SMT 問題の構造と

SMT ソルバの効率の相関について

指導教員 : 寺内　多智弘
早稲田大学　基幹理工学部
情報理工学科　寺内研究室
1w153084-8
竹井　友利
2021 年 2 月 1 日

要旨

充足可能性問題（satisfiability problem, SAT 問題）とは、命題論理式が与えられたとき、それに含まれる命題変数の値を真（true）,偽(false)に割り当てることで、命題論理式の値を真にできるか、という問題である。SAT ソルバとは、SAT 問題の充足可能性を判定するものである。
近年の研究[1]で、SAT ソルバの問題を解く効率（SAT ソルバの実行時間）と SAT 問題からつくるグラフのモジュラリティの間の相関性が指摘されている。

本研究では、SAT 問題で確認されている相関性が、SMT 問題に関しても確認できるかどうかを検証した。SMT 問題（Satisfiability Modulo Theories）とは、SAT 問題の命題変数を述語論理に置き換えたものである。

実験の結果、SAT 問題で指摘されている相関性が SMT 問題に関しても確認できた。

目次

1. はじめに

現実世界の様々な問題（ソフトウェア・ハードウェア検証、プランニング）を SAT/SMT 問題へ置き換えて、ソルバを用いて解くアプローチが普及している。[2] そのため、SAT/SMT 問題の実行速度と問題の構造の相関性について分析するのは有意義でないかと考える。

本研究では、既存研究で確認されている SAT 問題からつくるグラフのモジュラリティと SAT ソルバの実行時間の相関性が、SMT 問題についても、確認できるかを検証する。

2. 準備

2.1. SAT 問題[2][3]

充足可能性問題（satisfiablity problem）とは、1 つの命題論理式が与えられたとき、その式を満たす、論理式全体が真になるようなそれぞれの命題変数への真偽の割り当てが存在するかということをいう。

例.

$$(A \vee B) \wedge (C \vee A) \wedge C$$

A = false, B=true, C = true を代入すると、論理式全体は真になる。よってこの式は充足可能である。

2.2. SAT ソルバ[2][3]

SAT 問題を受け取り、SMT 問題の充足可能性を判定するためのプログラムである。
SAT ソルバは、連言標準形（Conjunctive Normal Form, CNF）の SAT 問題のみを入力として受け取る。

2.3. SMT 問題[2][3]

SMT 問題（Satisfiability Modulo Theories）とは、SAT 問題が命題論理を扱うのに対して、述語論理を扱う問題のこと。したがって、SMT 問題は関数や算術演算子を扱うことができるため、SAT 問題より豊かな表現力を持つ。

例.

$$\big((a + 2) > 0\big) \wedge ((b < 5) \vee (a < 4))$$

a = 3, b = 6 を代入すると、論理式全体は真になる。よってこの式は充足可能である。

2.4. SMT ソルバ[2][3]

SMT ソルバは SMT 問題の充足可能性を判定するためのプログラムである。SMT ソルバのアプローチとして Eager,Lazy の 2 種のアプローチがある。

Eager アプローチは、SMT 問題をすべて命題論理の充足問題に変換して全体を SAT ソルバで解く手法である。このアプローチは、各理論の論理式を、充足可能性が等価である命題論理の論理式に変換する必要がある。この変換には、論理毎の技法が必要で、必ず充足可能性を維持した変換ができるとは限らない。また、変換された命題論理式が長くなる可能性がある。

Lazy アプローチは、SAT ソルバと理論ソルバを利用して SMT 問題を解くという手法である。一般的な SMT ソルバは Lazy アプローチである。本実験で用いた SMT ソルバもこの Lazy アプローチである。

2.5. Tseitin 変換[4]

Tseitin 変換は 1968 年に G. S. Tseitin によって導入された充足同値な変換方法である。この変換方法により、比例したサイズの CNF を求めることが可能である。Lazy アプローチの SMT ソルバは CNF の論理式のみを入力として受け取る SAT ソルバを用いるので、SMT 問題が CNF でないときにこの変換を行っている。

まず、$(p_1 \wedge p_2 \wedge \ldots \wedge p_m) \vee (q_1 \wedge q_2 \wedge \ldots \wedge q_n)$ を A で表す。

$(p_1 \wedge p_2 \wedge \ldots \wedge p_m)$を新たな命題変数 r で置き換え、同時に r と$(p_1 \wedge p_2 \wedge \ldots \wedge p_m)$が同値であることを表す論理式を追加する。同様に、$(q_1 \wedge q_2 \wedge \ldots \wedge q_n)$についても新しい命題変数 s に置き換えると、以下の論理式 A'が得られる。

$$(r \vee s) \wedge \big(r \leftrightarrow (p_1 \wedge p_2 \wedge \ldots \wedge p_m)\big) \wedge (s \leftrightarrow (q_1 \wedge q_2 \wedge \ldots \wedge q_n))$$

