミルトン閉路問題では求解問題数が 33 問増加した．以上から，iSugar の有効性が確認できた．

本稿の構成は次の通りである．2 節で SAT と CSP について説明し，SAT 型制約ソルバーの問題点を挙げてその解決案を提示する．3 節でインクリメンタル SAT 解法とこれを利用する API について説明し，API の拡張を提案する．4 節で iSugar の設計と実装について述べる．5 節で具体的な問題に対する評価実験の結果を示して考察を行い，6 節で結論をまとめる．

2 SAT 型制約ソルバー

本節では，SAT と CSP について簡単に説明し，SAT 型制約ソルバーとその問題点について述べたのち，問題点を解決する方法の提案を行う．

2.1 SAT

充足可能性判定問題 (SAT) とは，与えられた命題論理式が真となる値の割当ての存在を判定する問題である．そのような値割当てが存在すれば充足可能 (SAT)，しなければ充足不能 (UNSAT) であるという．SAT 問題は通常，連言標準形 (CNF; Conjunctive Normal Form) で与えられる．CNF は節の連言であり，各節はリテラルの選言で表される．また，リテラルは命題変数またはその否定である．

SAT 問題を解くプログラムは SAT ソルバーと呼ばれる．SAT ソルバーは CNF を受け取ってそれが SAT か UNSAT かを判定し，SAT ならその解となる値割当てをひとつ返す．近年の SAT ソルバーには，求解過程で発生する矛盾から獲得した**学習節**を利用し，解を探索する CDCL アルゴリズム [9] などの高速化技術が取り入れられており，その性能は飛躍的に向上している [12]．これに伴い，直接求解することが困難な問題を SAT 問題に変換し，SAT ソルバーを用いて解くという手法が確立されている．

†1 http://baldur.iti.kit.edu/sat-race-2015/

2.2 CSP

制約充足問題 (CSP) とは，与えられた制約式をすべて満たすことができるような変数の値の割当てを決定する問題である．本稿では，整数の有限領域上の CSP を単に CSP と呼ぶ．CSP は，整数変数の有限集合，各変数の取り得る値の集合 (ドメイン)，制約の有限集合によって構成される．

CSP に関連する問題として，与えられた制約を満たしつつ特定の変数の値を最小化または最大化する制約最適化問題 (COP; Constraint Optimization Problem)，制約を満たす値割当てを列挙する解列挙問題などがある．また，CSP として定式化・求解する際に CEGAR (Counter-Example Guided Abstraction Refinement) [5] を用いると効率よく解ける問題もある．例として，ハミルトン閉路問題 (HCP; Hamiltonian Cycle Problem) においては CEGAR 解法の有効性が示されている [16]．これらは，いずれも CSP を繰り返し解くことで求解が可能である．

2.3 SAT 型制約ソルバー

CSP の解を探索するプログラムは制約ソルバーと呼ばれる．その中でも，CSP を一度 SAT 問題に変換 (SAT 符号化) して，SAT ソルバーを用いて元の問題の解を求めるシステムが注目されている．このようなシステムを **SAT 型制約ソルバー**という．代表的な SAT 型制約ソルバーである Sugar は，順序符号化法 [21] と呼ばれる SAT 符号化法を用いており，国際 CSP ソルバー競技会の GLOBAL 部門において，2008-2009 年の 2 年連続で優勝した [22]．また，ショップスケジューリング問題において当時未知であった最適値を決定するなど，多くの問題で成功を収めている [20]．

しかしながら，既存の SAT 型制約ソルバーには前述のインクリメンタル SAT 解法に対応していないという問題点がある．このことについて，Sugar を例に説明する．Sugar は CSP が記述されたファイルを読み込んで SAT 符号化した節をファイルに書き出し，SAT ソルバーを起動する．SAT ソルバーはファイルから CNF を読み込んで解の探索を行い，標準出力にその結果を出力する．これをもとに，Sugar が復号化を行う．このように既存の SAT 型制約ソルバーのほとんどは，SAT ソルバーが本体とは別のプロセスとして動作する疎な結合となっている．この結合方法では，前節で挙げたような CSP を複数回解く必要のある問題の求解において，以下の問題が生じる．

- 学習節をはじめとする，その後の探索に有益な情報を保持できない．
- 同一の探索空間を繰り返し調べるため，探索に無駄が多い．
- プロセス起動，ファイル I/O などの処理が複数回起こり，求解以外の部分におけるオーバーヘッドが大きい．

これらはいずれも，各 SAT 問題を解く度に SAT ソルバーを起動することが原因である．上記の問題点を解決するため，SAT 型制約ソルバーを**インクリメンタル SAT 解法**へ対応させる．

3 インクリメンタル SAT API の提案

本節では，インクリメンタル SAT 解法の概要および SAT Race 2015 より公開されたインクリメンタル SAT API の説明を行い，API の拡張の提案と設計，実装について述べる．

