# 整数計画法を用いた高速なSlitherlinkパズルの解法

情報処理学会論文誌 Vol.54 No.8 2103–2108 (Aug. 2013)

# 整数計画法を用いた高速なSlitherlinkパズルの解法

石濱 友裕[1,†1,a)] 久野 誉人[1,b)]

受付日 2012年12月26日, 採録日 2013年5月18日

**概要**：本論文では，人気のあるペンシルパズル “Slitherlink” の解法について議論する．多くのパズルがそうであるように，Slitherlink は NP 完全であり，整数計画法を使って求解が可能である．このパズルが，これまでに知られている方法よりも簡潔に定式化でき，はるかに高速に解けることを紹介する．

**キーワード**：ペンシルパズル，Slitherlink，整数計画法

# A Faster Solution to the Slitherlink Puzzle Using Integer Programming

Tomohiro Ishihama[1,†1,a)] Takahito Kuno[1,b)]

Received: December 26, 2012, Accepted: May 18, 2013

***Abstract:*** This paper addresses a solution to “Slitherlink”, one of popular pencil puzzles. Like many other puzzles, Slitherlink is NP-complete and can be solved using integer programming. We show that the puzzle can be formulated more concisely and solved much faster than in the existing formulation.

***Keywords:*** pencil puzzle, Slitherlink, integer programming

## 1. はじめに

ペンシルパズルとは，パズルに答の一部を徐々に書き込んでゆき，答を完成させるパズルの総称である．その手軽さから多くの人に楽しまれており，専門の雑誌もいくつか存在する．本論文で扱う Slitherlink パズルはペンシルパズルの 1 つで，格子の中に書かれている数字をヒントに，線分を結んで大きな 1 つの輪をつくるのが目的である（**図 1**）．

Slitherlink は，他の多くのパズルがそうであるように解の有無判定が NP 完全であり [14]，また ASP 完全[*1]であることも示されている [6]．2007 年には，格子を正 $k$ 角格子に一般化した $k$-Slitherlink も考案され，いくつかの $k$ について ASP 完全となることが分かっている [3], [12]．このパズルの計算機による厳密な解法として，整数計画法による解法 [9]，充足可能性による解法 [10]，二分決定図による解法 [7] がある．整数計画法による解法は，類似のパズル Slitherlinks を整数計画問題として定式化し，これを逐次解くことによって Slitherlink の解を得るというもので，2004 年に杉村によって考案された．杉村の解法が他の 2 つの解法に比べて経験的に高速であることは文献 [7] で報告されている．しかしながら，この解法が要する変数や制約式の数は多く，手で解ける問題ですら現実的な時間では解の得られないことがある．またヒューリスティクス [8] も存在するが，本論文では厳密な解法のみを扱うこととし，杉村とは異なる整数計画問題の定式化を行うことで変数と制約の数を大幅に減らせることを示す．また計算実験によって，杉村の解法よりも高速にパズルの解が得られることを実証し，既存の厳密解法に対する提案手法の優位性を示す．

本論文の構成は次のとおりである．2 章では Slitherlink のルールと杉村の提案した定式化を紹介し，3 章では新しい定式化の方法について詳しく解説する．4 章では杉村の解法と提案する解法との比較実験の結果を報告し，本研究

[1] 筑波大学大学院システム情報工学研究科
University of Tsukuba, Graduate School of Systems and Information Engineering, Tsukuba, Ibaraki 305–8573, Japan
[†1] 現在，新日鉄住金ソリューションズ株式会社
Presently with NS Solutions Corporation, Chuo, Tokyo 104–8280, Japan
[a)] ishihama@syou.cs.tsukuba.ac.jp
[b)] takahito@cs.tsukuba.ac.jp

[*1] 1 つ解が与えられたときに別解が存在するかを判定する問題が NP 完全となる．

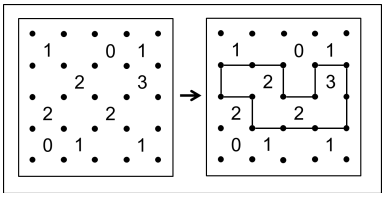

図 1 Slitherlink パズルと答え
**Fig. 1** Example of Slitherlink and its solution.

