抽象化経由発散に基づく新しいパズルの提案とパズルジェネレータの作成

A New Puzzle Based on Divergence via Abstraction and Its Generator

保里　和樹
Kazuki Hori
法政大学情報科学部ディジタルメディア学科
E-mail: kazuki.hori.8g@stu.hosei.ac.jp

Abstract

There are many types of puzzles, and many people have been enjoying puzzles since old days. Especially in recent years, puzzles for smart phones and tablets are increasing, and there are a growing number of people who enjoy such puzzles. However, most of existing puzzles are limited in shape although they adopt various rules. Sugiyama et al. proposed divergence via abstraction, and used it to generalize the shapes of two types of puzzles. Divergence via abstraction is a systematic approach by which existing puzzles can be converted to other media. In this paper, we generalize a square grid puzzle called Kakuro by using divergence via abstraction. Square grid puzzles can be regarded as graphs if each square is represented as a node and each side-by-side square relation is expressed as an edge. We visualize a Kakuro puzzle by using a graph visualization method called the Kamada-Kawai method together with three ways of visualizing edges. The first way of visualizing edges assigns a different color to each line of cells, the second one generates additional points for bending lines of cells, and the third one uses these two ways simultaneously.

1. はじめに

パズルには多くの種類があり，昔から多くの人に楽しまれている．特に近年はスマートフォンやタブレット上でパズルを楽しむ人が増えている．しかし，既存のパズルは多様なルールがあるにもかかわらず，限られた形状をしていることが多い．

パズルの形状を大きく変化させる研究として，杉山らの研究 [1] [2] [3]がある．杉山らは抽象化経由発散と呼ばれる体系的アプローチを提案し，このアプローチを用いて二種類のパズルの形状の一般化を行い，それを基に二つの新しいパズルを提案した．抽象化経由発散とは，既存のパズルは抽象化を経て他のメディアに変換できるとする体系的アプローチである．

本研究では，抽象化経由発散を杉山らの研究で扱われたパズルとは異なる種類のパズルに適用し，そのパズルを基により幅広い形状でのパズルの作成を可能とした新しいパズルを提案する．本研究では，加算パズルと呼ばれる格子状のパズルを扱い，各マスをノード，となりあうマスの関係をエッジと見ることで，パズルをグラフとして表現する．さらに，グラフ可視化技術と複数の接続方法を用いて，提案するパズルの可視化を行う．

2. 関連研究

2.1. 抽象化経由発散に基づくパズル

杉山らは抽象化経由発散の概念を提案し，本概念に基づく置換パズル [1] [3]と巡回パズル [1] [2]の二つのパズルを提案した．置換パズルとは，パズルの操作を要素間の置換として特徴づけられるパズルであり，ルービックキューブに代表される．一方，巡回パズルとは，パズルに同じ操作を繰り返すとパズルの状態が周期的に変化し一定回数後に元に戻るという特徴をもつパズルであり，ライツアウトに代表される．またこれら二つのパズルについてパズルジェネレータを作成し，それぞれの可視化を行った．置換パズルにおいてはパズルの対称性を重視した新しいグラフ可視化アルゴリズムを提案した．

2.2. Kamada-Kawai 法

Kamada-Kawai 法 [4]とは力指向アルゴリズムを基にしたグラフ描画アルゴリズムであり，シンプルな式で合理的な配置を計算することができる．この手法の特徴としてつながった要素間の距離が一定になるため，要素が密集する，つながった要素が離れすぎるといった状況をある程度防ぐことができる．

3. 加算パズル

本研究では抽象化経由発散を加算パズルに対して適用する．加算パズルとは，図 1 の B のような四角マスに数字を入れて，図 1 の A のような各三角マスから始まる縦または横の四角マスの列に入る数字の合計が，三角マスの中の数字と等しくなるようにするパズルである．三角マスの中の数字は，斜線の左下なら縦，右上なら横の列の値の合計を示す． また四角マスに入れる値は 1〜9 の整数のみで，縦横それぞれで同じ三角マスから連なる四角マスに同じ値を入れてはならない．