3.1 インクリメンタル SAT 解法

インクリメンタル SAT 解法とは，複数の SAT 問題を連続して効率的に解く手法である．以下で，その内容を説明する．

最新の SAT ソルバーの多くは，求解終了後も結果が SAT であれば，その探索で追加された学習節を次回以降の探索で利用できる．また，仮説 (部分的な真偽値の割当て) を与えて探索することも可能である．これらの機能により，節を追加した問題や仮説を変更した問題を複数回解く場合に，前回の探索において獲得した学習節を再利用できる．その結果，探索空間の枝刈りが行われ，効率よく解を探索することができる．

2.2 節で挙げた問題は，SAT 問題を繰り返し解く問題群である．したがって，インクリメンタル SAT 解法の利点を引き出し，求解時間を大幅に短縮することが期待できる．

表 1　既存 API IPASIR (C/C++)

関数名	動作
`const char * ipasir_signiture()`	インクリメンタル SAT を用いるライブラリの名前とバージョンを返す
`void * ipasir_init()`	新しいソルバーのオブジェクトを作成し，そのポインタを返す
`void ipasir_release(void * solver)`	ソルバーのリソースと確保した領域を解放する
`void ipasir_add(void * solver, int lit_or_zero)`	与えられたリテラルを現在の節に追加する．0 を与えるとその節を閉じる
`void ipasir_assume(void * solver, int lit)`	仮説となるリテラルをソルバーに与える
`int ipasir_solve(void * solver)`	与えられた仮説の下で SAT 問題の求解を行い，SAT なら 10，UNSAT なら 20，探索が中断されたら 0 を返す
`int ipasir_val(void * solver, int lit)`	求解結果が SAT である場合に，指定したリテラルの真偽値を返す
`int ipasir_failed(void * solver, int lit)`	求解結果が UNSAT であった場合に，与えた仮説がその証明に用いられていれば 1，そうでなければ 0 を返す
`void ipasir_set_terminate(void * solver, void * state, int(* terminate)(void * state))`	ソルバーに探索の終了要求を行うためのコールバック関数をセットする

3.2　既存 API

インクリメンタル SAT は注目度の高い技術であり，多くのソルバーが独自のインターフェースによってその機能を提供している．しかしながら，インターフェースの仕様がそれぞれ異なるため，アプリケーションから利用する際の SAT ソルバーの切り替えや各 SAT ソルバーのインクリメンタル SAT の性能比較が難しかった．

国際的な SAT 競技会である SAT Race 2015 では新しい競技部門として Incremental Library Track が導入された [†2]．Incremental Library Track の目的は，多くの異なる SAT ソルバーに共通のインクリメンタル SAT インターフェースを構築し，性能を比較することである．これを実現するため，表 1 に示す汎用的なインクリメンタル SAT API (IPASIR) が提案，公開された．以下，本稿ではこの API を**既存 API** と呼ぶ．

3.3　提案 API

表 1 に示した既存 API は，インクリメンタル SAT 解法を利用するための最低限の機能を提供している．しかしながら，この API は C/C++ からの利用を前提として設計されており，Java からの利用は考慮されていない．また，提供されている機能が少ないために，SAT ソルバーが持つ他の様々な機能を利用できない．加えて，処理がリテラル単位でしか行えないなど機能そのものも低レベルである．

そこで，API の利便性を高めるため，既存 API の拡張を行うことを提案する．本稿では，この拡張した API を**提案 API** と呼ぶ．提案 API の機能を提供する `iSAT Library` の設計方針は，以下の通りである．

- 既存 API の拡張
  既存 API で提供される関数は，そのまま利用可能である．
- 機能の高レベル化
  既存 API より高度なレベルでの記述を可能にする．例えば，リテラル単位でなく節単位での処理を実現する．これにより，ユーザーはより忠実に意図する処理の記述を行うことができ，コードの可読性，記述力が向上する．
- Java および C/C++ 両方の API の提供
  Java からの利用も対象とする．Java 用にはより高度なクラスを用意し，API をラップして動作するようにする．

†2 `http://baldur.iti.kit.edu/sat-race-2015/index.php?cat=rules\#api`

表 2　提案 API (C/C++)

関数名	動作
`int isat_vars(void * solver)`	ソルバーが保持する変数の個数を返す
`void isat_add_clause(void * solver, int[] clause, int length)`	節をまとめてソルバーに追加する
`int * isat_model(void * solver)`	結果が SAT であった場合に，全ての変数に対する真偽値割り当てを返す
`void isat_add_blocking_clause(void * solver, int[] literals, int length)`	得られた解の構成を禁止する否定節をソルバーに追加する
`void isat_freeze(void * solver, int literal)`	指定したリテラルを簡単化の対象から除外する
`int isat_is_eliminated(void * solver, int literal)`	指定したリテラルが簡単化されているかをチェックする

```
 1: int[][] cnf = ...
 2: com.sun.jna.Pointer solver = ...
 3: int maxLit = 0;
 4: for (int[] clause : cnf) {
 5:   for (int lit : clause) {
 6:     ipasir_add(solver,lit);
 7:     maxLit = Math.max(maxLit,Math.abs(lit));
 8:   }
 9:   ipasir_add(solver,0);
10: }
11: while (ipasir_solve(solver) == 10) {
12:   /* output solution */
13:   for (int i=1; i<=maxLit; i++) {
14:     int lit = ipasir_val(solver,i);
15:     ipasir_add(solver,-lit);
16:   }
17:   ipasir_add(solver,0);
18: }
```

