種々のリンクパズルへの応用

吉仲　亮，岩下　洋哲，川原　純，斎藤　寿樹，
鶴間　浩二，湊　真一

本稿では，フロンティア法による，与えられたグラフ上の特定の制約を満たす部分グラフ抽出の具体的応用例として，ナンバーリンクとスリザーリンクと呼ばれるパズルそれぞれのための解答器と問題生成器のアルゴリズムを提案し，実験結果を報告する．また，この技術を用いたスリザーリンクの問題作成支援についても議論する．

キーワード：スリザーリンク，ナンバーリンク，制約充足問題，ZDD

1. 序

近年，Knuth [1] は，グラフ上の s–t パスを ZDD (zero-suppressed binary decision diagram [2]) で高速に列挙するアルゴリズムを提案している．このアルゴリズムのアイディアに基づき特定の制約を満たす部分グラフを列挙する手法をフロンティア法と呼ぶ．フロンティア法は多様な制約に応用可能であるが，本稿では，同手法の汎用性，有効性をわかりやすく提示する応用例として，リンクパズルを取り上げる．ペンシルパズルと呼ばれる，紙の上の出題に対して鉛筆 1 本で挑戦するさまざまなタイプのパズルが提案され普及しているが，なかでもグリッドグラフなどのグラフ上に，与えられた制約を満たすようなパスやサイクルを見つけるパズルを，本稿ではリンクパズルと呼ぶ．さまざまなリンクパズルの中でも，本稿では，特に有名なナンバーリンクとスリザーリンクを取り上げることにする．それぞれのパズルについて，問題に対してすべての解を求めるアルゴリズムを示し，さらに，一定の条件を満たす問題を列挙するアルゴリズムも提案し，実験結果を示す．また，特にスリザーリンクの問題列挙アルゴリズムと ZDD 上の演算を利用した問題作成支援について議論する．例えばナンバーリンクの求解は，平面基板上の集積回路設計における配線を与えることと直接に関係しているなど，本稿は，さまざまな実務問題に対する解決手法の基本的な考え方を提供するものである．

なお，本稿のアルゴリズムの正当性の数学的な証明を含む技術的詳細は [3] を参照されたい．

よしなか　りょう
京都大学
〒 606–8501 京都府京都市左京区吉田本町 36–1
いわした　ひろあき，かわはら　じゅん
科学技術振興機構
さいとう　としき
神戸大学
つるま　こうじ
日本電機
みなと　しんいち
北海道大学/JST ERATO

2. リンクパズル

通常のナンバーリンクの問題インスタンスは，$m \times n$ の長方形のテーブルであり，その各セルは空白であるか，$1, \ldots, p$（ただし $2p \leq mn$）の自然数のいずれかが 1 つ記入されていて，$1, \ldots, p$ の各数字はちょうど 2 回ずつ出現しているものである．解答者はちょうど 2 回出現するすべての同じ数字同士の組を線で結ぶが，それらの線は互いに交わったり接してはならない．またそれぞれの線は各セルの中心点を水平もしくは垂直に結ぶことによって構成されなければならない[1]．各セルを頂点とし，セルの中心点を結ぶ水平線および垂直線を辺とすれば，ナンバーリンクは一般のグラフ上のパズルに直接に拡張できる．すなわち，一般化ナンバーリンクを定式化すると，問題は，グラフ $G = (V, E)$（V は頂点集合，E は辺集合）および互いに素でちょうど 2 頂点からなる p 個の集合 $V_1, \ldots, V_p \subseteq V$ として与えられ，それに対する解は，互いに頂点を共有しない G 上のパス p 本からなるパスマッチングであり，それぞれのパスの両端点はそれぞれの集合 $V_1, \ldots, V_p$

[1] すべてのセルを何らかの線分が通らなければならないという制約もしばしば導入されているか，もしくは暗黙の了解となっているが，ここではさしあたってこの制約は無視することにする．

