修士論文

ブロックカクタスグラフの効率的なグラフ同型性判定アルゴリズム

棟朝一世

主指導教員 上原 隆平

北陸先端科学技術大学院大学
先端科学技術専攻
（情報科学）

令和０７年１２月

Abstract

The graph isomorphism problem for block cactus graphs with n vertices and m edges can be solved in $O(n+m)$ time. Two graphs $G=(V,E)$ and $G'=(V',E')$ are isomorphic if and only if there exists a bijection $f: V \to V'$ such that any two vertices $u, v \in V$ have an edge in G if and only if the vertices $f(u), f(v) \in V'$ have an edge in G'. The graph isomorphism problem is to determine if G and G' are isomorphic. The computational complexity of the general graph isomorphism problem is not known. However, it is known only to belong to the complexity class NP, because if a bijection between the vertices of two graphs is given, it is verifiable in polynomial time whether that bijection actually defines a graph isomorphism. On the other hand, it is known that the graph isomorphism problem can be solved with efficient time complexity for certain graph classes. In particular, for graph classes that can be represented by a type of tree structure, efficient solutions are known by reducing the original problem to the graph isomorphism problem for trees.

Block cactus graphs can be represented by a tree structure using a PQ-Tree. For this tree representation, the graph isomorphism problem for block cactus graphs is reduced to the graph isomorphism problem for PQ-Trees. A block cactus graph is a graph in which all blocks are either cliques or cycles. A block cactus graph can be converted into a type of tree structure by representing its blocks as star graphs and connecting them at the cut vertices. We call this tree structure a Block-star tree. There is an important property that the vertices in a clique can be arbitrarily permuted, but the vertices in a cycle can only be permuted in their cyclic order or its reverse order. To represent this property of the block cactus graph in the Block-star tree, the vertices of the star graph representing a cycle can only be permuted in the original cyclic order or its reverse. To achieve this, we use a PQ-Tree which can assign constraints on the permutation of children to the vertices of a rooted tree. P-node in a PQ-Tree is a vertex that allows arbitrary permutations of its children. Q-node in a PQ-Tree is a vertex that only allows permutations of its children in forward or reverse order. By assigning P-node to the vertex of the star graph representing a clique and Q-node to the vertex of the star graph representing a cycle, the properties of the block cactus graph can be represented by the PQ-Tree.

The graph isomorphism problem for a PQ-Tree representing a block cactus graph is reduced to a string comparison problem. We call this string that represents the graph isomorphism the GI-Canonical representation. For a block cactus graph with n vertices and m edges, the GI-Canonical representation string can be constructed in $O(n+m)$ time. It uses algorithms for finding the lexicographically smallest cyclic permutation of a set of cyclic strings and the Longest Common Extension.

First, we find the center vertex of the Block-star tree and represent it as a rooted tree with that vertex as the root. Next, we perform a depth-first search on the rooted Block-star tree, simultaneously constructing the PQ-Tree and creating the GI-Canonical representation string. The process is simple if the root vertex becomes a P-node. The GI-Canonical representation of the PQ-Tree can be created in $O(n)$ time from an n vertices block cactus graph. However, if the root vertex becomes a Q-node, there is the most significant problem. We have to convert vertices of the cycle into a canonical sequence of vertices. To solve this problem, we define the canonical sequence of vertices that makes the lexicographically minimal list of GI-Canonical representation strings that can be created from the vertices of the cycle. We call this process root normalization. Finally, we introduce a data structure for the Longest Common Extension problem which enables lexicographical comparison of two GI-Canonical representation strings of length $O(n)$ in $O(1)$ time to improve the time complexity of the root normalization process. After $O(n)$ preprocessing, the GI-Canonical representation of a PQ-Tree with a Q-node root can be created in $O(n)$ time by using this data structure. Once the GI-Canonical representation is constructed, the graph isomorphism problem for block cactus graphs can be solved simply by comparing the strings.

