あみだくじ列挙アルゴリズムの実装と実験的評価

Implementation and Evaluation of Enumeration Algorithms for All Ladder Lotteries

舟山　諒

Ryo Funayama

令和４年２月

北海道大学工学部 情報エレクトロニクス学科
情報理工学コース
情報知識ネットワーク研究室

要 旨

本研究では, Yamanaka らが提案した置換 π に対するあみだくじを高速に列挙するアルゴリズムについて考察する. ここで, あみだくじ列挙とは置換 π を受け取り, π を表す横棒数最小のあみだくじを重複なく全て出力するとこをいう.

はじめに, Yamanaka らが提案した定数遅延列挙アルゴリズムを説明する. 次に, Yamanaka らの論文を参考にあみだくじ列挙アルゴリズムの実装を行う. 実装したプログラムは列挙の始点となるあみだくじの構築に $O(n^2)$ 時間, 以後 $O(n)$ 遅延時間で全ての最適あみだくじを出力する. さらに, 実装したプログラムにおいて, 全ての解の出力にかかる実行時間と使用メモリについて評価を行なった. 最後に, 本研究をまとめ, 今後の課題を述べる.

目次

第1章

はじめに

1.1 背景

あみだくじは, 同数の人と物があるときに, それらの間で, ランダムに割り当てを決める方法の１つであり, 日本では古くから知られている. 置換πが与えられたときに, 最小本数の横線を持つあみだくじをπに対する最適あみだくじと呼ぶ. あみだくじの列挙は, 組み合わせ論や代数学への応用が知られており, 情報科学において重要な問題である [6] [8].

1.2 目的

本研究では, Yamanaka らが提案した置換πに対する最適あみだくじを高速に列挙するアルゴリズム [7] について考察する.

Yamanaka らのアルゴリズムは，次のように深さ優先探索を用いて，全ての最適あみだくじを定数遅延で列挙する．まず解である全ての最適あみだくじの上に，親子関係を定義する．この親子関係を用いて，Avis らの**逆探索法** (*reverse search*) [1] に基づいて，家系木と呼ばれる解全体の上の根付き木を構成する．この家系木上で，根からスタートして深さ優先探索を行い，解である全ての最適あみだくじを出力する.

1.3 主結果

Yamanaka らの論文を参考にあみだくじ列挙アルゴリズムの実装を行なった. 実装したプログラムは列挙の始点となるあみだくじの構築に $O(n^2)$ 時間, 以後 $O(n)$ 遅延時間で全ての最適あみだくじを列挙する. さらに, 実装したプログラムにおける全ての解を出力する実行時間と使用メモリについて評価を行なった.

1.4 関連研究

あみだくじに関連する研究として, 任意の数の横線から構成されるあみだくじを列挙するもの [5] やグレイコードを用いて $n!$ 個の異なるあみだくじを構成するもの [2] がある. さらに, 逆順列の置換を与える最適あみだくじはプリミティブソーティングネットワークと強い関連があり, その数え上げなどが研究されている [3].

1.5 本稿の構成

本論文の構成は, 次の通りである. 2 章では, 基本的な用語についての定義を行う. 3 章では, Yamanaka ら [7] が提案したあみだくじ列挙の定数遅延アルゴリズムを紹介する. 4 章では, 3 章で紹介した Yamanaka らのアルゴリズムの C++言語による実装を与える. 5 章では, 3 章の Yamanaka らのアルゴリズムの実験的評価について説明する. 6 章では, 本論文のまとめを行い, 今後の課題を述べる.

第2章

準備

本章では, 本研究で用いる基本的な定義を与える. まず, 順列に関する定義を与える. 次に, 列挙アルゴリズムの定義と計算時間量の評価法を与え, 最後にあみだくじ列挙の定義を与える.

2.1 順列

本節では順列に関する定義を与える. 本論文では有限集合 S を, $S = \{1, 2, ..., n\}$ とする.

定義 1 (順列)**.** 有限集合 S の全ての要素をちょうど 1 度ずつ用いて作った順序列を, S の**順列** ($permutation$) という.

定義 2 (置換)**.** 有限集合 S から S への全単射 $\pi : S \to S$ を**置換** ($permutation$) という. 置換 π が $1 \mapsto p_1, 2 \mapsto p_2, ..., n \mapsto p_n$ という全単射のときに π を $\pi = (p_1, p_2, ..., p_n)$ と表す. $\pi = (3, 1, 4, 2)$ とすると $\pi(1) = 3, \pi(2) = 1, \pi(3) = 4, \pi(4) = 2$ である. 整数 x に対する置換は $\pi(x)$ の代わりに $x\pi$ と表記することもある. 置換 $\pi = (1, 2, ..., n)$ を π_e と書き, **単位置換** ($identical\ permutation$) という. 置換 $\pi = (n, n-1, ..., 1)$ を**逆置換** ($reverse\ permutation$) という. 2 つの置換 π_1, π_2 の**積** ($composition$)$\pi_1\pi_2$ を $\pi_1\pi_2(i) = \pi_2(\pi_1(i))(i = 1, 2, ..., n)$ と定義する.