さらに、変形すると、

$$(r \vee s)$$
$$\wedge (\mathrm{r} \vee \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_m) \wedge (\neg r \vee p_1) \wedge \ldots \wedge (\neg r \vee p_m)$$
$$\wedge (\mathrm{s} \vee \neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_n) \wedge (\neg s \vee q_1) \wedge \ldots \wedge (\neg s \vee q_n)$$

となる。

例.

n=100, m=100 のとき

$(r \vee s)$

$\wedge (\mathrm{r} \vee \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{100}) \wedge (\neg r \vee p_1) \wedge (\neg r \vee p_2) \wedge \ldots \wedge (\neg r \vee p_{100})$

$\wedge (\mathrm{s} \vee \neg q_1 \vee \neg q_2 \vee \ldots \vee \neg p_{100}) \wedge (\neg s \vee q_1) \wedge (\neg s \vee q_2) \wedge \ldots \wedge (\neg r \vee p_{100})$

となり、選言節は合計:1 + (1+ n) + (1 + m) = 203 個となる。

一方、分配法則の場合、選言節の合計は nm 個になるので 10000 個となる。

2.6. コミュニティ構造

コミュニティ構造とは、グラフ中のノードがいくつかのまとまりに分割したものを示す。高いコミュニティ構造を持つとは、各コミュニティ内では密なつながりを持ち、コミュニティ外とのつながりが疎である構造を持つということ。

2.7. モジュラリティ[5]

モジュラリティは、ネットワークの与えられた分割に対して、「同じグループg_iに属するノード同士が繋がるリンク数の全リンク数に占める割合」から「リンクがランダムに配置された場合のグループg_i内のリンク数の全体のリンク数に対する割合の期待値」を引いた値として定義される。

ノードの数 N、リンクの数 M のネットワークを考える。ノードを$\{g_1, g_2, \dots, g_C\}$のように C 個のグループに分ける。ネットワークの隣接行列を A として、行列成分A_{rs}はノード r,s 間に存在するリンク数とする。なお、本実験では、無向グラフかつリンクへの重みづけを考えないので、A の各成分は 1 か 0 のみになり、A は対称行列となる。したがって、隣接行列の成分の和は

$$\sum_{rs} A_{rs} = 2M$$

となる。

また、g_iに属するノードと、g_iに属するノードがつながるリンク数の合計の全リンク数に占める割合は以下のように書ける。

$$e_{ij} = \sum_{r \in g_i} \sum_{s \in g_j} \frac{A_{rs}}{2M}$$

したがって、「同じグループg_iに属するノード同士が繋がるリンク数の全リンク数に占める割合」はe_{ii}で表せる。

次に、「リンクがランダムに配置された場合のグループg_i内のリンク数の全体がリンク数に対する割合の期待値」について考える。

まず、M 本のリンクを半分に切り、2M 本の「手」（リンクの半分）を作る。次に、ノードに接続した「手」をランダムに 2 つずつ選択してつなぎなおす。ここで、ノード r の「手」の数をノード r の次数と呼び、以下のように表せる。

$$k_r = \sum_{s} A_{rs}$$

全ノードの次数の合計は

$$\sum_{r} k_r = \sum_{rs} A_{rs} = 2M$$

となる。上記のようにランダムにリンクをつなぐと、リンクの一方にg_j内のノードの「手」が選ばれる確率a_iは以下のように書ける。

$$a_i = \sum_{r \in g_i} \frac{k_r}{2M} = \sum_{r \in g_i} \sum_{s} \frac{A_{rs}}{2M} = \sum_{j} \sum_{r \in g_i} \sum_{s \in g_j} \frac{A_{rs}}{2M} = \sum_{j} e_{ij}$$

これは、少なくとも一方がg_j内のノードにつながれるリンク数の全リンク数に占める割合である。リンクの両端がg_j内のノードである場合の期待値は、a_i^2となる。

従って、モジュラリティは以下のように定義できる。

$$Q = \sum_{i} (e_{ii} - a_i^2)$$

例.

図 2.1 のようなネットワークがあったとするとモジュラリティは以下のように求める。

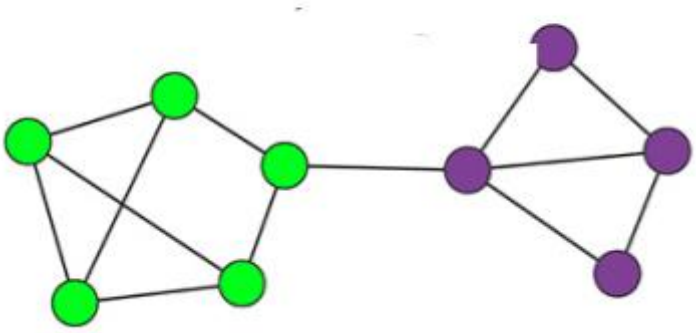

図 2.1 ネットワークの例

まず、緑のグループをg_1、紫 n グループをg_2とする。

するとモジュラリティ Q は

$$Q = \sum_{i} (e_{ii} - a_i^2) = (e_{11} - a_1^2) + (e_{22} - a_2^2)$$

となる。それぞれ求めていくと

$$e_{11} = \sum_{r \in g_1} \sum_{s \in g_1} \frac{A_{rs}}{2M} = \frac{2 * 7}{2 * 13}$$

