3 SATを用いた解法

応用

松永裕介 | 九州大学　田村直之 | 神戸大学

配線問題とSAT

配線問題は，マスとマスを結ぶ線を「引く」か「引かない」かという0/1の判断の結果が正しい解となっているかを判定する問題とみなせるので，比較的単純にSAT問題に定式化することができる．SATとはSATisfiabilityの略であり，通常は「充足可能性」と訳される．多くの場合，命題論理式[☆1]の充足可能性判定問題をSAT問題と呼び，SAT問題を解くプログラムのことをSATソルバーと呼ぶ．コンピュータ科学の視点から見ると，SAT問題はNP完全問題であることが知られており，現時点では問題のサイズに対して多項式時間で解けることを保証したアルゴリズムは存在しない．しかし，SAT問題を解く効率の良いアルゴリズムの研究は非常に活発に行われており，特に今世紀に入ってからの性能向上は目覚ましいものがある．

では配線問題を単純にSAT問題として定式化してそれを高速なSATソルバーで解けば問題は解決か，というとそう簡単な話ではない．配線問題の制約を命題論理式で表すやり方によって計算時間が大きく変わってくるのである．本稿ではDAシンポジウムアルゴリズムデザインコンテストでこれまでに得られた知見をもとにSATソルバーを用いて配線問題を解く解法についての解説を行う．なお，SAT技術や配線問題の応用に関する過去記事[1),2)]も参照されたい．

☆1 真(1)，偽(0)の2値を取る変数の否定，論理積および論理和から作られる論理式を命題論理式と呼ぶ．

グラフ問題としての定式化

まず，ここでは配線問題をより数学的に定義するためにグラフ理論の言葉で定義を行う．配線領域は格子グラフと考えることができる．格子グラフとは節点が格子状にならんだグラフで上下左右の節点間に枝が存在する．ここでは枝の向きは考慮しない（無向グラフ）ものとする．図-1に元の配線問題と対応するグラフを示す．

図-1(b)中の節点は白い節点と黒い節点に分類される．このうち白い節点は元の問題の空のマスに対応しており，黒い節点は元の問題の数字のマスに対応している．以降，白い節点を「自由節点」，黒い節点を「終端節点」と呼ぶことにする．同じ数字を持つ終端節点は必ず2個ずつ存在するものとする．このグラフ上での配線問題とは，互いに同じ数字を持った終端節点のペアの間を結ぶ経路を求める問題となる．以降，同じ数字を持った終端節点のペアを「ネット」と呼び，終端節点の数字を「ネット番号」と呼ぶことにする．ここで経路とは同じ節点

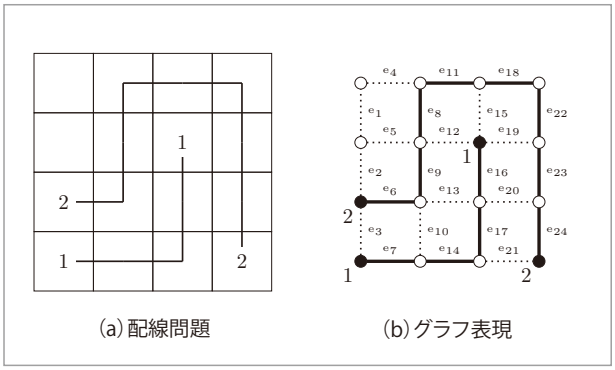


図-1　配線問題と対応するグラフ

に接続している枝の順列である．もちろん，1つの枝が2つ以上の異なる経路で用いられることはないし，1つの節点上で2つの経路が交差することも許さない[☆2]．以降は配線問題をグラフ上で経路を求める問題として説明する．

SAT問題へのエンコーディング

SATソルバーは（命題）論理式を扱うので与えられた問題を論理式の形で記述する必要がある．この操作をエンコーディング(encoding)あるいは符号化と呼ぶ．ほかの多くの問題の場合と同様にこのエンコーディングにはいくつかのやり方が考えられるが，ここでは基本的な制約に対する典型的なやり方を示す．まず配線問題の解である「経路」をどのように表すかを考えよう．

