高速SATソルバーの原理

鍋島 英知
nabesima@yamanashi.ac.jp

山梨大学

湊離散構造処理系プロジェクトERATO セミナー

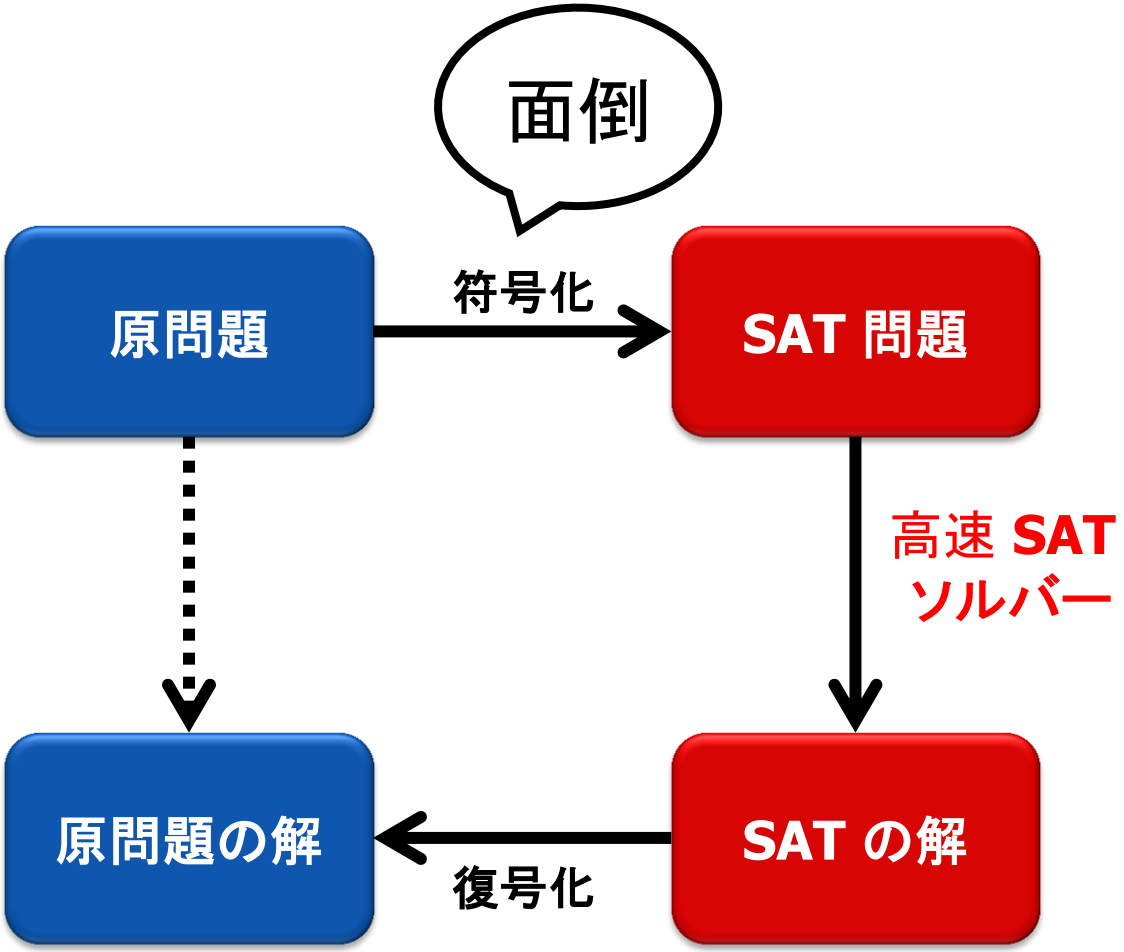
はじめに

- **SAT** = 命題論理式の充足可能性を判定する問題
  - 計算機科学における最も基本的で本質的な組合せ問題

- 90年代末頃からSAT ソルバーの性能が飛躍的に向上
  - 数百万変数からなる大規模な問題が求解可能に

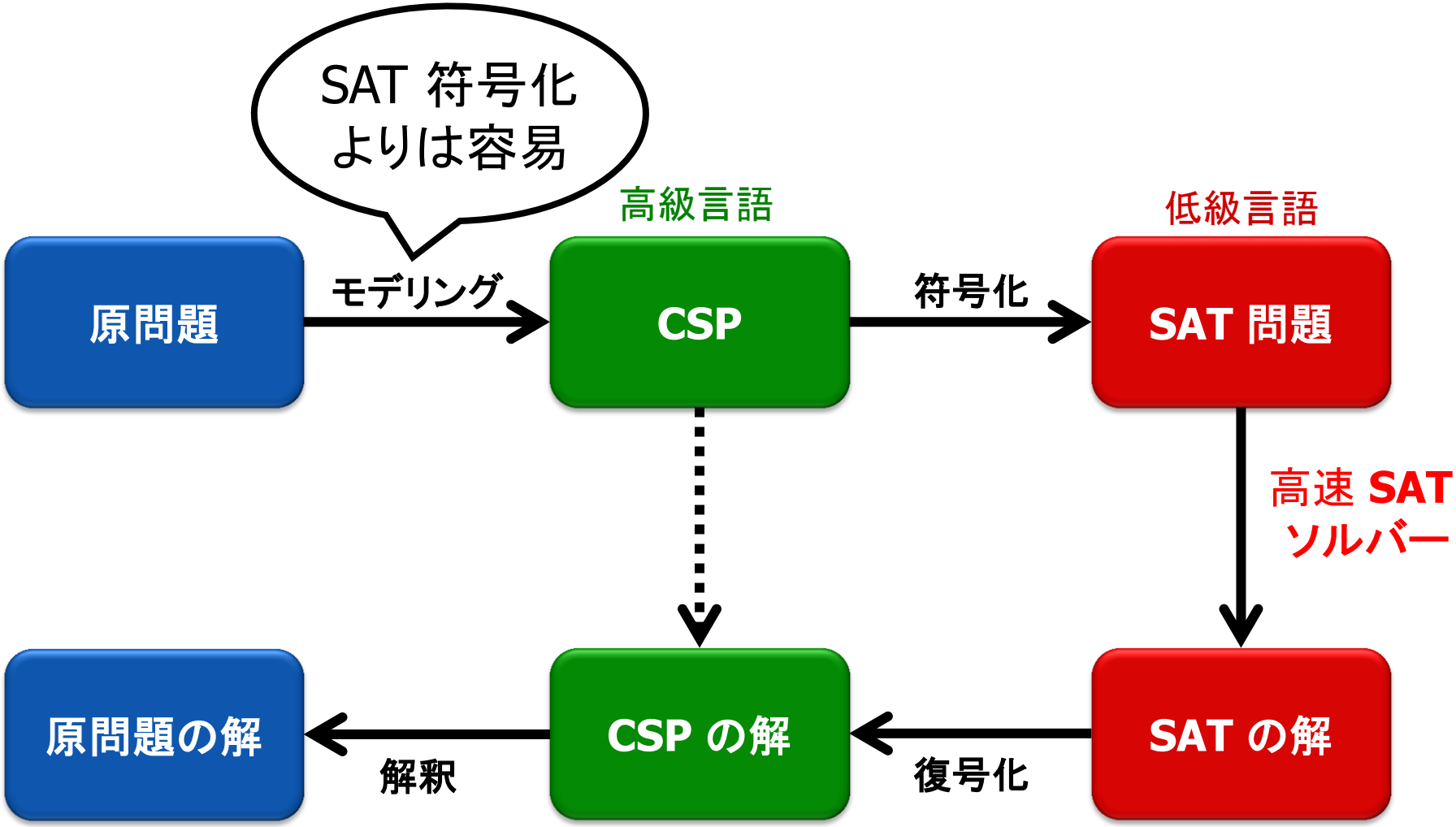
- システム検証，プランニング，スケジューリング，定理証明，制約充足問題など様々な分野で SAT が活用

SAT を利用した問題解決手法



解決策の1つとして

SAT 型 CSP ソルバーを利用



CSPSAT プロジェクト

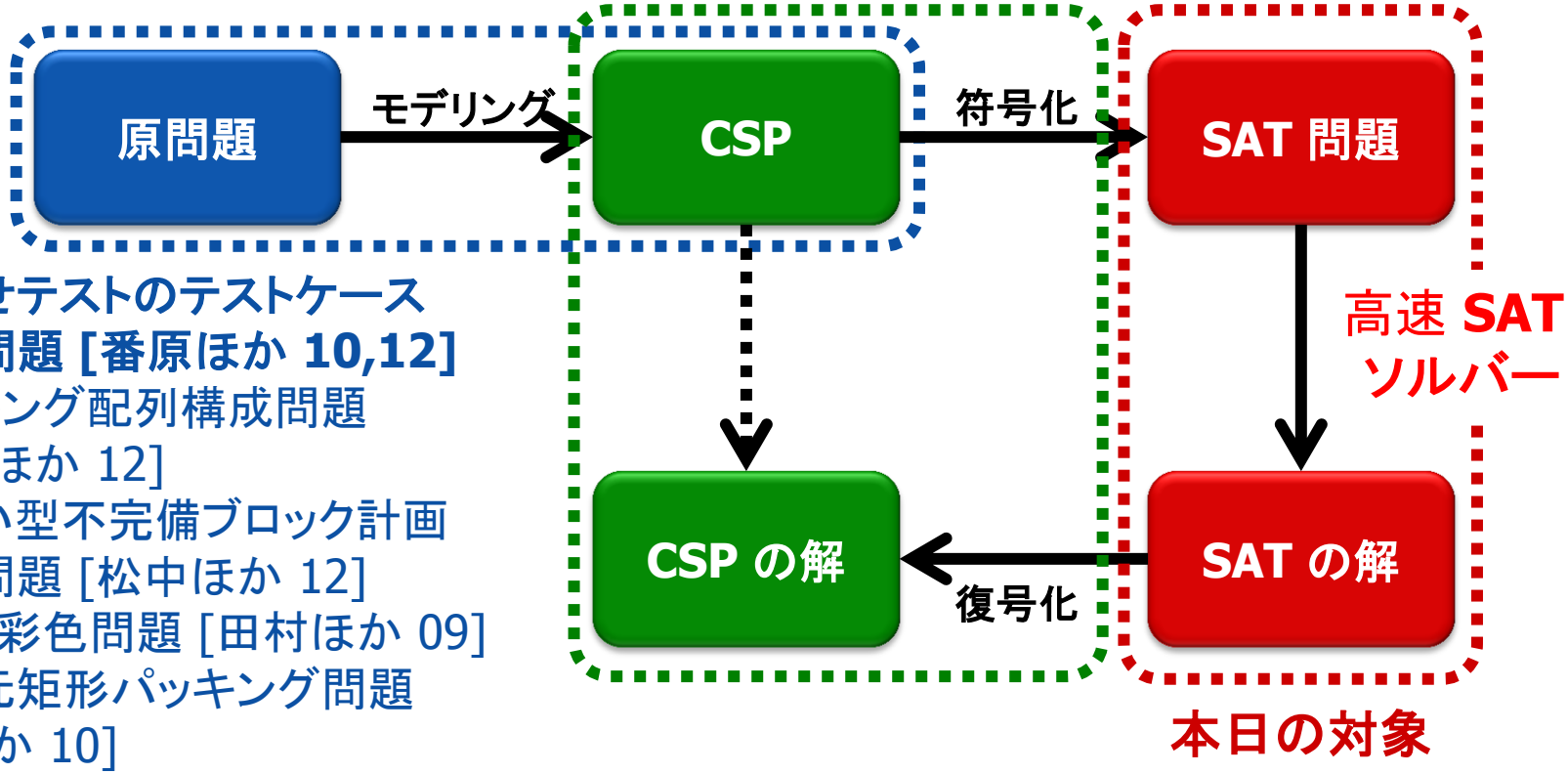
基盤A「制約最適化問題のSAT 変換による解法とその並列分散処理に関する研究」2008-2011

Sugar [Tamura+ 06]

- ✓ 整数有限領域上の CSP ソルバー
- ✓ 2008, 2009 年の CSP ソルバー競技会のグローバル制約部門のすべてで優勝

GlueMiniSat [Nabeshima+ 11]

- ✓ 学習節評価尺度 LBD に基づく SAT ソルバー
- ✓ SAT 2011 競技会の産業応用部門 CPU 時間 UNSAT クラス優勝



- ✓ **組合せテストのテストケース生成問題 [番原ほか 10,12]**
- ✓ パッキング配列構成問題 [則武ほか 12]
- ✓ 釣合い型不完備ブロック計画構成問題 [松中ほか 12]
- ✓ グラフ彩色問題 [田村ほか 09]
- ✓ 二次元矩形パッキング問題 [宋ほか 10]
- ✓ ジョブショップ・スケジューリング問題 [越村ほか 10]

目次

- SAT とは？
- 高速 SAT ソルバーの要素技術
  - アルゴリズム
  - ヒューリスティクス
  - 実装技術
- 高速 SAT ソルバーの理論的側面
- GlueMiniSat の紹介

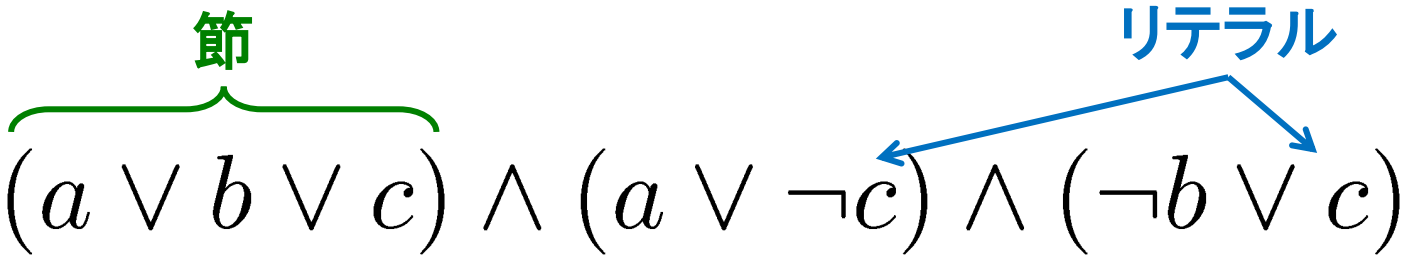
SAT とは？

- 命題論理の充足可能性判定 (Boolean satisfiability testing)
- 与えられた命題論理式を真にする値割当てが存在するか否かを判定する問題
  - 充足可能な真偽値割当が存在する: **SAT（充足可能）**
  - 充足可能な真偽値割当が存在しない: **UNSAT（充足不能）**
- 理論上も実際上も計算機科学の中心的課題
- 最初に NP 完全性が証明された問題 [Cook 71]
  - クラス NP に属する問題は，多項式時間で SAT に変換可能
  - NP 完全問題の例

    グラフ彩色の決定問題，ハミルトン閉路問題，素因数分解，生産・配送などのスケジューリング，時間割作成，レジスタ割り付け，テストケースの生成，数独，...