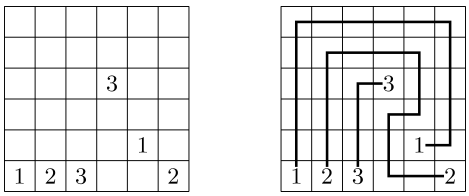

図 1　ナンバーリンクの問題例とその解答

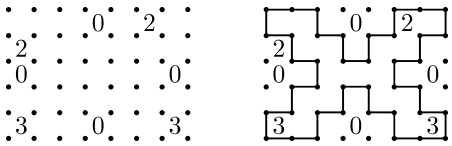

図 2　スリザーリンクの問題例とその解答

を成す．この端点対族 $\{V_1, \ldots, V_p\}$ を**ヒント**と呼ぶ．

通常の**スリザーリンク**の問題インスタンスは，$m \times n$ の格子状に配された頂点で，1×1 の単位正方形を構成する 4 頂点の中心には何も記入されていないか，もしくは 0 から 4 までの自然数のいずれか 1 つが記入されている．問題に対する解は，格子点を結ぶ水平線および垂直線の連結によって形作られるサイクルであって，問題中の自然数 h の周囲には，ちょうど h 本の単位長さの水平／垂直線が引かれていなければならない．通常のスリザーリンクにおいては，単位正方形を構成する 4 辺のうちサイクルの構成に使用する辺数が指示されるが，任意のグラフの任意の辺部分集合に対して使用辺数を指示できるように一般化できる．すなわち，一般化スリザーリンクを定式化すると，問題は，グラフ $G = (V, E)$ および辺の部分集合から自然数への部分関数 $H : 2^E \rightharpoonup \mathbb{N}$ として与えられ，それに対する解は G 上のサイクル（を成す辺集合）$C \subseteq E$ であり，$H(D)$ が定義されているなら $|C \cap D| = H(D)$ を満たしていなければならない．この使用辺数制約関数 H を**ヒント**と呼ぶ．

図 1 と 2 にナンバーリンクとスリザーリンクの問題と解答の例を示す．どちらのパズルも，解の存在の検証は NP 完全となることが知られている [4, 5]．

3. 準備

本稿ではグラフ $G = (V, E)$ 上の 2 頂点 $i, j \in V$ を端点とする辺を $e_{i,j}$（もしくは $e_{j,i}$）で表す．また，G 上のパスやサイクル，部分グラフを辺の部分集合 E' と同一視する．すなわち例えば図 3 左の 4 頂点完全グラフ K_4 において，頂点を順に 1,3,4,2 と辿るパスは，辺集合 $\{e_{1,3}, e_{2,4}, e_{3,4}\}$ と同一視される．K_4 における

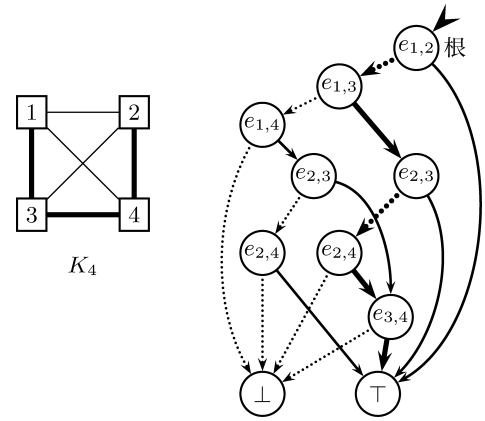

図 3　グラフ K_4（左）とその上のすべての 1–2 パスを表現する ZDD（右）．パス $\{e_{1,3}, e_{2,4}, e_{3,4}\}$ を強調表示している．