$$a_1 = \sum_{r \in g_1} \frac{k_r}{2M} = \frac{3+3+3+3+3}{2 * 13} = \frac{15}{26}$$

$$e_{22} = \sum_{r \in g_2} \sum_{s \in g_2} \frac{A_{rs}}{2M} = \frac{2 * 5}{2 * 13}$$

$$a_2 = \sum_{r \in g_2} \frac{k_r}{2M} = \frac{2+2+3+3}{2 * 13} = \frac{5}{13}$$

$$Q \doteqdot 0.41$$

となり、このようにモジュラリティは求められる。

3. SMT ソルバの効率とモジュラリティの相関性

3.1. 実験手法

SMT ソルバが SMT 問題を解く時間を計測する。SMT 問題からグラフを作成して、そのグラフのモジュラリティを求める。これを複数の問題に対して計測して、相関性があるか確認する。SMT ソルバには、Z3[6]を用いる。モジュラリティ計算には Gephi[7]というツールを使用する。

ベンチマークは http://smtlib.cs.uiowa.edu/benchmarks.shtml[8]にある QF_LIA を利用する。

具体的な実験手順は以下に従う。

(1) SMT 問題から以下のようにグラフを構築する。図 3.1 参照。

・頂点 ： 各変数を頂点とする

・辺 ：SMT 問題の論理式から CNF に変換した後の節内の変数を辺で結ぶ。

(2) グラフにおけるモジュラリティを Gephi で計算する。

図 3.1 のグラフを csv 形式のノードファイル(図 3.2 参照)と csv 形式のエッジファイル(図 3.3 参照)で表現したものを Gephi は入力として受け取る。

(3) Z3 を用いて、実行時間を計測する。

(4) 図 3.4 より、実行時間とベンチマークのデータ量が比例関係にあることが推測できるので、各ベンチマークのデータ量 1000kb であると仮定して実行時間を変換する。

(5) 変換した実行時間とモジュラリティの相関性を調べる。表 3.1、図 3.5

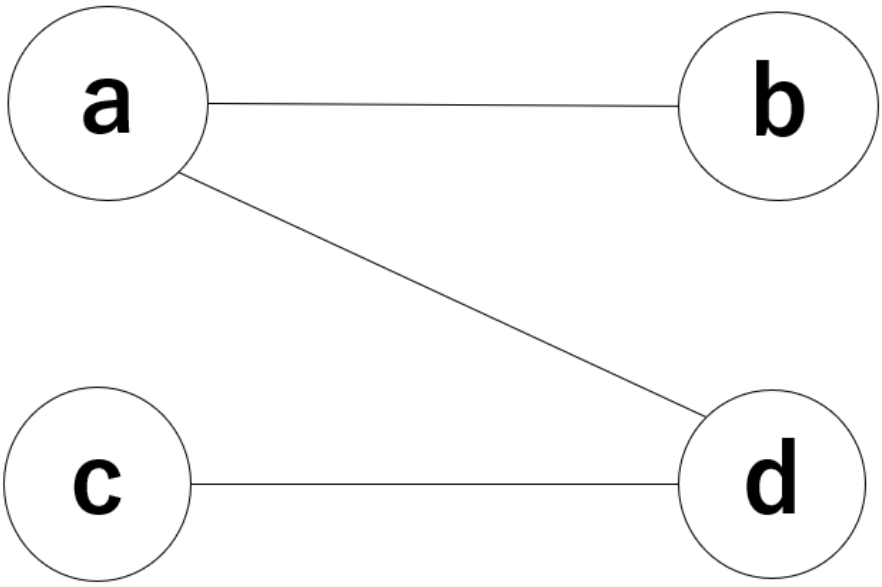

図 3.1 グラフ構築の例　$\big((a+d)>0\big)\wedge\big((b<5)\vee(a<4)\big)\wedge((c+d+1)>7)$

```
Id,Label
0,a
1,b
2,c
3,d
```