SAT と CNF

- SAT 問題は通常，連言標準形 (CNF; Conjunctive Normal Form) で与えられる
  - **CNF 式** は，複数の節の連言 (AND)
  - **節 (clause)** は，複数のリテラルの選言 (OR)
  - **リテラル (literal)** は，命題変数またはその否定



しばしば CNF 式をリテラル集合（節）の集合で表記

CNF 式への変換

- 任意の論理式は CNF 式に変換可能
  - ド・モルガン，分配律等により，論理的同値な式に変換可能．ただし，CNF 式が指数関数的に長くなる可能性がある
  - Tseitin 変換を利用すれば，充足同値な線形倍の CNF 式に変換可能．ただし新しい命題変数が必要

例： $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$ DNF 式

- 分配律では 2^n 個の節が必要

$$(z_1 \vee z_2 \vee \cdots \vee z_n) \quad (z_i \in \{x_i, y_i\},\ 1 \leq i \leq n)$$

- Tseitin 変換では $3n+1$ 個の節と n 個の新しい命題変数が必要

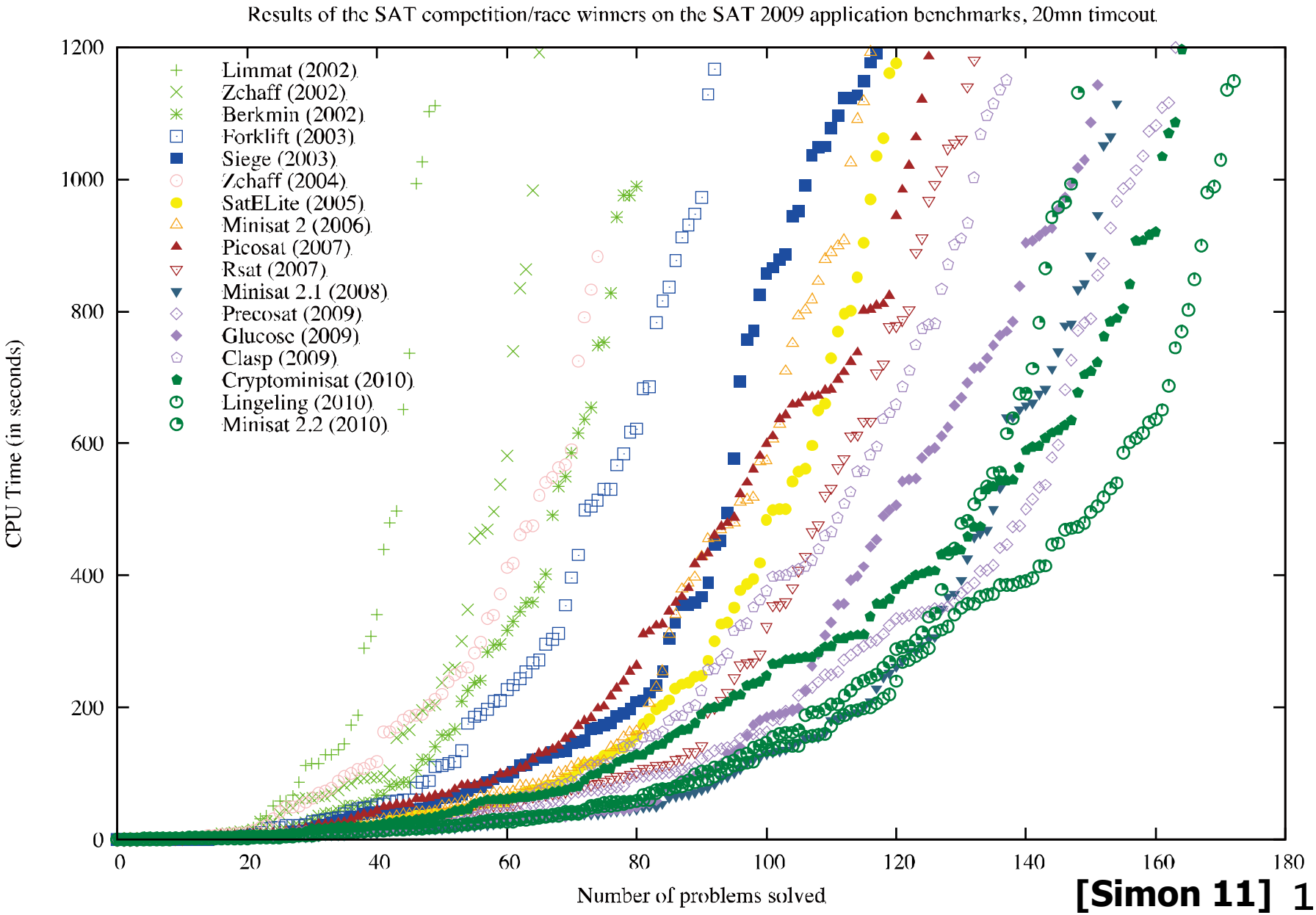
$$(p_i \leftrightarrow x_i \wedge y_i) \qquad (1 \leq i \leq n)$$
$$(p_1 \vee \cdots \vee p_n)$$

SAT ソルバー

- SAT 問題の充足可能性を判定するソフトウェア
  - 充足可能 (SAT) な場合，真偽値割当 を返す
  - 充足不能 (UNSAT) な場合，UNSAT を出力
    - ソルバーによっては，（極小の）充足不能な節集合を返す
    - Haifa-MUC [Ryvchin+ 11]（SAT 2011 競技会 MUS トラックで優勝）

- 多くのソルバーがインクリメンタル SAT に対応
  - 最適化問題において複数の SAT 問題を解く場合に，学習節の再利用を可能にする
  - ある仮説（部分真偽値割当）の下で SAT 問題を解く
  - その仮説を簡単化に利用しないため，仮説を変更した問題を解く場合に学習節の再利用が可能

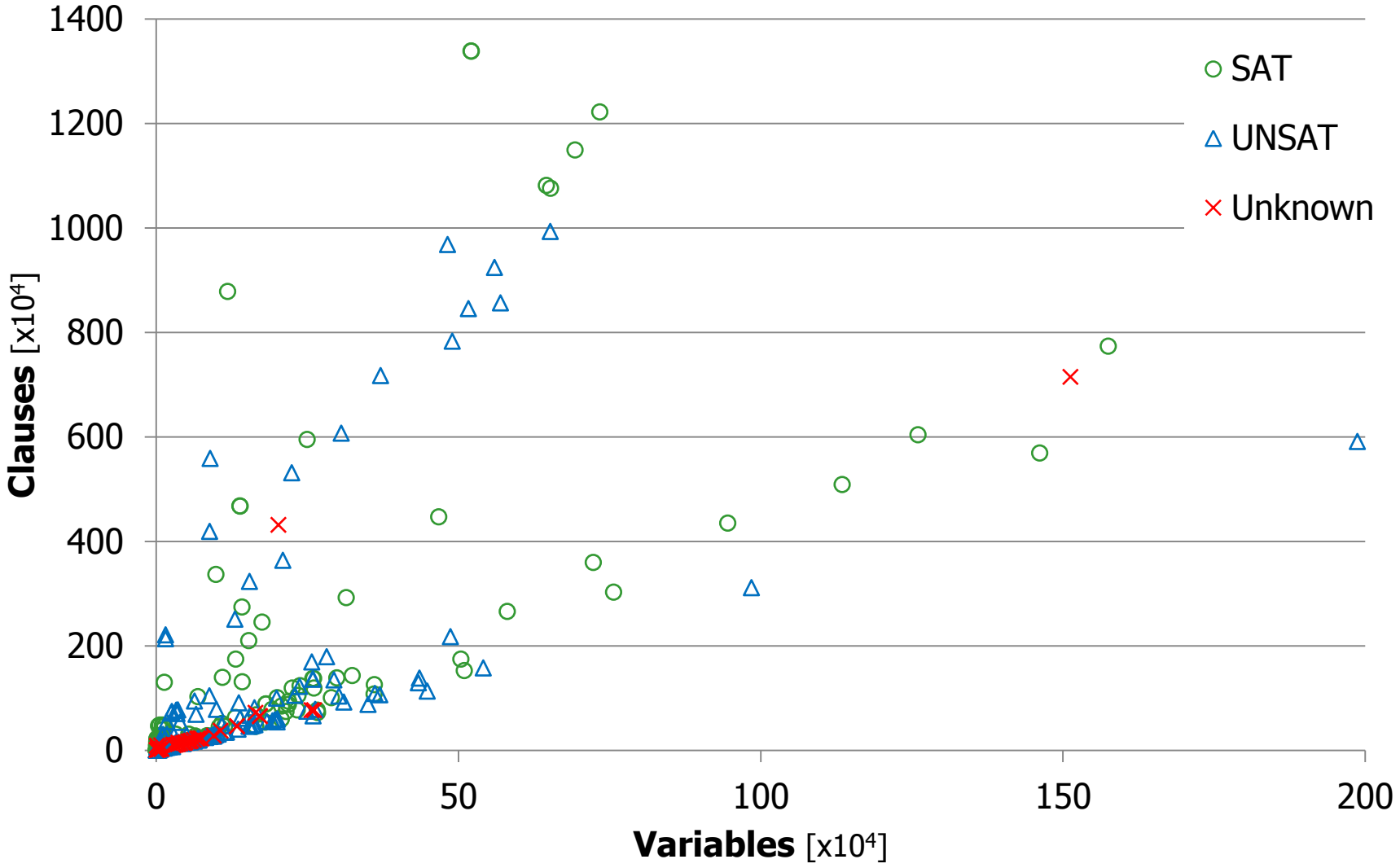
SAT ソルバーの性能向上の推移



[Simon 11]

実応用問題の大きさ

SAT 2011 競技会 Application 部門 300 問



実応用問題の大きさ

SAT 2011 競技会 Application 部門 300 問

	SAT		UNSAT		UNKNOWN	
	#V	#C	#V	#C	#V	#C
Min	170	1559	264	1,476	912	3,344
Max	10,950,109	32,632,955	1,987,351	5,908,940	1,512,832	7,144,473

※ Min, Max は，変数の数が最小または最大の問題を表す

個人的な意見

通常の PC では，変数の数は百万程度，節数は一千万程度が目安

MiniSat [Eén+ 03]

- 代表的 SAT ソルバー
  - 最新 SAT 技術の多くが（美しく）実装されている
  - SAT 2005 競技会では，MiniSat が Application 部門のすべてのクラスで優勝（CNF 式を簡単化するプリプロセッサを併用 ）
  - http://minisat.se/
- SAT コミュニティに拡張容易なソルバーを提供することが開発の目的
  - ソースコードは読みやすく，拡張しやすい設計
  - 多くの SAT ソルバーが MiniSat をベースに開発
  - 最初のバージョンは C++ で 600 行程度（コメント除く）
  - 最新版（2.2）は 3000 行程度（コメント除く）

個人的な意見 C++ の手本としても参考になる

目次

- SAT とは？
- 高速 SAT ソルバーの要素技術
  - アルゴリズム
  - ヒューリスティクス
  - 実装技術
- 高速 SAT ソルバーの理論的側面
- GlueMiniSat の紹介

SAT ソルバー

- 90年代末に技術的ブレイクスルーがあり，性能が飛躍的に向上
  - 矛盾からの節学習とバックジャンプ法 [Silva+ 99, Bayardo+ 97]
- 最新 SAT ソルバーは数百万変数の問題を扱うことが可能
- 最新 SAT ソルバーの要素技術
  - DPLL 手続き [Davis+ 62]
  - 矛盾からの節学習とバックジャンプ法 [Silva+ 99, Bayardo+ 97]
  - 監視リテラルによる高速単位伝搬 [Moskewicz+ 01]
  - 優れた変数選択ヒューリスティクス [Moskewicz+ 01]
  - リスタート戦略 [Gomes+ 98, Luby+ 93]
  - 部分解のキャッシング [Pipatsrisawat+ 07]
  - 充足済節の高速判定 [Jain+ 07][Schubert+ 07][Sorensson+ 08]
  - 学習節の優れた評価尺度 [Audemard+ 09, Nabeshima+ 11]

高速 SAT ソルバー

アルゴリズム

- 矛盾からの節学習とバックジャンプ
- 部分解キャッシング

ヒューリスティクス

- 変数選択戦略
- リスタート戦略
- 学習節評価尺度

実装技術

- 監視リテラル
- 充足済節の高速判定
- キャッシュミスヒットの低減

DPLL 手続き

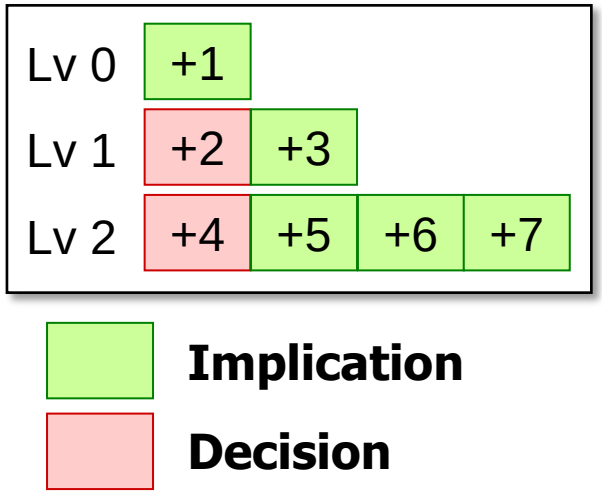
DPLL 手続き [Davis+ 62]

(1) 単位節（リテラルが1個の節）があれば，それを充足する
(2) 単位節がなければ，適当な変数を選択し真または偽を割り当てる
(3) 矛盾が発生した場合，最後に選択した変数の真偽値を反転する

{+1}　← 単位節
{+1,+2,+4}
{+2,+9}
{-1,+4,+9}
{-2,+3}　← 単位節
{-2,-5,+6}　← 単位節
{-1,-4,+5}　← 単位節
{-5,-3,+7}　← 単位節
{-6,+8}
{-7,-8}

Conflict!

Decision Stack



変数選択ヒューリスティクスにより2を選択し，真とする

変数選択ヒューリスティクスにより4を選択し，真とする

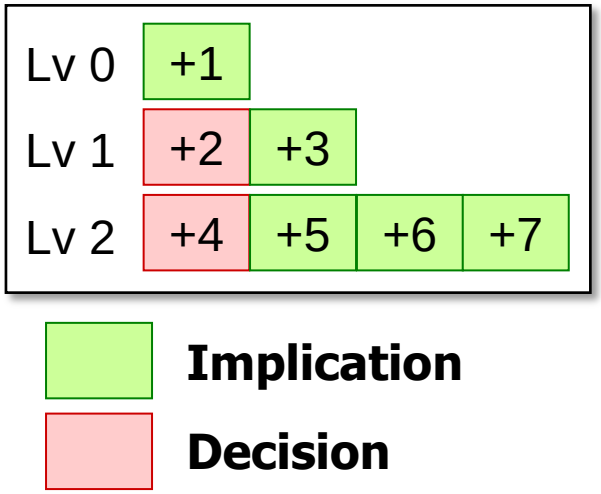
DPLL 手続き [Davis+ 62]

(1) 単位節（リテラルが1個の節）があれば，それを充足する
(2) 単位節がなければ，適当な変数を選択し真または偽を割り当てる
(3) 矛盾が発生した場合，最後に選択した変数の真偽値を反転する

{+1}
{+1,+2,+4}
{+2,+9}
{-1,+4,+9}
{-2,+3}
{-2,-5,+6}
{-1,-4,+5}
{-5,-3,+7}
{-6,+8}
{-7,-8}

Conflict!