頂点 1 から 2 への単純パスすべてを ZDD で表現したものを図 3 の右に示す．ZDD の詳説については本特集中の他稿に譲るが，本稿における用語法は以下のとおりとする．まず，与えられたグラフの部分グラフを ZDD で辺集合として表現するが，ZDD もグラフの一種であるから，2 種のグラフの区別のために，探索対象グラフでは頂点・辺という語を，ZDD では節点・枝という語を用いる．集合中に要素が含まれることを示す枝を **1–枝**（HI 枝とも）と呼び，含まれないことを示す枝を **0–枝**（LOW 枝とも）と呼ぶ（図 3 ではそれぞれ実線と点線で示してる）．集合が当該集合族に入っていることを示す終端節点を ⊤，入っていないことを示す終端節点を ⊥ で表記する．

3.1 フロンティア法によるパスマッチングの列挙

ナンバーリンクの解は特定の性質を満たすパスマッチング（互いに頂点を共有しないパスの集まり）であり[2]，また，スリザーリンクの解であるサイクルの真の部分グラフはパスマッチングである．われわれのこれらのパズルに対する解答器・問題生成器はフロンティア法によるパスマッチング列挙アルゴリズムを基にしており，本節ではパスマッチング列挙アルゴリズム（アルゴリズム 1）を詳説する．実際のところ，端点に制約のないパスマッチング列挙は Knuth の s–t パス列挙アルゴリズムよりも，考慮すべき制約が少ないため，より単純である．具体的には，端点の指定がなく，いかなる頂点も端点となりうるので，頂点がフロンティ

[2] ナンバーリンクパズルの解答列挙は，本特集他稿 [6] における複数端点対パス列挙と完全に等価であるが，後の問題生成器の説明の便宜上，ここではパスマッチング列挙を説明の出発点とする．

アルゴリズム 1　パスマッチングの列挙

```
Data: グラフ G = (V, E)
begin
    let N ← { 根節点 };
    foreach e ∈ E do
        foreach v ∈ N do
            v を e でラベル付ける;
            v の 0–枝の先に節点を作り，適切な mate
            関数を割り当て，N′ に入れる;
            if mate_v が辺 e を加えることを許す then
                v に 1–枝の先に節点を作り，適切な
                mate 関数を割り当て，N′ に入れる;
            else
                v の 1–枝を ⊥ に結線する;
            end
        end
        N′ 中で同じ mate 関数をもつ節点を合併する;
        N ← N′, N′ ← ∅;
    end
    N 中の唯一節点を ⊤ とする;
    return 完成した ZDD;
end
```

アを去る際にその次数を問題にする必要がない．

G の辺にはあらかじめ全順序を固定し，この順序に従って処理を行う．作成途中の ZDD における根から途中の節点 $v(\neq \bot)$ までのパス π は，対象グラフ G 上の部分グラフ（＝辺の集合）P_π に対応することになる．アルゴリズムは π が $\bot$ に辿り着かない限り，常に P_π がパスマッチングになるように ZDD を構築する．作成途中の ZDD の節点 v には，P_π の幾何的な情報を mate_v という V から $V \cup \{0\}$ への部分関数として記憶させる．その定義域は，現在までに処理が開始された辺（すなわち v 自体および v の上流の節点のラベル）および未だ処理が終わっていない辺の 2 種類の辺（v をラベルする辺はこの 2 種の性質をもつものとする）に接続する頂点であり，この頂点集合を**フロンティア**と呼ぶ．フロンティア中の各頂点 i について，s–t パスの列挙と同様に，

$$\mathrm{mate}_v(i) = \begin{cases} 0 & : P_\pi \text{ における } i \text{ の次数が } 2 \text{ のとき}, \\ i & : P_\pi \text{ における } i \text{ の次数が } 0 \text{ のとき}, \\ j & : P_\pi \text{ が } i\text{–}j \text{ パスを含むとき}, \end{cases}$$

と定義する[3]．現在までに得られているパスマッチングに特定の辺 $e_{k,l}$ を加えてもなおパスマッチングになるかどうかは mate 関数を調べることで判定可能であり，作成途中の ZDD の節点の mate 関数を用いて，その子節点の mate 関数が計算できることも s–t パス列挙と同様である．したがって，2 つの節点が同じ辺ラベルと同じ mate 関数をもつならば，その下流グラフは同型になるので，これらの節点は合併して 1 節点とする．以上がフロンティア法によるパスマッチング列挙のあらましである．

