平成 29 年度

修士学位論文

最大充足化ソルバを用いた誤り位置特定手法の近似解法の研究

Study on approximation algorithms of an error localization method using a Max-SAT solver

1205089　　米田裕司

指導教員　　高田喜朗

2018 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報システム工学コース

要 旨

最大充足化ソルバを用いた誤り位置特定手法の近似解法の研究

米田裕司

年々，ソフトウェアにはより高度な機能が求められ, プログラムは複雑化している．それに伴いプログラムに潜在する誤りを発見するのが困難になっている．また, ソフトウェアのバグを取り除くのに開発工程のうち多くの時間が費やされており, ソフトウェアの生産性に大きな影響を与えている．

そのため, 誤りを検出する方法としてモデル検査やデータフロー解析などの静的解析手法が研究されてきた．近年, 高速な SAT ソルバ (充足可能性判定器) の開発が進み，プログラムを表す論理式と仕様を表す論理式を生成することでプログラムの正当性を SAT 問題（充足可能性問題）に帰着させ SAT ソルバで検査する有界モデル検査法が知られるようになった．有界モデル検査も含め, 通常のモデル検査法は, エラー箇所及びその箇所までの実行系列を出力する．しかし, エラーの原因となる箇所は通常その実行系列の中のごく一部であり, 誤りを修正するためには利用者が実行系列の中のエラー原因を探さなければいけないという問題がある．

その問題に対して, Jose らは，充足可能性問題の拡張である最大充足化問題に対するソルバを使って, エラー箇所までの実行系列のうち修正するべきプログラム中の誤り位置の候補を特定する手法を提案した．この手法は, 有界モデル検査と同じ手法で命令列を表す論理式を作り, 潜在的な誤りの位置の検出をプログラムの仕様を満たしつつ命令列に対応する論理式をできるだけ満たすという最大充足化問題に帰着させて Max-SAT ソルバ（最大充足化ソルバ）で解くものである．

しかし，最大充足化問題を厳密に解くアルゴリズムは時間計算量が大きく，大きなソフト

ウェアの誤り位置を特定することは難しい.

そこで本研究では，最大充足化問題の近似解法をこの問題に適用することで高速に誤り位置の候補を特定する手法を提案し, 精度を既存手法と比較する.

キーワード　誤り位置特定, 静的解析, モデル検査, SAT ソルバ, Max-SAT ソルバ

Abstract

Study on approximation algorithms of an error localization method using a Max-SAT solver

Yuji YONEDA

More advanced functions are required for software year by year, software is becoming complicated. Accordingly, it is becoming difficult to find potential errors in programs. In recent years, satisfiability solvers, or SAT-solvers, have been actively developed, and high speed SAT-solvers have become available. The bounded model checking (BMC) is a well-known application of SAT-solvers to software verification. It is done by reducing correctness of programs to a satisfiability problem.

When a given program has an error, model checking tools provide the execution path to the location where an assertion fails. However, usually the real error of that program is in a little part of that execution path, and users have to search for the real error location by the themselves.

Jose et al. proposed a method for locating errors in programs, using a Max-SAT solvers, which is an extension of SAT-solvers. This method reduces error location of programs to a Max-SAT problem. This is done by generating logical expressions representing instruction sequences in the same way as the BMC. However, it is time-consuming to strictly solve the Max-SAT problem, and it difficult to identify the error location of a large software by this method. In this research, we propose a method for quickly identifying error location candidates, by applying approximation algorithms of Max-SAT problems. And then, we compare its accuracy with existing methods.

key words Fault localization, Static analysys, Model Checking, SAT Solver, Max-SAT Solver

目次

目次

図目次

表目次

第 1 章

はじめに

1.1 背景

年々，情報サービスの需要は増えている．20 年と比べると日本の情報サービス産業の年間売上高はおおよそ 2 倍ほどになっている [1]．それに伴いソフトウェアにはより高度な機能が求められ, プログラムは複雑化していき, プログラムに潜在する誤りを発見するのが困難になっている．また, ソフトウェア開発の工程のうち単体テスト・結合テストの割合はおおよそ 3 割を占めている [2]．そのため, ソフトウェアのバグを取り除くのに開発工程のうち多くの時間が費やされており, ソフトウェアの生産性に大きな影響を与えている．そのため, 誤りを検出する方法としてモデル検査やデータフロー解析などの静的解析手法が研究されてきた．

