MiniSat と遺伝アルゴリズムを組み合わせた短い証明の作成

和田 翔太

2024 年 12 月 27 日

1 概要

2 導入

3 準備

3.1 SAT

SAT とは命題論理式の充足可能性を判定する問題で Satisfiability Problem の頭 3 文字を取って SAT と呼ばれる。これは与えられた命題論理式を真にするような各変数への割り当てが存在するかどうか (充足可能かどうか) 判定するという問題である。命題論理式を真にするような各変数への割り当てが存在する場合は充足可能 (SAT), 各変数にどのような割り当てをしても全体が真にならない場合は充足不能 (UNSAT) となる。通常 SAT である場合はその解 (各変数への割り当て) を出力する。

3.1.1 定義

真を表す $\top$ または偽を表す $\bot$ を値にとる変数

$$x_1, x_2, ...$$

を命題変数と呼ぶ。命題変数 x_1 は 1 変数からなる論理式となる。命題論理式はこれらの命題変数を組み合わせることで表現される。

次に命題変数 $x_1, x_2, ...$ に対して操作を表す記号 $\vee, \wedge, \neg$ を導入する。これらの記号は論理演算子と呼ばれ、それぞれ論理和、論理積、否定と呼ばれる。

命題変数 x_i と x_j の論理和を $x_i \vee x_j$ で表す。x_i と x_j がどちらも偽である場合に偽になり、それ以外の場合は真となる。また、命題変数 x_i と x_j の論理積 $x_i \wedge x_j$ で表す。x_i と x_j がどちらも真である場合に真になり、それ以外の場合は偽となる。

$$x_i \vee x_j = \begin{cases} \bot & (x_i = x_j = \bot \text{の場合}) \\ \top & (\text{それ以外の場合}) \end{cases}, x_i \wedge x_j = \begin{cases} \top & (x_i = x_j = \top \text{の場合}) \\ \bot & (\text{それ以外の場合}) \end{cases}$$

命題変数 x に対してその否定を $\neg x_i$ で表す。x_i が真である場合に $\neg x_i$ は偽になり、x_i が偽である場合に $\neg x_i$ は真となる。

$$\neg x_i = \begin{cases} \bot & (x_i = \top \text{の場合}) \\ \top & (x_i = \bot \text{の場合}) \end{cases}$$

上記の論理演算子を命題変数と組み合わせたものは論理式となる。例えば命題変数 x_i, x_j に対して $\neg x_i$ は 1 変数からなる論理式に、$x_i \vee x_j, x_i \wedge x_j$ はそれぞれ 2 変数からなる論理式になる。

論理式 X_i, X_j に対しても命題変数と同様に論理式の論理和 $X_i \vee X_j$, 論理積 $X_i \wedge X_j$, 否定 $\neg X_i$ を定義することができる。

上記の 3 つの論理演算子とは他の演算子として排他的論理和 $\oplus$, 含意 $\rightarrow$, 同値 $\leftrightarrow$ があるが、今回の SAT の問題を表現する際に用いないので省略する。実際には排他的論理和 $\oplus$, 含意 $\rightarrow$, 同値 $\leftrightarrow$ は全て論理和 $\vee$, 論理積 $\wedge$, 否定 $\neg$ を用いて表現できるため、6 つ論理演算子で表現できる論理式は全て論理和 $\vee$, 論理積 $\wedge$, 否定 $\neg$ を用いて表現できる。

SAT の問題は通常、連言標準形 (conjunctive normal form; CNF) と呼ばれる特定の形式の論理式で表現される。命題変数 x_i に対して命題変数 x_i または命題変数の否定 $\neg x_i$ をリテラルと呼ぶ。リテラル $l_1, l_2, ...l_n$ に対してそれらを論理和 $\vee$ でつないだ論理式

$$l_1 \vee l_2 \vee ...l_n$$

を節と呼ぶ。SAT の問題 (CNF 式)F はこの節 $C_1, C_2, ...C_m$ を論理積 $\wedge$ で繋いだ論理式

$$C_1 \wedge C_2 \wedge ... \wedge C_m = (l_{11} \vee l_{12} \vee ...l_{1n_1}) \wedge (l_{21} \vee l_{22} \vee ...l_{2n_2}) \wedge ... \wedge (l_{m1} \vee l_{m2} \vee ...l_{mn_m})$$