実際のアルゴリズムの実装においては，複数の節点を作ったうえで，それらの合併の可否を検討し合併を行うような無駄はせずに，ある節点 u の子として新たな節点 v を作る前に，それが既存の節点 v' と合併しうるものになるなら，親節点 u からの枝を直接 v' に張るべきである．しかし，解答器・問題生成器の理解を容易にするため，この合併過程を持つアルゴリズムを基に説明をする．

4. フロンティア法によるリンクパズルソルバー

4.1 ナンバーリンクソルバー

ナンバーリンクパズルの解答列挙アルゴリズムは，前節のパスマッチングアルゴリズムに端点に関する制約を導入することで実現できる．問題が i–j パスを構成することを要請していたとする．このときいかなる ZDD 節点 v においても $\mathrm{mate}_v(i) = 0$ は許されないから，そのような辺の選択に対応する ZDD 節点は $\bot$ に合併させる．また，頂点 i がフロンティア集合から去る場合，すなわち i に接続する最後の辺の処理を終えるとき，$\mathrm{mate}_v(i) = k \neq j$ でありかつ k もすでにフロンティアから去っているなら，i–k パスが固定されており，これにいかなる辺を加えても i–j パスに成長することはないので，これも許されない．一方，問題が頂点 i についてパスの端点となることを要請していないならば，i がフロンティア集合から去るとき，その次数は 0 か 2 でなければならない．これらの制約が満たされているかどうかは mate 関数を観察することで判定できる．このように，制約に矛盾する辺の選択に対応する ZDD 節点を $\bot$ に合併していくことで，全パスマッチングのうち解のみを列挙できる．

4.2 スリザーリンクソルバー

スリザーリンクの解は，特定の制約を満たすサイクルである．与えられたグラフに対してサイクルとなるすべての部分グラフを ZDD で列挙するフロンティア法アルゴリズムは [6] で紹介されている．これに，スリザーリンクに特有の制約を考慮した修正を加える．インスタンスのヒントを $H : 2^E \rightharpoonup \mathbb{N}$ とする．アルゴ

[3] なお，$\mathrm{mate}_v(i) = j$ で j がフロンティア外のときは，特殊な記号 $*$ を導入して $\mathrm{mate}_v(i) = *$ とすることで，後述の節点の合併が促進され高速化ができる．しかしこのアルゴリズムは後節のリンクパズルの諸問題のためのアルゴリズムの準備として用いるため，このような最適化は行っていない．

リズムは mate 関数に加えて，count 関数を ZDD の各節点に割り当てる．count 関数は，H の定義域中の辺がどれだけ使われているかを管理する．count 関数の定義域は，H に同じである．2 節点の合併は，mate に加え count 関数も一致したときに実行する．作成途中の ZDD で根からのパス π を辿って節点 v に辿り着くとき，H の定義される各 D についてその count 関数の値を $\mathrm{count}_v(D) = |P_\pi \cap D|$ と定義する．ここで $P_\pi \subseteq E$ は π に対応する G 上のパスマッチングである．

この関数の更新はほとんど自明である．ある ZDD 節点 v が $e_{i,j}$ でラベル付けされているとき，その 1–枝側の子節点 u の count は

$$\mathrm{count}_u(D) = \begin{cases} \mathrm{count}_v(D) + 1 & : e_{i,j} \in D \text{ のとき}, \\ \mathrm{count}_v(D) & : e_{i,j} \notin D \text{ のとき}, \end{cases}$$