近年, 高速な SAT ソルバ (充足可能性判定器) の開発が進んでいる．SAT ソルバは毎年, 各実装の性能を競う競技会が開催され, 性能を競うことにより改良され続けている．近年は $10^6 \sim 10^7$ 個の変数を持った大規模な SAT 問題を非常に高速に解くことが可能な SAT ソルバが開発されており, プランニング問題や定理証明などの様々な分野に応用されている．また, プログラムを表す論理式と仕様を表す論理式を生成することでプログラムの正当性を SAT 問題（充足可能性問題）に帰着させ SAT ソルバで検査する有界モデル検査法が知られるようになった．

有界モデル検査も含め, 通常のモデル検査法は, エラー箇所及びその箇所までの実行系列を出力する. しかし, エラーの原因となる箇所は通常その実行系列の中のごく一部であり, 誤りを修正するためには利用者が実行系列の中のエラー原因を探さなければいけないという問題がある.

その問題に対して, Jose らは，充足可能性問題の拡張である最大充足化問題に対するソルバを使って, エラー箇所までの実行系列のうち修正するべきプログラム中の誤り位置の候補を特定する手法を提案した. この手法は, 有界モデル検査と同じ手法で命令列を表す論理式を作り, 潜在的な誤りの位置の検出をプログラムの仕様を満たしつつ命令列に対応する論理式をできるだけ満たすという最大充足化問題に帰着させて Max-SAT ソルバ（最大充足化ソルバ）で解くものである.

1.2　目的

Jose らの方法で誤り位置を特定することができるが, SAT 問題に対して最大充足化問題を厳密に解くことは時間計算量が非常に大きい. SAT 問題では最悪の場合は変数の数に対しての組み合わせを計算する必要があり, この組み合わせは指数オーダだが, 最大充足化問題はそれに加えて節の数に対する組み合わせを試さないといけない. そのため, 満たす制約を最大化するために, 最悪の場合膨大な組み合わせを求める必要があり, 時間計算量が大きく，大きなソフトウェアの誤り位置を特定することは難しい. そこで本研究では，最大充足化問題の近似解法をこの問題に適用することで高速に誤り位置の候補を特定する手法を提案し, 精度を既存手法と比較する.

1.3　本論の構成

本論文は, 本章を含め全 5 章で構成されている.

- 第 2 章は提案手法や周辺技術に関する技術と既存手法について説明する.
- 第 3 章は提案手法と提案手法を実装の処理の流れについて説明する.
- 第 4 章は提案手法の既存手法に対する性能を評価するための実験について説明する.
- 第 5 章は第 4 章の結果を受けての考察と今後の課題などについて述べる.

第 2 章

背景技術

2.1 充足可能性問題（SAT）

充足可能性問題（SAT 問題）とは, いくつかの論理変数から構成される論理命題 ϕ が与えられた時に $\phi = true$ となるような変数への値の割り当てが存在するか判定し, 存在すればその割り当ての一つを返すという問題である. なお, 論理変数とそれの否定をリテラルと呼び, リテラルを論理和で繋いだものを節と呼ぶ.

$$\phi_1 = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1) \tag{2.1}$$

$$\phi_2 = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1) \wedge (\neg x_2) \wedge (\neg x_3) \tag{2.2}$$

例えば, 論理命題 ϕ_1, ϕ_2 に対して, 式 (2.1), 式 (2.2) が与えられているとする. 式 (2.1) は $(x_1, x_2, x_3) = (false, true, false)$ の場合には $\phi = true$ となるためこの式は充足可能（SAT）となる.

式 (2.2) の場合は (x_1, x_2, x_3) のどの真偽値の割り当てでも $false$ となる. 従ってこの式は充足不能（UNSAT）となる.

SAT は NP 完全性が最初に証明された問題で, すべての入力列を効率よく解けるようなアルゴリズムが存在しないと予想されている. しかし, 近年では探索方法の改良により, 多くの入力例に対して SAT を高速に解く SAT ソルバと呼ばれるソフトウェアが提供されている.