Decision Stack



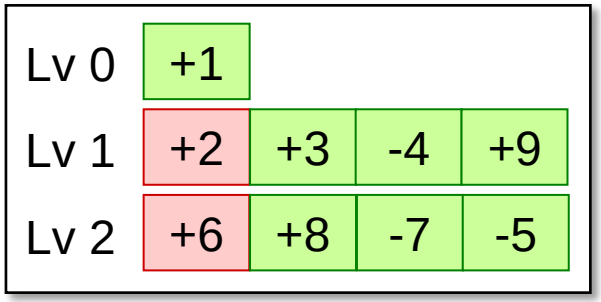
+4 の選択が間違いであったので +4 以降の推論を取消

DPLL 手続き [Davis+ 62]

(1) 単位節（リテラルが1個の節）があれば，それを充足する
(2) 単位節がなければ，適当な変数を選択し真または偽を割り当てる
(3) 矛盾が発生した場合，最後に選択した変数の真偽値を反転する

{+1}
{+1, +2, +4}
{+2, +9}
{−1, +4, +9}
{−2, +3}
{−2, −5, +6}
{−1, −4, +5}
{−5, −3, +7} ← 単位節
{−6, +8} ← 単位節
{−7, −8} ← 単位節

Decision Stack



変数 +4 に対し，
逆の真偽値割当を試みる

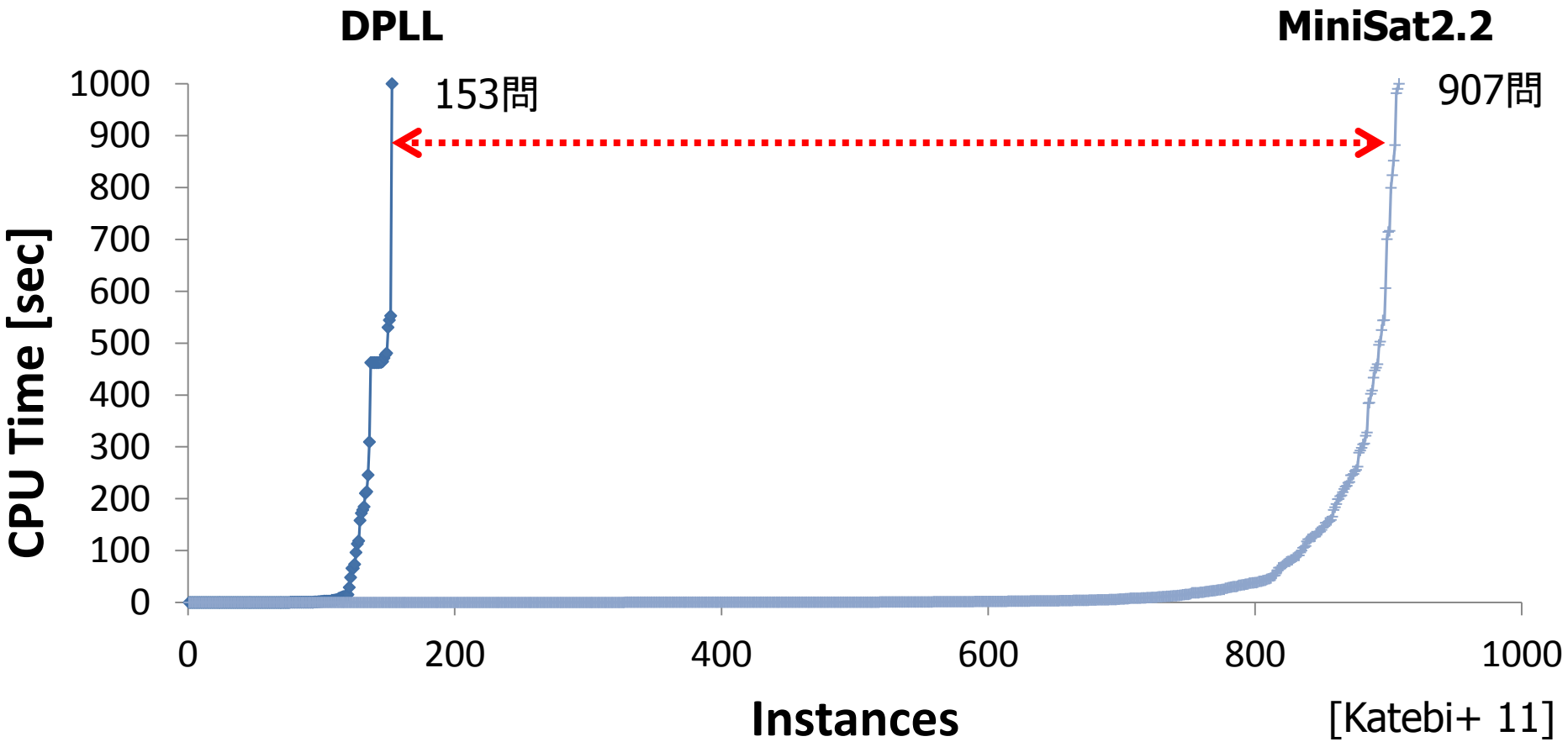
変数選択ヒューリスティクス
により6を選択し，真とする

Implication

Decision

この問題は **SAT** である

DPLL と最新 SAT ソルバーの比較



DPLL と最新 SAT ソルバーとの間には大きな開きがある

DPLL から CDCL へ

- 矛盾からの節学習（Conflict Driven Clause Learning）
- 矛盾が発生すると，矛盾の原因を節として抽出し，学習節 (learnt clause) として記憶
- 学習節は，その後の探索で探索空間を大幅に削減

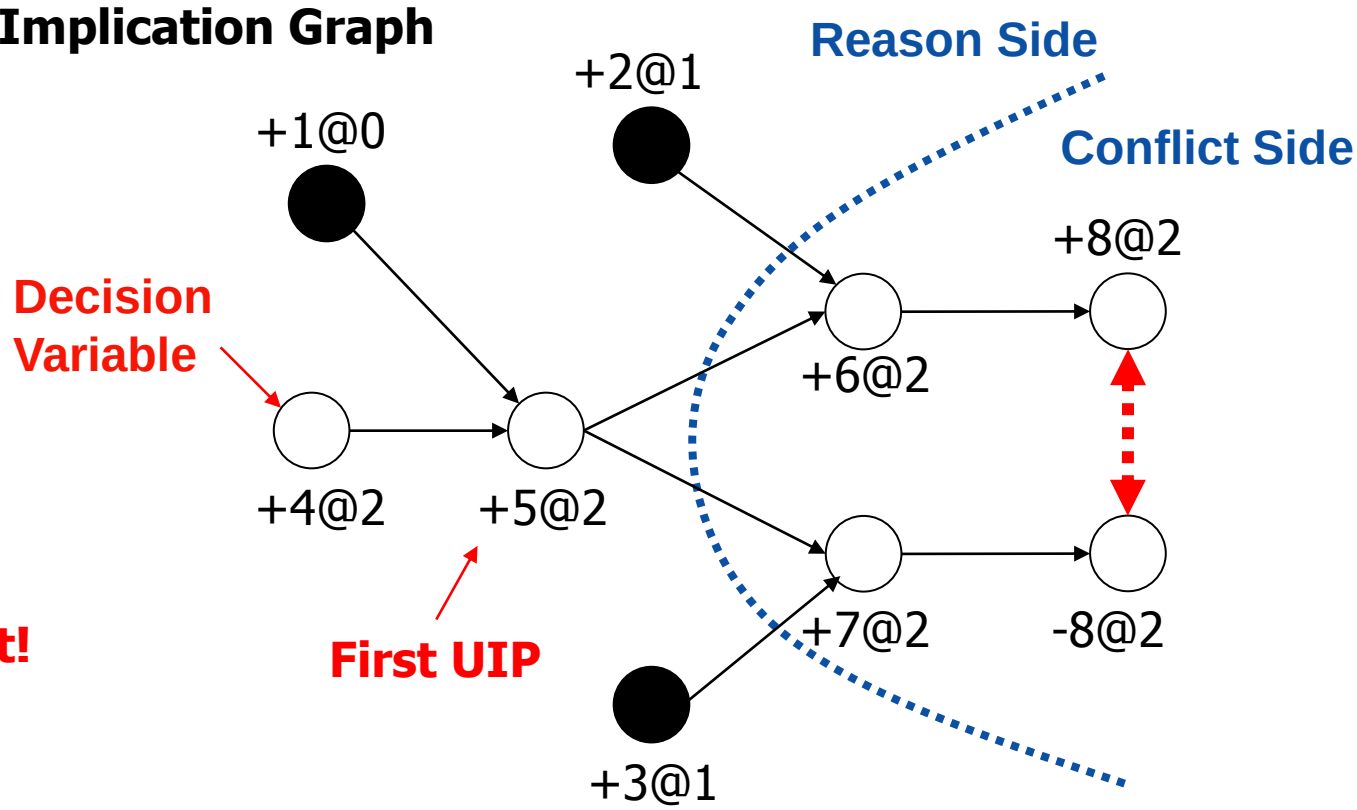
- CDCL により性能が大きく向上（ブレイクスルー）
- DPLL に節学習を導入したソルバーを CDCL ソルバーという

CDCL [Silva 99, Bayardo 97]

{+1}
{+1,+2,+4}
{+2,+9}
{-1,+4,+9}
{-2,+3}
{-2,-5,+6}
{-1,-4,+5}
{-5,-3,+7}
{-6,+8}
{-7,-8}

Conflict!

Implication Graph



Decision Stack

Lv 0	+1			
Lv 1	+2	+3		
Lv 2	+4	+5	+6	+7

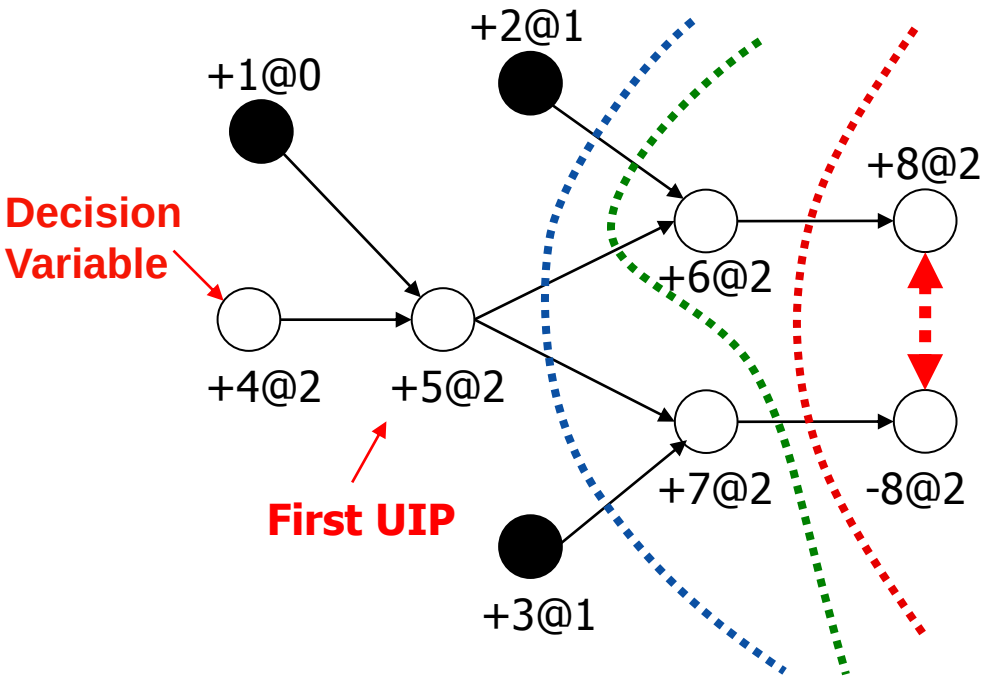
+2 ∧ +5 ∧ +3 がそろうと，必ず矛盾する

節 **-2 ∨ -5 ∨ -3** を学習する

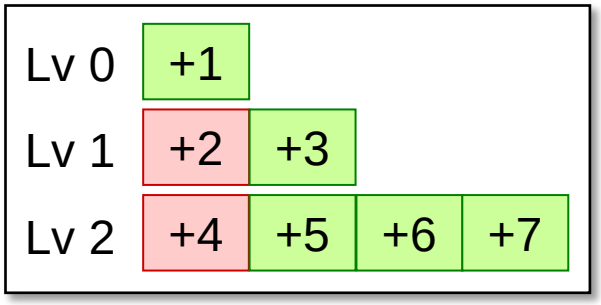
学習節は論理的帰結

{+1}
{+1,+2,+4}
{+2,+9}
{-1,+4,+9}
{-2,+3}
{-2,-5,+6}
{-1,-4,+5}
{-5,-3,+7}
{-6,+8}
{-7,-8}

Conflict!