図 3.2 グラフ（図 3.1）を表すノードファイル

```
Source,Target
0,1
0,2
2,3
```

図 3.3 グラフ(図 3.1)を表すエッジファイル

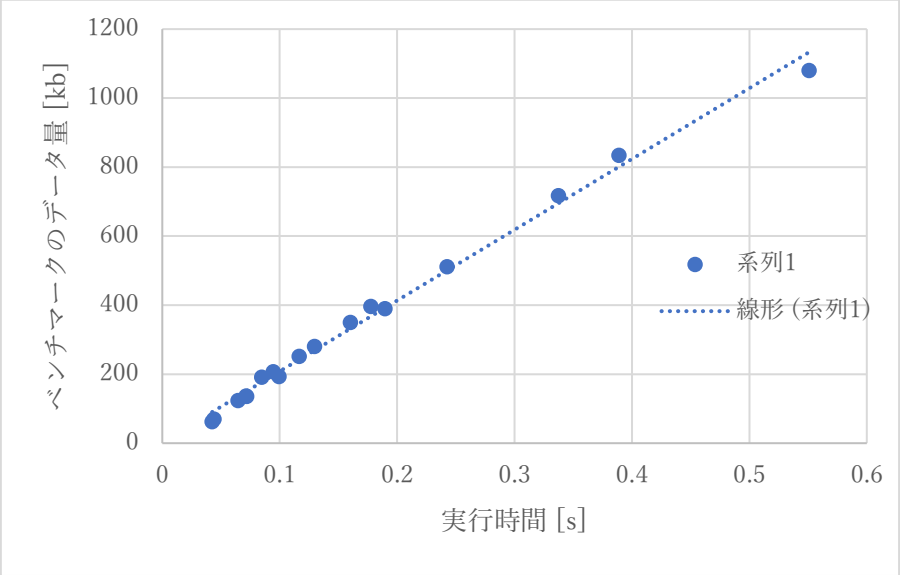

図 3.4 実行時間とベンチマークのデータ量の関係

3.2. 実験環境

デバイス名:DESKTOP-N7AGLEM

CPU: intel core i5-8299U CPU @2.50GHz 2.70GHz

実装 RAM: 8.00 GB

3.3. 実験結果

モジュラリティの値と Z3 の実行時間をそれぞれ表 3.1 に示す。
また、それをグラフに図示したものを図 3.5 に示す。

表 3.1　SMT ソルバの実行時間(変換後)とモジュラリティの関係

ソースファイル名	実行時間（変換後） [s]	モジュラリティ
ParallelPrefixSum_live_bgmc000	0.685621971	0.53
ParallelPrefixSum_live_bgmc002	0.519145161	0.619
ParallelPrefixSum_live_bgmc003	0.442586387	0.659
ParallelPrefixSum_live_blmc000	0.515041451	0.679
ParallelPrefixSum_live_blmc002	0.448232323	0.806
ParallelPrefixSum_safe_bgmc002	0.630972818	0.583
ParallelPrefixSum_safe_bgmc003	0.528691176	0.651
ParallelPrefixSum_safe_bgmc004	0.456772947	0.681
ParallelPrefixSum_safe_bgmc005	0.462439286	0.76
ParallelPrefixSum_safe_bgmc006	0.457845714	0.611
ParallelPrefixSum_safe_blmc000	0.519452555	0.624
ParallelPrefixSum_safe_blmc001	0.462571429	0.725
ParallelPrefixSum_safe_blmc002	0.485787179	0.753
ParallelPrefixSum_safe_blmc003	0.474315068	0.798
ParallelPrefixSum_safe_blmc004	0.470299861	0.813
ParallelPrefixSum_safe_blmc005	0.46613789	0.819
ParallelPrefixSum_safe_blmc006	0.51002963	0.837

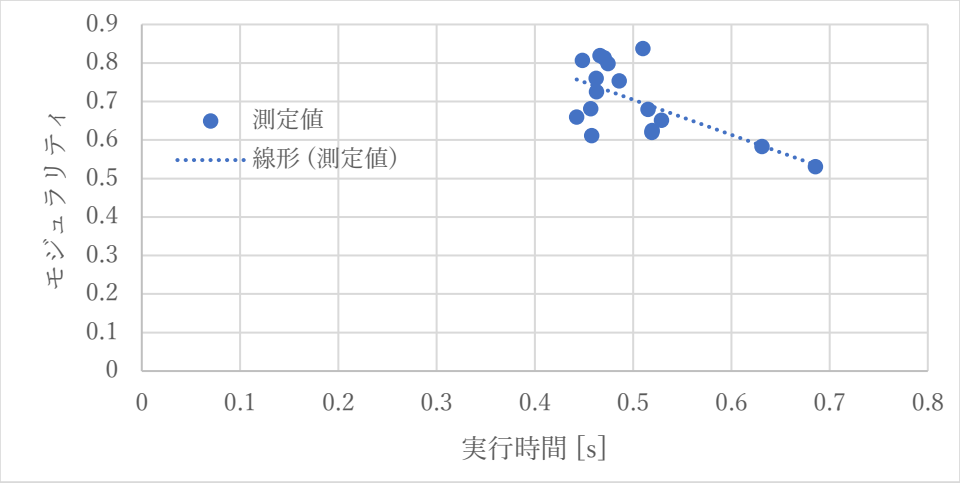

図 3.5　SMT ソルバの実行時間とモジュラリティの関係

4. SMT ソルバの効率とモジュラリティの相関性（重みづけあり）

4.1. 実験手法

SMT ソルバが SMT 問題を解く時間を計測する。SMT 問題からグラフを作成して、そのグラフのモジュラリティを求める。これを複数の問題に対して計測して、相関性があるか確認する。SMT ソルバには、Z3[6]を用いる。モジュラリティ計算には Gephi[7]というツールを使用する。