2.2 SAT ソルバ

多くの SAT ソルバでは, 連言標準形（CNF）で図 2.1 のような形式で論理式を入力する. “c”から始まる行はコメント行で SAT ソルバには無視される. “p”から始まる行は CNF に関する情報 “p cnf [変数の個数] [節の個数]” という形式で記述する. リテラルは整数値で表し, 変数と変数の否定を正負で表す. 節はスペース区切りで並べて記述し, 節の最後に “0” を追加する. 出力は図 2.2 のようになり, 論理式を充足する場合は “SAT” であることと SAT になる場合の各変数への真偽値の割り当てを整数値の正負で表す.

代表的な SAT ソルバとしては minisat, glueminisat, Treengeling などがある [6][7][8].

図 2.1　SAT ソルバへの入力ファイルの例

```
1 c
2 c コメント
3 c
4 p cnf 6 3
5 1 -3 4 0
6 -1 5 2 0
7 6 -4 0
```

図 2.2　SAT ソルバからの出力の例

```
1 SAT
2 -1 2 -3 4 -5 6 0
```

2.3　部分最大充足割当問題

部分最大充足割当問題 (Partial Max-SAT) は SAT 問題を拡張したもので SAT 問題は *true* となるべき節であるハード節からのみなるが, 部分最大充足割当問題は満たさなくても良いソフト節も含む. そして, このハード節をすべて満たしつつソフト節ができるだけ多く満たされるような割り当てを求める.

例として, ϕ_w のようなハード節 2 つと $(c, 1)$ で表されるソフト節 3 つからなる命題論理式を考える.

$$\begin{aligned}\phi_w = &(x_1 \vee x_2 \vee \neg x_3)\\ &\wedge (x_3)\\ &\wedge (x_1 \vee \neg x_3, 1)\\ &\wedge (\neg x_1 \vee \neg x_2, 1)\\ &\wedge (\neg x_3, 1)\end{aligned}$$

ハード節を満たす割り当ては, (x_1, x_2, x_3) に対して

- $(false, true, true)$
- $(true, false, true)$
- $(true, true, true)$

の 3 つで, このうち満たすソフト節の数が最大となるものは $(true, false, true)$ のみになるので, $(true, false, true)$ が正解となる.

この部分最大充足割当問題も高速に解くソルバがある.

2.4 Tseitin 変換

Tseitin 変換は命題論理式を充足同値な CNF に変換する方法である. 例えば, 命題論理式 $\phi = (x_1 \wedge x_2 \wedge ... \wedge x_m) \vee (y_1 \wedge y_2 \wedge ... \wedge y_n)$ を論理的に同値である CNF に変換すると

$$\phi = \bigwedge_{i=1...m, j=1...n} (x_i \vee x_j)$$

となり, mn 個の節ができる. ここで Tseitin 変換を行うと元の論理命題式に比例した大きさの CNF を求めることができる.

Tseitin 変換は以下の処理を再帰的に行うことで実現する.

- 論理式 ϕ 中の部分論理式 $X \wedge Y$ に対して, 新しい変数 α を用意し, ϕ を $\phi' \wedge c(\alpha \leftrightarrow (X \wedge Y))$ に置き換える. ただし, ϕ' は ϕ 中の $X \wedge Y$ を α で置き換えた式, $c(\alpha \leftrightarrow (X \wedge Y))$ は式 $\alpha \leftrightarrow (X \wedge Y)$ を後述の変換によって節の連言で表した式である. また, X と Y は Tseitin 変換済みであるとする.
- 論理式 ϕ 中の部分論理式 $X \vee Y$ に対して, 新しい変数 α を用意し, ϕ を $\phi' \vee c(\alpha \leftrightarrow (X \vee Y))$ に置き換える. ただし, ϕ' は ϕ 中の $X \vee Y$ を α で置き換えた式, $c(\alpha \leftrightarrow (X \vee Y))$ は式 $\alpha \leftrightarrow (X \vee Y)$ を後述の変換によって節の連言で表した式である. また, X と Y は Tseitin 変換済みであるとする.