のまとめを行う．

## 2. Slitherlink のルールと杉村の解法

Slitherlink のルールを文献 [14] の表記に従い，正確に記述すると以下のようになる．

**(a)** 盤面は長方形で，格子をなす点の集合として与えられる．格子間の単位長を 1 としたとき，その長方形の縦の長さ $m$ と横の長さ $n$ に対して，$m \times n$ を盤面の**サイズ**と呼ぶ．

**(b)** 4 つの点に囲まれた 1 辺の長さが 1 の正方形を**面**という．面には 0，1，2，3 のいずれかの数字が書かれていることがある．

**(c)** 目標は，格子上で隣接する 2 点間に線分を引いて，次の条件を満たす解を得ることである．

**(c1)** 盤面には 1 つの輪（初等的な閉路）しか存在しない．

**(c2)** 面に書かれた数と，その面を構成する 4 辺に引かれた線分の数は等しい．

**(c3)** 各点からでる線分の数は 0 本もしくは 2 本である．

**(c4)** 盤面の外側には線分を引かない．

ここで辺は，隣接する点の間の，線分が引ける領域のことを示す．**図 2** の 3 つの例は，中心の点が条件 (c3) を満たさないため，このパターンの線分は Slitherlink の解に含まれない．

点 $(i,j)$ の左上の面を $F(i,j)$ $(i=1,\ldots,m,\ j=1,\ldots,n)$ で表すことにする．面 $F(i,j)$ に数字が書かれている $(i,j)$ の集合を $P$ としよう．面に書かれた数字を

$$N(i,j) \in \{0,1,2,3\} \quad ((i,j) \in P)$$

で表す．杉村 [9] は，すべての辺に 0-1 変数を対応させてパズルを整数計画問題として定式化している．そのため，縦の辺に対応する変数 $v_{ij} \in \{0,1\}$ と，横の辺に対応する変数 $h_{ij} \in \{0,1\}$ が必要となる．変数の値が 1 のとき，対応する辺に線分が引かれる．制約条件は (c1)–(c4) から定式化できるが，(c1) の定式化にはさらに膨大な数の変数を追加する必要がある．杉村は，この欠点を克服するため，(c1) を無視した新たなパズル **Slitherlinks** を定義している．残りの 3 つの条件 (c2)–(c4) から，Slitherlink と違って複数の閉路のできることが許される．たとえば**図 3** で，右側の解は Slitherlink の解とならないが，Slitherlinks ではこの 2 つはどちらも解である．

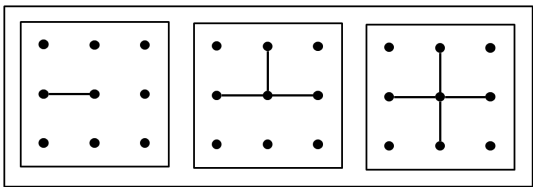

図 2 条件 (c3) を満たさない例
**Fig. 2** Examples that do not satisfy (c3).

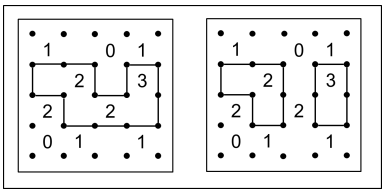

図 3 Slitherlinks の 2 つの解
**Fig. 3** Two solutions of Slitherlinks.

### 2.1 Slitherlinks の定式化

上記の定数，変数を用いると，盤面サイズ $m \times n$ の Slitherlinks は次のように定式化できる：

$$
\begin{aligned}
&\min \quad \sum_{i=0}^{m+1}\sum_{j=0}^{n} v_{ij} + \sum_{i=0}^{m}\sum_{j=0}^{n+1} h_{ij} \\
&\text{s.t.} \quad v_{i,j-1} + v_{ij} + h_{i-1,j} + h_{ij} = N(i,j) \quad ((i,j) \in P) \\
&\left.\begin{aligned}
&-v_{ij} + v_{i+1,j} + h_{ij} + h_{i,j+1} \geq 0 \\
&v_{ij} - v_{i+1,j} + h_{ij} + h_{i,j+1} \geq 0 \\
&v_{ij} + v_{i+1,j} - h_{ij} + h_{i,j+1} \geq 0 \\
&v_{ij} + v_{i+1,j} + h_{ij} - h_{i,j+1} \geq 0 \\
&v_{ij} + v_{i+1,j} + h_{ij} + h_{i,j+1} \leq 2
\end{aligned}\right\} \begin{pmatrix} i=0,\ldots,m \\ j=0,\ldots,n \end{pmatrix} \\
&v_{0,j} = v_{m+1,j} = 0 \quad (j=0,\ldots,n) \\
&h_{i,0} = h_{i,n+1} = 0 \quad (i=0,\ldots,m) \\
&v_{ij} \in \{0,1\} \quad (i=0,\ldots,m+1;\ j=0,\ldots,n) \\
&h_{ij} \in \{0,1\} \quad (i=0,\ldots,m;\ j=0,\ldots,n+1).
\end{aligned}
\tag{1}
$$