定義より経路とはつながった枝の「順列」であるが，元のグラフと枝の「集合」が与えられればそこから経路を求めることは容易である．そこで，枝の順列ではなく枝の集合によって1つの経路を表すことができる．また，異なるネットに属する経路は互いに交わることはなく，さらにその端にはネット番号を持った終端節点が接続しているのでどのネットに属しているかもあとから判別することができる．つまり経路として用いられている枝の集合によって解である経路を表すことができる．図-1の例では $\{e_6, e_7, e_8, e_9, e_{11}, e_{14}, e_{16}, e_{17}, e_{18}, e_{22}, e_{23}, e_{24}\}$ が解となる．これは，同図(b)の太線で描かれた枝に対応している．ただし，これは枝の集合であって順番に意味がないことに注意されたい．これをSATの論理式として記述するために，枝 e_i が解として選ばれるとき，真(1)となる命題変数 x_i を用意して配線結果を表すことにする．

次に，枝の集合が「経路」となっている条件を制約で表現することを考えよう．まず経路の端つまり順列における最初と最後の要素の場合を考える．もちろん経路の最初と最後の要素の枝と接続している節点は終端節点である．ある終端節点 v_a に4つの枝 e_1, e_2, e_3, e_4 が接続していると仮定する．すると，この4つの枝のうち解として選ばれるものはただ1つとなる．これを論理式で表すと以下のようになる．

$$x_1 \vee x_2 \vee x_3 \vee x_4 \quad (1)$$
$$\neg x_1 \vee \neg x_2 \quad (2)$$
$$\neg x_1 \vee \neg x_3 \quad (3)$$
$$\neg x_1 \vee \neg x_4 \quad (4)$$
$$\neg x_2 \vee \neg x_3 \quad (5)$$
$$\neg x_2 \vee \neg x_4 \quad (6)$$
$$\neg x_3 \vee \neg x_4 \quad (7)$$

ここで，記号‘∨’は「または」を表し，どちらか一方でも真であれば式全体も真となるものである．‘¬’は「否定」を表し，対象の変数[☆3]が偽のとき，真になるものである．これらの式のうち最初の式は真となる変数が最低でも1つある条件を表し，残りの式が2つ以上の変数が同時に真とならない条件を表している．実際のSAT問題が解く論理式とはこれらすべての式の論理積（∧）を取ったものである．つまり，同時にすべての式を真にする（充足する）変数の割り当てを求めることとなる．たとえば，e_4 を経路として選択して残りの枝を選択しない場合，$(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$ となるが，これはすべての式(1)－(7)を満足している．一方，どの枝も選ばれていない場合は $(x_1, x_2, x_3, x_4) = (0, 0, 0, 0)$ となり式(1)を満たすことができない．また，e_1 と e_3 の2つが選ばれた場合は $(x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$ となって式(3)が満たされない．

次に経路の中間の枝の条件について考える．この場合，経路上の連続した2つの枝が同じ自由節点に接続していることになる．逆に自由節点を中心に考えると，接続している枝のうち経路として選ばれて

☆2 グラフ理論の言葉では「節点素な経路を求める問題」と言う．

☆3 一般的には論理式に対しても否定演算子がかかることがあるがここでは否定演算子は変数にしか付かない．

いるのは0個あるいは2個である．そこで，終端節点の場合と同じように枝に対する命題変数を用いた論理式として制約を作ることができる．ここでは紙面の都合上，詳細は省略する．以上で枝の集合のなかで「経路のようなもの」を表す条件はできた．今ここで「経路のようなもの」と言った理由は2つある．1つはこの制約だけでは自由節点だけからなるループ経路ができてしまうことである．ただし，このループは後述するようにあとで取り除いてしまえばよい．もう1つは深刻な問題で，この制約だけでは異なるネットの終端節点が1つの経路でつながってしまうことを防げないということである．

そこで補助的に新たな命題変数を導入することを考える．これは各節点が何番目のネットの経路として用いられているか(あるいは用いられていないか)を表すためのもので，命題論理という縛りがなければ各節点につき1つずつ整数値を表す変数を導入すればよい．説明のためにこの仮想的な整数変数の場合の制約式を以下に示す．今，枝 e_{ij} によって節点 v_i と v_j が隣接しているものとする．枝 e_{ij} が経路として選ばれたことを表す命題変数を x_{ij} とし，節点 v_i, v_j のネット番号を表す整数変数をそれぞれ l_i, l_j とする．

$$x_{ij} \Rightarrow l_i = l_j \tag{8}$$

ここで $a \Rightarrow b$ は含意(implication)を表し，$\neg a \vee b$ と等価である．日本語で言い換えれば「もし a が真なら b も真となる」ということになる．さらに終端節点 l_k のネット番号が A である場合の条件として以下の制約を作る．