- 論理式 ϕ 中の部分論理式 $X \rightarrow Y$ に対して，新しい変数 α を用意し，ϕ を $\phi' \vee c(\alpha \leftrightarrow (X \rightarrow Y))$ に置き換える．ただし，ϕ' は ϕ 中の $X \vee Y$ を α で置き換えた式，$c(\alpha \leftrightarrow (X \rightarrow Y))$ は式 $\alpha \leftrightarrow (X \rightarrow Y)$ を後述の変換によって節の連言で表した式である．また，X と Y は Tseitin 変換済みであるとする．
- 論理式 ϕ 中の部分論理式 $\neg X$ に対して，新しい変数 α を用意し，ϕ を $\phi' \vee c(\alpha \leftrightarrow (\neg X))$ に置き換える．ただし，ϕ' は ϕ 中の $\neg X$ を α で置き換えた式，$c(\alpha \leftrightarrow (\neg X))$ は式 $\alpha \leftrightarrow (\neg X)$ を後述の変換によって節の連言で表した式である．また，X と Y は Tseitin 変換済みであるとする．

そして，

$\alpha \leftrightarrow (X \wedge Y)$ は

$$(\neg\alpha \vee X) \wedge (\neg\alpha \vee Y) \wedge (\alpha \vee \neg X \vee \neg Y)$$

$\alpha \leftrightarrow (X \vee Y)$ は

$$(\alpha \vee \neg X) \wedge (\alpha \vee \neg Y) \wedge (\neg\alpha \vee X \vee Y)$$

$\alpha \leftrightarrow (X \rightarrow Y)$ は

$$(\alpha \vee X) \wedge (\alpha \vee \neg Y) \wedge (\neg\alpha \vee \neg X \vee Y)$$

$\alpha \leftrightarrow (\neg X)$ は

$$(\alpha \vee X) \wedge (\neg\alpha \vee \neg Y) \wedge (\alpha \vee X)$$

と変換できることから, 元の論理式に比例したサイズの CNF を求めることができる．

2.5　有界モデル検査

有界モデル検査は SAT ソルバを使ったシステム検証の方法の一つで, プログラムの状態空間の到達可能性を SAT 問題に帰着して検査する．具体的には, 図 2.3 のようなプログラムが与えられた時に, プログラムを静的単一代入形式に変換する．図 2.3 の場合は図 2.4 のようになる．静的単一代入は各変数が 1 回のみ代入される形式である．こうして得られたプロ

図 2.3 検査対象のプログラムの例

```
1  void sub(int n) {
2      if (0 <= n && n <= 10) {
3        n += 5;
4        assert(5 <= n && n < 15);
5    }
6  }
```

図 2.4 検査対象のプログラムの例を静的単一代入形式に変換したもの

```
1  void sub(int n0) {
2      if (0 <= n0 && n 0<= 10) {
3        n1 = n0 + 5;
4        assert(5 <= n1 && n1 < 15);
5    }
6  }
```

グラムの各命令に対して, 各変数が取ることができる値を表す制約を生成する. この例では

$$(0 \leq \mathtt{n0} \wedge \mathtt{n0} \leq 10) \wedge (\mathtt{n1} = \mathtt{n0} + 5)$$

となる.

次に, `assert(5 <= n1 && n1 < 15)` が成り立つ, つまり, この条件外の場合があるか調べる必要がある. よって, プログラムの各変数の制約を満たしつつ条件を

$$(0 \leq \mathtt{n0} \wedge \mathtt{n0} \leq 10) \wedge (\mathtt{n1} = \mathtt{n0} + 5) \wedge \neg(5 \leq 1 \wedge \mathtt{n1} < 15)$$

それを SAT ソルバに入力して結果が SAT（充足可能）であれば仕様を満たさない場合が存在することがわかる. UNSAT（充足不能）であれば, 仕様を満たさない場合が存在しない.

つまり, 必ず仕様を満たすということがわかる.

こうして, 有界モデル検査によってプログラムが仕様を満たすかどうかと開始地点からエラー箇所までの実行経路が得られるが, エラー箇所までの実行経路に含まれる潜在的な誤りはそのごく一部である. そのエラー箇所を推測できるのが Jose らの提案した Max-SAT ソルバによる誤り位置の特定である.