定義 3 (互換)**.** 置換のうち特に 2 つの要素 x, y のみを入れ替え他の要素は動かさない置換 $\tau_{x,y} : x, y, z \in S, x \neq y, z \neq x, y \Rightarrow x\tau_{x,y} = y, y\tau_{x,y} = x, z\tau_{x,y} = z$ のことを**互換** ($transposition$) という. 互換 $\tau_{x,y}$ のうち $y = x + 1$ を満たすものを**隣接互換** ($adjacent\ transposition$) という. 任意の置換は隣接互換の積で表される.

定義 4 (転倒数)**.** 置換 $\pi = (p_1, p_2, ..., p_n)$ に関する**転倒数** $(inversion)$ を

$$inv(\pi) = \#\{(p_i, p_j) \mid i < j \ and \ p_i > p_j\}$$

と定義する.

2.2 列挙アルゴリズム

ある計算問題に対して, **列挙アルゴリズム** $\mathcal{A}$ $(enumeration \ algorithm)$ とは, 与えられた条件を満たす解を重複なく全て出力するアルゴリズムである. 一般にアルゴリズムの効率は, 問題の入力の大きさに対して, どのくらいメモリや時間を使うかによって測られる. しかし, 列挙アルゴリズムの場合は解の総数が入力に対して指数的に大きくなる場合がある. そのため, 列挙アルゴリズムの計算量を入力の大きさのみで測ると, 解を出力する時間が入力の大きさに対して指数的な時間になる. そこで, 入力のサイズを N, そのときの出力の数を M とする. 列挙アルゴリズム $\mathcal{A}$ の総計算時間が N と M の多項式で抑えられるような $\mathcal{A}$ の計算量を**出力多項式時間列挙** $(output\text{-}polynomial \ time)$ という. 次に, 列挙アルゴリズム $\mathcal{A}$ が 1 つ目の解を得るまでの時間, 及び任意の $i \in \{1, ..., M-1\}$ 番目の解から $i+1$ 番目の解を得るまでの時間, 最後の M 番目の解を得てから停止するまでにかかる時間のそれぞれが N の多項式で抑えられる場合, $\mathcal{A}$ の計算量を**多項式時間遅延** $(polynomial \ delay)$ という. 最後に, $\mathcal{A}$ の計算時間が N について多項式, かつ M に関して線形である場合, $\mathcal{A}$ の計算量を**ならし多項式時間** $(amortized \ polynomial \ time)$ という

2.3 あみだくじ

あみだくじ $(ladder \ lottery)$ とは古くから伝わるくじの方式の 1 つであり, いくつかの**縦線** $(line)$ と, 隣り合う 2 本の縦線の間に挿入されたいくつかの**横線** (bar) から構成される. あみだくじは, 数字列の置換を与えるともみなせる. すなわち, それぞれの縦線の上端に異なる数字が当てられ, 縦線に沿って数字が下向きに伝わる. 伝わる途中に横線があったならば, その横線が接続している 2 つの縦線を伝わっている数字が交換され, その後改めて下向きに伝わる. このような数字の交換を繰り返し, 縦線の下端まで全ての数字が伝わったとき, 置換された数字列を得る.

$\pi = (p_1, p_2, ..., p_n)$ を表すあみだくじ L は n 本の縦線と, 隣接する縦線間を繋ぐいくつかの横線で構成される. 左から i 番目の縦線を縦線 i と呼ぶ. n 本の縦線の上端は π に対応し, 下端は π_e に対応する. また, p_i が縦線の上端から下端まで移動するときに通る**経路** $(route)$ を経路 p_i と呼ぶ. π を表すあみだくじ L がちょうど $inv(\pi)$ 個の横線を持つとき, L を**最適あみだくじ** $(optimal\ ladder\ lottery)$ という.

定義 5 (あみだくじ列挙)**. あみだくじ列挙** $(enumerate\ all\ ladder\ lotteries\ )$ とは, 入力として $\pi = (p_1, p_2, ..., p_n)$ が与えられたとき, それを満たす最適あみだくじ L を全て重複なく出力することである.