Supervisor: Prof. Hiroshi Hosobe

図 1 の例では，A の三角マスは斜線の左下に 12 が入っているため，このマスの下に連なる三つの四角マスに入れる値の合計を 12 にしなくてはならない．この時，2, 5, 5 や 4, 4, 4 といった同じ値を用いた組み合わせを使うことはできない．

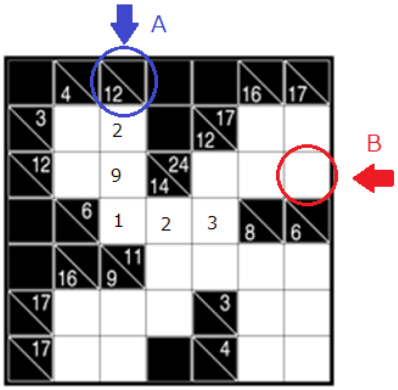

図 1　加算パズルの例.

4. 提案手法

4.1. パズルの抽象化

加算パズルは，各列に記入した数字の合計が決められた値と等しくなるようにするパズルである．よって加算パズルを定義するためには，各要素に入る値，各要素の列への所属とその順序，各列の要素の値の合計が定まっていればよい．このことから加算パズルを抽象化したもの M を次のように定式化できる．

$M = (X, C, \varphi, L)$
$X = \{x_1, x_2, \ldots, x_n\}$ ：要素の集合
$C = \{1,2,3,4,5,6,7,8,9\}$：要素に入る値の集合
$\varphi : X \to C$：各要素に入る値
$L = \{(t_1, r_1), (t_2, r_2), \ldots, (t_m, r_m)\}$：列の集合
$r_i = (x_{j1}, x_{j2}, \cdots, x_{jki})$
$(p \neq q$ のとき $\varphi(x_{jp}) \neq \varphi(x_{jq}))$
$t_i = \varphi(x_{j1}) + \varphi(x_{j2}) + \cdots + \varphi(x_{jki})$

以下に簡単な例を示す．

$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$
$C = \{1,2,3,4,5,6,7,8,9\}$
$\varphi : x_1 \to 1, x_2 \to 2, x_3 \to 3, x_4 \to 4, x_5 \to 5,$
$x_6 \to 6, x_7 \to 7, x_8 \to 8$
$L = \{(t_1, r_1), (t_2, r_2), \ldots, (t_m, r_m)\}$
$r_1 = (x_1, x_2, x_3), r_2 = (x_6, x_7, x_8),$
$r_3 = (x_1, x_4, x_7), r_4 = (x_2, x_5, x_7),$
$r_5 = (x_2, x_4, x_6), r_6 = (x_3, x_5, x_6), r_7 = (x_3, x_8)$
$t_1 = \varphi(x_1) + \varphi(x_2) + \varphi(x_3) = 1 + 2 + 3 = 6$
$t_2 = 21, t_3 = 12, t_4 = 14, t_5 = 12, t_6 = 14, t_7 = 11$

4.2. パズルの可視化

定義したパズルを可視化するとき，要素の位置をランダムに決定してしまうと，配置によっては列の把握が困難になってしまう可能性がある．そのため要素がある程度わかりやすい配置になるよう，本研究では各要素の配置に Kamada-Kawai 法を用いる．ただし要素をただ繋いだだけでは列の判別ができなくなってしまうため，以下の通り，列ごとに色分けする等の対策を行う．

1. 列ごとに色分けをする．単純に列ごとにエッジの色分けをして表示することで判別を容易にする．ただしこの方法には，列の数が多くなるにつれ，似た色の列同士が交差する箇所がでてくる可能性が増え，判別が困難になることが予想される．
2. 曲げるための折曲点を別に用意する．要素同士を単純につないだ場合に列の判別が困難になる原因として，線同士が交差した点で列を曲げてしまうと，その列が次にどの要素に向かっていくかわからなくなってしまうことが挙げられる．そこで要素と要素を結んだ直線上に曲がるための折曲点を新たに作成し，そこでのみ曲がるようにすることで列同士が交差した点では曲がることがなくなり，列の判別が容易になると期待できる．
3. 上記の二つの方法を併用する．これら二つの方法は同時に使用することが可能なため両方利用する．この方法では同じ色の列が交差しても列の判別が可能となり，折曲点を生成する方法のみの場合と比べて列の判別が容易になると期待できる．