図 1　既存 API を用いた解の列挙

```
1: int[][] cnf = ...
2: ISatSolver solver = ...
3: for (int[] clause : cnf)
4:   solver.add_clause(clause);
5: while (solver.solve()) {
6:   solver.model();
7:   /* output solution */
8:   solver.add_blocking_clause();
9: }
```

図 2　提案 API を用いた解の列挙

```
; Input Example  ; Output Example
p cnf 3 4        1 -2 -3
1 2 3 0          -1 -2 3
-1 -2 0          -1 2 -3
-1 -3 0
-2 -3 0
```

図 3　問題と出力の例

3.4　iSAT Library の設計

提案 API で新たに提供する関数の一部を，表 2 に示す．以下で，iSAT Library の設計方針を述べる．

API は C/C++ で記述し，Java から利用する際には，JNA (Java Native Access) を用いて API を呼び出す．これにより，C/C++ と Java の両方から API を利用することが可能になる．Java 側には，JNA で API を利用するためのインターフェースと，より高レベルなラッパークラスを作成する．

API の使用例として，Java から既存 API，提案 API のそれぞれを用いて解の列挙を行うプログラムを図 1，2 に示す．なお図 2 では，提案 API のラッパークラスのメソッドを利用している．入力と得られる解は，図 3 のようになる．以下で，既存 API によるプログラムの流れを説明する．1 行目，2 行目でそれぞれ CNF と SAT ソルバーを与える．3 行目の変数 maxLit は，リテラルの番号のうち最大のもの，すなわち SAT ソルバーに追加された変数の数を表す．4 行目から 10 行目が節の追加処理である．節中の各リテラルに対して追加処理を行い (6 行目)，リテラル番号の更新を行う (7 行目)．全てのリテラルを追加し終えたら，0 を追加してその節が終了したことを SAT ソルバーに知らせる (9 行目)．これを全ての節に対して繰り返す．次に 11 行目から 18 行目で解の列挙を行う．求解結果が SAT なら (11 行目)，解の表示を行った後に (12 行目)，得られた解の構成を禁

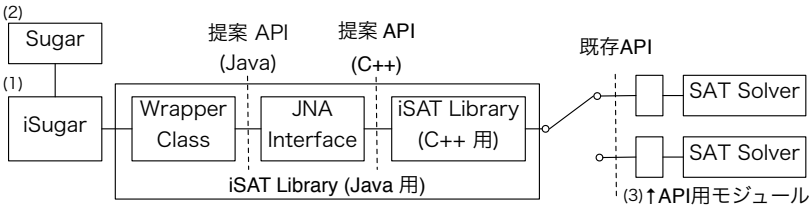

図 4　システム全体の構成

止する否定節を追加する (13〜16 行目)．このときも否定節の最後に 0 を加える必要がある (17 行目)．

このように，既存 API を利用する場合には補助的な変数の導入や，同じ記述の繰り返しなどが生じ，冗長なコードになる．提案 API では，新たな変数などを用いることなく同じ処理を 9 行で行える．また，各処理の内容も分かりやすくなっており，既存 API の問題点を解決している．

3.5　実装上の工夫

JNA による関数呼び出しにはマーシャリング (型変換) などの前処理があり，通常に比べて大きなオーバーヘッドが存在する．そのため，関数の呼び出し回数が多くなるとその処理時間が無視できなくなる．このオーバーヘッドを調べるため，簡単な予備実験を行った．リテラル数が 10 の節 10,000,000 個に対して，既存 API で節の追加処理を行ったところ，全ての節を追加するのに 260.53 秒かかった．そこで，複数の節をまとめて SAT ソルバーに追加する関数を作成した．一度に追加する節数を変えたところ，10 個の場合で 1.65 秒，1000 個の場合で 0.25 秒となり，オーバーヘッドが削減できた．この結果から，Java から利用する場合には節をバッファリングして，一度に複数個を SAT ソルバーに追加するようにした．これらの工夫は，ラッパークラスで実装している．

4　iSugar の設計と実装

2 節および 3 節を踏まえて，本稿では以下の 2 点を目的とし，インクリメンタル SAT 解法の利用によりこれを実現した**インクリメンタル SAT 型制約ソルバー**を提案する．