で表現される。例えば論理式 $(x_1 \vee x_2) \wedge (\neg x_3 \vee x_4)$ は CNF 式であるが、論理式 $(x_1 \wedge x_2) \vee (\neg x_3 \vee x_4)$ は CNF 式ではない。また節をリテラルの集合で表し、CNF 式を節の集合で表すことがある。先ほどの例の CNF 式 $(x_1 \vee x_2) \wedge (\neg x_3 \vee x_4)$ は $\{\{x_1, x_2\}, \{\neg x_3, x_4\}\}$ といった集合で表現される。

最後に充足可能、充足不能について定義する。命題変数についてその割当 (未割当も含めた) を写像

$$\nu : X \to \{\top, \bot, u\} (X \text{ は命題変数の集合とし、} u \text{ は未割当であることを表す})$$

で定義する。この写像 ν をリテラル l, 節 $C = \{l_1, l_2, ..., l_n\}$, CNF 式 $F = \{C_1, C_2, ..., C_m\}$ についても割当ができるように以下のように拡張する。

$$\nu(l) = \begin{cases} \nu(x) & (l = x \text{ の場合}) \\ \neg\nu(x) & (l = \neg x \text{ の場合}) \end{cases} (\text{ただし} \neg u = u \text{ とする})$$
$$\nu(C) = \nu(l_1) \vee \nu(l_2) \vee ... \vee \nu(l_n)$$
$$\nu(F) = \nu(C_1) \wedge \nu(C_2) \wedge ... \wedge \nu(C_m)$$

CNF 式 F に対して充足可能 (SAT) とはある割当 ν が存在して

$$\nu(F) = \top$$

となることをいう。そして CNF 式 F に対して充足不能 (UNSAT) とは任意の割当 ν に対して

$$\nu(F) = \bot$$

となることをいう。

3.1.2 例

- $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$
  - $x_1 = \top$, $x_2 = \bot$ とすると全体が真となるので SAT となる。
- $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$
  - x_1, x_2 に $\top$, $\bot$ どちらを代入した場合でも全体は常に偽になるため UNSAT となる。

3.2 SAT solver

充足可能性問題を解くソルバーのことを SAT solver と呼ぶ。充足可能性問題は NP 完全に属し、解くのに時間がかかる。一番愚直な解き方として、各変数に $\top$ と $\bot$ を代入して全体が真になるかどうかを判断する方法がある。しかしこの場合最悪 2^n 回確認をしなければならない。そのため、速く解くために様々な手法が用いられる。

3.2.1 DPLL アルゴリズム

DPLL(Davis-Putnam-Logemann-Loveland) アルゴリズムとは単位伝搬 (Unit propagation) を用いたアルゴリズムのことである。CNF 式の命題論理式は各節が論理積で繋がった形をしているため、全体が真となるためにはその各節が真とならなければならない。もし仮にある節において、1 つのリテラルのみが未定義でその他のリテラルが偽になる場合 ($x \wedge \bot \wedge \bot ...$)、その節が真になるためには未定義のリテラルが真にならなければならない。このようにして未定義のリテラルが真になるように割り当てを行うことを単位伝搬という (またそのような節を単位節と呼ぶ)。DPLL アルゴリズムは以下のような流れで問題を解いていく

1. 単位節がある場合、それを充足するように変数に割り当てを行う (単位伝搬)
2. 単位節がない場合、適当な変数を選択し真または偽を割り当てていく
   - 割り当てによって単位節ができた場合、単位伝搬を行う
   - 矛盾が発生した場合、最後に選択していた変数の真偽値を反転 (すでに反転していた場合はその 1 つ前の変数を反転) させる
3. 全体が真になった場合は SAT(とその割り当て) を、すべての割り当てにおいて全体が偽になった場合は UNSAT を返す