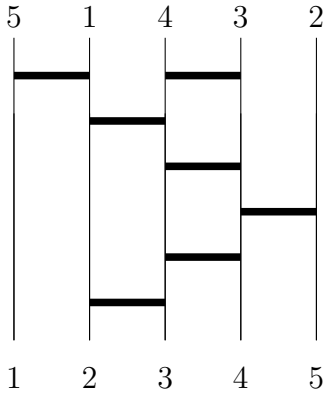

図 2.1: $\pi = (5, 1, 4, 3, 2)$ に対する最適あみだくじ

第3章

あみだくじ列挙アルゴリズム

本章では, Yamanaka ら [7] が提案したあみだくじ列挙アルゴリズムについて説明する. まず, あみだくじ列挙を行う際に用いる局所置換と, 家系木についての説明を行い, 最後に Yamanaka ら [7] の定数遅延あみだくじ列挙アルゴリズムを説明する.

3.1 局所置換

本節ではある最適あみだくじを異なる最適あみだくじに遷移させる操作である**局所置換** (*local swap*) についての説明を行う. この操作は, 代数学におけるあみだくじのブレイドリレーションを利用したものである.

図 3.1 の (a) から (b) への局所置換を b_u に対する**左置換** (*left swap*) と呼び, b_u は左置換されたという. 図 3.1 において b_u は経路 5 の上 (右) から下 (左) に移されている. b_u の右端と b_y の右端の間に他の横線の端点がなく, かつ b_u と b_y の左端の間にちょうど 1 つの b_x の右端がある場合, b_u は左置換可能であるという. 同様に図 3.1 の (b) から (a) への局所置換を b_d に対する**右置換** (*right swap*) と呼び, b_d は右置換されたという. この操作により b_d は経路 5 の下 (左) から上 (右) に移されている. b_d の左端と b_w の左端の間に他の横線の端点がなく, かつ b_d と b_w の間にちょうど 1 つの b_z の左端がある場合, b_d は右置換可能であるという.

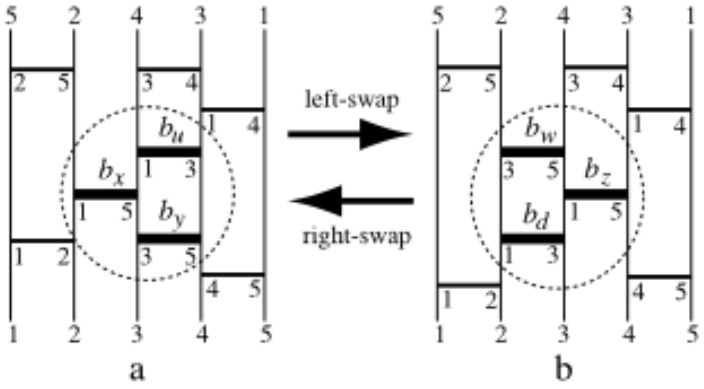

図 3.1: ローカルスワップ

3.2 家系木

本節ではπを表す最適あみだくじにおける木構造T_πを定義する. T_πの各頂点はπを表す最適あみだくじに対応しており, 各辺は１回の局所置換で遷移可能な２つのあみだくじ間の関係を表している. 頂点LとL'が連結であるとき, 局所置換を繰り返すことであみだくじLからL'を構成できることを表している.

S_πを$\pi = (p_1, p_2, ..., p_n)$を表す最適あみだくじの集合とし, $L = L_n$をS_πに含まれるあみだくじとする. iを$p_i = n$を満たすものとする. 経路nはL_nを２つの領域に分ける. 経路nによって分けられた上側の部分をL_n^U, 下側の部分をL_n^Lと表す. L_nから経路nを取り除き, L_n^UとL_n^Lを合わせたあみだくじをL_{n-1}とする. L_{n-1}は$\pi = (p_1, p_2, ..., p_{i-1}, p_{i+1}, ..., p_n)$を表すあみだくじである.

定義 6 (クリーンレベル)**.** L_n^Uに横棒が存在しないときL_nをn-cleanであるという. Lが$n \geq i \geq k$を満たす任意のiについてi-cleanであり$(k-1)$-cleanでないときLはクリーンレベルkであるという.

定義 7 (ルートあみだくじ)**.** $k = 1, 2, ..., n$についてL_kがk-cleanであるとき, LをルートあみだくじといいRで表す. Rのクリーンレベルは１である.

補題 8. S_πにおけるRは一意である.

Proof. ルートあみだくじ$R, R' \in S_\pi, R \neq R'$について考える. $R \neq R'$であるため$R_i \neq R'_i$を満たすような整数iが存在する. そのようなiのうち最小ものを考える. もし, R_iの経

路 i が縦線 k の上端から始まっているなら, R'_i では $l \neq k$ となるような縦線 l から始まっているはずである. これは π が異なることとなり矛盾. □

L をクリーンレベルが k である最適あみだくじとする. $p_j \geq k$ となる p_j に対して $(q_1, q_2, ..., q_{n_L})$ を π において p_j より大きく p_j より左側に位置する数のリストとする. L において経路 p_j はまず左に n_L 回移動し, 次に $p_j - j + n_L$ 回右に移動する. L は経路 k の下と経路 $k-1$ の上側に少なくとも 1 つの横棒を持つ. この領域のことを**活性領域** $(active\ region)$ と呼ぶ. L のクリーンレベルが $k = n+1$ であるときは, 経路 n の上側を活性領域とする. R の活性領域は空である.