- 様々な高速 SAT ソルバーと密に結合できる．
- COP，解列挙問題，CEGAR 解法による問題を高速に求解可能である．

本節では，提案システムの実現である iSugar の設計と実装について述べる．

4.1　設計

iSugar では符号化した節を，ファイルを経由することなく `iSAT Library` の関数を通して直接 SAT ソルバーに与え，解も関数の戻り値として直接受け取る．SAT ソルバーは iSugar と同一のプロセスで動作し，求解後も終了せずに情報を保持する．両者の間でファイルアクセスは行わず，節や解などの情報はすべて整数や配列のデータとして受け渡しを行う．CSP の読み込みや SAT 符号化・復号化などの処理は iSugar のメインプログラムから Sugar のメソッドを呼び出すことで行う．

iSugar と SAT ソルバーの結合には，iSAT Library を利用する．C/C++ 側では各 SAT ソルバーに対応する API 用のモジュールを設計する．これによって，API を介していろいろな SAT ソルバーをプラグインできる．API 用モジュールのプログラムは短く，MiniSat 2.2[†3] の場合で 100 行程度である．なお，SAT Race 2015 に提出されたソルバーなど，既存 API に対応しているものであれば，特別な作業を行うことなく iSugar との結合が可能である．

以上から，システム全体の構成は図 4 のようになる．

†3　http://minisat.se/MiniSat.html

4.2 実装

ベースには Sugar 2.2.1，結合する高速 SAT ソルバーには GlueMiniSat 2.2.5 [25](以下 GlueMiniSat) を選択した．実装内容は主として SAT ソルバーを利用するためのプログラムの実装と，各問題の求解を行うためのプログラムの実装に分けられる．以下でそれぞれについて具体的に述べる．

4.2.1 SAT ソルバーとの結合

- Sugar のクラスやメソッドを利用し，与えた問題ファイルを処理するためのメイン部分のプログラムを作成した (図 4-(1))．
- Sugar において節をファイルに書き出す部分の処理を SAT ソルバーに直接追加する処理に変更するため，関連する Sugar のメソッドをオーバーライドした (図 4-(2))．
- GlueMiniSat の仕様に合わせて，API 用のモジュール部分を実装した．(図 4-(3))．
  これには，SAT Race 2015 が提供している，MiniSat 2.2 に対応する API 実装のプログラムを参考とした．

4.2.2 種々の問題への対応

2.2 節で取り上げた COP，解列挙問題，CEGAR を用いた問題について，iSugar に実装した求解アルゴリズムを以下に示す．COP については，Sugar および iSugar それぞれの求解手順を図を用いて説明する．解列挙問題，CEGAR を用いた問題については，iSugar での手順のみを簡単に記す．

(COP)

ここでは，目的変数 v $(l \leq v \leq u)$ を最小化する COP を扱う．

目的変数の値の絞り込みの方法には，二分探索や線形探索などがある．インクリメンタル SAT では，SAT の判定に比べて時間を要する UNSAT の判定回数は極力少ないほうがよい．二分探索では多くの場合に少ない探索回数で解が求まるが，最適値未満の範囲での探索を 2 回以上行うことがあり，その場合には UNSAT を同じ回数判定しなければならない．一方で，線形探索では最適値未満の範囲での探索が最後の 1 回だけなので UNSAT の判定は 1 回でよいが，探索回数が多くなってしまう．

```
begin
制約を SAT 符号化し SAT 問題 Ψ を生成
Ψ を ファイルに出力
while Dom(v) が空でない do
    if Dom(v) のサイズが k 以上 then
        d := ⌊(l + u)/2⌋
    else
        d := u − 1
    end if
    v ≤ d を表す節をファイルに追加
    SAT ソルバーを起動して解く
    if SAT then
        s := 求まった v の値
        Dom(v) から s 以上の値を削除
    else
        Dom(v) から d 以下の値を削除
    end if
end while
s を最適値として出力
end
```

図 5 Sugar による COP の求解アルゴリズム

```
begin
(iSugar の起動とともに SAT ソルバーが起動)
制約を SAT 符号化し SAT 問題 Ψ を生成
Ψ を SAT ソルバーに追加
while Dom(v) が空でない do
    if Dom(v) のサイズが k 以上 then
        d := ⌊(l + u)/2⌋
    else
        d := u − 1
    end if
    ipasir_assume(solver, (v ≤ d を表すリテラル))
    ipasir_solve(solver)
    if SAT then
        s := 求まった v の値
        Dom(v) から s 以上の値を削除
        ipasir_add(solver, (v ≤ s を表すリテラル))
    else
        Dom(v) から d 以下の値を削除
        ipasir_add(solver, (v > d を表すリテラル))
    end if
end while
s を最適値として出力
end
```

