高難度一筆書きパズルの生成について

内川 伶 † 山本 修身 † †
† 名城大学理工学部 † † 名城大学情報工学部

1. はじめに

本研究では，サンフランシスコのゲーム会社 Thekla によって 2016 年に作られたゲーム The Witness (制作者： Jonathan Blow) に登場するパズルの高難度の問題を生成する方法について考える．このパズルのルールは以下のとおり: 図 1 に示すような $n \times m$ の縦横の線によって構成された盤面がある．線で囲われた四角のうちいくつかに 1 から 3 の数が与えられており，さらに 2 つの格子点が始点，終点として与えられている．始点から終点までを繋ぐ自己交差のない道を見つけるのがこのパズルの目的である．ただし，数が与えられた四角についてはその四角を通過する道の断片の個数がこの数と一致する必要がある．この数のことをヒントと呼ぶ．また，以降 $n \times m$ の縦横の線によって構成された盤面の問題のことを $(n-1) \times (m-1)$ の問題と呼ぶ．

このパズルが問題として有効であるためには解が唯一存在する必要がある．本研究ではこの条件を満たしつつ，解くことが難しい問題を生成することを目的とする．図 1 は 6×6 のヒント数 11 の問題であり，条件を満たす道は太線で示したもののみである．

2. パズルを解くためのアルゴリズム

このパズルの問題はいくつかの方法で解くことができる．まず，単純なバックトラックにより解くことができるが，問題のサイズが大きくなると多くの時間がかかるようになる．一方，ZDD [1] と呼ばれるデータ構造を利用して解くこともでき，こちらはサイズの大きい問題も比較的高速に解くことができる．

3. パズルを生成するためのアルゴリズム

問題の生成は最初に解となる道をランダムに決定し，難度の高い問題の特徴を最もよく満たすものをバックトラックによる全探索で生成するようにした．なお，全探索といっても解が唯一にならないヒントの組み合わせは枝刈りするため，実際に調べ上げるパターンはヒント数を n として 2^n よりも少ない．解が唯一かどうかは前述のバックトラックで確認している．

難度の高い問題の特徴は色々と考えられるが，本研究ではヒントの個数最小化，ヒントの位置のばらつき最大化，バックトラックによって解いた場合の探索木の最大化（以下「探索木の最大化」という）の 3 通りの最適化を考えた．これら 3 つの最適化手法で 4×4 の問題をそれぞれ 30 問ずつ生成し，人力で解くのにかかる時間を計測した．いずれの最適化手法でも簡単なものから難しいものまで幅広い難易度の問題が生成されたが，

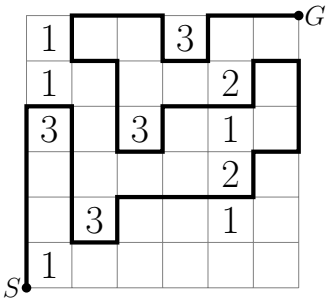

図 1: 6×6 のヒント数 11 の問題.

表 1: $n \times n$ の問題を 1 問生成するための計算時間[1] (ms).

	$n=4$	$n=5$
ビームサーチ $(k=n^2/2)$	82.6	767.5
ビームサーチ $(k=n^2)$	159.4	1550.2
ビームサーチ $(k=2n^2)$	290.6	2740.0
全探索	34.9	35530.7

解くのに 30 秒以上の時間を要した問題の数は，ヒントの個数最小化が 8 問，ヒントの位置のばらつき最大化が 7 問，探索木の最大化が 11 問で探索木の最大化が最も多かった．また，15 秒以内に解けた問題の数については，ヒントの個数最小化が 17 問，ヒントの位置のばらつき最大化が 12 問，探索木の最大化が 8 問で探索木の最大化が最も少なかった．以上より，総合的に見て難度の高い問題の生成に最も適した最適化手法は探索木の最大化であると考えられる．

全探索による問題の生成では計算量が多く，サイズが最大でも 5×5 の問題しか生成できなかった．そこでビームサーチを用いてよりサイズの大きい問題を生成できないかと考えた．ビームサーチは全探索と比べ計算量を抑えることが可能だが，最適解が見つけられるとは限らない [2]．今回は解となる道の個数の少なさをビームサーチの評価関数として，ヒントを追加していきながら探索することで，ヒントが少なめの問題を生成するようにした．解となる道の数え上げには，ZDD をベースにして構築された Python で書かれたグラフ列挙ライブラリ Graphillion [1] を用いた．ビームサーチと全探索の問題の生成にかかる時間を比較したものを表 1 に示す．なお，表の括弧内の k はビーム幅を表しており，この値が大きいほど探索によってより質の良い解が得られやすくなる [2]．表を見ると，$n=4$ では全探索の方が生成にかかる時間が短かいが，$n=5$ ではビームサーチの方が全探索と比べ生成にかかる時間が 10 分の 1 以下になっていることが確認できる．また，全探索では厳しかった 6×6 の問題も生成することができた．なお，ビームサーチによって生成された問題に含まれるヒントの個数は，全探索によって生成された問題と比較して 1%程度多いだけであった．

4. まとめと今後の課題

全探索によって 3 通りの最適化手法でサイズ 5×5 までの問題を生成することができた．また，ビームサーチを利用してヒントが少なめの問題をサイズ 6×6 まで生成することができた．

今後はビームサーチでの異なる評価関数の使用や，特徴的なヒントの排除によって，難度の高い問題がより生成されやすくなるようにしたいと考えている．また，7×7 や 8×8 などよりサイズの大きい問題の生成も今後の課題である．

参考文献

[1] 湊真一，超高速グラフ列挙アルゴリズム –〈フカシギの数え方〉が拓く, 組合せ問題の新アプローチ–．森北出版，2015.

[2] Peter Norvig, *Paradigms of AI Programming: Case Studies in Common Lisp*. Morgan Kaufmann Publishers, 1991.

[1]使用した PC： MacBook Air (M1, 2020)，チップ： Apple M1, メモリ： 8 GB，使用言語： Python 3.11.7 (ビームサーチ)，clang 15.0.0 (全探索).