次に $L \in S_\pi - \{R\}$ を満たす L について親子関係を定義する. L をクリーンレベルが k である最適あみだくじとする. L の活性領域には少なくとも 1 つの横線が存在する. 横線 b が l と $l+1$ 番目の縦線に端点をもっているとき, 経路 $k-1$ より上の縦線 l の横棒の最下端が b であり, かつ経路 $k-1$ より上の縦線 $l+1$ の最下端が b であるとき, b を**上近傍線** $(upward\ visible\ bar)$ と呼ぶ. 経路 $k-1$ に含まれる上近傍線の中で最も右にあるものを L の**活性線** $(active\ bar)$ と呼ぶ. $L \in S_\pi - \{R\}$ の活性線に対して左置換を行なってできたあみだくじを $P(L)$ と表し, L の親あみだくじと呼ぶ. また L を $P(L)$ の子あみだくじと呼ぶ. $P(L)$ のクリーンレベルは L 以下であり, $P(L)$ の活性領域における横線の数は L よりも少ない.

補題 9. 任意の $L \in S_\pi - \{R\}$ は $P(L) \in S_\pi$ を満たす.

Proof. 局所置換は順列を保持する置換操作である. □

補題 10. $L \in S_\pi - \{R\}$ に関する列 $L, P(L), P(P(L)), ...,$ は $R \in S_\pi$ を末項にもつ.

Proof. $L \in S_\pi$ に対してクリーンポテンシャル C を $C(L) = (s,t)$ と定義する. ここで, s は L のクリーンレベルを表し, t は L の活性領域における横線の数を表す. $C(L_1) = (s_1, t_1), C(L_2) = (s_2, t_2)$ となる $L_1, L_2 \in S_\pi$ に対して $(1) s_1 \leq s_2$ または $(2) s_1 = s_2$ かつ $t_1 < t_2$ を満たすとき, L_1 は L_2 よりクリーンであるという. 任意の $L \in S_\pi$ に対して $P(L)$ は L よりクリーンである. $C(R) = (1,0)$ であるため, R は S_π のうち最もクリーンである. 以上より, 任意の $L \in S_\pi$ に対して操作列 $C(L), C(P(L)), C(P(P(L))), ...$ は必ず $C(R)$ を末項にもつ. □

以上の議論より S_π における木構造 T_π を構成することができる. T_π の根は R に対応し, 各頂点は S_π の最適あみだくじ, 各辺は遷移可能な最適あみだくじの関係を表している.

補題 11. 任意の 2 つの最適あみだくじ L, L' について, 局所置換を任意の回数行うことで L を L' に遷移可能である.

3.3 あみだくじ列挙アルゴリズム

本節では, Yamanaka ら [7] によって提案された S_π における最適あみだくじを列挙するアルゴリズムについて説明する.

$L \neq R$ を $\pi = (p_1, p_2, ..., p_n)$ に対する最適あみだくじとする. L のクリーンレベルを k とする. このとき, 経路 k の右側にある各横線は $x \geq k$ の経路に含まれるが, 活性領域には $x \geq k-1$ のどの経路にも含まれない横線が少なくとも 1 つ存在する. $x \geq k-1$ の経路はまず左に移動し, その後折り返し, 最後に右に移動する. $x \geq k-1$ の各経路において左に移動する横線の後に最初に右に移動する横線が b であるとき, b を経路 x における**折り返し線** $(turn\ bar)$ と呼ぶ.

補題 12. L をクリーンレベル k をもつ最適あみだくじとする. 経路 $x \geq k-1$ 上の折り返し線のみが右置換可能である.

Proof. L のクリーンレベルは k であるため, 各経路 $x \geq k-1$ はまず左に移動し, 折り返し線に到達したのちに右に移動する. 図 3.1 において, b_d が右置換可能であるのは b_d の左端と b_u の左端の間に他の横線の右端が接続されていない状態のときである. そのような状態を満たすのは経路 $x \geq k-1$ 上の折り返し線のみである. □

L の横線 b を右置換して得られるあみだくじを $L[b]$ と表す. L の子はある b に対しては $L[b]$ であるが, すべての $L[b]$ が L の子であるとは限らない. b が $L[b]$ の活性領域に含まれている場合のみ $L[b]$ は L の子である. ここで $L[b]$ が L の子であるかどうかを以下のように分類する. ここで, L のクリーンレベルを k とし, 経路 x の右側の部分を $R(x)$, 左側の部分を $L(x)$ とする.