Decision Stack



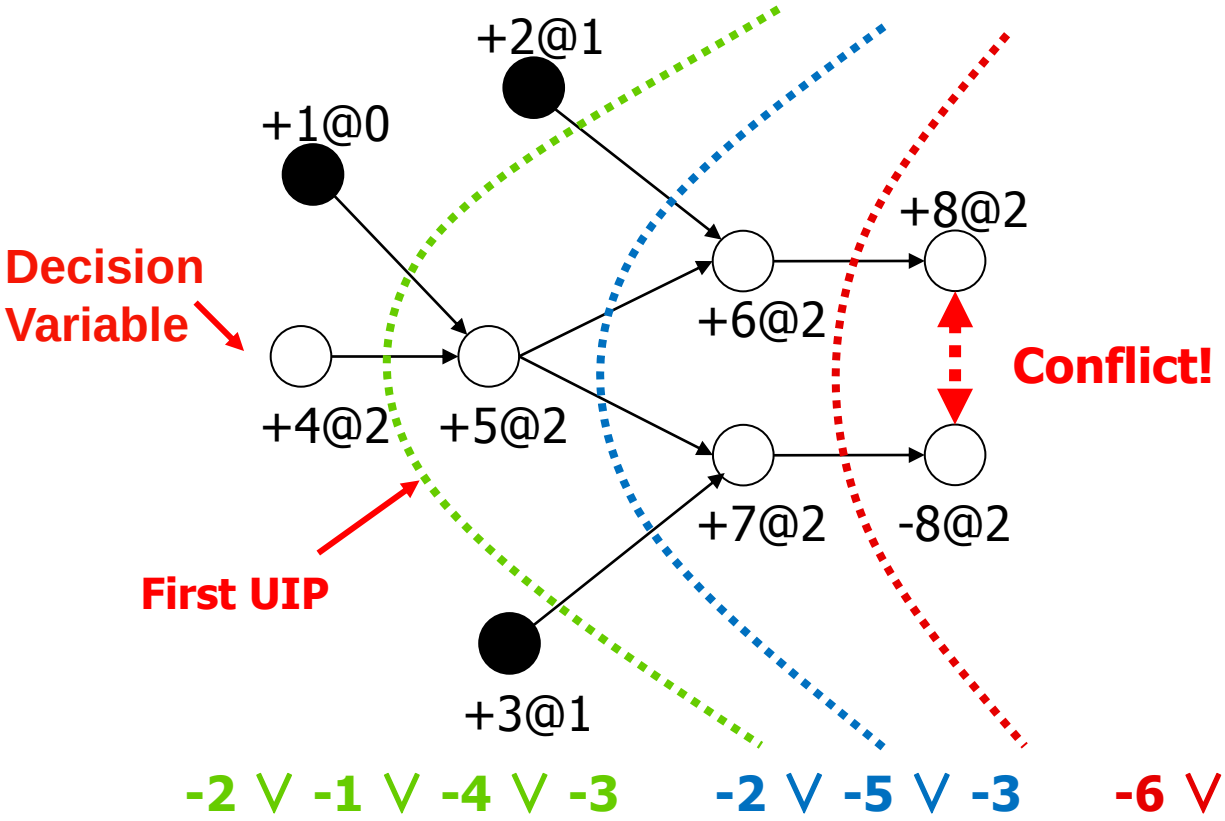
{-6,+8} {-7,-8}

{-6,-7} {-2,-5,+6}

{-2,-5,-7} {-3,-5,+7}

{-2,-3,-5}

Implication Graph と学習節



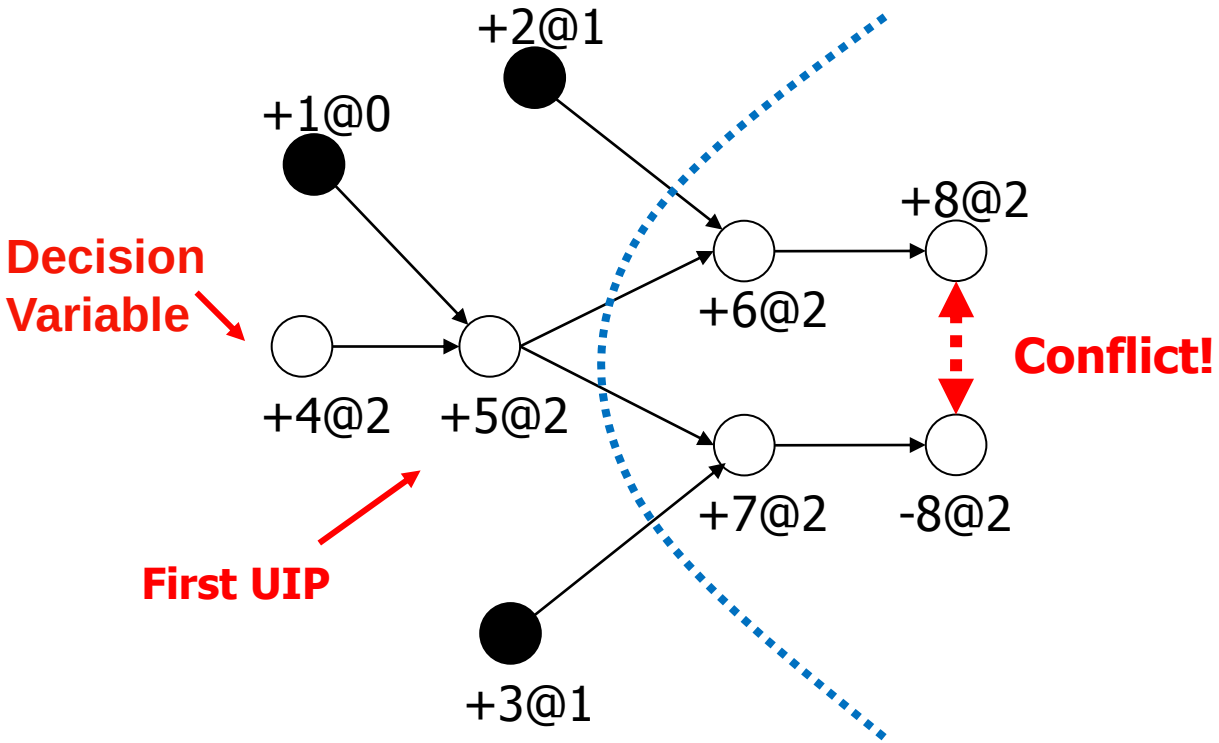
長い ← 学習節の長さ → 短い

少ない ← 単位伝搬に利用される可能性 → 多い

多い ← 矛盾に至る単位伝搬の刈り取り量 → 少ない

First UIP の直後で分割すると効果的な学習節が獲得可能 [Zhang+ 01]

学習節とバックジャンプ



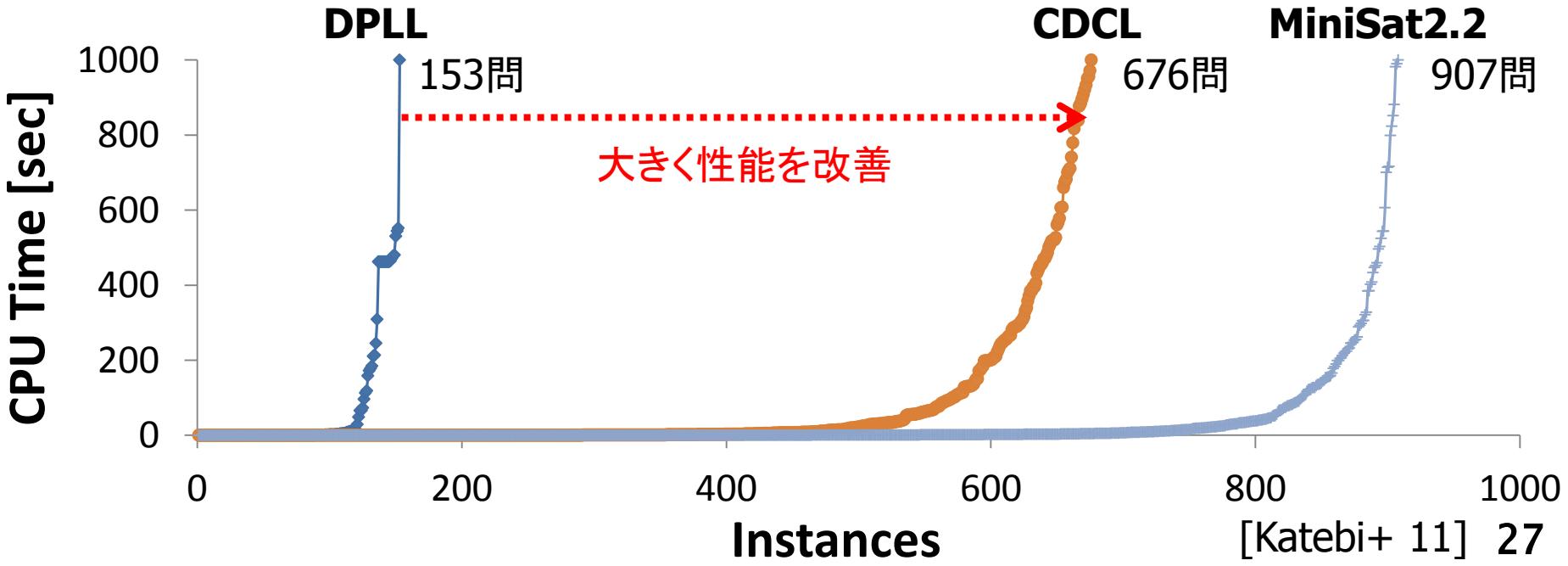
-2 ∨ -5 ∨ -3 ↔ 2 ∧ 3 → -5

- 命題変数 2 と 3 は Lv 1 で真
- 学習節 -2 ∨ -5 ∨ -3 は論理的帰結であるため，本来ならば Lv 1 において 5 を偽にできたことになる
- そこで Lv 1 までバックジャンプし，-5 を含意する

学習節は単位融合では起こせない含意を新たに生み出す [Pipatsrisawat+ 08]

CDCL の効果

- CDCL ソルバーは，矛盾が発生するたびに学習節を獲得
- 学習節は、同様の矛盾発生を防ぐ有用な知識
  - 必ず矛盾に至る単位伝搬の連鎖の発生を防ぐ
  - 学習節を獲得するたび，無駄な真偽値の組合せが除去され，探索空間が削減されていく
- CDCL は DPLL でのみでは起こせない単位伝搬を生み出す



高速 SAT ソルバー

アルゴリズム

- 矛盾からの節学習とバックジャンプ
- 部分解キャッシング

ヒューリスティクス

- **変数選択戦略**
- **リスタート戦略**
- **学習節評価尺度**

実装技術

- 監視リテラル
- 充足済節の高速判定
- キャッシュミスヒットの低減

DPLL 手続き

変数選択ヒューリスティクス

CDCL ソルバーのアルゴリズム

(1) 単位節があれば，それを充足（単位伝搬）
(2) 単位節がなければ，適当な変数を選択し真または偽を割り当て
(3) 矛盾した場合，その原因を解析し，予防する節を学習

- 変数選択ヒューリスティックスは SAT ソルバーの性能に大きな影響を与えるため，これまでに多くの手法が提案されている

- **基本方針**：頻出するリテラルに偽を割り当てる(Fail-First Principle)
  - 多くの節を短くし，単位伝搬を促進
  - 現在の真偽値割当に可能性がないことを早期に発見

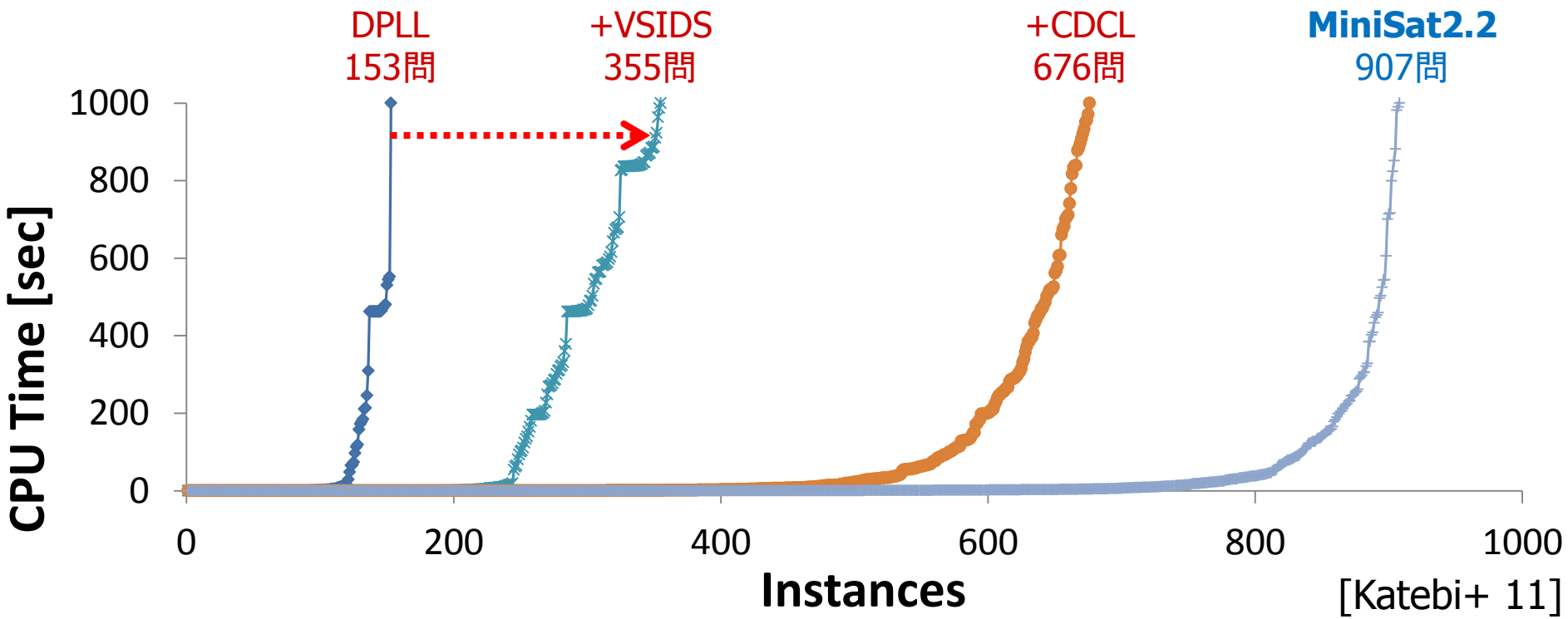
初期の変数選択ヒューリスティクス

- **MOMS** (Maximum Occurrence in clauses of Minimum Size) [Freeman 95], **BOHM** [Böhm+ 96]
  - 短い未充足の節に多く出現するリテラルを優先して選択

- 欠点
  - 各リテラル x に対し，未充足の節における x の出現回数を求める必要がある
  - 真偽値割当が更新されるたび（単位伝搬，バックトラック）に再計算する必要があり高コスト

VSIDS [Moskewicz+ 01]

- 多くの CDCL ソルバーで採用されている代表的ヒューリスティクス
- Variable State Independent Decaying Sum の略
- ソルバー Chaff で導入され，性能を大きく改善 [Moskewicz+ 01]



[Katebi+ 11]

CDCL ほどではないが，性能を大きく改善

VSIDS (MiniSat 版)

(0) 各変数ごとに優先度を表すカウンタを用意し，0 で初期化

(1) 学習節が追加されるたび，その節中の変数のカウンタを増加 (VSI)

(2) 変数選択においては，最も高い優先度を持つ未割り当ての変数を選択し，それを偽とする（一定の確率で優先度にかかわらず変数をランダムに選択）

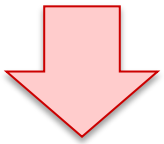
(3) 定期的にすべてのカウンタの値を定数で割り減少させる (DS)

特徴

- 変数が含まれる節の状態（充足済 or 未充足）に関わらずカウンタを更新
- よって計算が軽い（単位伝搬・バックトラック時に何もしないため）
- (3) がなければ，学習節が追加されるたびに，それに含まれる変数の優先度が増加するので，優先度は変数の出現回数の大まかな近似にはなっている
- (1) および (3) より，最近追加された学習節に含まれる変数が優先的に選択されるため，最近の学習節を充足させる働きがある