図 6 iSugar による COP の求解アルゴリズム

そこで iSugar では，はじめは二分探索を行い，ドメインの幅があるしきい値 k 以下となるところから線形探索に切り替える**混合探索**を採用した．

混合探索による Sugar と iSugar の COP 求解アルゴリズムを図 5，6 に示す．iSugar ではループ内で SAT ソルバーの起動がなく，学習節を保持したまま

表 3　OSS の実験結果 (単位：秒)

問題	Sugar	iSugar	問題	Sugar	iSugar	問題	Sugar	iSugar
j6-per0-0	283.33	88.81	j7-per0-0	69041.28	29946.84			
j6-per0-1	111.56	1.57	j7-per0-1	228.09	43.20	j8-per0-1	43397.22	18574.38
j6-per0-2	19.86	4.95	j7-per0-2	156.18	24.64	j8-per0-2	204.76	126.51
j6-per10-0	94.91	3.25	j7-per10-0	359.69	40.98	j8-per10-0	1011.20	296.53
j6-per10-1	94.63	1.87	j7-per10-1	287.05	3.67	j8-per10-1	5852.87	386.66
j6-per10-2	115.01	2.37	j7-per10-2	976.82	367.81	j8-per10-2	5432.23	2859.95
j6-per20-0	106.40	1.97	j7-per20-0	173.05	1.28	j8-per20-0	409.46	5.19
j6-per20-1	124.13	2.12	j7-per20-1	348.01	10.85	j8-per20-1	442.77	3.45
j6-per20-2	170.83	2.39	j7-per20-2	287.49	21.62	j8-per20-2	431.00	4.94

2 回目以降の探索を行う．

(解列挙問題)

1. 問題を読み込み，SAT 符号化した節を SAT ソルバーに追加する．
2. SAT 問題を解き．UNSAT なら終了する．
3. 得られた解を禁止する否定節を SAT ソルバーに追加して 2. へ戻る．

(CEGAR を用いた問題)

1. 元の問題から緩和問題を生成し，SAT 符号化した節を SAT ソルバーに追加する．
2. SAT 問題を解き．UNSAT なら終了する．
3. 得られた解が元の問題の制約を満たしているかどうかを確認する．満たしている場合，それを解として終了する．
4. その解を禁止する否定節を追加して 2. へ戻る．

5　性能評価

実装した iSugar を用いて性能評価の実験を行った．CPU 時間は Linux Ubuntu 14.04(Xeon 3.00GHz, メモリ 16GB) 上で計測した．

5.1　COP に対する評価

COP としてオープンショップスケジューリング問題 (OSS; Open Shop Scheduling Problem) を扱う．OSS のベンチマーク問題の中でも，j* 問題は Sugar が解くまで未解決であった問題を含んでおり，求解困難な問題として知られている．その中から特に難解な j6，j7，j8 の 26 問について，Sugar と iSugar でそれぞれ求解した結果の CPU 時間を表 3 に示す．探索手法を切り替える k の値は 100 とした．

表 4　N-クイーン問題の実験結果 (単位：秒)

N	解の総数	Sugar	iSugar
4	2	0.50	0.30
5	10	0.49	0.49
6	4	0.51	0.52
7	40	1.12	0.50
8	92	1.66	0.49
9	352	7.14	0.90
10	724	52.70	2.42
11	2680	(T.O.)	13.31
12	14200	(T.O.)	62.90
13	73712	(T.O.)	(T.O.)

実験の結果，26 問全てにおいて求解時間の短縮に成功した．26 問の速度比の幾何平均は 16.41 となり，iSugar は Sugar の 16 倍程度高速であることが分かった．これより，OSS ではインクリメンタル SAT 解法が有効であることが示された．

5.2　解列挙に対する評価

解列挙問題として N-クイーン問題を扱う．Sugar は解列挙に対応していないため，インクリメンタル SAT 解法を用いずに解く機能を iSugar に実装した．具体的には SAT ソルバーを別プロセスとして呼び出して問題を解き，解を得るたびに CNF ファイルにその解を除去する制約を追加して再度問題を解く手続きを繰り返すことで解列挙を行う．本稿では，このシステムを Sugar と表記する．両システムにおける CPU 時間を比較したものを表 4 に示す．問題のサイズは

$4 \leq N \leq 13$，制限時間は 3600 秒とした．(T.O.) は制限時間内に求解できなかったことを表す．

表 4 の通り，解の総数が大きくなるにつれて iSugar の求解時間が Sugar に比べて小さくなっていくことがわかり，$N = 10$ で 21 倍，$N = 11$ で 270 倍以上の高速化に成功した．また，$N = 13$ に関しては，3600 秒以内に Sugar が 6303 個，iSugar が 73712 個全ての解を発見した (iSugar は他に解がないことを判定できず終了した)．この結果から，解が多いほどインクリメンタル SAT 解法の利点が引き出されていることが確認できた．