となる．0–枝側の子節点 u' については $\mathrm{count}_{u'}(D) = \mathrm{count}_v(D)$ である．

ヒントと矛盾するパスマッチングに対応する ZDD の節点 v は $\bot$ に合併する．すなわち，多すぎる辺を用いる場合 ($\mathrm{count}_v(D) > H(D)$) や，今後すべての辺を使用しても H の制約を満たせない場合 ($H(D) - \mathrm{count}_v(D)$ が D 中の未処理辺の数を上回る場合) である．

4.3 実験

われわれは 4.1 節と 4.2 節のアルゴリズムを C++ で実装し，ニコリ社の例題集 [7, 8] から適当に問題を抜粋し，解答させた．実験に用いた計算機の CPU は AMD Opteron Processor™ 8393, 3.09 GHz で 512 GB のメモリを搭載している．比較のため Sugar 制約ソルバー [9] を用いたほか，スリザーリンクには web 上で公開されているソルバー SLINK [10] を用いた．結果を表 1（ナンバーリンク）と表 2（スリザーリンク）に示す．

スリザーリンクに特化したプログラムである SLINK が最も高速に動作する一方，SAT ベースのソルバーである Sugar は，汎用性の反面で，ソルバーとしてはわれわれの提案アルゴリズムよりも遅い傾向が見られる．われわれのアルゴリズムはグラフ上のリンク列挙アルゴリズムという，両者の中間的な汎用性（特殊性）をもつ手法に基づいており，その意味で妥当な結果が得られたと言えよう．

なお，今回の実験は唯一解を持つ問題のみを対象としているが，われわれのアルゴリズムはすべての解を 1 回の実行で求められる点に注意されたい．一般のソルバーは複数解を求めるように設計されていない．Sugar では特定解を除外する論理式を与えることで別解を求めることもできるが，すべての解を求めるためには解の数だけプログラムを実行しなければならない．

表 1 ナンバーリンク問題に対する解答時間（秒）

問題番号 (size)		Sugar	提案手法
1	(8×8)	1.179	0.008
15	(8×8)	1.336	0.004
30	(10×10)	1.896	0.016
43	(10×10)	1.924	0.072
52	(10×10)	1.520	0.012
64	(10×10)	1.704	0.136
72	(10×10)	1.388	0.012
79	(10×10)	1.496	0.220
85	(20×15)	7213.079	862.518
99	(20×15)	14.097	1658.772

表 2 スリザーリンク問題に対する解答時間（秒）

問題番号 (size)		Sugar	SLINK	提案手法
1	(11×11)	5.460	0.001	0.002
12	(11×11)	4.696	0.004	0.121
25	(11×11)	6.496	0.001	0.002
37	(11×11)	8.773	0.004	0.022
43	(19×11)	8.349	0.001	0.014
54	(19×11)	7.380	0.008	0.075
68	(25×15)	12.433	0.004	0.107
77	(25×15)	13.281	0.004	0.191
89	(37×21)	48.035	0.016	3.933
96	(37×21)	186.612	0.104	10.540

5. ナンバーリンクの問題列挙

5.1 アルゴリズム

ナンバーリンクの問題に対する解が，すべての頂点を網羅しないとき，その解を**短絡解**と呼ぶ．ナンバーリンクの**良問**とは，唯一解をもち，かつその解が短絡解でない問題をいう．本節では，グラフ G を固定したときに良問を与えるヒントを ZDD で列挙するアルゴリズムを提案する．以下ではヒント H と問題 (G, H) を同一視する．

まず準備として，解を 1 つ以上もつ可解な問題ヒントの列挙を考えよう．グラフ上のパスマッチング P を構成する各パスの端点対 $V_1, \ldots, V_p$ からなる集合 $T = \{V_1, \ldots, V_p\}$ は，少なくとも P という解を 1 つもつ問題とみなせる．可解問題列挙のアルゴリズムは，3.1 節のパスマッチングの列挙アルゴリズムと全く同