5. 実装

本研究では実装に Processing を用いた．またパズルの内容を入力するためのもの，パズルを可視化し実際に遊ぶためのものの二つに分けて実装した．

5.1. 入力内容・入力画面

コンピュータでパズルを定義する場合，人の手で要素の数，各要素に入る値，列の数，各列に所属する要素とその順番が入力されれば残りの項目はプログラムで求められるので，これらの項目を入力する形でパズルジェネレータを作成した(図 2)．パズルの内容は図 3 のような形でテキストファイルとして保存される．

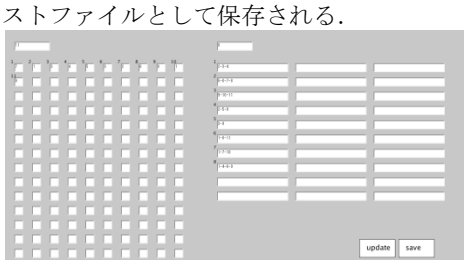

図 2　入力画面.

図 3　記録内容.

5.2. 表示方法・表示画面

まずパズルの情報をテキストファイルから読み込む．このときテキストファイルには各列に所属する要素の値の合計と実際に表示する列の情報がないためここで設定する．また Kamada-Kawai 法で利用するため各要素にどの要素がつながっているかの情報もここで設定する．次に Kamada-Kawai 法を用いて各要素の位置を決定する．このときパズルの中心が画面の中心からずれてしまい，この後画面下部に設置するボタン(図 4 の下部)に重なってしまうことがある．そこで Kamada-Kawai 法を実行した後，最も画面の右側と左側の要素の x 座標からパズルの中心の x 座標を調べ，パズルの中心の x 座標が画面の中心の x 座標に来るようにすべての要素をスライドする．また最も画面の上側の要素の y 座標を調べ，画面下側のボタンに要素が重なってしまわないよう最も上の要素が画面の上部の端に近くなるようすべての要素をスライドする．その後ボタンを設置しパズルを表示する．ボタンは入力したい数字を選択するためのもの，入力した回答が正しいかチェックするためのもの，答えがわからなかったとき回答を表示するものを実装した．

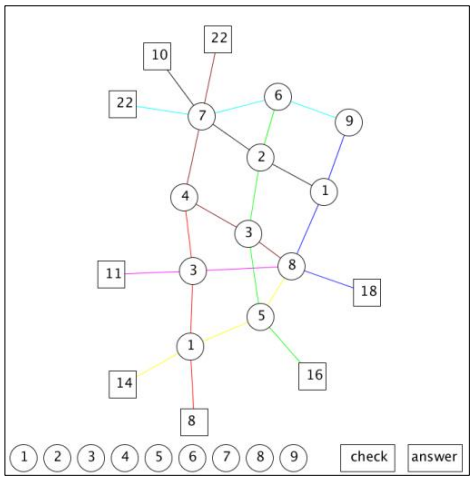

図 4　表示画面．

6. 実験

本ジェネレータでどの程度の規模のパズルを適切に作成できるか，要素の接続方法を変えて実験を行う．

6.1. 実験方法

規模を変えたいくつかのパズルに対して，3 種類の接続方法をためす．

6.1.1. 色分け

列ごとに色分けをする．色の配分のしかたは事前に設定していた 13 色を順番に配分する方法と，各列にすべての色からランダムに色を設定する方法をためす．

6.1.2. 折曲点の生成

列を曲げるための折曲点を別に用意する．各列の合計を表示している場所を始点とし，そこからその列の最初の要素を結んだ直線上の最初の要素から一定距離離れた場所に一つ目の折曲点を設定する．次に一つ目の折曲点と二つ目の要素を結んだ直線上の二つ目の要素から一定距離離れた場所に二つ目の折曲点を設定，というように最後の要素の一つ前まで折曲点を設定していく．その後合計を表示している場所，一つ目の折曲点，二つ目の折曲点，…，最後の折曲点，最後の要素，とつないでいく．この方法と，最後の要素でも折曲点を設定しそこを終点とする方法の二つをためす．