5.3 CEGAR に対する評価

CEGAR を用いる問題としてハミルトン閉路問題 (HCP) を扱う．CEGAR による HCP の定式化および符号化には論文 [16] の方法を用いた．iSugar で解く際には得られた解に部分閉路が存在するかどうかを調べる必要があるため，頂点や辺などのグラフの情報を保持し，閉路の抽出などを行うクラスを新たに作成した．また Sugar はこの解法にも対応していないため，解列挙と同様に SAT ソルバーと疎に結合して動作する機能を iSugar に実装した．

ベンチマーク問題には COLOR04[†4] の 119 問を用いた．問題をグラフの頂点数により分類し，それぞれの問題数および Sugar，iSugar が解けた問題数を表 5 に示す．制限時間は 500 秒とした．反復回数は SAT ソルバーが解を探索した回数，単位求解時間は求解に要した CPU 時間を反復回数で割ったものであり，平均は各ソルバーが解けた問題に対する値である．なお，各問題についての詳細な結果は付録にまとめる．

表 5 の通り，全 119 問のうち Sugar が解けたのは 71 問であったのに対し，iSugar では 104 問の求解に成功した．差分の 33 問は全て，Sugar で解けず iSugar で解けた問題である．Sugar と比べて，iSugar は頂点数が多い問題でもよく解けていることが分かる．また反復回数では iSugar が Sugar の約 1/5 になっており，より早く解を発見できた．反復回数の減少は学習節を保持することによるものだと考えられるが，正確な原因については調査を行う必要がある．また，両方で解けた 71 問について，両ソルバーでの求解時間のスキャッタープロットを図 7 に示す．図 7 より，規模が大きい問題ほど速度比の改善がみられ，iSugar が大幅に求解性能を向上していることが分かる．なお，71 問での iSugar の平均反復回数は 14.00，単位求解時間平均は 0.01 秒であった．

これより，インクリメンタル SAT 解法の利用により各探索回の求解時間を平均的に短縮でき，大規模な問題に対する性能が向上できたことが示された．

表 5　HCP の実験結果

頂点数	問題数	Sugar	iSugar
1〜100	26	26	26
101〜500	60	41	**52**
501〜	33	4	**26**
合計	119	71	**104**
平均反復回数		114.00	23.00
単位求解時間平均 (秒)		0.28	0.27

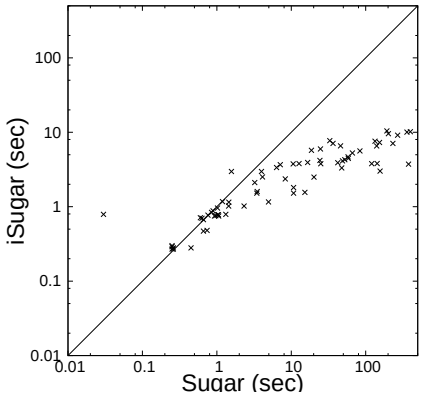

図 7　両方が解けた問題の求解時間比較

6 まとめと今後の課題

本稿では，SAT Race 2015 で公開されたインクリメンタル SAT API をより高機能化し，Java アプリケーションからの利用に対応するよう拡張した API として，`iSAT Library` の提案を行った．また，SAT ソルバーを繰り返し起動することなく，制約最適化

†4 `http://mat.gsia.cmu.edu/COLOR04/`

問題や解列挙問題を効率的に解くことを目的として，インクリメンタル SAT 解法が利用可能な SAT 型制約ソルバーを提案した．提案システムとして `iSAT Library` に対応した iSugar を設計・実装し，インクリメンタル SAT 解法が有効であると考えられる問題群に対して既存のシステムとの性能比較を行い，以下の結果を得た．

- OSS
  26 問全てで求解時間を短縮し，Sugar と比べ平均で 16 倍高速に求解できた．
- N-クイーンの解列挙
  $N = 10$ で 21 倍，$N = 11$ で 270 倍以上の高速化に成功した．
- HCP の CEGAR 解法
  119 問中 104 問を解き，Sugar より 33 問多く求解できた．

以上から，インクリメンタル SAT 解法を利用することにより全体的な性能改善が達成でき，提案したシステムの有用性が示された．

今後の研究課題には，求解方法の見直しによる性能の向上， API のさらなる拡張による機能の充実などが挙げられる．提案 API で追加する機能の案としては，極小の充足不能な節集合の抽出 [8] [15] や，擬似ブール制約の取り扱い [7] [19]，ソルバーの並列化などが考えられる．

参 考 文 献

[1] Barták, R.: On-line Guide to Constraint Programming, `http://kti.ms.mff.cuni.cz/~bartak/constraints/`, 1998.

[2] Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y.: Symbolic Model Checking without BDDs, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999), LNCS 1579*, 1999, pp. 193–207.

[3] Biere, A., Heule, M., van Maaren, H., and Walsh, T.(eds.): *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications (FAIA), Vol. 185, IOS Press, 2009.