この制約条件が (c2)–(c4) に等しいことを確認しよう．まず 1 つめの制約は $F(i,j)$ を構成する 4 辺についての制約である．$N(i,j)=k$ のとき，この 4 辺のうち $k$ 辺に線分が引かれるため，この制約は (c2) に等しい．2–6 つめの制約はすべて点 $(i,j)$ についての制約である．2–5 つめは，すべての変数の符号がちょうど 1 回ずつ反転している．このことから，どこか 1 つの辺に線分が引かれると，他の少

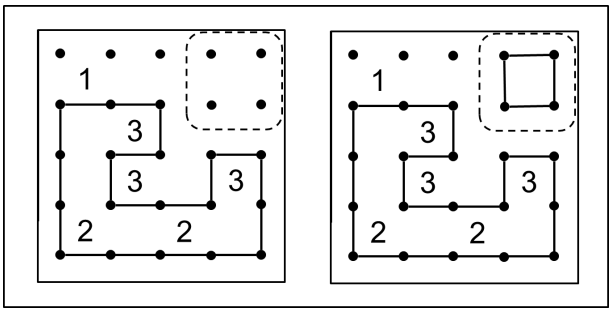


図 4 最小化の理由
**Fig. 4** Reason for minimization.

なくとも 1 つの辺にも線分が引かれることが分かる．さらに 6 つめの制約を合わせると，点 $(i,j)$ から伸びる線分の数は 0 本か 2 本になり，(c3) が満たされる．また盤面の周囲の面や点を内側と同じように扱えるように，変数 $v_{0,j}$，$v_{m+1,j}$ と $h_{i,0}$，$h_{i,n+1}$ も用いている．7–8 つめの制約は，これらの変数が条件 (c4) を満たすことを保証している．

問題 (1) は線分の数の最小化を目的としているが，その理由は最終的な目標が Slitherlinks ではなく Slitherlink にあるためである．たとえば，**図 4** のように右上の面とその周辺に数字がない問題を考えよう．このとき，右上の領域では (c2) の制約が効かないので，(c3)–(c4) を満たす限りどのように線分を引いても，Slitherlinks の解として認められる．この例では，考えられる解は図 4 の 2 つである．左側の解は Slitherlinks の解であり，かつ Slitherlink の解でもある．一方，右側の解は Slitherlinks の解ではあるが，Slitherlink の解ではない．したがって，Slitherlink の解を得るためには，線分の数が最小になるような解を選ぶ必要がある．

### 2.2 Slitherlink の厳密解法

杉村は，式 (1) を解いて Slitherlinks の解を求めたのち，複数の閉路が存在する場合には切除平面法を用いてこれを除去している．Slitherlinks の解に含まれる最も小さな閉路を構成する縦の辺の集合を $L_v$，横の辺の集合を $L_h$ としよう．この閉路を構成する辺の数を $k$ とすれば，

$$\sum_{(i,j)\in L_v} v_{ij} + \sum_{(i,j)\in L_h} h_{ij} \le k-1$$

を制約として加えることによって，この閉路を解に現れないようにできる．これを Slitherlinks の解に閉路が 1 つになる（Slitherlink の解となる）まで繰り返すことで，Slitherlink の解を生成できる．