$$l_k = A \tag{9}$$

これで同じネット番号を持つ終端節点が1つの経路で結ばれることとなる．残念なことにSATソルバーは整数の変数を受け付けないので0か1かの2値の命題変数を（複数）用いてエンコーディングする必要がある☆4．いま，ネット数を K とすると経路として用いられない場合も含めて $K+1$ 個の値を区別する必要があるので最もコンパクトなエンコーディングは $\log_2(K+1)$ 個の命題変数を用いた2進符号化である．単純に整数を2進表現で表し各ビットに命題変数を対応させればよい．「2つの整数が等しい」と「2つの整数の2進表現の各ビットが等しい」は同値なので上の2つの制約式はそのまま同様の命題論理式に変換できる☆5．

ネット番号に関してはもう1つ「枝 e_{ij} が経路として選ばれていないとき，両端の節点 v_i と v_j が同じネット番号を持つことはない」という制約を考えることもできる☆6．実はこちらの条件を2進符号化で表すのは少し面倒である．「2つの整数が等しくない」は「2つの整数の2進表現のうち1つ以上が異なる」となってしまって各ビットごとに独立した式で表すことができないからである．

ネット番号の符号化のもう1つの考え方として K 個の命題変数を用意するワンホット(one-hot)符号化も考えられる．これは同時にたかだか1つの変数しか真にならない符号化である．ある節点 v_i に対して $b_{i1}, b_{i2}, \ldots, b_{iK}$ という K 個の変数を用意して j 番目のネットの経路として用いられたときに b_{ij} のみ1にして残りの変数をすべて0にする，もしも経路として用いられなかった場合にはすべての変数を0にする，という符号化を行う．こうすることで隣接している節点間のネット番号が異なるという条件は各ビットごとに「両方の変数が共に1にならない」，という条件で表すことが可能となる．ワンホット符号化の欠点は，必要な変数の数がネット数に比例することである．2進符号化ではネット数の対数(底は2)なのでネット数が数十を超えるとその差

☆4 述語論理の充足可能性判定問題を解く SMT（Satisfiability Modulo Theories）ソルバーを用いれば整数変数の等式を扱うことも可能である．興味ある読者は試してみてもらいたい．

☆5 ただし，ビット数倍の式が必要となる．なお，$a = b$ は $(\neg a \vee b) \wedge (a \vee \neg b)$ と同値．

☆6 厳密にはこれは違反ではなく冗長な経路の条件であるが，アルゴリズムデザインコンテスト 2017 では違反条件と定められている．

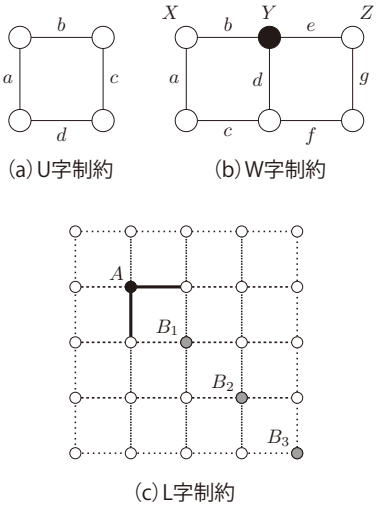
は顕著となる．ただし簡潔な論理式が得られるので変数の数の差がそれほど多くない場合にはワンホット符号化の方が有利な場合もある．

性能改善のためのヒューリスティック

ここまでに述べたエンコーディングに追加の制約条件を付加することで計算時間を短縮することを目的としたヒューリスティック（発見的手法）が考案されている．ここでヒューリスティックと呼んでいるものは一種のエンコーディングの工夫である．なぜ，追加の条件を加えると速く解けることがあるのか，ということに関してはさまざまな理由やケースがあり統一的に説明することは難しいが，下記の具体的な例からその一端を汲み取ってほしい．

U 字制約

特に定まった命名規則はないので本稿ではU 字制約と称する（以下同様）．図 -2(a) における 4 つの枝のうち 3 つ以上が同時に用いられることはない，という制約である．これは前述の「選ばれていない枝の両端に同じネット番号が割り振られることはない」，というルールのもとでは完全な制約である[☆7] たとえば図中で a, d, c が経路として選ばれている場合には前述のルール違反となる．もしこのルールがなかったとしても，b のみを選んで残りの 3 つの枝を選ばない解があるのでこの制約を付加したことで配線可能な問題に対して「配線不能」と答えることはない．