様に ZDD を構築しながら，各節点に新たな情報を付加・管理する．構築される ZDD において根から各節点までのパス π が表すパスマッチング P_π 中のパスには，両の端点がフロンティアを去っているもの（以下，**固定端点対**と呼ぶ）と，少なくとも一端がフロンティアに入っているものの 2 種類があるが，後者は mate 関数によって管理されていたから，前者の固定端点対の集合 T を新たに記憶すれば十分である．4.1 節で，頂点がフロンティアを去る際に固定される i–j パスが確認できることを述べた．そのような場合に端点対 $\{i, j\}$ を固定端点対集合 T に加えることになる．このように，固定端点対集合は単調に成長する．ただし，一般には，ある ZDD 節点 v に到達する根からのパス π_1 と π_2 について，対応する 2 つのパスマッチング P_{π_1} と P_{π_2} の固定端点対集合は異なりうる．したがって，各節点 v に至る根からのパス $\pi_1, \pi_2, \ldots$ について，固定端点対集合 $T_{\pi_1}, T_{\pi_2}, \ldots$ を集めた族 S_v を節点 v に割り当てる．そして，i–j パスが固定されるとき，これらの固定端点対集合それぞれに $\{i, j\}$ を加える．アルゴリズム 1 において 2 節点 v_1, v_2 が合併するとき，合併後の ZDD 節点の固定端点対集合族は $S_{v_1} \cup S_{v_2}$ となる．最終的に終端節点 $\top$ に割り当てられた族 $S_\top$ 中の端点対集合が，解をもつ問題のヒントになる．

次に，解をもつのみならず，その解が唯一でありかつ全頂点を使用する，という制約を考慮する．このため，各節点 v に割り当てられた族 S_v の要素のうち，良問に成長しないことがわかっている固定端点対集合（**悪の集合**と呼ぶ）の族 $B_v \subseteq S_v$ を別に管理する．ある問題 T が異なる 2 解をもつならば，それぞれの解に対応する ZDD 上の根から終端節点への 2 つの異なるパス π_1 と π_2 があって $T_{\pi_1} = T_{\pi_2} = T$ である．これらのパスが節点 v で合流するとき，v に至る π_1 と π_2 の接頭パス π_1' と π_2' についても $T_{\pi_1'} = T_{\pi_2'}$ である．このように，1 つの固定端点対集合 $T_{\pi_1'} = T_{\pi_2'}$ が異なるパスに由来し合流するときに，この固定端点対集合は悪の集合 B_v に入れられる．すなわち，アルゴリズム 1 で節点 v_1 と v_2 の合併をして新たな節点 v とするとき，

$$B_v = (S_{v_1} \cap S_{v_2}) \cup B_{v_1} \cup B_{v_2}$$

が合併後の悪の集合族になる．これによって複数解を持つ問題を除外できる．また，短絡解の排除は次のように実現できる．頂点 i がフロンティアから去る際に，i の次数が 0 ならば，これは短絡解になるから，対応 ZDD 節点 u に付された固定端点対集合はことごとく悪の集合になる．すなわち $B_u = S_u$ となる．この場合の悪の集合族の計算は，他の節点との合併の前に行われる．最終的に $S_\top \setminus B_\top$ が良問集合である．

アルゴリズムの実装にあたっては，巨大な端点対集合族 S_v, B_v の管理方法が問題となる．巨大な集合族の管理と演算は本来的に ZDD の得意とするところであるから，われわれは ZDD パッケージを用いてこれを実現した．すなわち，フロンティア法で構築する ZDD 上の各節点に，2 つの ZDD を割り当てた．ここで端点対集合族を管理する ZDD の節点ラベルは端点対である．

5.2 問題生成実験

実験には，AMD Opteron Processor™ 6136, 2.40 GHz CPU および 256 GB のメモリを搭載する計算機を用い，良問を探索した．例えば 5×6 正方格子上の全良問 784,030,205 個（対称性を考慮しない）は 3,553 秒で求められた．6×6 正方格子上の最小ヒント数は 3 対であることを確認し，その全良問 304 個を 12,161 秒で列挙した．図 1 はその一例である．しかし，6×6 正方格子上の全良問の探索は 24 時間以上かけても終了しなかった．さらに，正方格子以外の形状のグラフについても生成実験を行った．$3 \times 3 \times 2$ の 3 次元格子では，25.7 秒で全良問 557,907 個を列挙できた．また，図 4 左のようなグラフ上の全良問は全部で 55,854,502 個，この列挙には 1 時間程度を要した．これらのうち最小ヒント数 3 対による問題は 432 問である．一例を図 4 右に示す．