## 3. 解法の高速化

杉村の方法は，Slitherlinks が辺に線分を引いてゆくパズルであることを考えると自然な定式化であるが，縦辺，横辺にそれぞれ変数が必要なため，盤面のサイズが $m\times n$ ならば，少なくとも $2mn$ 個の変数が必要になる．さらに条件 (c3) を表す制約の本数は，各点に 5 つの不等式が必要になるので全部で約 $5mn$ 本となる．このため，問題がある程度の大きさになると，整数計画法で厳密な解を求めることは困難である．そこで，辺ではなく面に注目することで，制約，変数の数をどちらも大幅に抑える定式化を提案する．ここで注意すべきことは，**「輪を構成する線分を引くことと，輪の内外を指定することが等価」**な点である．たとえば，**図 5** の左側のように輪を構成する線分が引かれているとしよう．このとき，輪の内側と外側が一意に定まり，右側の連結な集合が得られる．逆に，図 5 右側の集合から，左の輪を復元することも可能である．

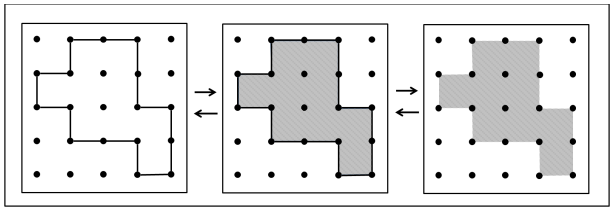

図 5 重要な観察
**Fig. 5** Key observation.

以上の観点から，まず各面が輪の内側か否かを 0-1 変数で表し，Slitherlinks の条件を満たす連結な領域（もしくは領域群）を整数計画法を使って求める．得られた集合から輪を復元することで，対応する Slitherlinks の解を得るというのが提案するアプローチである．

### 3.1 新しい Slitherlinks の定式化

定式化に用いる変数は，$F(i,j)$ が輪の内側にあるとき 1，外側のときに 0 となる 0-1 変数として，$x_{ij}\in\{0,1\}$ を用いる．Slitherlinks を解いたとき，輪が入れ子状に生成される場合がある．そこで一番外側の輪に含まれる面の変数を 1，その内側の面の変数を 0，さらにその内側の面の変数を $1,\cdots$ として交互に定めることにする．

まず，条件 (c2) について考えよう．1 つの面 $F(i,j)$ のまわりの 4 辺のうち，いくつかに線分が引かれているものとする．この線分は輪の内側と外側の間に引かれているので，$F(i,j)$ が輪の内側（$x_{ij}=1$）のときには，線分を挟んで $F(i,j)$ に隣接する面は輪の外側に位置する．逆に，$F(i,j)$ が輪の外側（$x_{ij}=0$）のときには，線分を挟んで隣接する面は輪の内側である．つまり，$(i,j)\in P$ ならば，

$$x_{i-1,j}+x_{i+1,j}+x_{i,j-1}+x_{i,j+1}=N(i,j) \quad (\text{if } x_{ij}=0),$$
$$x_{i-1,j}+x_{i+1,j}+x_{i,j-1}+x_{i,j+1}=4-N(i,j) \quad (\text{if } x_{ij}=1)$$

が成り立つ．さらにこの式は，以下のように 1 つにまとめることができる：

$$x_{i-1,j}+x_{i+1,j}+x_{i,j-1}+x_{i,j+1} = N(i,j)(1-x_{ij})+(4-N(i,j))x_{ij}.$$

次に条件 (c3) を定式化する．ある点 $(i,j)$ について，こ

の点を共有する 4 つの面の内外の組合せは $2^4$ 通りあるが，そのうち回転や反転して同じものを除くと，**図 6** の 6 つに分類できる．この 6 つについて，中心の点 $(i, j)$ から伸びる線分を数えると，左下の図形では線分の数が 4 つとなって制約を満たさない．つまり (c3) において除外すべきパターンは，図 6 の左下の図形とそれを反転させた図形の 2 つになる．点 $(i, j)$ の周りの面に対応する変数 $x_{ij}$，$x_{i+1,j}$，$x_{i,j+1}$，$x_{i+1,j+1}$ に対して，次の式の値を考えよう：

$$\rho_{ij} = x_{ij} + (1 - x_{i+1,j}) + (1 - x_{i,j+1}) + x_{i+1,j+1}.$$

明らかに $0 \le \rho_{ij} \le 4$ であるが，0 あるいは 4 となるのは先ほどの除外すべきパターンのときのみで，それ以外のパターンのときは $1 \le \rho_{ij} \le 3$ を満たす．つまり (c3) の条件は次の 2 式によって与えることができる：

$$x_{ij} + (1 - x_{i+1,j}) + (1 - x_{i,j+1}) + x_{i+1,j+1} \le 3,$$
$$x_{ij} + (1 - x_{i+1,j}) + (1 - x_{i,j+1}) + x_{i+1,j+1} \ge 1.$$