2.6　Max-SAT Solver を使った誤り位置の特定

最大充足化ソルバを用いた誤り位置特定はアサーション違反までの実行系列の中から修正するべき箇所を特定する．この手法では有界モデル検査と同じく命令列を表す論理式を生成する．また, 有界モデル検査の方法と逆でアサーション違反がない場合を SAT とする．各命令の動作を表す制約を $I = (ins_0, ..., ins_{n-1})$, 各命令の位置に対応する変数を $L = (loc_0, ..., loc_{n-1})$ とする．プログラムに満たしてほしい仕様を $assert$ とする．Partial Max-SAT ソルバへ入力する制約は以下のようにする．以下の式の $(loc_i, 1)$ はソフト節であり, 重みが 1 であることを意味している．それ以外の節はハード節である．

$$assert \wedge \Big(\bigwedge_{0 \leq i < n} loc_i \leftrightarrow ins_i \Big) \wedge \Big(\bigwedge_{0 \leq i < n} (loc_i, 1) \Big) \tag{2.3}$$

この論理式を Partial Max-SAT ソルバに入力して L の要素のうち $false$ が割り当てられた位置が誤りである．

第 3 章

提案手法

既存手法では, 正しい命令の最大化つまり満たすソフト節の最大化だったが, 提案手法では 満たすソフト節の最大化ではなく, 閾値 m 以上の個数を満たすという制約を解く. 既存手法でのソフト節の各命令と対応する制約の集合を I とすると以下の式の C のような制約を追加する.

$$Comb_k I = \{i | i \in \mathcal{P}(I) \wedge |i| = k\} \tag{3.1}$$

$$C_m = \bigvee_{J \in Comb_m I} \bigwedge_{i \in J} i \tag{3.2}$$

最適化問題でなく制約問題を解くため既存手法より高速に誤り位置をある程度特定できることが期待できる.

以下, 提案手法を実現するシステムの構成と処理の流れを述べる.

3.1 システム構成

3.1.1 ツール

提案システムを構成するに当たって以下のソフトウェアを用いた.

- NodeJS (JavaScript 実行環境)：検査対象のプログラム用の言語内 DSL の表現と, 制約式の生成, 制約式から CNF への変換, 逆変換に使用する.
- GlueMinisat 制約式の解を得るために使う.

3.2　処理の流れ

処理は以下の流れで行う.

- アサーションを含む検査対象のプログラムを JavaScript の言語内 DSL で表現された図 3.1 のようなプログラムを用意する. 検査対象のプログラミング言語は論理演算, 比較演算, 加減算, 条件分岐, 代入の機能を備える.
- その検査対象のプログラムを評価すると木構造で表された制約式が得られる. また, この制約式に上の式で示した C_m の制約を加える. この時に元の変数と論理変数との対応関係を保存する.
- その制約式に対して Tseitin 変換を再帰的に適用する. この際, SAT ソルバに CNF ファイルで入力するため, 論理変数と変数の番号との対応関係を保存する.
- CNF ファイルを SAT ソルバに入力し, 制約式を充足する番号で表された論理変数への値の割り当てを得る.
- 番号で表された論理変数への割り当てと論理変数と変数との対応関係から, 元の論理変数へ値の割り当てを得る.
- 論理変数への値の割り当てから, 既存手法でのソフト節の各命令と対応する集合 I の解が得られる. I の解の要素のうち $false$ となっている部分に対応する命令を誤り箇所とする.

3.3　プログラムの命令から制約式への変換規則

検査対象のプログラムから制約式への変換規則を示す.

32bit 整数の変数・定数 A, B, X を $A = (a_0, a_1, ..., a_{31})$, $B = (b_0, b_1, ..., b_{31})$, $X = (x_0, x_1, ..., x_{31})$ と列として定義する.

図 3.1 検査対象のプログラムの入力ファイルの例

```
1 (() => {
2 let ret = and(
3   // if (0 <= n0 && n0 < 10) {
4   ift(
5     and(le(cv(0), 'n0'), lt('n0', cv(10))),
6     //  n += 5;
7     add('n1', 'n0', cv(5)),
8     //  assert(10 <= x1 && x1 < 21);
9   // }
10  )
11 );
12
13 ret =
14   and(ret,
15     asserts(le(cv(5), 'x1'), lt('x1', cv(15))));
16
17 return ret;
18 })();
```

3.3.1 $A = B$

$A = B$ を表す式は以下の制約式で表す. これは単純に各 bit に対応する論理値が等しいかを表現する.