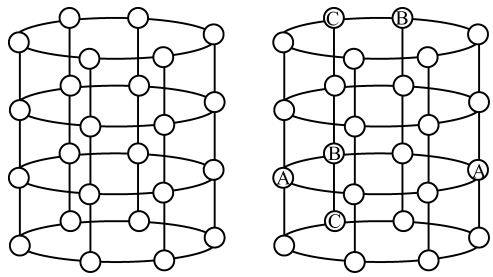

図 4 非正方格子グラフと生成されたナンバーリンク問題例

6. スリザーリンクの問題列挙と問題作成支援

6.1 問題列挙

解が唯一に定まるスリザーリンクの問題を**良問**と呼ぶ．グラフ $G = (V, E)$ とその部分グラフとなるサイクル $C_* \subseteq E$ が与えられたときに，そのサイクルのみを解とするようなスリザーリンクの良問を与えるヒ

ント関数を列挙する手法について議論する．問題のヒント関数 $H : 2^E \rightharpoonup \mathbb{N}$ の定義域としては，グラフの辺の全部分集合 2^E を考慮に入れるのではなく，例えば通常のスリザーリンクでは定義域には単位正方形を成す 4 辺の集合のみが現れるように，ヒント最大定義域 $F \subseteq 2^E$ はあらかじめ限定されているものとする（さもなくば辺集合の冪集合の冪集合が探索空間となり，絶望的な計算資源が必要になる）．その際，最も「親切な」ヒント H_* の定義域は F そのものであり，その値は各 $D \in F$ について

$$H_*(D) = |C_* \cap D|$$

と決まる．ヒント関数の列挙といっても，定義域が決まればその値は C_* によって一意に定まるから，われわれが実際に列挙するのは，F の部分集合 F' で，F' をちょうど定義域とする H_* の部分関数 $H_*|_{F'}$ が良問となるようなものすべてということになる．この列挙も ZDD 上で行うため，G の各辺に対応する変数に加え，F の各要素に対応する変数を用意した．ここで $\mathcal{C} \subseteq 2^E$ を G 上の全サイクル集合とすると，良問を与える定義域 $F' \subseteq F$ の条件は

$$\forall C \in \mathcal{C} \setminus \{C_*\}, \exists D \in F', |C \cap D| \neq H_*(D) \quad (6.1)$$

となる．ここで $\mathcal{C}$ を表す ZDD は，サイクル列挙アルゴリズムによって得られているから，あとは ZDD パッケージの標準的な代数演算を組み合わせることで，上の論理式 (6.1) を満たす F' を ZDD で列挙できる．

6.2 問題作成支援

ZDD パッケージの代数演算を活用することで，単に良問すべてを列挙するだけでなく，柔軟な問題作成が支援できる．良問の定義上，良問ヒントの定義域を拡張したものは常に良問であるが，より解答はやさしくなる．ヒント数が最小であったりヒント定義域が極小になるような難度の高い良問のみを抽出したり，人手で作成した問題の冗長なヒントを指摘することなどは，ZDD パッケージの代数演算の組み合わせによって容易に実現できる．また，格子グラフ上のスリザーリンクでは，一般に “0” や “3” は情報量が大きく解答を容易にする傾向があるのに対し，“2” が記入されたセルは 4 辺の使用方法が 6 通りもあるため情報量が少ない．こういったヒントの情報量の重みを加味して難問を抽出することも，ZDD の得意とするところである．