目 次

図 目 次

表 目 次

第1章　背景

2つのグラフ $G=(V,E), G'=(V',E')$ が与えられた時、G, G' に対して、$u,v \in V, \{u,v\} \in E \Leftrightarrow \phi(u), \phi(v) \in V', \{\phi(u), \phi(v)\} \in E'$ となる全単射 ϕ が存在する時、G と G' はグラフ同型であるという。一般のグラフ同型性判定問題は計算量的観点の難しさが分かっていない。自明な事実として一般のグラフ同型性判定問題は、計算複雑性クラス NP に属することが知られている。なぜなら、2つのグラフの頂点の全単射が与えられれば、それが実際にグラフ同型性を与える全単射かどうかは多項式時間で検証可能だからである。しかし、一般のグラフ同型性判定問題はクラス P に属するかどうかは分かっておらず、NP 完全であるかどうかも分かっていない [8]。つまり一般のグラフの同型性判定問題は計算複雑性の位置付け自体が分かっていない NP 問題である。この位置付けを研究にするために、一般のグラフの同型性判定問題を多項式時間で還元できる問題をグラフ同型完全問題という [8]。これはある問題がグラフ同型完全であることを示し、その問題に関する計算複雑性に関する進捗があれば全てのグラフ同型完全な問題に関する進捗につながるためである。特に二部グラフのグラフ同型性判定問題はグラフ同型完全であることが示されている [8]。また弦グラフのグラフ同型性判定問題もグラフ同型完全であることが示されている [3]。

一方で、特定のグラフクラスに関しては効率的な計算量でグラフ同型性判定問題が解決できることが分かっている。区間グラフは Booth,Lueker による PQ-Tree というデータ構造を利用することによりグラフ同型性判定問題を線形時間で解決可能であることが示されている [3]。置換グラフは Colburn により置換グラフを modular decomposition tree で表現することでグラフ同型性判定問題を線形時間で解決できることが示されている [9]。これらのグラフクラスに共通する点は、グラフ同型性を失わずにある種の木構造で表現できる点である。これは木のグラフ同型性判定問題は線形時間で解決できることが示されているため [2]、グラフ同型性を失わずに木構造で表現することは極めて強力な還元となるためである。

ブロックカクタスグラフは全てのブロックがサイクルかクリークであるグラフであり、全てのブロック同士があるカット頂点により結合されているグラフである。そのため全てのブロックとカット頂点をそれぞれ 1 つの頂点に対応させたグラフを作ることで木構造で表現することが可能である。よって効率的な計算時間でのグラフ同型性判定問題の解法が期待できるグラフクラスであるが、そういった結果はまだ知られていない。

第2章　準備

2.1　定義と用語

グラフとは、頂点集合 V と辺集合 $E \subseteq \{\{u,v\}|u,v \in V, u \neq v\}$ の組 $G=(V,E)$ である。$V(G)$ はグラフ G における頂点集合、$E(G)$ はグラフ G における辺集合を表す。本論文では、グラフは自己ループや多重辺を含まない単純無向グラフであるとする。**部分グラフ**とは、グラフ $G=(V,E)$ の辺と頂点を含むグラフ $H=(V',E'), V' \subseteq V, E' \subseteq E$ のことである。グラフ G がある性質 P を満たし、G に辺または頂点を追加すると性質 P を失うならば、G を性質 P に関する**極大なグラフ**という。グラフ $G=(V,E)$ において長さ k の**パス**とは、互いに異なる頂点の列 $(v_0,v_1,\ldots,v_k)$ で、任意の $i=0,1,\ldots,k-1$ に対して、辺 $\{v_i,v_{i+1}\} \in E$ が存在するものである。長さ k のパス $(v_0,v_1,\ldots,v_k)$ において v_0 をパスの **始点**、v_k をパスの**終点**という。**サイクル**とは、長さ 3 以上のパスであり、パスの始点と終点が一致するパスのことである。サイクルからなるグラフのことを**サイクルグラフ**という。**クリークグラフ**とは、そのグラフに属する任意の相異なる 2 頂点が辺を持つグラフのことである。**距離**とは、グラフ $G=(V,E)$ においてある二頂点 $v_i,v_j \in V$ の v_i から v_j へのパスのうち最も長さの小さいパスの長さをいう。グラフ $G=(V,E)$ における距離は記法 $\mathrm{dist}(v_i,v_j)$ で表現する。**連結なグラフ**とは、そのグラフに属する相異なる任意の二頂点間の間にパスが存在するグラフのことである。**連結成分**とは、そのグラフにおける連結なグラフのうち、極大なグラフのことである。**カット頂点**とは、連結なグラフにおいて削除すると、連結なグラフでなくなる頂点のことである。