(a) U字制約　(b) W字制約　(c) L字制約

図 -2　U 字制約，W 字制約，L 字制約

W 字制約

これも U 字制約の拡張で図 -2(b) において節点 Y が終端でないときには a, c, f, g の 4 つの枝が同時には選ばれないという制約である．Y が終端でなく，a, c, f, g が経路として用いられていた場合，その代わりに b, e を経路として選んでも X と Z を結ぶ経路を作ることができるため，元の問題が配線可能の場合にはこの制約を付加しても「配線不能」と答えることはない．

全マス使用制約

これは一般的な配線問題というよりアルゴリズムデザインコンテストで用いているオリジナルのナンバーリンク問題が課している暗黙の制約と呼べるもので，すべての節点がいずれかの経路に用いられるという制約である．もちろんこの制約を加えることによって本当は答えがあるのに解なしという誤った答えを出す場合もあるが，全マスを使う解が存在する問題に対しては圧倒的なスピードアップとなることが多い．全マスを使う解がない問題に対してもすぐに‘UNSAT（充足不能）’と分かることが多いので，最初にこの制約付きでトライしてみてUNSAT のときに次のヒューリスティックを試すのが効果的と思われる．全マス使用制約は非常に単純で自由節点に接続する枝の制約に関して「0 個か 2 個の枝が選ばれる」という条件を「常に 2 個の枝が選ばれる」に変更するだけでよい．

☆7　ここでは制約を付加した問題が，元の問題で解がある場合には必ず解を見つけることができる場合に「完全な制約」と呼んでいる．

L 字制約

これはナンバーリンクの問題でありがちなパターンに基づく制約で，図 -2(c) のように節点 A においてL 字に折れ曲がる経路を許す条件として，内側 45 度の方向の節点（図中の B_1, B_2, ...）のいずれかに終端節点があることを制約として加えるものである．これはそうでない場合に B_1, B_2, B_3 のどれかの節点が経路として用いられることがなくなって無駄になってしまう，というアイディアに基づいている．全マスを使用する解の場合にはこの制約は自動的に満たされるが，この制約を満たしつつ全マス使用制約を満たさない問題は存在するので全マス使用制約よりは緩い制約である．ただし，この制約を満たさない解が存在する問題もあるのでこの制約を加えて解が求まらなかった場合にはこの制約を外して再度試す必要がある．

解の出力

配線問題をエンコーディングした論理式の変数の中には元の配線問題に対する解を表す枝の選択の有無を表す変数以外にも補助的に追加された変数が含まれている．SAT ソルバーがこの論理式に対して ‘SAT’ と答えたときはこれらすべての変数に対する真偽の割り当てが決まった答え[☆8] が返される．そこから配線問題としての解を作るには，SAT ソルバーの答えから枝の選択の有無の情報だけを取り出し，その情報を用いて終端節点から選ばれた枝をたどって経路を作ればよい（図 -1(b) 参照）．終端節点から経路をたどることで自由節点のみからなる独立したループ経路は自然と無視される．これだけで配線問題の解としては正しい経路を求めることができるが，多くの場合，配線問題はただ引ければよいというのではなく「なるべく配線長を短く」とか「なるべく折れ曲がり回数を少なく」という目的関数を持つ場合が多い．そこで現実的な手法としてはいったん，SAT ソルバーを用いて可能解を得たあとでこれらの目的関数の値を最適化するように解を改善する方法が考えられる．1 つの具体的な方法としては，SAT で求められた解（経路集合）から 1 つの経路を取り出し，その経路だけ「引き剥がし」て「再配線」する手法が考えられる（図 -3）．この場合，その他の経路が用いている節点および枝を使用できなくしたグラフ上での経路探索問題となる．グラフ上の最短経路を見つけるアルゴリズムや格子グラフ上で最短かつ最小曲がりの経路を見つけるアルゴリズムは効率の良いものが知られている．図 -3 の例ではまず経路 2 を引き剥がし経路長と折れ曲がりが最小となる経路を求める (b)，続いて経路 1 を引き剥がして経路長と折れ曲がりが最小となる経路を求めて最終結果 (c) を得る．