図 5 の上段に示すような，格子グラフ G，サイクル C_* および最大定義域 F を入力として与えた場合 ($|F| = 36$)，全良問は 1,669,424 個，うちヒント域極小良問は 1,850 個，ヒント数最小良問は 4 個であった．

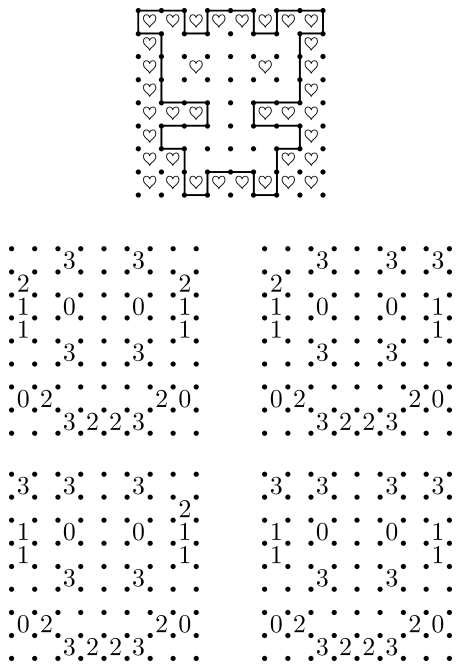

図 5 スリザーリンク問題生成例．記号 “♡” の周囲 4 辺からなる辺集合が F の定義域に入る．

図 5 の中下段にヒント数最小良問 4 個を示す．なお，この計算に要した時間は，4.3 節で使用した計算機上で約 2 秒であった．しかし一方で，すべてのセルを定義域に含めた場合 ($|F| = 64$) は 1 日かけても計算は終了しなかった．

7. 結

本稿では，フロンティア法を用いてナンバーリンクおよびスリザーリンクの解答列挙アルゴリズムおよび問題列挙アルゴリズムを提案した．すでに提案されているパズルソルバーと異なり，われわれのアルゴリズムはすべての解を一度に求めることができる．また，もちろん各パズル特有の知識を利用して高速化を図ることも可能であろうが，本稿ではフロンティア法が，ほかのリンクパズルのみならず，さまざまな制約を満たす部分グラフ抽出問題に柔軟に応用可能である一般的な手法であることを強調するために，ソルバーアルゴリズムは単純なアイディアの合成にとどめた．本稿で示した手法は，適切な補助関数を導入することによって，ヤジリン，スラローム，ましゅ，などのよく知られた（もしくはよく知られていない）ほかのリンクパズルにも応用が可能であると思われる．

参考文献

[1] D. Knuth, *The Art of Computer Programming, Vol. 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.

[2] S. Minato, Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of 30th international Design Automation Conference*, *DAC '93*, pp. 272–277, New York, USA, 1993.

[3] R. Yoshinaka, T. Saitoh, J. Kawahara, K. Tsuruma, H. Iwashita and S. Minato, Finding all solutions and instances of numberlink and slitherlink by ZDDs. *Algorithms*, **5**(2), pp. 176–213, 2012

[4] M. R., Kramer and J. van Leeuwen, Wire-routing is NP-complete. Report No. RUU-CS-82-4, Department of Computer Science, University of Utrecht, Utrecht, 1982.

[5] 八登崇，スリザーリンクの NP 完全性について，情処研報 **2000-AL-74-4**, pp. 25–32, 2000.

[6] 川原純，湊真一，グラフ列挙索引化技法の種々の問題への適用，オペレーションズ・リサーチ, **57**(11), pp. 604–609, 2012.

[7] ナンバーリンク 1, ニコリ，1989.

[8] スリザーリンク 1, ニコリ，1992.

[9] 田村直之，パズルを Sugar 制約ソルバーで解く．http://bach.istc.kobe-u.ac.jp/sugar/puzzles/

[10] 伊戸川暁，スリザーリンク解答プログラム，http://www2.ttcn.ne.jp/~itogawa/product/slitherlink.html