ベンチマークは http://smtlib.cs.uiowa.edu/benchmarks.shtml[8]にある QF_LIA を利用する。重みづけの効果を検証したいので、第 3 章と同じベンチマークを使用する。

具体的な実験手順は以下に従う。

(1) SMT 問題から以下のようにグラフを構築する。図 3.1 参照。
- ・頂点 ： 各変数を頂点とする
- ・辺 ：SMT 問題の論理式から CNF に変換した後の節内の変数を辺で結ぶ。
  辺があらわれるたびに、辺の重みを 1 加算していく。

(2) グラフにおけるモジュラリティを Gephi で計算する。
図 4.1 のグラフを csv 形式のノードファイル(図 4.2 参照)と csv 形式のエッジファイル(図 4.3 参照)で表現したものを Gephi は入力として受け取る。

(3) Z3 を用いて、実行時間を計測する。

(4) 図 3.4 より、実行時間とベンチマークのデータ量が比例関係にあることが推測できるので、各ベンチマークのデータ量 1000kb であると仮定して実行時間を変換する。

(5) 変換した実行時間とモジュラリティの相関性を調べる。表 4.1、図 4.5

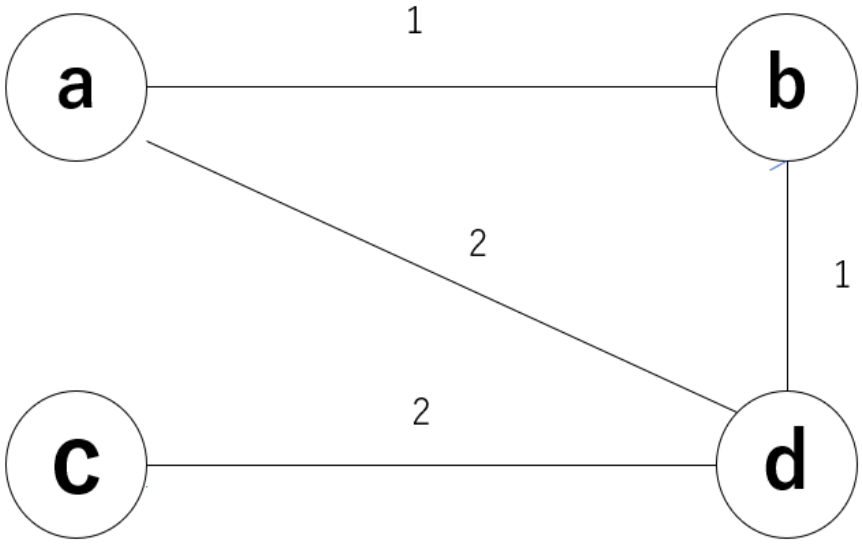

図 4.1 グラフ構築の例

$\big((a+d)>0\big)\wedge\big((b<5)\vee(a<4)\vee(d<8)\big)$

$\wedge((c+d+1)>7\vee(c<2))$の場合

```
Id,Label
0,a
1,b
2,c
3,d
```

図 4.2 グラフ（図 4.1）を表すノードファイル

```
Source,Target,Weight
0,1,1
0,3,2
1,3,1
2,3,2
```

図 4.3 グラフ(図 4.1)を表すエッジファイル

4.2. 実験環境

デバイス名:DESKTOP-N7AGLEM
CPU: intel core i5-8299U CPU @2.50GHz 2.70GHz
実装 RAM: 8.00 GB

4.3. 実験結果

モジュラリティの値と Z3 の実行時間をそれぞれ表 4.1 に示す。
また、それをグラフに図示したものを図 4.4 に示す。

表 4.1　SMT ソルバの実行時間(変換後)とモジュラリティの関係

ソースファイル名	実行時間（変換後） [s]	モジュラリティ
ParallelPrefixSum_live_bgmc000	0.685621971	0.617
ParallelPrefixSum_live_bgmc002	0.519145161	0.73
ParallelPrefixSum_live_bgmc003	0.442586387	0.768
ParallelPrefixSum_live_blmc000	0.515041451	0.787
ParallelPrefixSum_live_blmc002	0.448232323	0.826
ParallelPrefixSum_safe_bgmc002	0.630972818	0.591
ParallelPrefixSum_safe_bgmc003	0.528691176	0.709
ParallelPrefixSum_safe_bgmc004	0.456772947	0.75
ParallelPrefixSum_safe_bgmc005	0.462439286	0.765
ParallelPrefixSum_safe_bgmc006	0.457845714	0.787
ParallelPrefixSum_safe_blmc000	0.519452555	0.69
ParallelPrefixSum_safe_blmc001	0.462571429	0.752
ParallelPrefixSum_safe_blmc002	0.485787179	0.751
ParallelPrefixSum_safe_blmc003	0.474315068	0.772
ParallelPrefixSum_safe_blmc004	0.470299861	0.796
ParallelPrefixSum_safe_blmc005	0.46613789	0.825
ParallelPrefixSum_safe_blmc006	0.51002963	0.83