学習節の評価尺度

- 矛盾からの節学習は 高速 SAT ソルバーの基盤技術
  - 学習節は必ず矛盾に至る単位伝搬の連鎖の発生を防ぐ
  - DPLL のみでは起こせない単位伝搬を生み出す
- CDCL ソルバーは矛盾が発生するたびに学習節を獲得
  - もし学習節を保有しないと，同じ失敗を繰り返すことになり，探索は進まなくなる（DPLL と同じになる）
  - 学習節は問題サイズに対して指数関数的に生成されるため，すべて保有することは困難であり，単位伝搬速度も低下



学習節の定期的な削減が必要

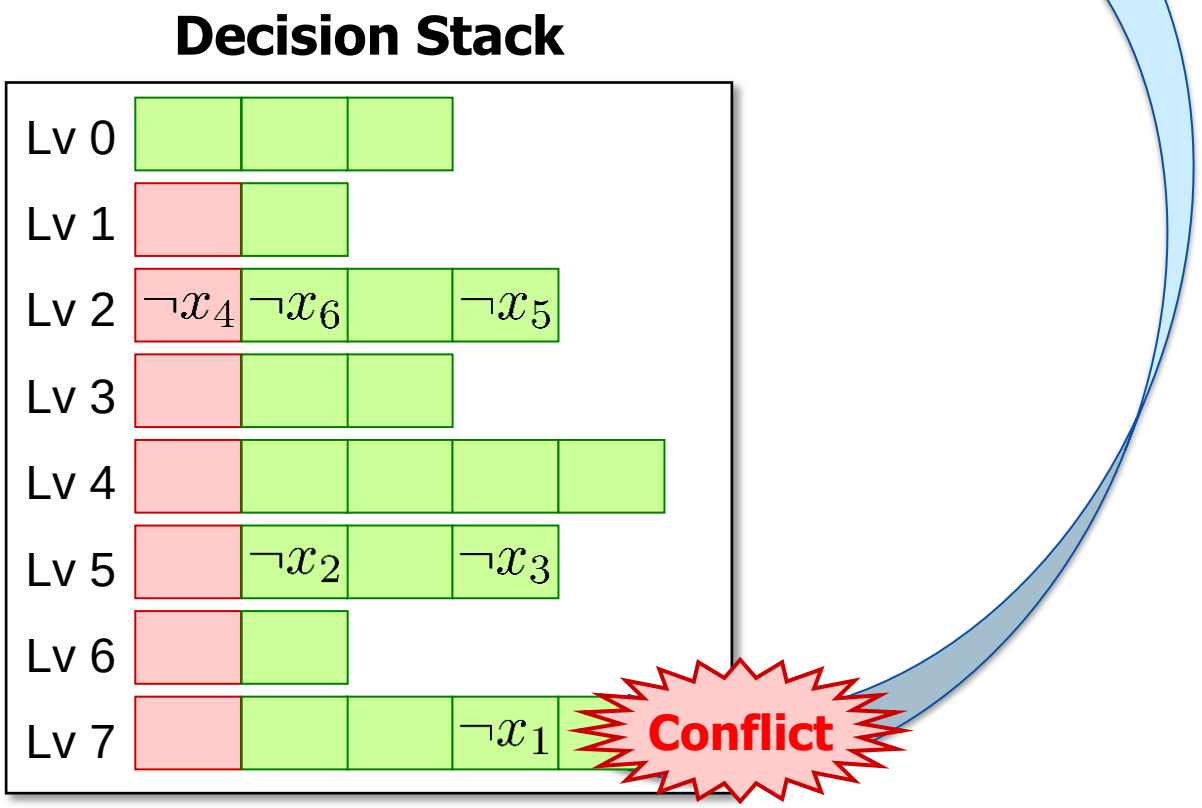
学習節の評価尺度

- **長さ**
  - 短い節は枝刈り能力が高い
  - 並列 SAT ソルバーでは短い節を共有
- **活性度**
  - **Chaff** [Moskewicz+ 01] , **MiniSat** [Eén+ 03]
  - 学習節に活性度を付与．活性度が閾値以下の節を削除
  - 矛盾の原因解析時に，学習節が矛盾導出に関わっていたならば，その学習節の活性度を向上させる
  - 最近使われていない学習節は破棄される
- **リテラルブロック距離 (Literal Blocks Distance; LBD)**
  - 将来使われる可能性が高い学習節を識別
  - 例え長い節であっても，使われる可能性が高いならば保持し続ける
  - **glucose** [Audemard+ 09] で導入された
    - ✓ SAT 2009 競技会 Application 部門で総合2位，UNSAT クラス優勝

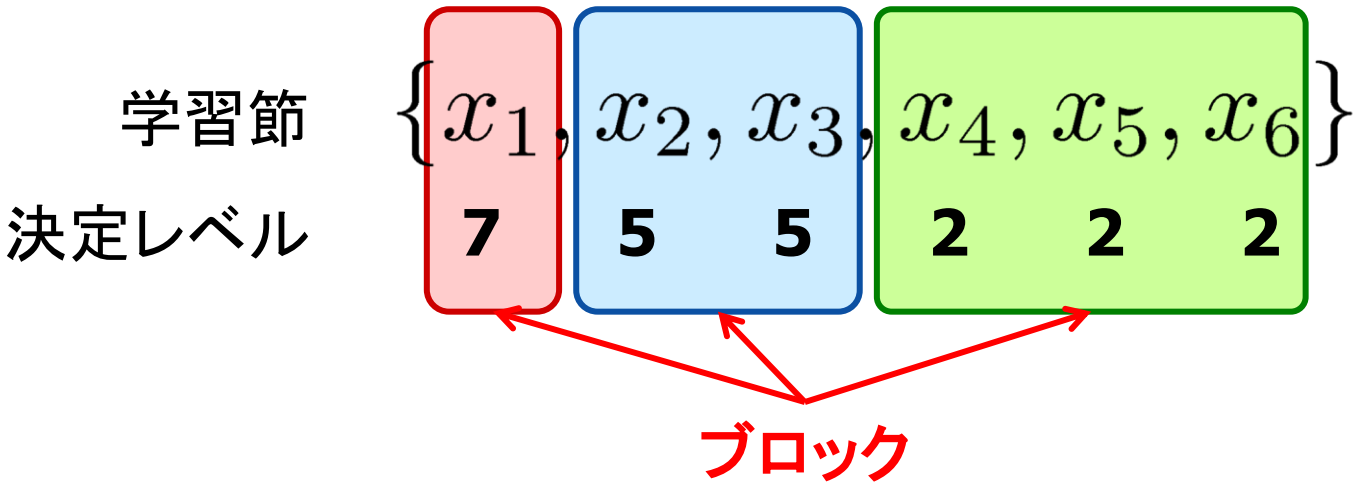
LBD [Audemard+ 09]

学習節 $\{x_1, x_2, x_3, x_4, x_5, x_6\}$

決定レベル **7 5 5 2 2 2**



LBD [Audemard+ 09]



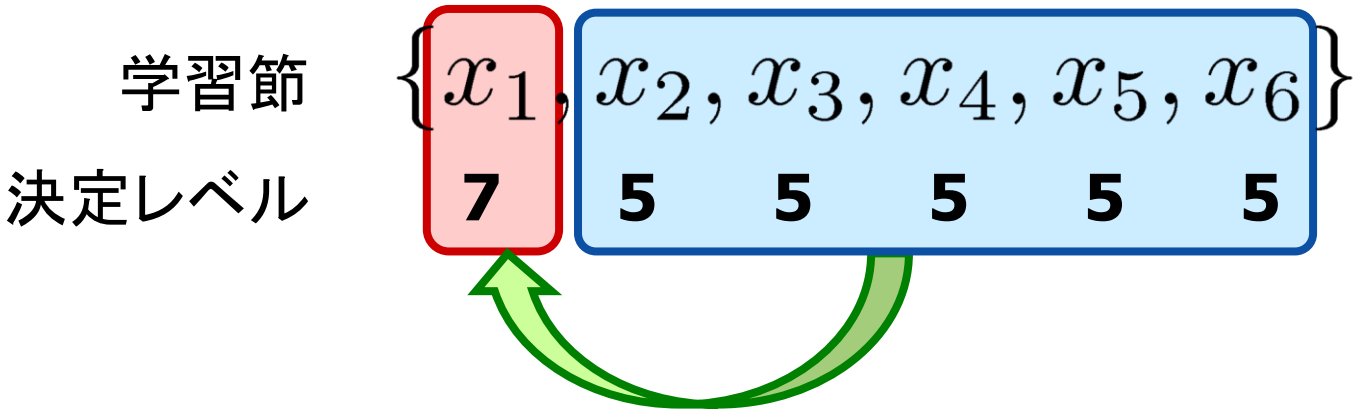
- 同じレベルで割り当てられたリテラル群をブロックと呼ぶ
- 節の LBD を，その節中のブロック数で定義