Type1: b が折り返し線で右置換可能である

そのような横線は経路 $k-1, k, ..., n$ 上にのみあらわれる. 図 3.2 に示すあみだくじの近

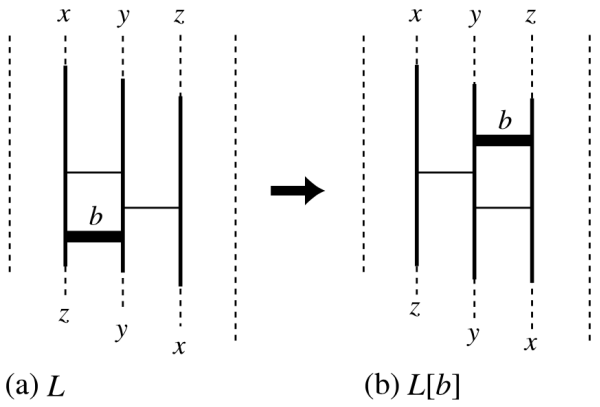

図 3.2: Type1 の図

傍を考える. 経路 x, y, z はその近傍を通り b は経路 y の折り返し線である. b は経路 y の折り返し線であるため $y \geq k-1$ を満たす. L は $x > y > z$ を満たす最適あみだくじである. b は経路 $j > x$ に含まれず, b は経路 $y(y < x)$ と経路 $z(z < x)$ に含まれるので $L[b]$ は $x - clean$ ではない. このとき, $L[b]$ のクリーンレベルは $x+1$ に増加し, $L[b]$ の活性領域には b しか存在しないので, b は $L[b]$ の活性線となる. 以上より $L[b]$ は L の子である.

Type2: b は右置換可能であるが折り返し線ではない

そのような横線は $L(k) \cap L(k+1) \cap ... \cap L(n)$ に存在する. 右置換で b が $j \geq k$ となる $R(j)$ に移動した場合, $L[b]$ のクリーンレベルは $j+1$ となり, b は新しい活性領域の唯一の横棒となる. よって b は $L[b]$ の活性線であるため, $L[b]$ は L の子である. 右置換で b が $R(k-1)$ に移動した場合, $L[b]$ のクリーンレベルは k のままであり, b は活性領域に付け加えられる. L の活性線が縦線 s に右端を持つとする. $L[b]$ の b が $t \geq s-1$ となる縦線 t に右端を持つ場合, b は $L[b]$ の活性線となる. それ以外の場合は b は活性線とならない. よって, $t \geq s-1$ を満たす場合のみ $L[b]$ は L の子である. 右置換で b が $x < k-1$ となる $R(x)$ に移動した場合, $L[b]$ のクリーンレベルは k のままであり, b は $L[b]$ の活性線ではない. よって $L[b]$ は L の子ではない.

以上の考察よりルートあみだくじを始点として現在のあみだくじからその子あみだくじを再帰的に出力していくことで, あみだくじの列挙を行うことができる. ルートあみだくじの構築は $O(n^2)$ 時間, 親あみだくじから, 子あみだくじの構築は $O(1)$ 時間で行うことができる.

第4章

あみだくじ列挙アルゴリズムの実装

本章では，3 章で述べた Yamanaka らのあみだくじ列挙アルゴリズムの実装について説明する．さらに，この章の実装したアルゴリズムでは，解の出力の仕方が元のアルゴリズムと異なるので，この版のアルゴリズムの計算時間量についても述べる．

4.1 あみだくじの表現

本節では, あみだくじを表現するデータ構造を与える．計算機上であみだくじを表現する方法によって, あみだくじの操作にかかる計算時間量及びあみだくじを記憶するために必要な領域量が異なる．

$\pi = (p_1, p_2, ..., p_n)$ を表すあみだくじを, 図 4.1 に示すように 4 方向連結リストで表す．リストに対して要素の挿入, 削除は最悪/平均時間計算量 $O(1)$ で行うことができる．リストの位置 pos に対して, pos.up, pos.down, pos.left, pos.right は, 位置 pos の上, 下, 左, 右の位置を返す．pos.left 及び pos.right がある要素の位置を返す場合, 現在のあみだくじの位置から隣の縦線に横線が接続されていることを表す．また, pos.line は現在の要素が左から何番目の縦線を表しているか, pos.num はその位置を通る p_i の値を返す．また, あみだくじの上端と下端を表す 4 方向連結リストの要素を特に upper, lower とする．縦線 i の上端は upper$[i]$, 縦線 i の下端は lower$[i]$ で表す．これらの要素へのアクセスは $O(1)$ で可能である．1 つのあみだくじを表す 4 方向連結リストの出力は $O(n)$ 時間である．