[4] Bordeaux, L., Hamadi, Y., and Zhang, L.: Propositional Satisfiability and Constraint Programming: A Comparative Survey, *ACM Computing Surveys*, Vol. 38, No. 4(2006).

[5] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H.: Counterexample-Guided Abstraction Refinement, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, Emerson, E. A. and Sistla, A. P.(eds.), Lecture Notes in Computer Science, Vol. 1855, Springer, 2000, pp. 154–169.

[6] Eén, N. and Sörensson, N.: Temporal Induction by Incremental SAT Solving, *Electronic Notes in Theoretical Computer Science*, Vol. 89, No. 4(2003).

[7] Eén, N. and Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 2, No. 1-4(2006), pp. 1–26.

[8] Liffiton, M. H. and Sakallah, K. A.: Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints, *J. Autom. Reasoning*, Vol. 40, No. 1(2008), pp. 1–33.

[9] Marques-Silva, J. P. and Sakallah, K. A.: GRASP: A Search Algorithm for Propositional Satisfiability, *IEEE Transactions on Computers*, Vol. 48, No. 5(1999), pp. 506–521.

[10] Metodi, A. and Codish, M.: Compiling finite domain constraints to SAT with BEE, *TPLP*, Vol. 12, No. 4-5(2012), pp. 465–483.

[11] 鍋島英知: SAT によるプランニングとスケジューリング, 人工知能学会誌, Vol. 25, No. 1(2010), pp. 114–121.

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

[13] Nabeshima, H., Soh, T., Inoue, K., and Iwanuma, K.: Lemma Reusing for SAT based Planning and Scheduling, *Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS 2006)*, 2006, pp. 103–112.

[14] Rossi, F., Beek, P. v., and Walsh, T.: *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA, 2006.

[15] Ryvchin, V. and Strichman, O.: Faster Extraction of High-Level Minimal Unsatisfiable Cores, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, Sakallah, K. A. and Simon, L.(eds.), Lecture Notes in Computer Science, Vol. 6695, Springer, 2011, pp. 174–187.

[16] Soh, T., Le Berre, D., Roussel, S., Banbara, M., and Tamura, N.: Incremental SAT-Based Method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem, *Logics in Artificial Intelligence*, Springer International Publishing, 2014, pp. 684–693.

[17] Soh, T., Tamura, N., and Banbara, M.: Scarab: A Rapid Prototyping Tool for SAT-Based Constraint Programming Systems, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July*

8-12, 2013. Proceedings, Järvisalo, M. and Gelder, A. V.(eds.), Lecture Notes in Computer Science, Vol. 7962, Springer, 2013, pp. 429–436.

[18] Stojadinovic, M. and Maric, F.: meSAT: multiple encodings of CSP to SAT, *Constraints*, Vol. 19, No. 4(2014), pp. 380–403.

[19] Tamura, N., Banbara, M., and Soh, T.: Compiling Pseudo-Boolean Constraints to SAT with Order Encoding, *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, IEEE Computer Society, 2013, pp. 1020–1027.

[20] 田村直之, 多賀明子, 番原睦則, 宋剛秀, 鍋島英知, 井上克巳: ショップ・スケジューリング問題の SAT 変換による解法, スケジューリング・シンポジウム *2007* 講演論文集, 2007, pp. 97–102.

[21] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2(2009), pp. 254–272.

[22] Tamura, N., Tanjo, T., and Banbara, M.: System Description of a SAT-based CSP Solver Sugar, *Proceedings of the 3rd International CSP Solver Competition*, 2008, pp. 71–75.

[23] 田村直之, 丹生智也, 番原睦則: 制約最適化問題と SAT 符号化, 人工知能学会誌, Vol. 25, No. 1(2010), pp. 77–85.

[24] Tanjo, T., Tamura, N., and Banbara, M.: Azucar: A SAT-Based CSP Solver Using Compact Order Encoding, *Theory and Applications of Satisfiability Testing – SAT 2012*, Cimatti, A. and Sebastiani, R.(eds.), Lecture Notes in Computer Science, Vol. 7317, Springer Berlin Heidelberg, 2012, pp. 456–462.

[25] 鍋島英知, 岩沼宏治, 井上克巳: GlueMiniSat 2.2.5: 単位伝搬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, コンピュータ ソフトウェア, Vol. 29, No. 4(2012), pp. 4_146–4_160.

[26] 田村直之, 丹生智也, 番原睦則: SAT 変換に基づく制約ソルバーとその性能評価, コンピュータ ソフトウェア, Vol. 27, No. 4(2010), pp. 183–196.

A HCP の実験結果

HCP の実験結果詳細を表 6 に示す.