以上のことから提案する Slitherlinks の定式化は次の最適化問題としてまとめられる：

$$\begin{aligned}
\max \quad & \sum_{i=0}^{m+1} \sum_{j=0}^{n+1} x_{ij} \\
\text{s.t.} \quad & x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} \\
& \quad = N(i,j)(1-x_{ij}) + (4 - N(i,j))x_{ij} \qquad ((i,j) \in P) \\
& x_{ij} + (1 - x_{i+1,j}) + (1 - x_{i,j+1}) + x_{i+1,j+1} \le 3 \\
& x_{ij} + (1 - x_{i+1,j}) + (1 - x_{i,j+1}) + x_{i+1,j+1} \ge 1 \\
& \qquad (i = 1, \ldots, m-1;\ j = 1, \ldots, n-1) \\
& x_{0j} = x_{m+1,j} = 0 \quad (j = 0, \ldots, n+1) \\
& x_{i0} = x_{i,n+1} = 0 \quad (i = 0, \ldots, m+1) \\
& x_{ij} \in \{0, 1\} \quad (i = 0, \ldots, m+1;\ j = 0, \ldots, n+1).
\end{aligned} \tag{2}$$

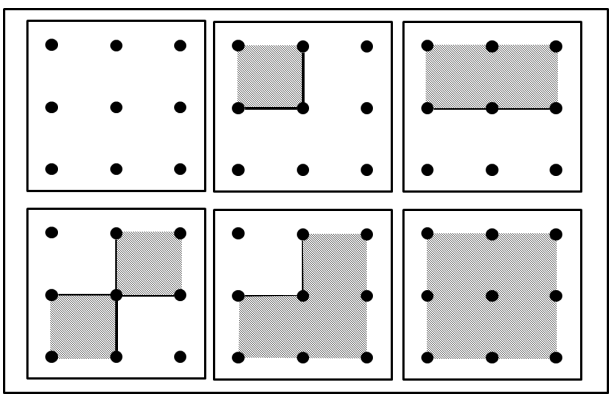

**図 6** 点を共有する 4 面の配置
**Fig. 6** Possible alignments of four squares sharing a vertex.

式 (1) と同様，条件 (c4) を満たすために 4–5 本めの制約を加えている．この定式化では図 4 のような場合，最大化でも最小化でも余計な輪ができてしまう場合がある．Slitherlink の解は，変数の数が 1 となる面の多い Slitherlinks の解に等しいことが多く，最終的に Slitherlink を解くことが目的なので式 (2) では変数の値が 1 となる面の数の最大化を行っている．

### 3.2 Slitherlink の新たな厳密解法

Slitherlinks の解に含まれる閉路の数は，シードフィルアルゴリズム [2] で数えることができる．これは，ある点とその点を含む閉領域内のすべての点を列挙するアルゴリズムで，得られた解に含まれるすべての閉領域に番号付けができる．閉領域の数が 2 つのとき，初等的な閉路が 1 つしか存在しない（閉路の内側と外側の閉領域だけである）ため，これが Slitherlink の解である．それ以外の場合，複数の閉路が存在するので，余計な閉路の除去を行う．

Slitherlinks の解に含まれる最も小さな閉領域 $L$ について，$L$ の境界からなる閉路を $C_L$ とする．閉路 $C_L$ の 1 つ外側の面の集合を $L_{out}$，1 つ内側の面の集合を $L_{in}$ とする．このとき $L_{out}$ に含まれる面に対応するすべての変数の値が 1（または 0）で，かつ $L_{in}$ に含まれる面に対応するすべての変数の値が 0（または 1）のとき，変数の値が 0 と 1 の面の間には線分が引かれるために閉路 $C_L$ ができる．逆に，この条件を満たさなければ閉路 $C_L$ は生じないので，切除平面法で追加する式は次のいずれかである：

$$\sum_{(i,j) \in L_{in}} x_{ij} + \sum_{(i,j) \in L_{out}} (1 - x_{ij}) \le |L_{in}| + |L_{out}| - 1,$$
$$\sum_{(i,j) \in L_{in}} (1 - x_{ij}) + \sum_{(i,j) \in L_{out}} x_{ij} \le |L_{in}| + |L_{out}| - 1.$$