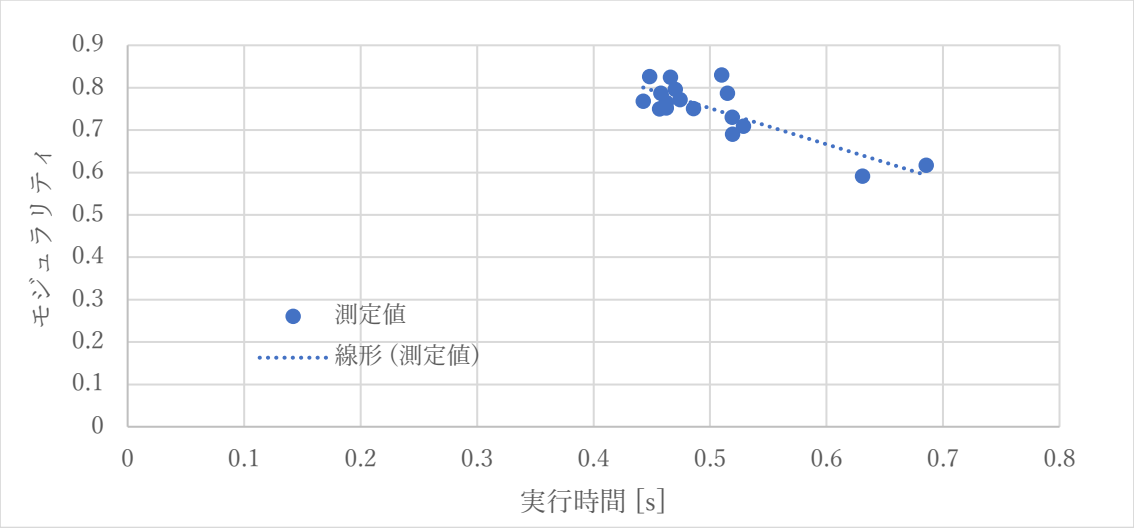

図 4.4　SMT ソルバの実行時間とモジュラリティの関係

5. 考察

図 3.5,図 4,4 を参照すると、SMT ソルバの実行時間とモジュラリティの相関性を確認できた。モジュラリティが大きくなると、実行時間は短くなり、小さくなると実行時間は長くなるという相関性であることがわかる。
また、重みづけ有りの方が、近似直線と測定値の差が小さくなり、より相関性があると考えられる。

6. 今後の課題

本実験では、適切なベンチマークを 17 個しか見つけられなかった。
以下の原因が考えられる。
１つ目の原因は、実験環境に十分な演算能力がなかったことである。２つ目の原因は、適当なベンチマークを見つけられたが作成したグラフを作成するプログラムがベンチマーク内の ite 演算を処理できなかったことである。
今後の課題として、高い演算能力を持つ CPU での実験を行うこと、グラフ作成プログラムの改善、プログラムの改善が考えられる。
また、重みづけの方法として、節の中で複数の辺を構成できるとき（$((c + d + 1) > 7 \vee (c < 2))$の場合、c,d 間の辺の重みは２）も、カウントしたが、節の中はカウントしない、重みを変えるなどいろいろな重みづけをして実験をして、最も実行時間と相関性のある重みづけを探りたい。

謝辞

本研究を進めるにあたり、ご指導いただきました早稲田大学情報理工学科の寺内 多智弘教授に心より深く感謝致します。また、多くの助言をいただきました寺内研究室の山内君に心より深く感謝いたします。

参考文献

[1] C. Ans ´ otegui, J. Gir ´ aldez-Cru, and J. Levy,
"The community structure of SAT formulas", SAT 2012, pp.410-423, 2012.
[2]「SAT/SMT ソルバのしくみ」< https://www.slideshare.net/sakai/satsmt >
2021/01/02 アクセス.
[3]「SAT ソルバ・SMT ソルバの技術と応用」
< https://www.jstage.jst.go.jp/article/jssst/27/3/27_3_3_24/_pdf > 2021/01/02 アクセス.
[4]「ソフトウェア科学特論：命題論理」< https://tamura70.gitlab.io/lect-proplogic/org/proplogic.html >2021/01/02 アクセス.
[5] Newman, M. E. J. (2006). "Fast algorithm for detecting community structure in networks".
[6]「Z3」, < https://github.com/Z3Prover/z3 >, 2021/01/02 アクセス.
[7]「Gephi – The Open Graph Viz Platform」, < https://gephi.org/ >, 2021/01/02 アクセス.
[8] 「SMT-LIB Benchmarks」, < http://smtlib.cs.uiowa.edu/benchmarks.shtml >,
2021/01/02 アクセス.