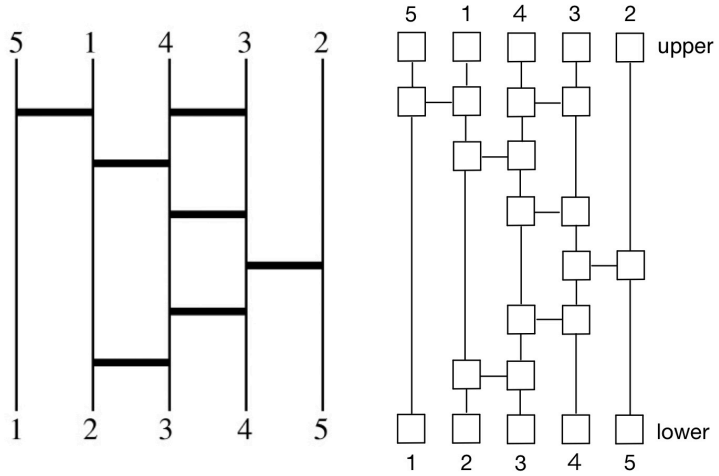

図 4.1: あみだくじの表現

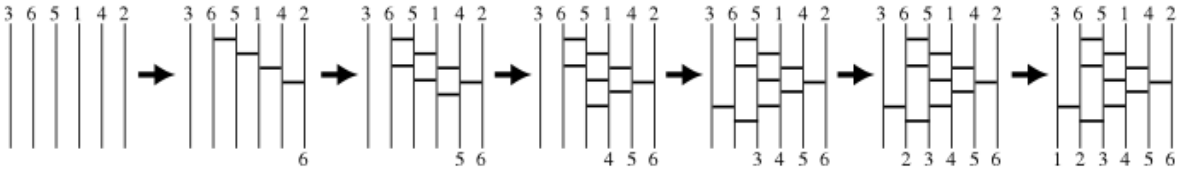

図 4.2: $\pi = (3, 6, 5, 4, 1, 2)$ に対するルートあみだくじの構築

4.2 ルートあみだくじ R の構築

本節ではルートあみだくじ R の構築法とその時間計算量について述べる. Algorithm1 に疑似コードを示す. ルートあみだくじ R の構築は図 4.1 のような手順で行う.

まず, 与えられた置換 $\pi = (p_1, p_2, ..., p_n)$ に対して, 値が最大である p_i が縦線 n の下端に移動するように縦線 i から縦線 n まで上端から順に横線を追加していく. 次に, π のうち p_i の次に値が大きい p_j を選ぶ. p_j の経路上に既に配置された横線があるなら, あみだくじの規則に従い, その経路の最下端まで移動する. その後, p_j が現在の位置から縦線 $n-1$ の下端に移動するように横線を追加していく.

以上の操作を繰り返し行うことで, ルートあみだくじ R を構築できる. $1, 2....n$ のそれぞれの経路に対して, 既に配置された横線に沿った移動及び横線の追加は $O(n)$ であるため, ルートあみだくじの構築は $O(n^2)$ で行うことができる.

4.3 あみだくじの列挙

本節ではあみだくじ列挙アルゴリズムの実装とその時間計算量について述べる．本列挙アルゴリズムは3章で説明した家系木T_πの根からT_πを探索することで, 全ての最適あみだくじを列挙する．また列挙にかかる時間計算量についても述べる．Algorithm2に疑似コードを示す.

find-all-ladder-lotteriesは家系木T_πの根であるルートあみだくじRから探索を開始するアルゴリズムである．このアルゴリズムでは入力として$\pi = (p_1, p_2, ..., p_n)$を受けとる．受け取った$\pi$に対してルートあみだくじRを構築し, その後アルゴリズムfind-all-childrenを実行する．ルートあみだくじの構築は4.2節で説明したAlgorithm1を用いる．

find-all-childrenは親あみだくじから全ての子あみだくじを出力するアルゴリズムである．このアルゴリズムは入力として親あみだくじLとLのクリーンレベルk, 活性線bを受け取る．posをあみだくじ上での現在の位置を表すポインタとする．まず, クリーンレベル$k-1$以上の経路iについてその経路の折り返し線までposを移動させる．その後, posが位置している折り返し線が右置換可能であれば3.3節で説明したType1, Type2に沿ってLの子あみだくじとなる$L[b]$を構成し, 再帰的にfind-all-childrenを実行する．折り返し線までのposの移動は$O(1)$時間, 局所置換は$O(1)$時間で実行可能であるためこのアルゴリズムでは$O(1)$遅延時間で全ての子あみだくじを探索可能である．