多層問題への拡張

アルゴリズムデザインコンテストでは 2015 年までは単層の配線問題を扱ってきた．これらの問題

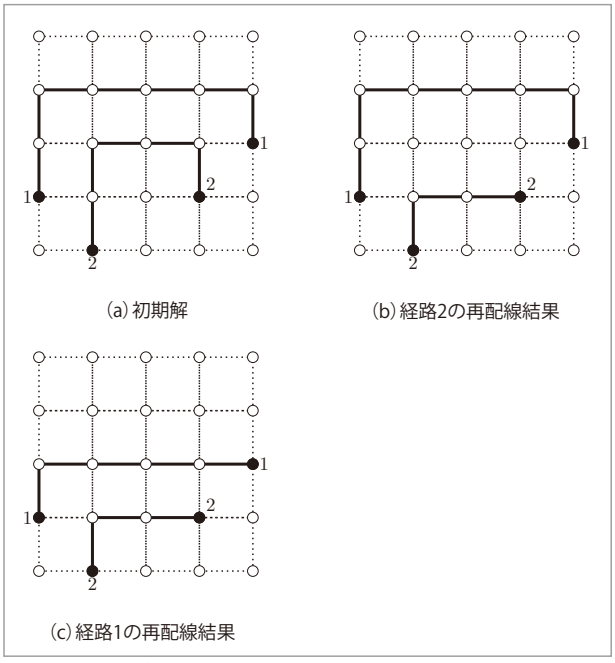


図 -3　引き剥がし・再配線

☆8　モデルと呼ばれる．

に対してはSATソルバーを用いた手法は相性が良く，ほとんどすべての問題を一瞬で解くことができている．2016年以降は問題が多層の配線問題に拡張されている．ただし，2016年と2017年ではルールが少し異なっており，2016年の問題は異なる層を結ぶ経路はあらかじめ定められたビア(via)をただ1回だけ用いるというルールであるのに対して，2017年の問題は任意の場所で異なる層間を結ぶことができるルールとなっている．これはある意味多層配線問題というより3次元配線問題と呼ぶことができる．図-4 (a)に2016年のタイプの問題の例を示す．これは2層の配線問題で，層をまたぐことのできる位置がビア（図中の2重丸の節点）のみに制限されている．ネットは全部で3本あるが，このうち層をまたいだネットは番号1と2の2本であり，ビアもaとbの2カ所である．この場合，ネットとビアの組合せは2通りある，$(1, a), (2, b)$と$(1, b), (2, a)$である．ここで，前者の対応付けを選べば図-4(b)のような配線結果を得ることができる．一方，後者の対応付けを選ぶと1層目においてネット3に阻害されてネット1とビアbを結ぶことができない．このように2016年のタイプの問題は‘ネットとビアの割り当て問題’＋‘複数の単層配線問題’とみなすことができる．もちろん，ネットとビアの割り当ての場合の数はネット数[☆9]（＝ビア数）の階乗だけ存在するが，ネットiがビアjに割り当てられたときに1となる命題変数v_{ij}を導入することで明示的にネットとビアの割り当てを行うことなくビア割り当て問題と配線問題を1つのSAT問題として解くことが可能である．図-4の例の場合，ビアaがネット1に対応するときに1となる変数をv_{1a}とし，ネット1に対応するときに1となる変数をv_{2a}とする．この2つの変数が同時に1になることはないので，$(\neg v_{1a} \vee \neg v_{2a})$という制約を追加しておく．また，どちらか1つは1にならないといけないので$(v_{1a} \vee v_{2a})$という制約も追加しておく．あとはビアaの1層目，2層目の節点に対するネット番号の制約をv_{1a}=1なら‘1’，v_{2a}=1なら‘2’というふうに条件付きにすれば残りの制約条件は単層の配線問題とまったく同じものを用いることができる．このように異なる種類の制約も論理式の形で表すことができればひとかたまりのSAT問題として解くことができるというところがSATを用いた手法の利点の1つである．アルゴリズムデザインコンテストの例題を用いた実験ではこのやり方でビアを用いた多層配線問題をうまく解くことができている．

翻って2017年の問題に対してはあまり結果が芳しくない．その理由はいくつか考えられるが，1つは全マス制約やL字制約といったヒューリスティックが3次元配線問題の場合には効果的でないことである．これらの制約は平面上で配線する場合の特徴から考え出されたものであり，配線経路が上下に逃げることができる場合にはあまり有効には働かない．また，そもそも考慮しなければならないグラフの枝数が増えてしまうという要因もある．たとえば