✓ ブロック中のリテラル群は，ある単位伝搬の連鎖により同時に偽が割り当てられる可能性がある（例： at-most-one 制約）

✓ LBD では，ブロックをあたかも1つのリテラルと見なし，ブロック数の短い節を将来使われる可能性が高い節として評価（長さの一般化）

Glue Clause

- 特に LBD = 2 の節を glue clause と呼ぶ
- Glue clauses は，例え非常に長くても，単位伝搬を促進する可能性が期待できる

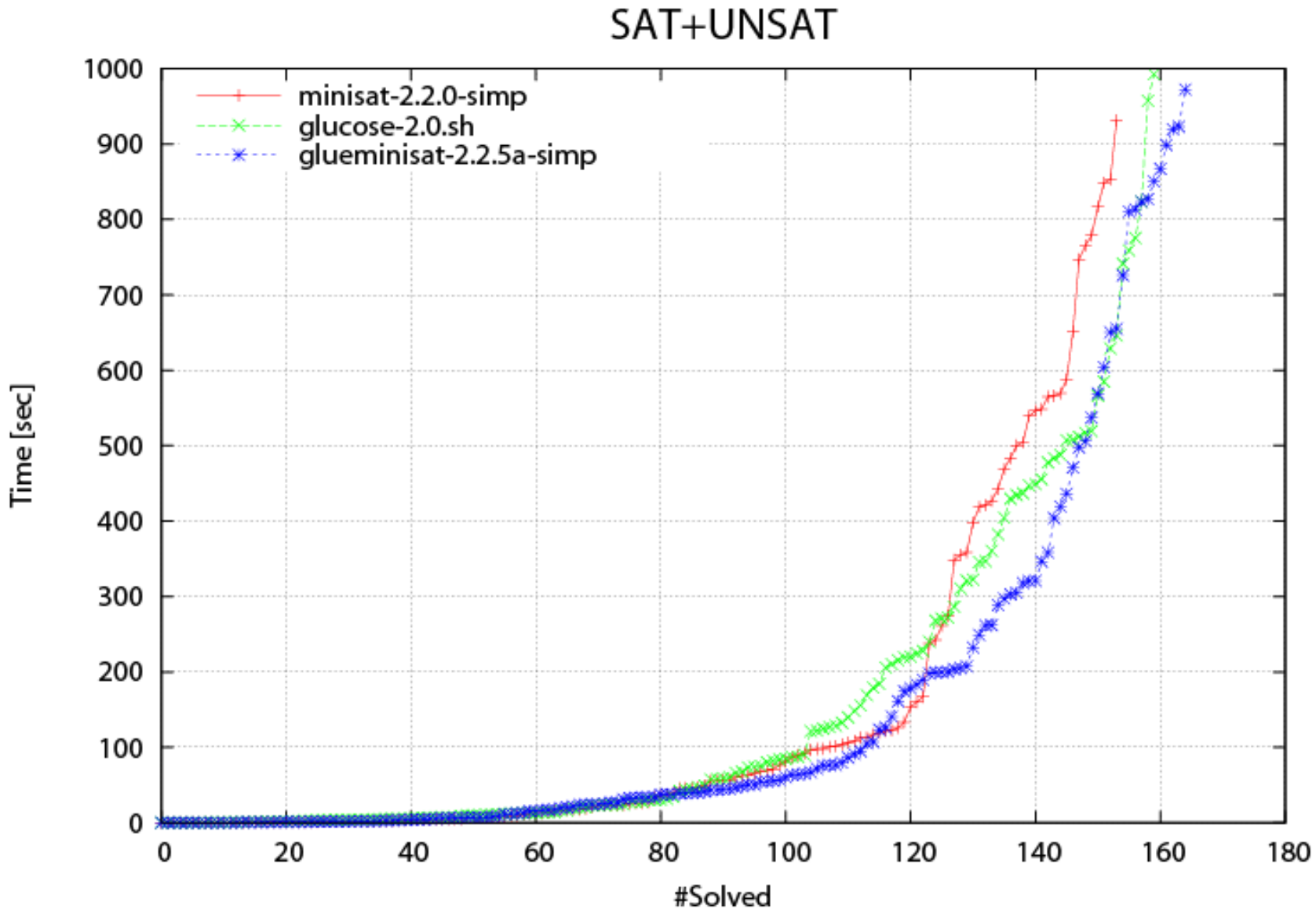


$x_2 \sim x_6$ は将来も同時に出現する可能性が高く，その場合 x_1 の伝搬につながる

Glucose, GlueMiniSat は，獲得した glue clauses を決して削除しない

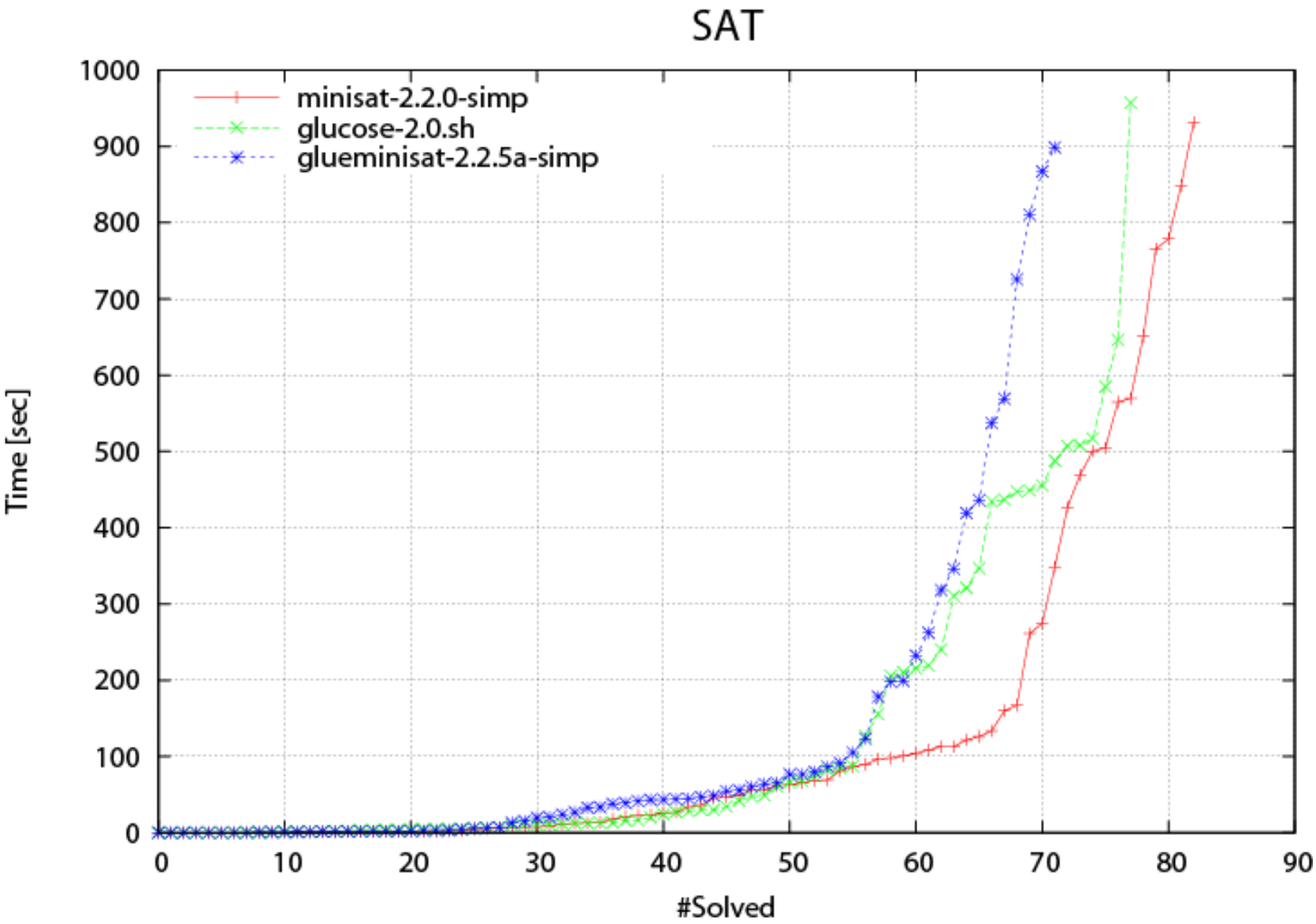
LBD の評価

SAT 2011 競技会 Application 部門



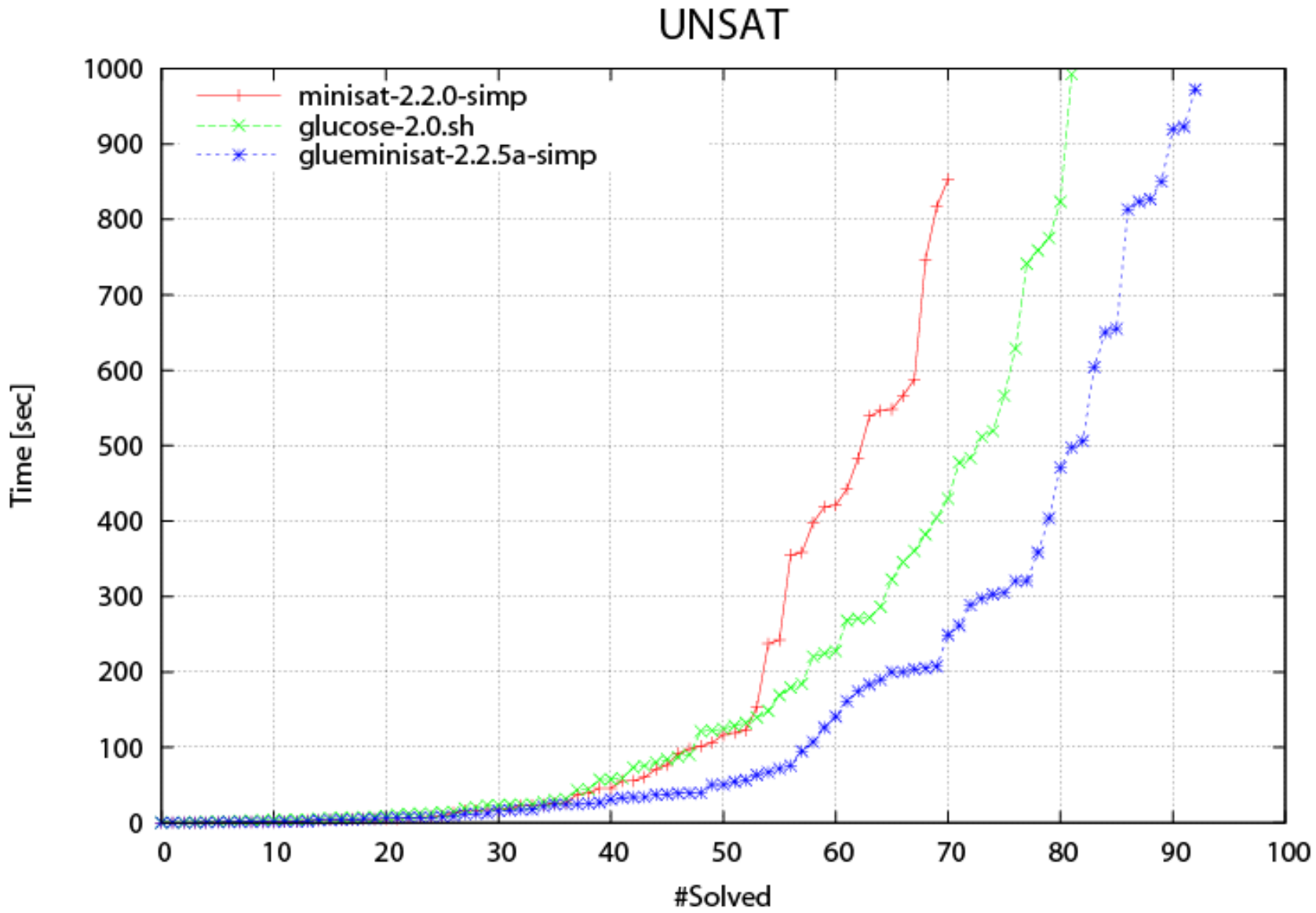
LBD の評価

SAT 2011 競技会 Application 部門

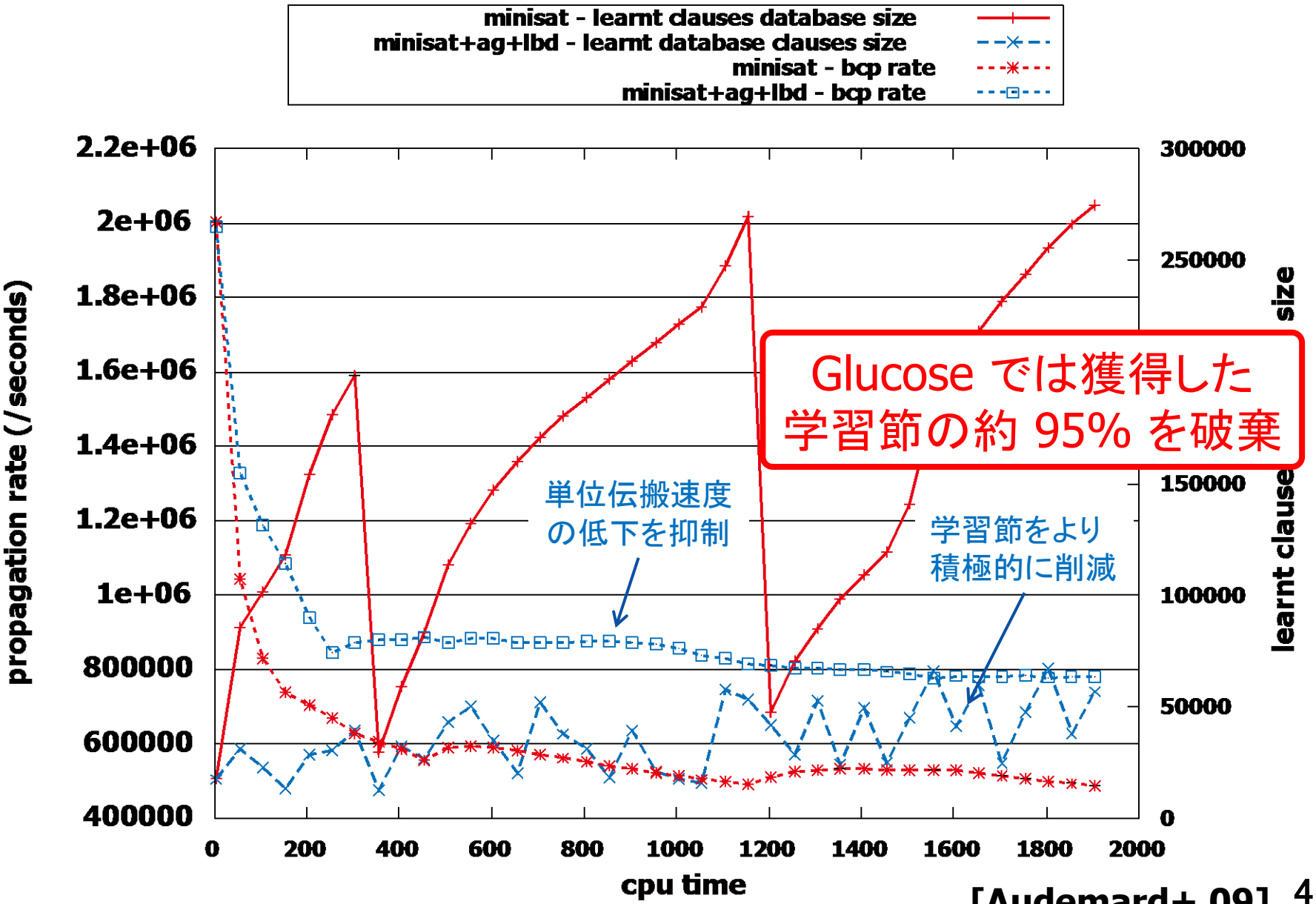


LBD の評価

SAT 2011 競技会 Application 部門



LBD では学習節を積極的に削減可能



[Audemard+ 09]

高速 SAT ソルバー

アルゴリズム

- 矛盾からの節学習とバックジャンプ
- 部分解キャッシング

ヒューリスティクス

- 変数選択戦略
- リスタート戦略
- 学習節評価尺度

実装技術

- **監視リテラル**
- **充足済節の高速判定**
- **キャッシュミスヒットの低減**

DPLL 手続き

監視リテラル [Moskewicz+ 01]

- SAT ソルバーの実行時間の 70～90% は単位伝播の処理で占められる

CDCL ソルバーのアルゴリズム

(1) 単位節があれば，それを充足（単位伝搬）
(2) 単位節がなければ，適当な変数を選択し真または偽を割り当て
(3) 矛盾した場合，その原因を解析し，予防する節を学習

効率の良い単位節検出アルゴリズムが不可欠

単位節の素朴な検出方法

- カウンターを利用

$\{x_1, x_2, x_3, x_4\}$

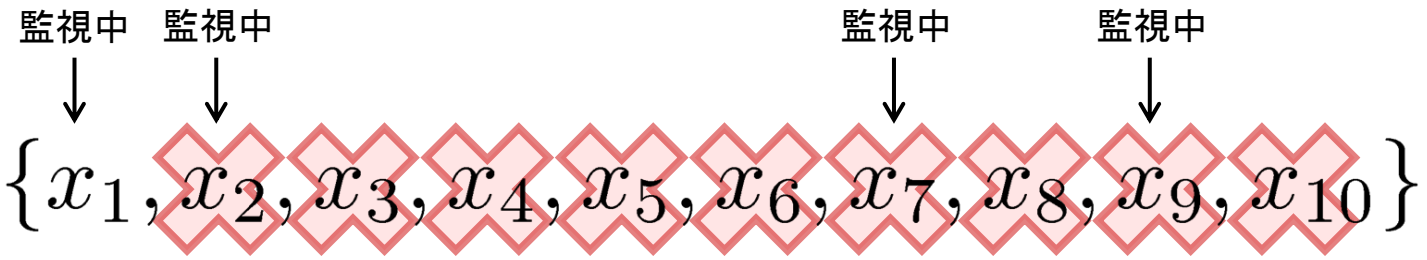
true	false	undef
0	3	1

  - 各節がカウンターを保持
  - 節中のリテラルのうち，真と偽が割り当てられたリテラルの数をカウント
  - もし真が 0 かつ偽が "節長-1" なら単位節

欠点

- リテラル x の真偽値割当が更新されるたび，x を含むすべての節のカウンタを更新する必要がある
- 単位伝搬，変数選択による値割り当て，バックジャンプのいずれの場合もカウンタを更新する必要がある

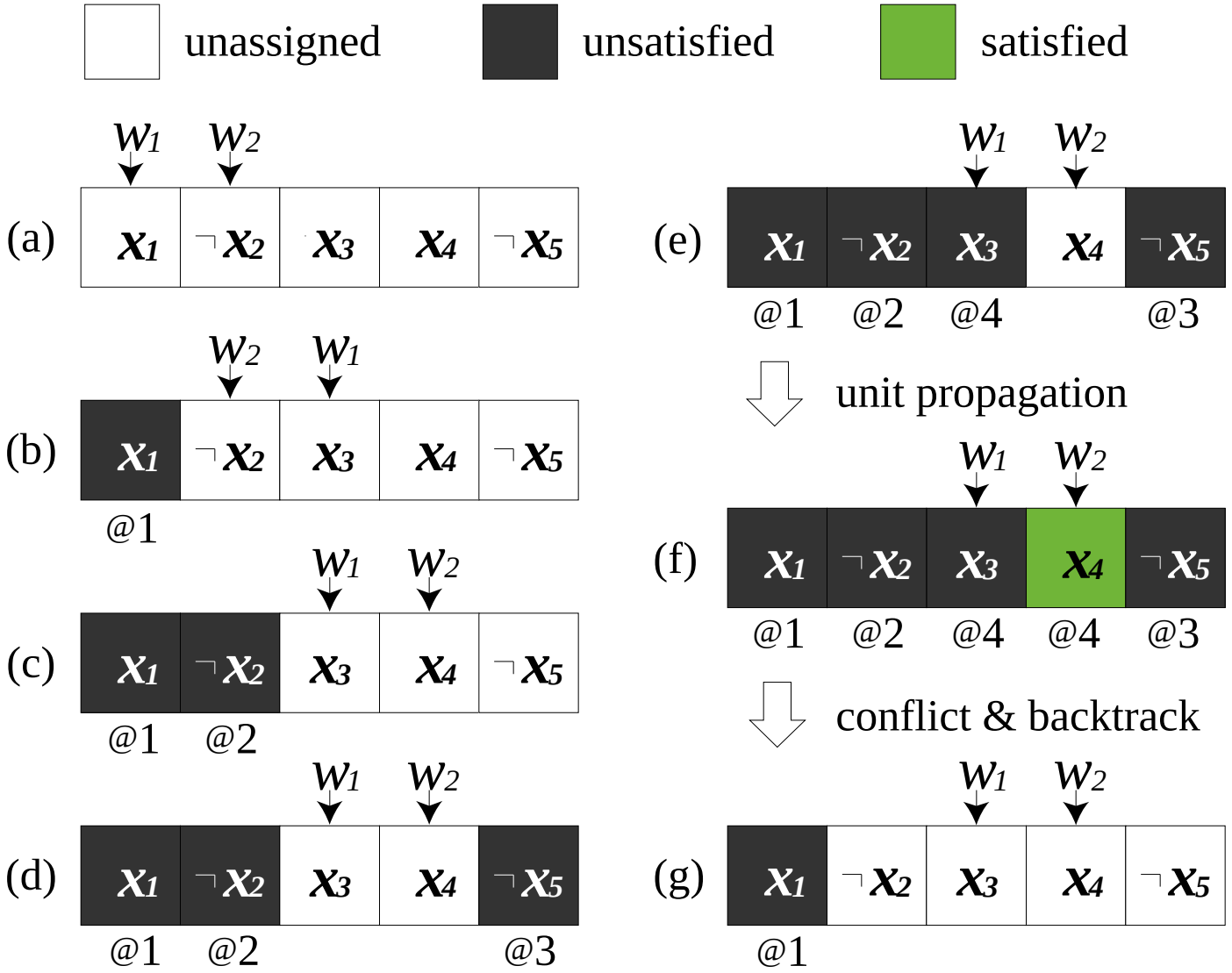
監視リテラルによる単位節検出



監視中のリテラルが偽になったら，はじめて節をチェックすれば良い

- 単位節になる直前の状態は，節中のリテラルのうち2つのみが未割り当てで（これを x_i , x_j とする），残りが偽の場合
- x_i , x_j のいずれかに偽が割り当てられたときその節は単位節
- よって単位節検出のために節中のすべてのリテラルを調べる必要はなく，未割り当てのリテラル2つのみを監視すれば十分