以上のアルゴリズムでは, あるあみだくじを出力してから次のあみだくじを出力するまでにT_πの深さ分の遅延時間が生じてしまうことがある．そこで, あみだくじの出力には**前後順序探索** (*prepostoder*) [4]を用いる．この方法では, 家系木T_πにおいて深さが偶数の頂点では行きがけで, 深さが奇数の頂点では帰りがけであみだくじを出力する．この方法を用いることにより, ならし解析であみだくじを$O(1)$遅延時間で列挙することができる．ただし, この版のアルゴリズムでは1つのあみだくじの出力に$O(n)$時間かかるため, 遅延時間は$O(n)$となる．

以上の考察により, ルートあみだくじの構築に$O(n^2)$時間, その後, 全てのあみだくじを$O(n)$遅延時間で出力することができる．

Algorithm 1 ルートあみだくじを構成するアルゴリズム

```
1: function MAKE-ROOT(π = (p_1, p_2, ..., p_n))
2:     for i = 1 to n do
3:         upper[i].down ← lower[i]
4:         upper[i].line ← i
5:         lower[i].up ← upper[i]
6:         lower[i].line ← i
7:     end for
8:     for i = n to 1 do
9:         startLine ← 1
10:        while π[startLine]≠ i do
11:            startLine++
12:        end while
13:        pos ← upper[startLine]
14:        restartLine ← startLine
15:        while pos.down ≠ NULL do
16:            if pos.left ≠ NULL then
17:                pos ← pos.left
18:                restartLine ← restartLine −1
19:            else
20:                pos ← pos.down
21:            end if
22:        end while
23:        for j = restartLine to i − 1 do
24:            insertBar(pos, j) // 横棒の追加
25:        end for
26:    end for
27: end function
```

Algorithm 2 全ての最適あみだくじを列挙するアルゴリズム

```
1: function FIND-ALL-CHILDREN(k, b) // k はクリーンレベル, b は活性線
2:     if 根からの深さが偶数 then
3:         あみだくじを出力
4:     end if
5:     for i = n to k − 1 do
6:         pos ← upper[π[i]]
7:         折り返し線の位置まで pos を移動
8:         if 折り返し線 t が右置換可能 then
9:             if i = k − 1 then
10:                if pos.line ≥ b.line−1 then
11:                    pos が示す折り返し線 t を右置換
12:                    FIND-ALL-CHILDREN(i + 1, t)
13:                end if
14:            else
15:                pos が示す折り返し線 t を右置換
16:                FIND-ALL-CHILDREN(i + 1, t)
17:            end if
18:        end if
19:    end for
20:    if 根からの深さが奇数 then
21:        あみだくじを出力
22:    end if
23: end function
24:
25: function FIND-ALL-LADDER-LOTTERIES(π = (p_1, p_2, ..., p_n))
26:    MAKE-ROOT(π)
27:    FIND-ALL-CHILDREN(1, NULL)
28: end function
```

第5章

実験

本章では, 実装したあみだくじ列挙アルゴリズムに対し計算機実験を行う. まず, 実験の目的を説明する. 次に実験の方法と実験環境を説明し, 主結果を与える.

5.1 目的

実験では実装したプログラムの正しさを確認するため, n が 10 以下の全ての逆置換について最適あみだくじの個数が Yamanaka らの論文 [7] に記載されているものと一致することを確認する. また, 実装したプログラムにおける全ての最適あみだくじを列挙する平均計算時間, 平均使用メモリを調べる.

5.2 方法と環境

今回の実験は以下の環境で行なった. OS は macOS Monterey 12.0.1, メモリは 16GB, CPU は 2 GHz intel Core i5, OS 種別は 64bit, 使用言語は C++, コンパイラは g++ 11.1.0 である. 実行時間, 使用メモリは各入力に対して 10 回計測を行い, その平均値を求めた.

5.3 結果

5.3.1 プログラムの確認

任意の $n \leq 10$ に対する逆置換 $\pi = (n, n-1, ..., 1)$ において, 出力される最適あみだくじの個数を確認した. 実装したプログラムにおける, 逆置換 π に対する最適あみだくじの

個数を表 5.1 と表 5.2 に示す. これは, Yamanaka らの論文内に記載されていたものと一致する. また、Yamanaka らの論文に記載されていた $\pi = (5, 6, 3, 4, 2, 1)$ に対して構成される家系木 T_π 及びその各頂点を表す最適あみだくじについて, 実装したプログラムに同様の π を与えたときの出力結果から構成される家系木が一致していることも確認した. 以上のことから, 実装したプログラムは正しく動作していると考えられる.