定義 2.1 (グラフ同型)**.** *2* つのグラフ $G=(V,E), G'=(V',E')$ が与えられた時、G,G' に対して、$u,v \in V, \{u,v\} \in E \Leftrightarrow \phi(u),\phi(v) \in V', \{\phi(u),\phi(v)\} \in E'$ となる全単射 ϕ が存在する時、G と G' は**グラフ同型**であるという。G と G' がグラフ同型である場合、$G \cong G'$ という記法を用いる。

定義 2.2 (グラフ同型性の保持)**.** *2* つのグラフ $G=(V,E)$, $G'=(V',E')$ に対し、写像 f による変換を $f(G), f(G')$ とする。G と G' がグラフ同型であるとき、$f(G)$ と $f(G')$ もグラフ同型であるならば、写像 f は**グラフ同型性を保持**するいう。以下の命題を満たす写像 f はグラフ同型性を保持する。

$$G \cong G' \quad \Longrightarrow \quad f(G) \cong f(G')$$

木は、サイクルを含まない連結なグラフである。木においてある 1 つの頂点を**根頂点**という特別な頂点とする木を**根付き木**と呼ぶ。根付き木 $G = (V, E)$ において、根頂点 $v_0 \in V$ からの長さ k のパス $(v_0, v_1, \ldots, v_k)$ において、$v_j (j < i)$ を v_i の**先祖頂点**と呼ぶ。u が v の先祖頂点であるとき、v を u の**子孫頂点**と呼ぶ。v の**先祖頂点**の中で v と辺を共有する頂点を v の**親頂点**と呼ぶ。u が v の親頂点であるとき、v を u の**子頂点**と呼ぶ。根付き木において子頂点を持たない頂点を**葉頂点**と呼ぶ。根付き木において葉頂点以外の頂点を**内部頂点**と呼ぶ。根付き木 $G = (V, E)$ において葉頂点以外の任意の頂点 $v \in V$ において、v の子頂点の集合 $\{v_0, v_1, \ldots, v_k\}$ に列 $(v_0, v_1, \ldots, v_k)$ が定められている根付き木を**順序木**と呼ぶ。木において最も長いパスの長さを**木の直径**と呼ぶ。木 T の直径を R とすると、$r = \left\lfloor \frac{R}{2} \right\rfloor$ を**木の半径**と呼ぶ。根付き木において根頂点から葉頂点までの距離の最大値を**木の高さ**と呼ぶ。根付き木において根頂点からある頂点までの距離をその**頂点のレベル**と呼ぶ。根付き木 T において任意の頂点 v を根頂点として v の子孫頂点を全て含む部分グラフ $H \subseteq T$ を v を根頂点とする**部分木**と呼ぶ。グラフにおいてある頂点を始点として、始点からの距離が大きくなるように頂点を探索していき、探索ができない場合、探索した直前の頂点に戻り探索を続けて、全ての頂点を探索していくアルゴリズムを**深さ優先探索**と呼ぶ。グラフ G における連結な部分グラフ $H \subseteq G$ のうち、カット頂点を持たない極大な部分グラフ H を**ブロック**と呼ぶ。連結グラフ $G = (V, E)$ において、相異なる 2 つの部分グラフ $H_1, H_2 \subset G$ があるカット頂点 $v \in V$ を削除すると異なる連結成分に属する場合、H_1, H_2 は v で**結合されている**という。すべてのブロックがクリークグラフまたはサイクルグラフであるグラフを**ブロックカクタスグラフ**と呼ぶ。グラフ $G = (V, E)$ においてある一つの頂点 $v \in V$ と v 以外の頂点と共有する辺 $E = \{\{v, v'\} \mid v' \in V, v' \neq v\}$ からなる木を**スターグラフ**と呼ぶ。スターグラフ構造を持つ木の中心頂点を**スターグラフの中心頂点**と呼ぶ。