付録

SMT ソルバの実行時間を求めるプログラム

```
open Z3

let smtlib2_to_assertions (ctx: Z3.context) (smt2file_name : string) : Z3.Expr.expr list =
  let astvec = SMT.parse_smtlib2_file ctx smt2file_name [] [] [] [] in
  Z3.AST.ASTVector.to_expr_list astvec

let  to_string_exprs (expr_list : Z3.Expr.expr list) : string =
  List.fold_left(fun acm e -> Printf.sprintf "%s%s:¥n" acm (Z3.Expr.to_string e)) "" expr_list

let print_exprs (expr_list : Z3.Expr.expr list) : unit =
  Printf.printf "%s¥n" (to_string_exprs expr_list)

let is_satisfiable (ctx :Z3.context) (exprs : Z3.Expr.expr list) : Z3.Solver.status =
  let solver = Z3.Solver.mk_solver ctx None in
  Z3.Solver.check solver exprs

let get_time f ctx (exprs : Expr.expr list) : Z3.Solver.status =
  let t = Sys.time() in
  let f_ctx_exprs = f ctx exprs in
  Printf.printf "Execution time : %fs¥n" (Sys.time() -. t);
  f_ctx_exprs

let _ =
  let ctx = Z3.mk_context [("model","true");("proof","true")] in
  let argument1 =
    try List.hd (List.tl (Array.to_list Sys.argv))  with
      Failure _ -> (Printf.printf "Usage:./a.out file.smt2¥n"); exit 1
  in
  let exprs = smtlib2_to_assertions ctx argument1 in
  Printf.printf "%s¥n" (Z3.Solver.string_of_status (get_time is_satisfiable ctx exprs));
```