$$A = B \equiv \bigwedge_{i=\{0\ldots31\}} (a_i \leftrightarrow b_i)$$

3.3.2　$A > B, A \geq B$

$A > B, A \geq B$ を表す式は以下の制約式で表す．A, B の各ビットを上位から見ていって，A の方が大きいか判定する論理式を表現する．

$A \geq$ は先ほど示した規則を用いて $(A = B) \wedge (A > B)$ とする．

一時変数として T, E を導入する．

$$
\begin{aligned}
T &= (t_0, t_1, ..., t_{31}) \\
E &= (e_0, e_1, ..., e_{31}) \\
A > B \equiv\ & (t_{31} \leftrightarrow (t_{31} \wedge \neg t_{31})) \\
& \wedge (e_{31} \leftrightarrow (a_{31} \leftrightarrow b_{31})) \\
& \wedge \bigwedge_{i=\{30...1\}} (t_i \leftrightarrow (e_{i+1} \wedge (a_i \wedge \neg b_i))) \\
& \wedge \bigwedge_{i=\{30...1\}} (e_i \leftrightarrow (e_{i+1} \wedge (a_i \leftrightarrow b_i))) \\
& \wedge (t_0 \leftrightarrow (e_1 \wedge (a_i \wedge \neg b_i))) \\
& \wedge \bigwedge_{i=\{0...31\}} t_i \\
A \geq B \equiv\ & (A = B) \wedge (A > B)
\end{aligned}
$$

3.3.3　$X = A + B$

$X = A + B$ は以下の論理式で表す．全加算器を表す論理式と繰り上がりを表す一時変数 C を導入することで 32bit の加算器の論理回路を表現する．

$$fa_s(a, b, c) = a \otimes b \otimes c$$
$$fa_c(a, b, c) = (a \wedge b) \vee ((a \otimes b) \wedge c)$$

一時変数として C を導入する．

$$\begin{aligned} C &= (c_0, c_1, ..., c_{31}) \\ X = A + B &\equiv (x_0 \leftrightarrow (a \otimes b)) \\ &\quad \wedge (c_0 \leftrightarrow (a_0 \wedge b_0)) \\ &\quad \wedge \bigwedge_{i=\{0...30\}} (x_i \leftrightarrow fa_s(c_{i-1}, a_i, b_i)) \\ &\quad \wedge \bigwedge_{i=\{0...30\}} (c_i \leftrightarrow fa_c(c_{i-1}, a_i, b_i)) \\ &\quad \wedge (x_{31} \leftrightarrow fa_s(c_{30}, a_{31}, b_{31})) \end{aligned}$$

第 4 章

実験結果

4.1 評価方法

提案システムの検出精度を評価するために以下のような実験を行う．正しいプログラムにアサーションを満たさない原因となる誤りをランダムに入れた 30 命令のプログラムを複数用意する．そのプログラムに対して提案手法を適用し, またその計算にかかった時間で既存手法の検査を中断し近似解を求めたものと比較することで評価を行う．SAT ソルバには glueminisat 2.2.8, Partial Max-SAT ソルバには MaxHS 2.9.0 を用いた．

具体的にはまず, プログラム P は n 個の命令からなるとする．プログラム P のアサーションを満たそうとした場合に各命令が存在可能か存在不可能かを真偽値で表した列 I を考える．存在不可能な命令が，検出された誤り位置である．デバッグの容易さから考えて出力する誤り位置の範囲は狭い方が良い．従って, 列 I の中の $false$ である要素を最小化した列 I_{min} に近いものが求まることが好ましい．列 I_{min} は既存手法によって求まる．このとき, I_p を提案手法を適用して I_{min} の近似値を求めたもの．I_p は, m 個以上の $true$ を含む．I_e を既存手法を使うが提案手法を適用した場合にかかった時間で計算を中断させ I_{min} の近似値を求めたもの．とする．そして, 既存手法と提案手法の近さは I_{min} と I_p の間で真偽値の異なる命令の数, 既存手法と既存手法で計算を打ち切る方法の近さは I_{min} と l_e の間で真偽値の異なる命令の数として, 近さを比較する．