3.2.2 変数選択における重み

DPLL アルゴリズムにおいて変数選択をする際に未割当の変数が複数ある場合、どの変数を選択すべきかという問題がある。一番シンプルな方法としてランダムに選ぶ方法もあるが、実際にはどのようにして変数を選択するかが決まっている。多くのソルバーにおいてはあるルールに従って各変数に重み (スコア) をつけることで変数選択をする時に重みが一番大きい変数を選択するようになっている。後述するソルバーの 1 つである minisat においては、矛盾が発生した際にその矛盾を引き起こすのに関わった変数 (その時に作った学習節に含まれる変数) のスコアが上がるように計算することで、直近の矛盾に関わった変数ほどスコアが高くなっている。変数選択の際には一番重みが大きい変数を選択している。

3.2.3 矛盾からの節学習 (CDCL solver)

DPLL アルゴリズムに基づいた高速 SAT solver の大きな特徴であり、これを用いた solver はしばしば CDCL(conflict-driven clause learning) solver と呼ばれる。節学習の流れとして、まず、DPLL アルゴリズムにおいて矛盾が起きた際にどの変数への割り当てが矛盾を引き起こしたのかなどの原因を調べる。そして今後変数割当や単位伝搬を進めていく中でさきほどの矛盾を引き起こした変数らへの割当を防ぐために新しい節を学習節として新しく節を加える。これを続けていくことで DPLL アルゴリズム単体で解の探索をするよりも高速に探索をすることができる。

3.2.4 監視リテラル

SAT solver の実行時間の 70% から 90% は単位伝播の処理で占められているため、効率良く単位節を検出することができればより高速に解を探索することができる。素朴な方法として各変数 x_i に対して x_i を含む節のリストを用意しておき、x_i に値が割り当てられた時にそのリストを走査することで状態の変化した節を確認する方法がある。しかしこの方法だとすでに充足されている節の確認を行う必要があり無駄が多くなってしまう。監視リテラルを用いた方法は節の中の未割当な変数 2 つを監視する方法である。節が単位節になる直前

の状態は節中のリテラルのうち 2 つのみが未割当で残りのリテラルに偽が割り当てられている状態となっている。どちらかのリテラルに偽が割り当てられた時に節は単位節となるため、節中の全てのリテラルを監視する必要はなく未割当のリテラル 2 つのみを監視するだけで効率良く単位節を検出することができる。

3.3 DRAT

DRAT(Deletion Resolution Asymmetric Tautology) とは証明の表記法の 1 つであり、DRUP(Deletion Reverse Unit Propagation) と呼ばれる証明の表記法を一般化したものである。数学の問題を証明する際にいくつかの補題を組み合わせて証明をするように、DRAT の証明も主にいくつか補題を表す節を並べることで作成される。DRAT の証明の各行は、補題を表す節かそれまでにある特定の節を削除する意味を持つ削除節からなり、最後の行は矛盾を表す長さ 0 の節 (リテラルを 1 つも持たない節) となっている。ここで証明の検証方法を説明するためにいくつか変数を取り入れる。CNF 形式の問題を F、CNF 形式の証明を P として、証明が持つ節の数 (証明の行数) を $|P|$ とする。$i \in \{0, 1, ..., |P|\}$ に対して F_P^i を次のように定義する。ただし、L_i を証明 P における i 行目の節とする。

$$F_P^i = \begin{cases} F & (i = 0 \text{ の場合}) \\ F_P^{i-1} \setminus \{L_i\} & (L_i\text{が削除節の場合}) \\ F_P^{i-1} \cup \{L_i\} & (\text{それ以外}) \end{cases}$$

証明を検証する際には、証明の各行の節を RUP チェックと RAT チェック両方で検証を行いその節を導くことができるかを検証する。

RUP チェック　F_P^{i-1} に関して L_i を単位伝搬のみで導出することができる ($F_P^{i-1} \cup \{\overline{L_i}\}$ について単位伝搬のみで矛盾を導くことができる)

RAT チェック　F_P^{i-1} に関して L_i が l_i について以下の性質を持っているか

- 任意の $C \in F_P^{i-1}$ について $\bar{l}_i$ が C に含まれる場合、F_P^{i-1} に関して $(C \setminus \bar{l}_i) \cup L_i$ を単位伝搬のみで導出することができる。