6.1.3. 併用

上記の二つの方法を同時に使用する．色分けについては事前に色を設定しておく方法のみ使用し，折曲点を用いる方法は両方の方法を使用する．

6.2. 実験結果

6.2.1. 色分け

事前に使用する色を決めておく方法は図 5 のように列数が 13 より少ないパズルは問題がなかったが，13 より多くなると同じ色の列が交差してしまい列の判別ができなくなってしまうことがあり，列の数が多くなるにつれて同じ色の列が交差する数が多くなった．色をランダムに設定する方法では，予想していた以上に列の判別が可能かどうか安定せず，また視認が困難な色が設定されてしまうこともあり，あまりよい結果は得られなかった．

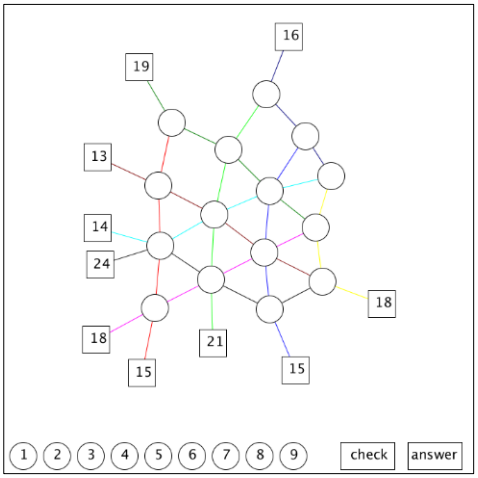

図 5　色分けの実行例．

6.2.2. 折曲点の生成

最後の要素を終点とする方法は，すべての要素がばらばらに配置されているパズルでは規模によらずしっかりと折曲点がすべて表示され列の判別が可能だった．しかし一部でも要素同士が密集した場所ができてしまうと，折曲点が近くの要素と重なって見えなくなってしまい列の判別ができなくなってしまうことがあった．またまれに二つの線が同じ方向から同じ要素につながってしまい，折曲点が同じ場所に設定されて判別できなくなる問題が発生した．またどこで列が終了しているかがわかりにくく，すぐに列を判別することは難しかった．最後の要素でも折曲点を設定する方法でも同じ問題が発生したが，図 6 のようにどこで列が終わるかはっきりしたことで列の判別はもう一つの方法と比べて容易になった．折曲点

の増加により要素と折曲点が重なってしまう可能性が少し増えてしまうが，全体的なわかりやすさはこちらのほうがよいと考えられる．

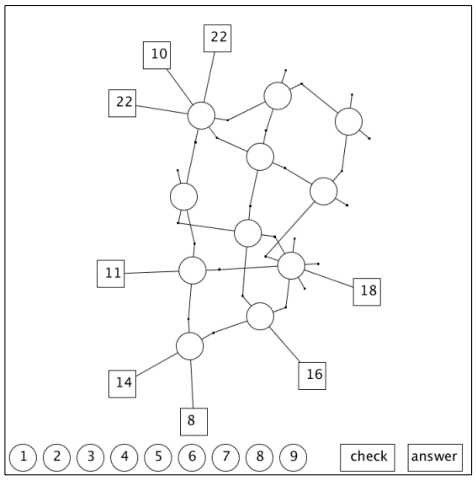

図 6　折曲点の生成の実行例．

6.2.3. 併用

二つの方法を同時に使用することで，同じ色の列が交差してしまっても判別が可能となった．しかし折曲点が別の要素と重なってしまった場合は折曲点を生成する方法と同様に列の判別が困難になってしまった．また折曲点を用いる二つの方法を比較すると，色分けしたことにより最後の要素を終点とする方法でも列の判別は容易になった(図 7)が，最後の要素でも折曲点を生成する方法がより判別しやすかった．