表 4.1 提案手法と既存手法の最適解との近さの比較

m	1	2	3	4	5
提案手法	13.3	11.6	12	10.3	10
既存手法で提案手法と同じ時間処理した場合	-	-	-	0	0

4.2 結果

実験結果は表 4.1 のようになった．表で “–” となっている部分は, Max-SAT ソルバが打ち切った時間でハード節を満たせてないため近似解が出力されない場合である．

この結果から, m が小さい間は提案手法で Max-SAT ソルバより高速に処理できることがわかる．しかし, m が 4 以上になると厳密解を出す方法の方が早くなる．これは, 30 命令のうち m 個を選ぶ組み合わせを使って制約を構成したため, 指数関数的に節が増えたからだと考えられる．

この問題は m 個以上の命令が正しいかに対応する変数が $true$ であるものの個数の管理を基数で行い制約を作ることで解決できると思われる．

第 5 章

まとめ

本研究では, プログラムの誤り位置を検出する Jose らの手法に対する近似的な解法を提案した. 提案手法は本来最適化問題として解くところを既存手法のソフト節に対応する節において閾値以上の個数を満たすという妥協した条件の制約問題として解くことである程度の精度で高速に計算することを目指した.

その結果, 閾値が小さい間は既存手法よりも高速に解けることがわかったが, ある一定以上になると厳密解を求める方法より遅いことがわかった.

本研究では, まだ検討が十分とは言えないが, 多くの問題点や課題が明らかになった. 以下, 現状でわかっている問題点や課題を示す.

$m = 3$ から $m = 4$ の間で Max-SAT ソルバが全く解けない状態からいきなり, 最適解が求まるということが起きている. なぜ, このようなことが起きたのか入力ファイルの種類・規模を増やして調査する予定である.

m 個以上の命令が正しいことを表現する制約の節の数が m の指数より大きいオーダで増えるため m が小さい間しか求められないという問題があることがわかった. m が小さい場合はあまり実用にならないため, 今後デバッグツールなどに応用していくことを考えるとより良い制約を考える必要がある. 例えば, m の個数を基数で管理し, 正しい命令位置の個数を加算器のような論理式を組み合わせて表すことでこの問題を克服できると思われる.

現時点では SAT ソルバ 1 種類, 最大充足化ソルバ 1 種類で評価したが, 今後は様々なソルバを組み合わせて評価する予定である.

また, 提案システムでは制約式に対する簡単化・最適化を全く行っていなかったため, 各種最適化を行った場合に結果がどうなるかも評価したい.

この制約を改良して, 改めて実験により評価することが今後の課題である.

謝辞

本研究を行うにあたり, 様々な面で多くのご指導を頂きました主指導教員である本学情報学群高田喜朗准教授に深く感謝いたします.

また, 副査を引き受けてくださった鵜川始陽准教授, 松崎公紀准教授に心より感謝いたします. 最後に, 有益なコメントを頂きました高知工科大学高田研究室の皆様に深く感謝いたします.

参考文献

[1] 情報処理推進機構,“情報サービス・ソフトウェア産業の現状”, https://jinzaiipedia.ipa.go.jp/casestudies-2/it_efforts/about_itedu/itservice_genjyo (2018/2/2).

[2] 情報処理推進機構, ソフトウェア・エンジニアリングセンター,“ソフトウェア開発データ白書 2010-2011 2584 プロジェクト 定量データ分析で分かる開発の最新動向,” p.204 (2011). https://jinzaiipedia.ipa.go.jp/casestudies-2/it_efforts/about_itedu/itservice_genjyo.

[3] 井上克巳，田村直之 “特集「最近の SAT 技術の発展」にあたって,” 人工知能学会誌, 25 巻 1 号, p.56 (2010).

[4] 土屋達弘,“充足可能性判定を利用したモデル検査,” コンピュータソフトウェア, 29 巻 1 号, pp.19–29 (2012) .

[5] M.Jose, R.Majumder, “Cause Clue Clause: Error Localization using Maximum Satisfiability,” PLDI’11, pp.437–446.

[6] E.Niklas, S.Niklas, “MiniSat,” http://minisat.se/ (2017/3/23).

[7] “GlueMiniSat: A SAT solve based on glue clauses and agressive dynamic restart,” http://glueminisat.nabelab.org/ (2018/1/23).

[8] “Lingeling, Plingeling and Treengeling”, http://fmv.jku.at/lingeling/ (2018/1/31).