表 6　HCP の実験結果 (単位：秒)

Instance	Sugar	iSugar	Instance	Sugar	iSugar	Instance	Sugar	iSugar
1-FullIns_3	0.93	0.75	DSJC500.1	268.93	9.15	mulsol.i.3	(T.O.)	(T.O.)
1-FullIns_4	10.74	1.52	DSJC500.5	(T.O.)	24.79	mulsol.i.4	(T.O.)	(T.O.)
1-FullIns_5	(T.O.)	6.23	DSJC500.9	(T.O.)	46.48	mulsol.i.5	(T.O.)	(T.O.)
1-Insertions_4	1.44	1.02	DSJR500.1	(T.O.)	4.29	myciel3	0.45	0.28
1-Insertions_5	155.88	3.01	DSJR500.1c	(T.O.)	48.85	myciel4	0.74	0.48
1-Insertions_6	(T.O.)	74.89	DSJR500.5	(T.O.)	24.99	myciel5	2.32	1.02
2-FullIns_3	0.03	0.79	fpsol2.i.1	(T.O.)	8.33	myciel6	8.31	2.36
2-FullIns_4	142.51	3.81	fpsol2.i.2	(T.O.)	(T.O.)	myciel7	120.52	3.79
2-FullIns_5	(T.O.)	135.47	fpsol2.i.3	(T.O.)	(T.O.)	qg.order100	(T.O.)	(T.O.)
2-Insertions_3	0.66	0.47	games120	15.22	1.56	qg.order30	(T.O.)	13.55
2-Insertions_4	10.74	1.81	homer	0.60	0.71	qg.order40	(T.O.)	30.28
2-Insertions_5	(T.O.)	24.67	huck	0.26	0.28	qg.order60	(T.O.)	122.55
3-FullIns_3	4.95	1.16	inithx.i.1	(T.O.)	(T.O.)	queen10_10	10.62	3.76
3-FullIns_4	(T.O.)	4.25	inithx.i.2	(T.O.)	(T.O.)	queen11_11	7.12	3.67
3-FullIns_5	(T.O.)	(T.O.)	inithx.i.3	(T.O.)	(T.O.)	queen12_12	12.70	3.80
3-Insertions_3	0.66	0.67	jean	0.25	0.27	queen13_13	58.44	4.46
3-Insertions_4	47.59	3.32	latin_square_10	(T.O.)	136.05	queen14_14	48.78	4.13
3-Insertions_5	(T.O.)	(T.O.)	le450_15a	36.43	7.08	queen15_15	83.54	5.62
4-FullIns_3	3.49	1.60	le450_15b	1.02	0.96	queen16_16	45.96	6.57
4-FullIns_4	(T.O.)	16.00	le450_15c	397.81	10.18	queen5_5	0.76	0.77
4-FullIns_5	(T.O.)	(T.O.)	le450_15d	(T.O.)	9.93	queen6_6	1.45	1.15
4-Insertions_3	1.03	0.78	le450_25a	231.74	7.09	queen7_7	3.44	1.52
4-Insertions_4	(T.O.)	15.20	le450_25b	153.24	7.29	queen8_12	6.35	3.36
5-FullIns_3	20.21	2.50	le450_25c	358.79	10.05	queen8_8	4.11	2.51
5-FullIns_4	(T.O.)	124.32	le450_25d	192.61	10.42	queen9_9	3.98	2.97
abb313GPIA	(T.O.)	22.66	le450_5a	18.68	5.73	school1	0.85	0.84
anna	0.25	0.29	le450_5b	24.66	5.97	school1_nsh	0.89	0.88
ash331GPIA	0.63	0.70	le450_5c	133.36	7.59	wap01a	(T.O.)	54.86
ash608GPIA	1.19	1.17	le450_5d	32.94	7.72	wap02a	(T.O.)	58.37
ash958GPIA	1.57	2.97	miles1000	24.25	4.19	wap03a	(T.O.)	167.95
david	0.25	0.30	miles1500	66.24	5.27	wap04a	(T.O.)	179.07
DSJC1000.1	(T.O.)	20.91	miles250	0.26	0.27	wap05a	(T.O.)	19.12
DSJC1000.5	(T.O.)	104.70	miles500	24.90	3.78	wap06a	(T.O.)	20.26
DSJC1000.9	(T.O.)	360.99	miles750	376.39	3.73	wap07a	(T.O.)	42.26
DSJC125.1	3.23	2.11	mug100_1	1.06	0.75	wap08a	(T.O.)	46.10
DSJC125.5	16.62	3.93	mug100_25	1.05	0.78	will199GPIA	(T.O.)	7.05
DSJC125.9	140.48	6.55	mug88_1	1.32	0.79	zeroin.i.1	57.97	4.68
DSJC250.1	52.47	4.28	mug88_25	0.97	0.76	zeroin.i.2	(T.O.)	(T.O.)
DSJC250.5	201.99	9.58	mulsol.i.1	42.20	3.90	zeroin.i.3	(T.O.)	(T.O.)
DSJC250.9	(T.O.)	13.66	mulsol.i.2	(T.O.)	(T.O.)			