監視リテラルによる単位節検出



(d) (g) において何もしなくて良いことが大きな利点

監視リテラルの利点

①ある変数 x に値が割り当てられたとき，x を含むすべての節ではなく，x を監視中の節のみを走査するだけで済むため，調べる節数を大幅に削減可能

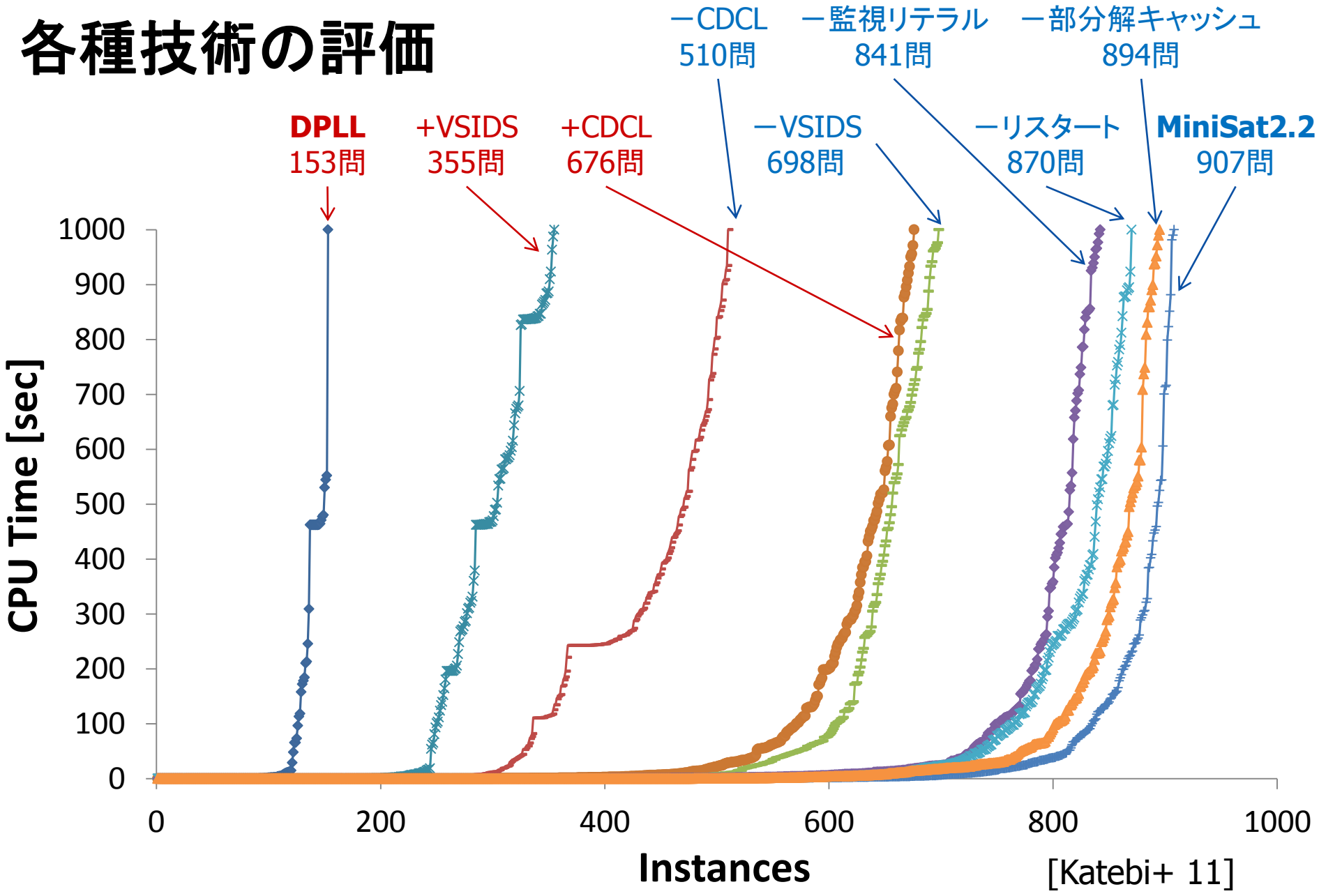
②バックジャンプ時には単に変数を未割り当てに戻すだけで良い

- 監視中のリテラルを元に戻す必要がない（バックジャンプ処理が非常に軽い）

③CPU データキャッシュのミスヒット率を大幅に低減

- 変数 x に値を割り当てると x を監視中の節数が大きく減る
- 矛盾によるバックジャンプが起きると，最近値を割り当てた変数に再び値を割り当てることがしばしばあり，このとき x を監視中の節数は少なくなっているため，最初に値を割り当てたときよりも高速に値を割り当てることが可能
- これは節を格納しているメモリへのアクセス回数を大きく減らし，その結果キャッシュのミスヒット率が低減する
- Zhang らは，GRASP などの従来手法と比較して，ミスヒット率の低減だけでも約3倍の速度の向上が得られることを示している [Zhang+ 03]

各種技術の評価



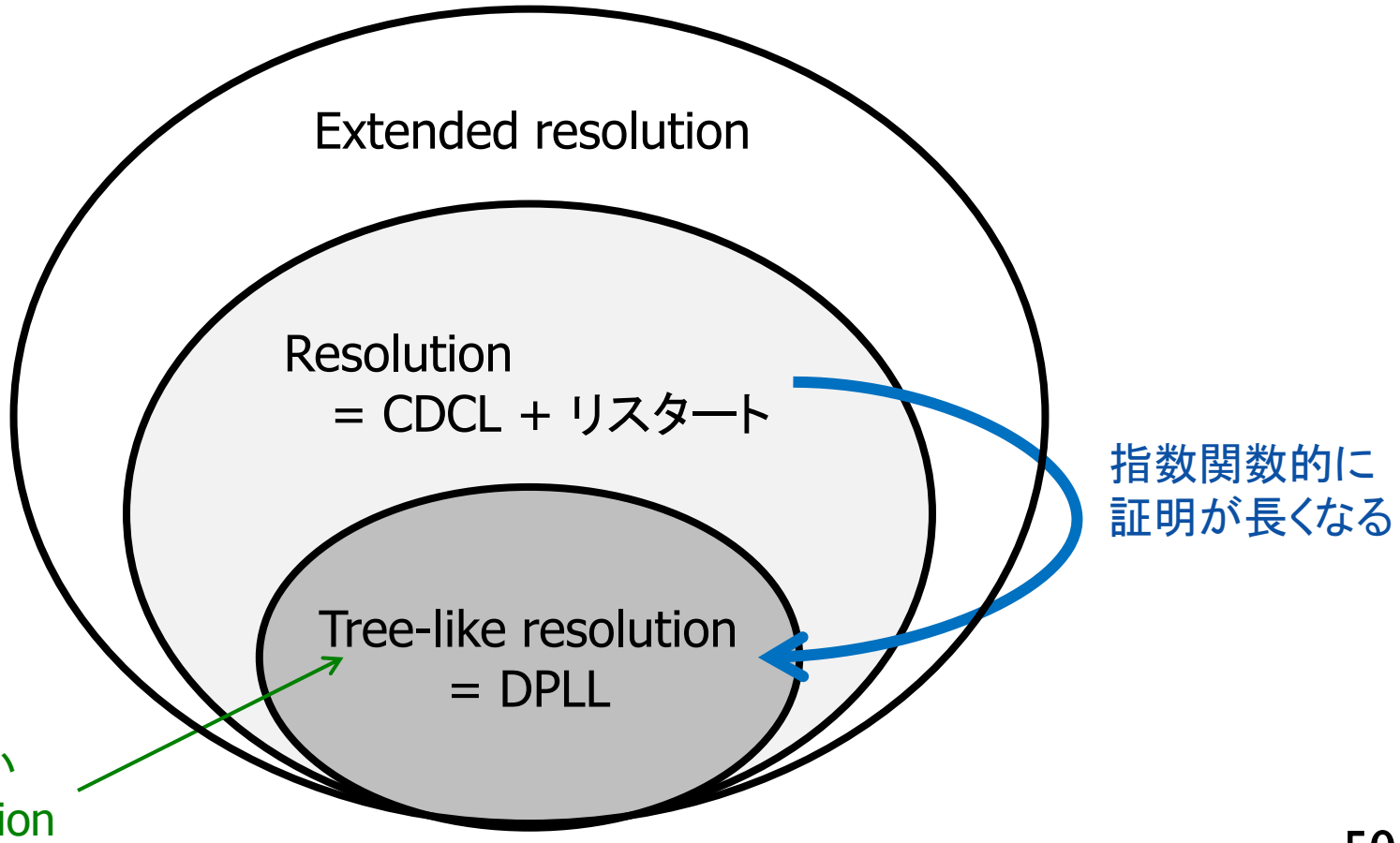
CDCL >> VSIDS >> 監視リテラル > リスタート > 部分解キャッシング

目次

- SAT とは？
- 高速 SAT ソルバーの要素技術
  - アルゴリズム
  - ヒューリスティクス
  - 実装技術
- 高速 SAT ソルバーの理論的側面
- GlueMiniSat の紹介

SAT ソルバーと融合法

- CDCL + リスタートは，融合法と同程度に短い証明を生成可能（融合法の最短証明の高々多項式倍）[Pipatsrisawat+ 09]



拡張融合法 [Tseitin 83]

- 融合法に拡張規則を追加

  ϕ : CNF 式　l_1, l_2: ϕ に現れるリテラル　x: 新規変数

  証明の任意の時点で $x \leftrightarrow l_1 \vee l_2$ を導入してよい

- 通常の融合法よりも強力（短い証明を生成できる場合がある）[Cook 76]

拡張融合法の例

$$\frac{\overline{l_1} \vee \alpha_1 \quad c_1 \qquad \overline{l_1} \vee \alpha_2 \quad c_2 \qquad \cdots \qquad \overline{l_1} \vee \alpha_k \quad c_k}{\overline{l_1} \vee \beta}$$

$$\frac{\overline{l_2} \vee \alpha_1 \quad c_1 \qquad \overline{l_2} \vee \alpha_2 \quad c_2 \qquad \cdots \qquad \overline{l_2} \vee \alpha_k \quad c_k}{\overline{l_2} \vee \beta}$$

$$x \leftrightarrow l_1 \vee l_2$$

$$\overline{x} \leftrightarrow \overline{l_1} \wedge \overline{l_2}$$

$$\frac{\overline{x} \vee \alpha_1 \quad c_1 \qquad \overline{x} \vee \alpha_2 \quad c_2 \qquad \cdots \qquad \overline{x} \vee \alpha_k \quad c_k}{\overline{x} \vee \beta}$$

証明の圧縮が可能

拡張融合法に基づくソルバ

- GlucoseER [Audemard+ 10]
  - 獲得した学習節が連続して $\overline{l_1} \vee \alpha_1$, $\overline{l_2} \vee \alpha_1$ という形式のとき $x \leftrightarrow l_1 \vee l_2$ を追加
- TinisatX [Huang 10]
  - 獲得した学習節の長さが 30 より大きい場合に，その節から（決定レベルが浅い順に？）2つのリテラル $\overline{l_1}$, $\overline{l_2}$ を選択し，$x \leftrightarrow l_1 \vee l_2$ を追加

鳩ノ巣問題のような通常の融合法が苦手とする問題で効果を実証

目次

- SAT とは？
- 高速 SAT ソルバーの要素技術
  - アルゴリズム
  - ヒューリスティクス
  - 実装技術
- 高速 SAT ソルバーの理論的側面
- GlueMiniSat の紹介

GlueMiniSat 2.2.5
[Nabeshima+ 11][鍋島ら 12]

MiniSat2.2 + Glucose1.0 + α

[Eén+ 03] [Audemard+ 09]

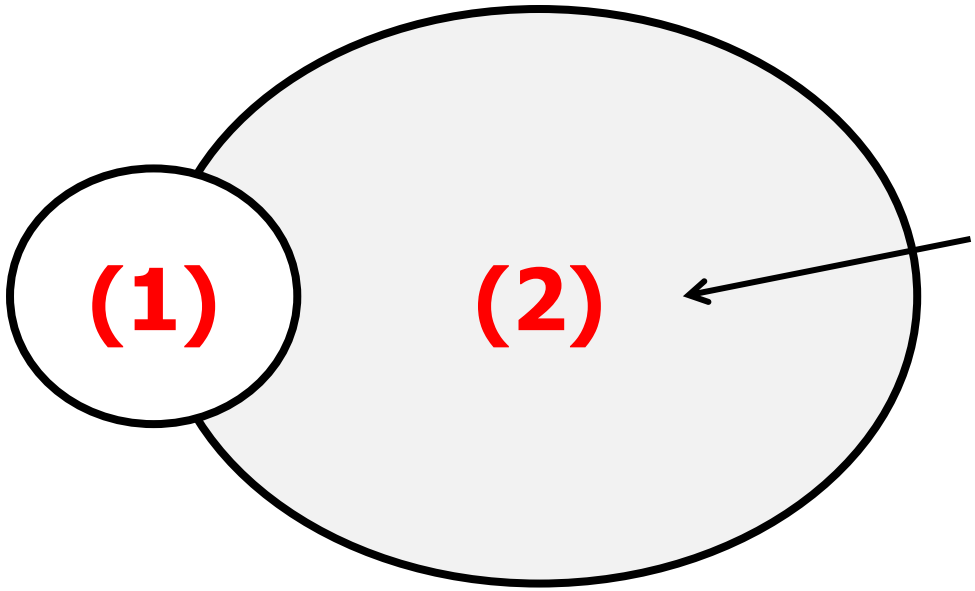
- 学習節の評価尺度 LBD の改善
- 良い学習節を獲得するための積極的リスタート戦略