5.3.2 実行時間と使用メモリの計測

実装したプログラムにおいて $n \leq 11$ の逆置換 π に対する実行時間と使用メモリの計測を行なった. 計測はあみだくじの出力部を含めたものと, 出力部を除いたもので行なった. 出力はファイル出力とした. 実行時間が 3600sec を超えるものについては途中で実行を中止した. 結果を表 5.1 と表 5.2 に示す.

この結果より, あみだくじの出力部を除くと, 一定のメモリ使用量でプログラムを実行できることが確認できた. さらに, 出力部を除いたものでは $n = 10$, ファイル出力を行うものでは $n = 9$ までの逆置換を表す最適あみだくじを制限時間内に列挙できることが確認できた.

表 5.1: 逆置換 π に対する最適あみだくじの個数と実行時間 (あみだくじの出力なし)

π の長さ	最適あみだくじの個数	実行時間 [/sec]	使用メモリ [MB]
1	1	0.000145	5.7
2	1	0.000178	5.7
3	2	0.000224	5.7
4	8	0.00026	5.7
5	62	0.000264	5.7
6	908	0.000429	5.8
7	24,696	0.003769	5.8
8	1,232,944	0.102417	5.8
9	112,018,190	8.569281	5.8
10	18,410,581,880	1513.261804	5.8
11	-	-	

表 5.2: 逆置換 π に対する最適あみだくじの個数と実行時間 (あみだくじの出力あり)

π の長さ	最適あみだくじの個数	実行時間 [/sec]	使用メモリ [MB]
1	1	0.000239	5.7
2	1	0.000268	5.7
3	2	0.000276	5.7
4	8	0.000357	5.8
5	62	0.000815	5.8
6	908	0.015789	5.8
7	24,696	0.238094	7.2
8	1,232,944	15.682315	101.3
9	112,018,190	1848.224719	8480.0
10	-	-	
11	-	-	

第6章

おわりに

本研究では, あみだくじ列挙アルゴリズムについて考察し, 実装を行なった. さらに, 実装したプログラムにおける平均実行時間, 平均使用メモリを計測した.

主結果として, 実装したプログラムにおいては, 列挙の始点となるあみだくじの構築に $O(n^2)$ 時間, 以後 $O(n)$ 遅延時間で全ての最適あみだくじを出力できることを確認した. また, あみだくじの出力部を除いたものでは, 任意の $n \leq 10$ に対する逆置換を表す最適あみだくじを, 一定のメモリ使用量で列挙できることを確認した.

今後の課題として, あみだくじをよりコンパクトに表すデータ構造を考察することや, さまざまな手法によるあみだくじの列挙を研究することなどが考えられる.

謝辞

本研究を行うにあたり, 北海道大学工学部情報エレクトロニクス学科情報理工学コース情報知識ネットワーク研究室の有村博紀教授に, 研究内容や論文執筆について御指導, 御助言をいただきました. 研究の開始から最後まで親身になりアドバイスをいただき, 時には長時間にわたる打ち合わせにつきあっていただきました. 心から感謝しています.

また, 研究室の皆様には, 研究室の活動や資料作成方法などのさまざまな場面で, 大変お世話になりました. 本研究を進めるにあたり, 皆様のご協力をいただけましたことに感謝申し上げます.

最後に, 心身ともに健康な状態で, 大学生活を送ることができるように支えてくれた両親に深く感謝します.

参考文献

[1] David Avis and Komei Fukuda. Reverse search for enumeration. *Discret. Appl. Math.*, 65(1-3):21–46, 1996.

[2] Patrick Di Salvo. *Amidakuji: Gray Code Algorithms for Listing Ladder Lotteries*. PhD thesis, University of Guelph, 2021.

[3] Jun Kawahara, Toshiki Saitoh, Ryo Yoshinaka, Shin-ichi Minato, and Ryo Yoshinaka Shin-ichi Minato. Counting primitive sorting networks by πdds by. 2011.

[4] Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. *National Institute of Informatics, Technical Report E*, 4:2003, 2003.

[5] Katsuhisa Yamanaka and Shin-ichi Nakano. Efficient enumeration of all ladder lotteries with k bars. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 97(6):1163–1170, 2014.

[6] Katsuhisa Yamanaka and Shin-ichi Nakano. Enumeration, counting, and random generation of ladder lotteries. *IEICE Transactions on Information and Systems*, 100(3):444–451, 2017.

[7] Katsuhisa Yamanaka, Shin-ichi Nakano, Yasuko Matsui, Ryuhei Uehara, and Kento Nakada. Efficient enumeration of all ladder lotteries and its application. *Theoretical Computer Science*, 411(16-18):1714–1722, 2010.

[8] 田中勇真. あみだくじ数え上げ問題に対する高速解法, オペレーションズ・リサーチ: 経営の科学. 59(10):595–600, 2014.