SMT 問題から csv 形式のノードファイルとエッジファイルを出力するプログラム

```
open Z3

type node = {id : int; label : string}

let to_string_exprs (expr_list : Z3.Expr.expr list) : string =
  List.fold_left (fun acm e -> Printf.sprintf "%s%s," acm (Z3.Expr.to_string e)) "" expr_list ^ "¥n"

let print_exprs (expr_list : Z3.Expr.expr list) : unit =
  Printf.printf "%s" (to_string_exprs expr_list)

let rec get_all_variable (result_list : Expr.expr list) (elem_list : Expr.expr list) : Expr.expr list =
  match elem_list with
    [] -> result_list
    | e :: elem_list ->
      let getArgs_e : Expr.expr list = Expr.get_args (e : Expr.expr) in
      if (List.length (getArgs_e : Expr.expr list) > 0) then begin
        let res : Expr.expr list = get_all_variable result_list getArgs_e in
        get_all_variable res elem_list
        end
      else
        if(Expr.is_numeral (e : Expr.expr)) then
          get_all_variable result_list elem_list
        else begin
          (*append e to res_list *)
          (* [e] :: result_list *)
          let val_list = List.append [e] result_list in
          get_all_variable val_list elem_list
          end


let rec print_all_variable (elem_list : Expr.expr list) : unit =
  match elem_list with
   [] -> ()
  | e :: elem_list ->
    let getArgs_e :Expr.expr list = Expr.get_args (e :Expr.expr) in
```

```
    if ( Expr.is_numeral (e : Expr.expr) ) then
      print_all_variable elem_list
    else
      (* let getArgs_e :Expr.expr list = Expr.get_args (e :Expr.expr) in *)
      if (List.length (getArgs_e : Expr.expr list) > 0) then begin
        print_all_variable getArgs_e;
        print_all_variable elem_list
        end
      else
        Printf.printf "variable : %s¥n" ( Expr.to_string (e : Expr.expr) )

let rec print_astvec (res_channel : out_channel) (expr_list : Expr.expr list) : unit =
  match expr_list with
    [] -> ()
    | expr :: expr_list ->
      let args_of_expr = Expr.get_args expr in
      let res_expr_list : Expr.expr list = get_all_variable [] args_of_expr in
      print_exprs (res_expr_list);
      output_string res_channel (to_string_exprs (res_expr_list));
      print_astvec res_channel expr_list

let rec is_already_exist (node_list : node list) (e : string ) : bool =
  match node_list with
    [] -> false
    | n :: node_list ->
      if (String.compare n.label e = 0) then
        true
      else
        is_already_exist node_list e

let rec set_node (res_list : node list) (i : int) (expr_list : Expr.expr list) : node list =
  match expr_list with
  [] -> res_list
  | expr :: expr_list ->
    if (is_already_exist res_list (Expr.to_string expr)) then
      set_node res_list i expr_list
```

```
      else
        let res_node_string = Expr.to_string expr in
        let n : node = {id = i; label =   res_node_string} in
        let after_res_list : node list = List.append [n] res_list in
        set_node after_res_list (i+1) expr_list

let rec get_node_list (res : node list) (i : int) (expr_list : Expr.expr list) : node list =
  match expr_list with
  [] -> res
  | expr :: expr_list ->
    let args_of_expr = Expr.get_args expr in
    let res_expr_list : Expr.expr list = get_all_variable [] args_of_expr in
    if (List.length res_expr_list > 1) then begin              (*set node info from variables*)
      let res_node_list : node list = set_node res i res_expr_list in
      get_node_list res_node_list (List.length res_node_list) expr_list
      end
    else
      get_node_list res i expr_list

let rec get_id (node_list : node list) (l : string) : int =
  match node_list with
  [] -> (-1)
  | n :: node_list ->
    if(String.compare n.label l = 0) then
      n.id
    else
      get_id node_list l

let rec get_var_of_node_id (res : int list) (node_list : node list) (var_list : Expr.expr list) : int list =
  match var_list with
    [] -> res
    | var :: var_list ->
      let res_ = List.append [(get_id node_list (Expr.to_string var))] res in
      get_var_of_node_id res_ node_list var_list

let rec put_edge (matrix : int array array) (i : int) (node_id : int list) : int array array =
```

```
  match node_id with
  [] -> matrix
  | j :: node_id ->
    matrix.(i).(j) <- 1;
    put_edge matrix i node_id

let rec set_edge (matrix : int array array) (node_id : int list) : int array array =
  match node_id with
  [] -> matrix
  | i :: node_id ->
    let matrix_ = put_edge matrix i node_id in
    set_edge matrix_ node_id

let rec set_matrix (matrix : int array array) (node_list : node list) (expr_list : Expr.expr list) :int array array
=
  match expr_list with
  [] -> matrix
  | expr :: expr_list ->
    let args_of_expr = Expr.get_args expr in
    let res_expr_list : Expr.expr list = get_all_variable [] args_of_expr in
    if (List.length res_expr_list > 1) then (*if there is edge*)
      let node_id : int list = get_var_of_node_id [] node_list res_expr_list in
      let added_matrix : int array array = set_edge matrix node_id in
      set_matrix added_matrix node_list expr_list
    else
      set_matrix matrix node_list expr_list

let rec print_edge (res_edge_channel : out_channel) (i : int) (j : int) (num_of_node : int) (matrix : int array
array) : unit =
  if(j = num_of_node)then
    ()
  else begin
    if(matrix.(i).(j) = 1) then begin
      output_string res_edge_channel (Printf.sprintf "%d,%d¥n" i j);
      print_edge res_edge_channel i (j+1) num_of_node matrix
      end
```

```
    else
      print_edge res_edge_channel i (j+1) num_of_node matrix
  end

let rec call_print_edge (res_edge_channel : out_channel) (gyou_i : int) (num_of_node : int) (matrix : int array
array) : unit =
  if(gyou_i = num_of_node) then
    ()
  else begin
    print_edge res_edge_channel gyou_i 0 num_of_node matrix;
    call_print_edge res_edge_channel (gyou_i + 1) num_of_node matrix
  end

let rec print_node (res_node_channel : out_channel) (node_list : node list) : unit =
  match node_list with
  [] -> ()
  | n :: node_list ->
    output_string res_node_channel (Printf.sprintf "%s,%s¥n" (string_of_int n.id) n.label);
    print_node res_node_channel node_list

let convert_to_cnf (ctx:Z3.context) (exprs : Z3.Expr.expr list) : Z3.Expr.expr list =
  let goal = Z3.Goal.mk_goal ctx false false false in
  Z3.Goal.add goal exprs;
  let tseitin_tactic = Z3.Tactic.mk_tactic ctx "tseitin-cnf" in
  let apply_result : Z3.Tactic.ApplyResult.apply_result =  Z3.Tactic.apply tseitin_tactic goal None in
  Printf.printf "num_of_subgoals = %d¥n" (Z3.Tactic.ApplyResult.get_num_subgoals apply_result);
  let goal_ : Z3.Goal.goal = Z3.Tactic.ApplyResult.get_subgoal apply_result 0 in
  Z3.Goal.get_formulas goal_

let _ =
  let ctx = Z3.mk_context [("model", "true");("proof","true")] in
  let argument1 =
    try List.hd (List.tl (Array.to_list Sys.argv))  with
      Failure _ -> (Printf.printf "Usage:./a.out file.smt2¥n"); exit 1
  in
  let astvec =  SMT.parse_smtlib2_file ctx argument1 [] [] [] [] in
```

```
(*
  let astVecString = AST.ASTVector.to_string astvec in
  Printf.printf "astVecString %s¥n" astVecString;
*)

  let exprList_of_astvec : Expr.expr list = AST.ASTVector.to_expr_list astvec in
  let exprList_after_tseitin : Expr.expr list = convert_to_cnf ctx exprList_of_astvec in



(*
  let res_channel : out_channel = open_out "res_variable.csv" in
  print_astvec res_channel exprList_of_astvec;
*)

  let node_list : node list = get_node_list [] 0 exprList_after_tseitin in
  let num_of_node : int = List.length node_list in
  Printf.printf "num_of_node = %d¥n" num_of_node;
  let inited_matrix :int array array = Array.make_matrix num_of_node num_of_node 0 in
  let res_matrix = set_matrix inited_matrix node_list exprList_after_tseitin in
  let res_edge_channel : out_channel = open_out "res_edge.csv" in
  output_string res_edge_channel "Source,Target¥n";
  call_print_edge res_edge_channel 0 num_of_node res_matrix;
  let res_node_channel : out_channel = open_out "res_node.csv" in
  output_string res_node_channel "Id,Label¥n";
  print_node res_node_channel node_list
```