- SAT 2011 競技会の Application 部門にて **GlueMiniSat** がクラス優勝
  - 逐次ソルバー UNSAT クラス優勝
  - 逐次ソルバー SAT+UNSAT クラス2位
  - 並列ソルバー UNSAT クラス2位（逐次ソルバーとして）

	MiniSat 2.2	Glucose 2.0	GlueMiniSat 2.2.5
Base solver	-	MiniSat2.2	MiniSat2.2
Var selection heuristics	VSIDS	VSIDS	VSIDS
Evaluation criteria of learnts	LRU	LBD	Pseudo LBD & LRU
Reduction strategy of learnts	Exponential $(\#C/3)*1.1^d$	Linear 4000+300x+α	Linear 20000+10000x
Restart strategy	Luby	Dynamic (LBD)	Aggressive (LBD & DLV)
Priority handling for binary clauses		✓	
Clause Minimization		✓	
Phase caching	✓	✓	✓
Fast identification of satisfied clauses	✓	✓	✓
Memory management	Single area	Single area	Single area

LBD 計算の詳細

- 学習節 C が glue
  - **(1)** 矛盾原因解析から獲得した C の LBD が 2 の場合
  - **(2)** C が単位伝搬で使用されたとき，現在の真偽値割当に基づき LBD を再計算し，それが 2 の場合



Glue clauses のうち，条件 **(2)** より生成されるものが大半を占める

条件 **(2)** が支配的条件

Pseudo LBD

条件	(1)	(2)
Leant clause Decision Lv	$\{ L_1, L_2, L_3, L_4, L_5, L_6 \}$ 7 5 5 5 4 4	$\{ L_1, L_2, L_3, L_4, L_5, L_6 \}$ 7 7 5 5 4 4
LBD	3	3
Pseudo LBD	3 same as LBD	4 LBD + 1

- GlueMiniSat は pseudo LBD ≦ 3 の節を削除せず保有し続ける
  - 条件 (1) の学習節は，常に単位リテラルブロックを含むため，ブロック数が 3 であっても，ある程度単位伝搬の促進が期待できる
  - 条件 (2) の学習節は，単位リテラルブロックが含まれない場合がある．よって，これまで通り LBD ≦ 2 の節を保有する

GlueMiniSat 2.2.5 のリスタート戦略

- LBD に関するリスタート戦略 (Glucose と同様)

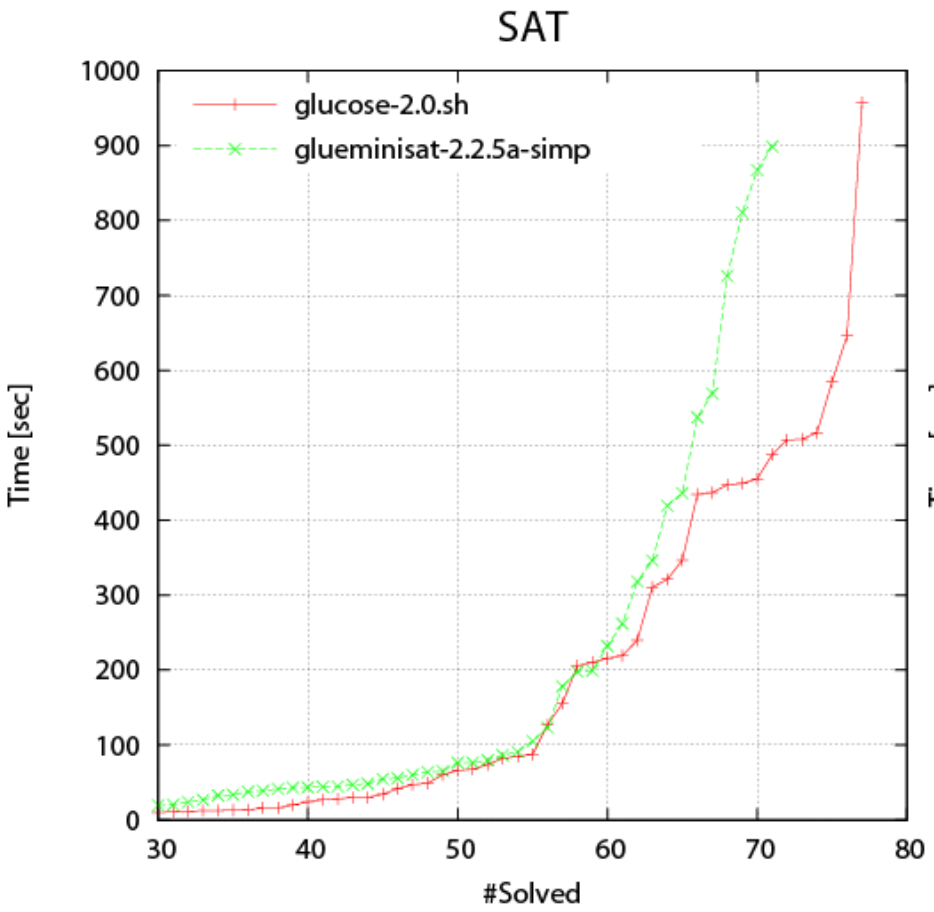
最近50件の学習節の LBD 平均 * **0.8** > すべての学習節の（削除済み含む）LBD 平均

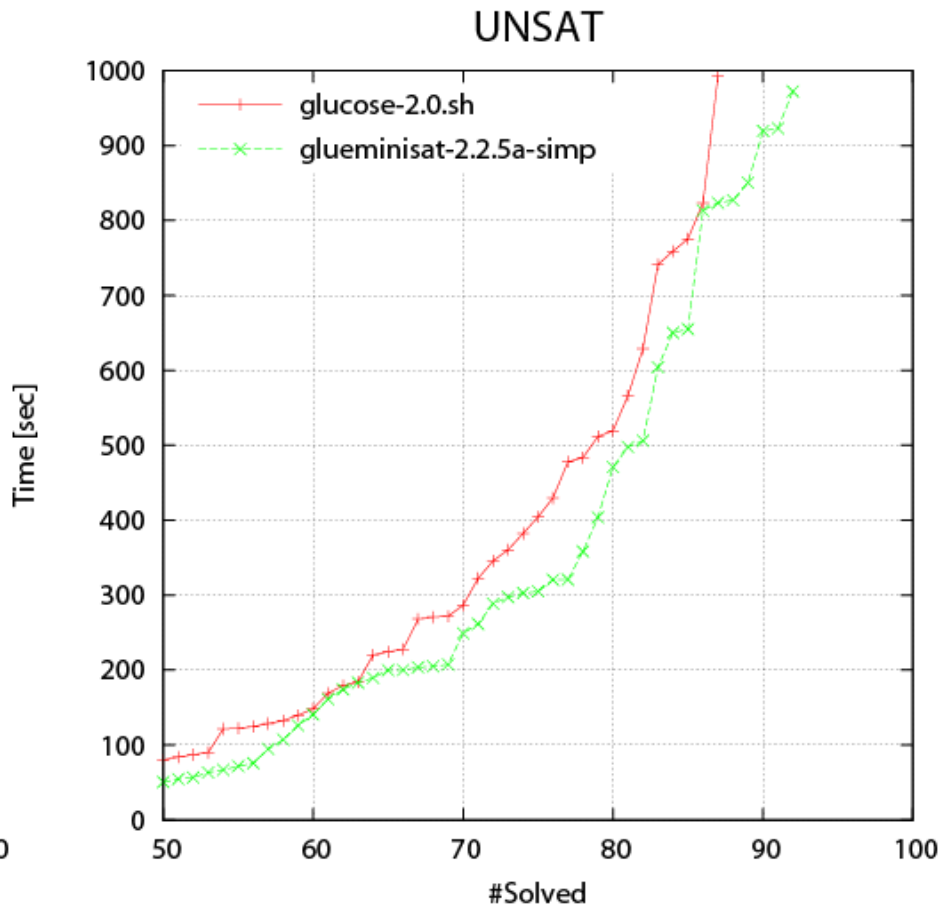
- 決定レベルに関するリスタート戦略

最近50件の矛盾の決定レベル平均 * **1.0** > すべての矛盾の決定レベル平均

- いずれかの基準を満たすとリスタート
- 非常に積極的にリスタートする

SAT 競技会 2011 Application 部門

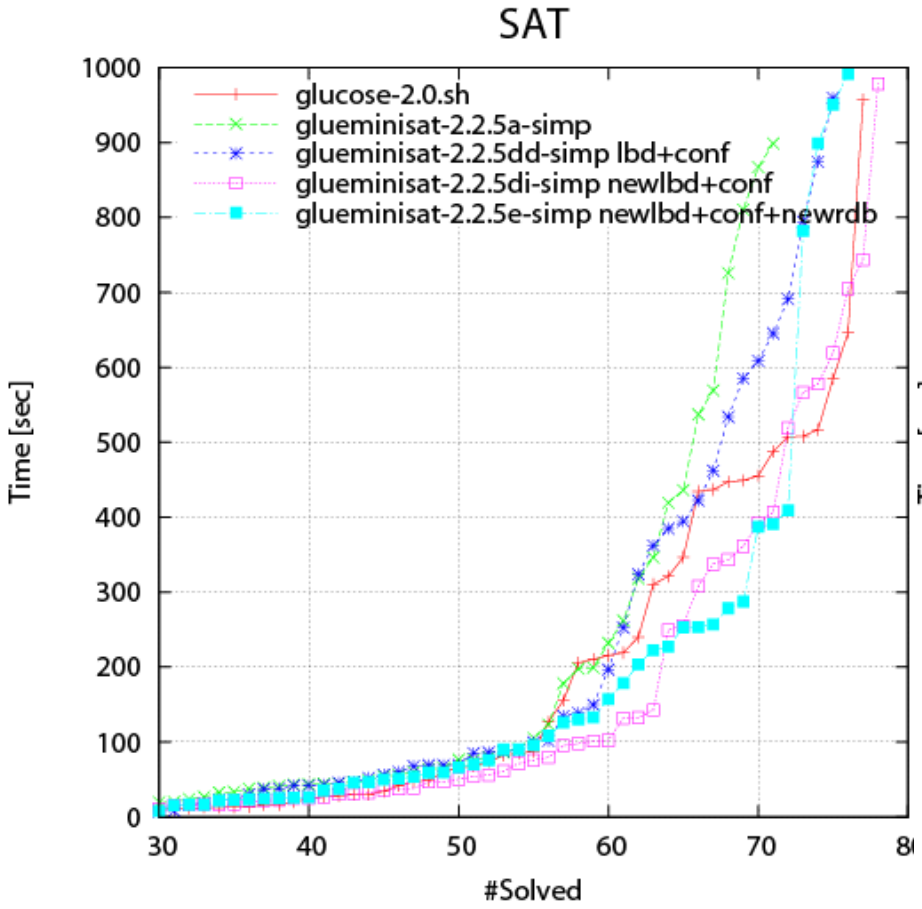


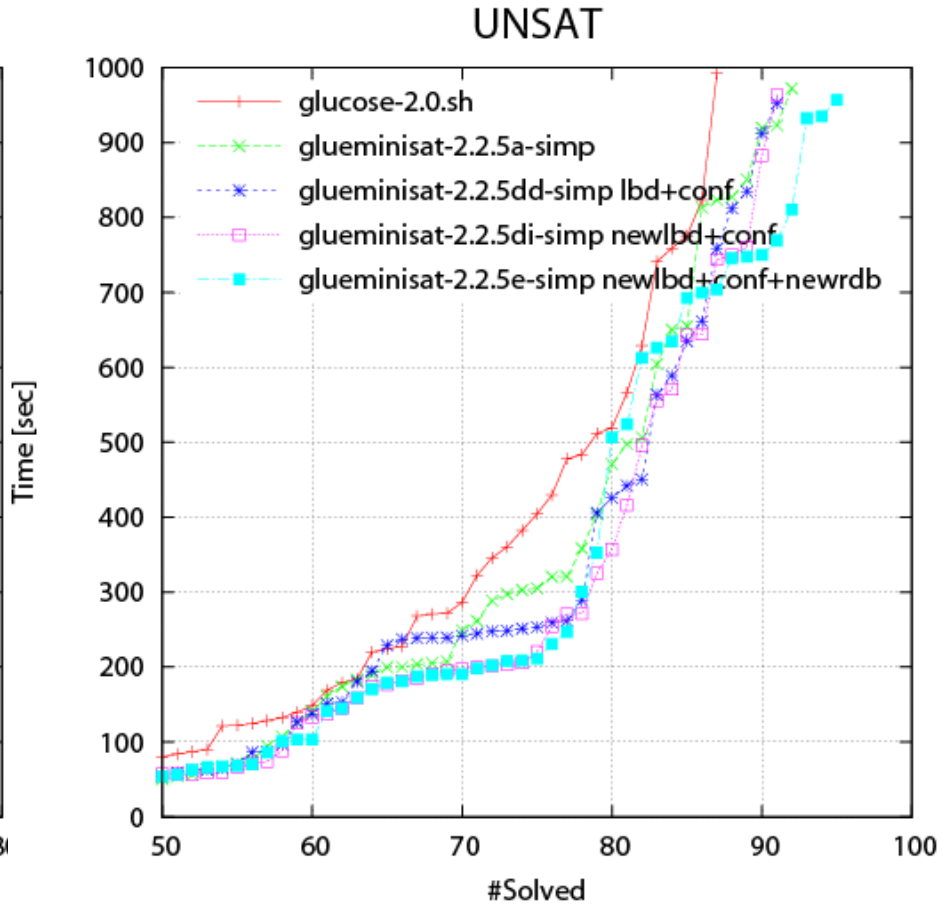


SAT と UNSAT で性能が対照的

UNSAT 証明にご利用ください

GlueMiniSat 2.2.6a





リスタート戦略，LBD 計算，学習節削減戦略を改善

今後の予定

- UNSAT の強化
  - bi-asserting & empowerment clauses の学習 [Pipatsrisawat+ 08]
  - 拡張融合法の導入
    - 置換対象のリテラル群の選択基準の検討（頻度？）

$$x \leftrightarrow l_1 \vee \cdots \vee l_n$$

- UNSAT コアの計算
  - 各学習節の証明をメモリ上に保存する手法 [Biere 08][Shacham+ 07][Aśın+ 08]
  - 各入力節に選択変数を追加する手法（素朴な手法）

ご清聴ありがとうございました