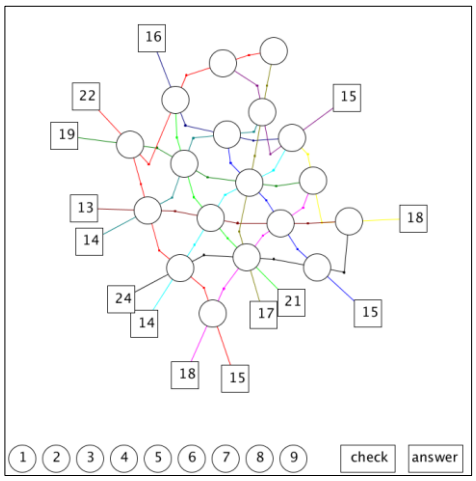

図 7　併用の実行例．

7. 議論

3 種類の方法を比べると，列の数が少ない場合は色分けが優秀だったが，列の数が多くなるにつれ折曲点を生成する方法のほうが列の判別が可能なパズルが多くなった．また併用したものは全体的に折曲点のみの場合より判別が容易だったため，列数が少ない場合は色分けのみ，列数が多い場合は併用したものを使用するとよいと考えられる． 折曲点を利用する方法は，実験結果から始点と終点がはっきりしていると列の判別が容易になると考えられる．今回提案したパズルでは各列に所属する要素の値の合計を表示する場所が始点としての役割をしていたため，始点として特別に点を用意することはしなかったが，同じようなマス目のパズルにこの方法を用いる場合は始点を用意したほうがよいと考えられる．

問題点として，要素が密集した場所ができてしまうと，特に折曲点を利用する方法で適切に列の表現ができなくなってしまう場合がある．この問題の原因として，作成するパズルに対して画面が小さすぎること，Kamada-Kawai 法の位置決定に問題があることの二つが考えられる．一つ目の問題はパズルの全体が画面内に収まらなくてもよいとして，画面をスクロールするなどして全体を見ることができるようにすることで解決できると考えられる．二つ目の問題を今回の手法を変えずに解決するためにはパズルの形状を制限しなくてはならないため，他のアルゴリズムを併用する 等の対策を考える必要がある．

8. おわりに

本研究では抽象化経由発散を用いてパズルの形状を一般化し，それを基により幅広い形状で作成可能なパズルの提案とそのパズルの可視化を行った．小規模，中規模のパズルの可視化は期待通りにできたが，大規模なパズルや一部の小・中規模のパズルはうまく可視化できなかった．本研究ではパズル全体を画面内に収めたが，パズル全体が画面内に収まらなくてもよいとするならばより大規模なパズルの可視化も可能だと考えられる．今後の課題は，今回適切に表示できなかったパズルの可視化を可能とすること，他の格子状のパズルに対しても今回の手法を適用し，適切に表現可能か調べること，また本研究ではパズルの難しさ等を考慮していないため，パズルの難易度がどのように変わっているか調べることである．

文　献

[1] K. Sugiyama, R. Osawal and S.-H. Hong, "Puzzle Generators and Symmetric Puzzle Layout," *Proc. Asia-Pacific Symposium on Infomation Visualisation (APVIS2005)*, pp. 97-105, 2005.

[2] 前田篤彦, 杉山公造, 間瀬健二, "巡回パズルのメディア変換とパズル・ジェネレータの試作," 情報処理学会研究報告：ヒューマンインターフェース(HI), no. 101, pp. 41-48, 2002.

[3] 前田篤彦, 杉山公造, 間瀬健二, "置換パズルのメディア変換とパズル・ジェネレータの試作," 情報処理学会研究報告：ヒューマンインタフェース(HI), no. 101, pp. 33-40, 2002.

[4] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7-15, 1989.