2 つの式のうち，得られた Slitherlinks の解で違反している方を追加し，これを Slitherlinks が複数の閉路を持たなくなるまで繰り返すことで Slitherlink の解を生成する．

## 4. 比較結果とまとめ

杉村の方法と提案する定式化を比較すると，変数と制約式の数は**表 1** のようになる．条件 (c2), (c4) に関する制約式の数は同じであるが，変数に関しては約 1/2，(c3) に関する制約式も 2/5 未満に減少している．

実際に Slitherlink を解いて 2 つの方法を比較したものが**表 2** である．比較項目は以下の 3 つである：

(a) 切除平面法に対する反復数
(b) Slitherlink の計算時間（秒）
(c) Slitherlinks の計算時間（秒/反復数）

また，筆者もパズルを解き，計算時間の測定を行った．実験には Core i7 3.33 GHz の PC を使い，アルゴリズムは Octave [1] 上で実装した．また整数計画法ソルバ

表 1 変数と制約の比較
**Table 1** Comparison of the problem sizes.

| | 既存解法 | 提案解法 |
|---|---|---|
| 変数 | $(m+1)(n+2)+(m+2)(n+1)$ | $(m+2)(n+2)$ |
| (c2) の制約 | $\|P\|$ | $\|P\|$ |
| (c3) の制約 | $5(m+1)(n+1)$ | $2(m-1)(n-1)$ |
| (c4) の制約 | $2(m+1)+2(n+1)$ | $2(m+1)+2(n+1)$ |

表 2 実験結果
**Table 2** Numerical results.

| | | 既存解法 | | | 提案解法 | | | 筆者 |
|---|---|---|---|---|---|---|---|---|
| 問題 | サイズ | (a) | (b) | (c) | (a) | (b) | (c) | (b) |
| 1 | $6\times6$ | 1 | 0.047 | 0.047 | 1 | 0.031 | 0.031 | 51 |
| 2 | $10\times10$ | 2 | 0.219 | 0.109 | 1 | 0.047 | 0.047 | 140 |
| 3 | $10\times10$ | 6 | 0.516 | 0.086 | 1 | 0.047 | 0.047 | 159 |
| 4 | $10\times10$ | 3 | 0.297 | 0.099 | 1 | 0.047 | 0.047 | 125 |
| 5 | $10\times10$ | 12 | 47.844 | 3.987 | 1 | 0.063 | 0.063 | 1524 |
| 6 | $10\times10$ | 7 | 7.25 | 0.188 | 2 | 0.141 | 0.07 | 765 |
| 7 | $10\times10$ | 4 | 0.75 | 0.188 | 3 | 0.125 | 0.042 | 1070 |
| 8 | $10\times10$ | 10 | 82.063 | 8.206 | 5 | 0.203 | 0.041 | 2253 |
| 9 | $10\times10$ | 2 | 0.297 | 0.128 | 2 | 0.063 | 0.031 | 171 |
| 10 | $10\times10$ | 6 | 3.891 | 0.648 | 7 | 0.203 | 0.029 | 506 |
| 11 | $10\times18$ | 7 | 5.516 | 0.788 | 4 | 0.438 | 0.109 | 240 |
| 12 | $10\times18$ | 7 | 2.188 | 0.313 | 2 | 0.156 | 0.078 | 271 |
| 13 | $10\times18$ | 9 | 2.469 | 0.274 | 3 | 0.344 | 0.115 | 375 |
| 14 | $14\times20$ | 4 | 10.547 | 2.637 | 2 | 0.938 | 0.469 | 891 |
| 15 | $14\times20$ | – | – | – | 5 | 3.875 | 0.775 | 4802 |
| 16 | $14\times20$ | – | – | – | 14 | 30.672 | 2.191 | 3263 |
| 17 | $14\times20$ | 12 | 196.22 | 16.352 | 5 | 3.594 | 0.719 | 1233 |
| 18 | $14\times24$ | – | – | – | 20 | 34.937 | 1.747 | 570 |
| 19 | $14\times20$ | – | – | – | 12 | 69.5 | 5.792 | 718 |
| 20 | $20\times30$ | – | – | – | 34 | 2372.7 | 69.786 | 1620 |