実際に検証するツールとして drat-trim が存在する。このツールのもう 1 つの特徴として元の証明を短くすることができる。検証を行う際にもとの証明の中で使用されなかった節を削除する方法で元の証明よりも短い証明を出力している。

3.3.1 学習節と DRAT

CDCL ソルバーが作成する学習節はこの DRAT の証明になっている。先ほど述べたように、学習節は以降の探索の際に矛盾を引き起こした割当になることを防ぐ役割を持っている。例えば学習節 C が $l_1 \vee l_2 \vee ... \vee l_n$ を形をしているとする。この学習節が作られた際には $l_1 = \bot, l_2 = \bot, ..., l_n = \bot$ という割当によって矛盾が引き起こされていたことがわかる ($\overline{C} = \bar{l}_1 \wedge \bar{l}_2 \wedge ... \wedge \bar{l}_n$)。つまり $\overline{C}$ を仮定した時に単位伝搬のみで矛盾を導くことができるため、RUP チェックによってこの学習節を導くことができる。したがって、CDCL ソルバーが問題を解く際に作成されていく学習節を並べていくと問題が UNSAT である時の DRAT の証明になっている。この作り方において、証明の中に削除節は存在しない (後述する今回の実験で使うソルバーにおいては削除節が作成される)。

3.3.2 証明の長さ

実験の目標とする短い証明を作ることについて、ここでは証明の長さを証明の節の中で補題を表す節の数で表す。削除を表す節はそれ自体が存在しなくても証明として成り立つため、削除を表す節については考慮しない。

3.4 minisat

SAT ソルバーの 1 つであり、今回証明を作成するのに使用しているソルバーである。

特長の 1 つとして詳細を理解しやすいという点がある。既存の最先端のソルバーを改良することは、仮に問題領域や SAT ソルバーに関する技術を深く理解していても 10000 行を超えるプログラムを理解しなければならず、極めて時間のかかる作業になってしまう。同様にゼロからソルバーを構築しようとしても多くの時間を費やす必要がある。このように時間がかかってしまう理由としては、現代のソルバーで用いられている技術は十分に文書化されている一方で、実装に必要な詳細が十分に提示されていないことが挙げられる。

これに対して minisat は

- プログラム全体のコード行が 6000 行と比較的少ない
- 矛盾からの節学習、監視リテラル、変数への重み付けといった既存のソルバーに多く採用されている技術が使用されている
- 実装の詳細について説明した論文が存在する
- オープンソースである

ため、改造がしやすかったり独自のソルバーをゼロから構築しやすくなっている。

今回は変数選択へ介入を行うため、ソルバーの改造が必要であるという理由から、minisat を使用した。

3.5 遺伝アルゴリズム

遺伝アルゴリズム (G.A.: genetic algorithm) とは、John Holland によって発明された、最適化アルゴリズムのことである。このアルゴリズムは自然進化に見られる過程のいくつかを模倣して構築されている。この進化は染色体と呼ばれる生物の構造を符号化するための有機的な装置上で生じており、1 個体の生物は染色体を翻訳する過程により作られる。染色体のコード化と翻訳の過程の特徴として、

- 進化は生命体そのものにではなく、それを符号化した染色体に対して操作を行う過程からなる
- 自然淘汰は、染色体とそれをデコードした結果できた構造体の環境への適応度を結びつける。この過程により、より適応した染色体はより頻繁に再生に用いられる。
- 再生と呼ばれる今ある染色体から新しい染色体を生成する操作の過程で、進化が起きると考えられる。突然変異と呼ばれる操作によって親のものとは異なった染色体が生成され、組み替えの過程を通して両者の染色体の構成物質が組み合わされてどちらの親の特徴を持つ全く異なった染色体が作られる。

といった特徴がある。

4　実装

4.0.1　証明を作る

- 前処理
- 学習節の追加

4.0.2　変数選択への介入

4.0.3　遺伝アルゴリズム

- オペレータ
  - 交叉
  - 突然変異
- その他パラメータ

5　問い

6　実験

7　結果

8　考察

9　まとめ