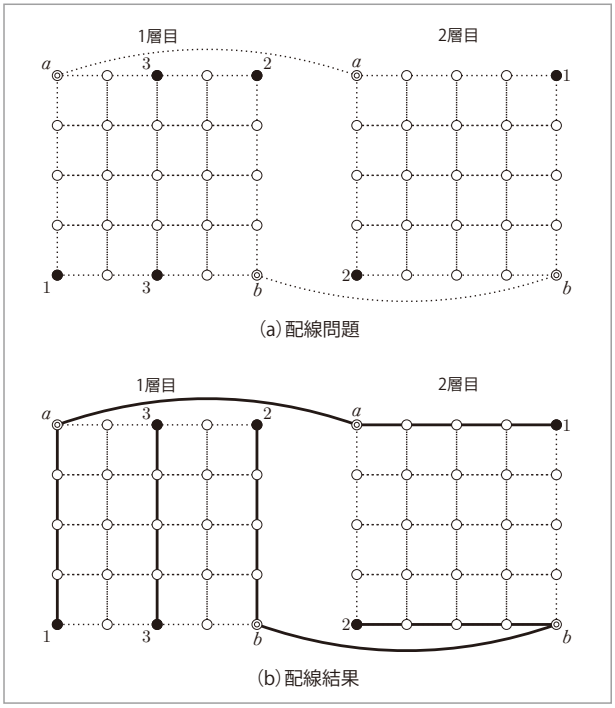


図-4　ビア付きの多層配線問題の例

☆9　厳密には異なる層間を結ぶネットの数である．

図 -4 の問題を 2017 年のタイプの問題に変換するには 2 つのビア a と b を取り去って，1 層目と 2 層目のすべての節点の間に層をまたぐ枝を追加すればよい．すると枝の数は 25 本増えることになる☆10．元々の枝数は 80 本であったのでこれは無視できない増加である．

2017 年に SAT ソルバーを用いたチームの 1 つは，層の数を順次増やしながら解く手法を導入し，この問題に対処している．

(1) i=1
(2) i 層までについて，まだ結ばれてない数字のペアだけを結ぶ部分問題を考え，求解する．
(3) 充足不能なら，すでに結んだ線のうち最長の線を消して (2) に戻る．
(4) 充足可能の場合，i が層数に一致していれば終了．そうでなければ $i = i$+1 として (2) へ．

これでもうまく解けない場合があるが，このように部分問題に分割して解く方法も有効である．

今後の挑戦

SAT ソルバーを用いる解法の利点は，その宣言性と汎用性だ．基本的には問題に対する解の条件だけを宣言的に記述すればよい．また，配線問題だけでなくどのような問題にも対応できる．

しかし，複雑な問題になるとさまざまな工夫が必要になる．これは，どのような論理式で表すかというエンコーディングの工夫だけではない．問題に条件を追加したり，逆に条件を削除して緩和したりし，SAT ソルバーを複数回使用する方法も有用だ．本稿で述べた「全マス使用制約」などの制約を追加する方法や，層の数を順次増やしながら解く方法などがそれに当たる．

SAT ソルバーにおける技術も年々発展している．たとえば，充足不能の場合にその直接的な原因となっている節だけを取り出す方法なども開発され，最適解の探索に利用されている．また，SAT の拡張である MaxSAT，擬似ブール制約，SMT，ASP (Answer Set Programming) などのソルバーの発展も著しい．これらの手法の詳細については文献 1) およびそれらと同じ特集に含まれているほかの記事などを参照されたい．

今回の多層配線問題に対する手法もまだまだ発展可能だと思われる．読者諸氏の挑戦を期待したい．

☆10 ビア付きの場合にはビアの層間の枝は考慮する必要がない．

参考文献
1) 番原睦則，鍋島英知：SAT 技術の進化，情報処理，Vol.57, No.8, pp.704-709 (Aug. 2016).
2) 田村直之，宋　剛秀，番原睦則：SAT とパズル―問題をいかに SAT ソルバーで解くか―，情報処理，Vol.57, No.8, pp.710-715 (Aug. 2016).

(2017 年 11 月 24 日受付)

■松永裕介（正会員） matsunaga@ait.kyushu-u.ac.jp
九州大学大学院システム情報科学研究院准教授．博士 (工学)．LSI の設計自動化技術，SAT 技術等に興味を持つ．

■田村直之（正会員） tamura@kobe-u.ac.jp
神戸大学情報基盤センター教授．学術博士．論理プログラミング，制約プログラミング，SAT 技術，パズル等に興味を持つ．