として GLPK [5] を使用している．実験に使用した問題は [4], [11], [13] から適当な 20 題を選んだ．

表 2 で，Slitherlink，Slitherlinks の計算時間を比較すると，いずれも提案手法の方が高速に解が得られていることが分かる．盤面のサイズを大きくしていくと杉村の方法では $14\times20$ 以上のサイズで解けない問題が現れ始めるが，提案する方法では数秒で解けてしまう．このことから，大幅な高速化を実現できたと結論できる．最も大きな $20\times30$ の問題では，筆者が解いたときよりも長い計算時間がかかっているが，整数計画問題として解く場合，計算時間は人間の解きやすさ以上に問題サイズに依存するためと考えられる．

以上の実験結果から，杉村の方法に比べてすべての問題で計算時間が改善され，これまで解くことができなかったサイズの問題まで解けることも判明した．

## 参考文献


[1] Eaton, J.W.: GNU Octave (online), available from ⟨http://www.gnu.org/software/octave/⟩ (accessed 2012-11-01).

[2] Glassner, A.S. (Ed.): Graphics gems, Heckbert, P.S.: *A seed fill algorithm*, pp.275–277, Academic Press (1990).

[3] Kölker, J.: Selected Slither Link Variants are NP-complete, *Journal of Information Processing*, Vol.20, pp.709–712 (2012).

[4] KWON-TOM LOOP (online), available from ⟨http://www.kwontomloop.com/⟩ (accessed 2012-05-30).

[5] Makhorin, A.O.: GLPK (GNU Linear Programming Kit) (online), available from ⟨http://www.gnu.org/software/glpk/⟩ (accessed 2012-10-20).

[6] Yato, T. and Seta, T.: Complexity and completeness of finding another solution and its application to puzzles, *IEICE Trans. Fundamentals of Electronics, Communications and Computer*, Vol.86, pp.1052–1060 (2003).

[7] Yoshinaka, R., Saitoh, T., Kawahara, J., et al.: Finding All Solutions and Instances of Numberlink and Slitherlink by ZDDs, *Algorithms*, Vol.5, pp.176–213 (2012).

[8] 伊戸川暁：スリザーリンク解答プログラム（オンライン），入手先 ⟨http://www2.ttcn.ne.jp/itogawa/product/slitherlink.html⟩（参照 2013-03-11）．

[9] 杉村由花：整数計画法を用いたスリザーリンクの解法，東京大学 2004 年度卒業論文 (2004).

[10] 田村直之：スリザーリンク・パズルを Sugar 制約ソルバー

で解く（オンライン），入手先 ⟨http://bach.istc.kobe-u.ac.jp/sugar/puzzles/slitherlink.html⟩（参照 2013-03-08）．
[11] ニコリ：スリザーリンクのおためし問題（オンライン），入手先 ⟨http://www.nikoli.co.jp/ja/index.html⟩（参照 2012-05-20）．
[12] 温井康介，上嶋章宏：六角格子，三角格子上でのスリザーリンクの ASP 完全性について，情報処理学会 AL，アルゴリズム研究報告会，pp.129–136 (2007)．
[13] 藤原博文：JAVA とパズルの世界，パソコン初心者の館（オンライン），入手先 ⟨http://www.pro.or.jp/~fuji/java/index.html⟩（参照 2012-05-20）．
[14] 八登崇之：スリザーリンクの NP 完全性について，*IPSJ SIG Notes*, pp.25–31 (2000)．

**石濱 友裕**（正会員）

2011 年筑波大学情報科学類卒業．2013 年筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程修了．同年新日鉄住金ソリューションズ株式会社入社．現在に至る．

**久野 誉人**

1983 年東京工業大学社会工学科卒業．1988 年同大学大学院理工学研究科博士課程単位取得退学．工学博士．東京工業大学社会工学科助手，筑波大学電子・情報工学系講師等を経て現在，筑波大学システム情報系教授．研究分野は数理最適化，特に非凸計画問題の大域的最適化が専門．日本 OR 学会，Mathematical Optimization Society，INFORMS 会員．