定義 2.3 (辞書順)**.** *2* つの列 $A = (a_1, a_2, \ldots, a_k)$ と $B = (b_1, b_2, \ldots, b_l)$ が与えられたとする。A が B に対して辞書順で小さい（$A < B$）とは、以下のいずれかの条件が満たされる場合をいう。

1. あるインデックス i $(1 \leq i \leq \min(k, l))$ が存在し、全ての $j < i$ に対して $a_j = b_j$ であり、かつ $a_i < b_i$ である。
2. A は B の接頭辞である。すなわち、$k < l$ かつ、全ての $j \leq k$ に対して $a_j = b_j$ である。

定義 2.4 (循環列)**.** 長さ n のある列 $S = (s_1, \ldots, s_n)$ が与えられたとする。このとき、以下の条件を満たす列 S_i の集合を S の循環列という。

$$\{S_i = (s_i, s_{i+1}, \ldots, s_n, s_1, \ldots, s_{i-1}) \mid 1 \leq i \leq n\}$$

第3章　ブロックカクタスグラフの PQ-Tree表現

本章では、ブロックカクタスグラフのグラフ同型性判定問題を解くために *PQ-Tree*[3] の概念を導入する。グラフ同型性判定問題の基本戦略はグラフをグラフ同型性を失わない木構造に変換し変換先の木構造のグラフ同型性判定問題を解くことである。木構造のグラフ同型性判定問題は、効率的に解く手法が知られているため、この帰着は有効な戦略となる。本稿のアプローチはまずブロックカクタスグラフをブロックスターツリーと呼ばれるより扱いやすい木構造に変換する。しかし、このブロックスターツリーへの変換は頂点の並び順に関する重要な情報を表現しきれず、グラフ同型性が保持されないという課題を持つ。この課題はブロックカクタスグラフにおけるクリークブロックの頂点の並び順は自由であるのに対し、サイクルブロックの頂点の並び順は自由でなく、逆順にすることのみ許されるという順序制約の違いに起因する。この制約の違いを表現するため、本章ではPQ-Tree というデータ構造を導入する。PQ-Tree が持つ頂点の並べ替えのルールをブロックスターツリーに適用することで、ブロックカクタスグラフをグラフ同型性を保持したまま木構造として表現できる。

3.1　ブロックカクタスグラフの木構造への変換

この章ではブロックカクタスグラフをある種の木構造に変換できることを示す。ブロックカクタスグラフを木構造に変換する理由は、4 章で後述する木のグラフ同型性判定アルゴリズムを本論文のブロックカクタスグラフのグラフ同型性判定問題で利用するためである。

3.1.1　二重頂点連結成分分解

無向グラフ $G = (V, E)$ が与えられた時、任意の相異なる 3 つの頂点 $v, w, a \in V$ に対して、a を通らない v, w 間のパスが存在する時、G を二重頂点連結成分という。また v, w 間の任意のパスが必ず a を通る時、a はカット頂点となる。グラフをカット頂点により二重連結成分の集合に分解することを **二重頂点連結成分分解**という。二重頂点連結成分分解では、辺の集合に対して 2 つの辺が同じサイクルに

含まれる場合、2 つの辺を同値とし、この同値関係に基づいたグラフの同値類を $C_1, C_2, \ldots, C_k$ として分解する。各 C_i は極大な二重連結成分であるという性質を持つ。ブロックカクタスグラフを二重頂点連結成分に分解すると、一つ一つの連結成分はサイクルかクリークに分解される。よって、ブロックカクタスグラフは二重頂点連結成分分解によりサイクルグラフとクリークグラフのブロックの集合に分解できる。無向グラフ $G = (V, E)$ が与えられた時、二重頂点連結成分分解は Tarjan[4] が提案した深さ優先探索を用いた方法により $O(|V| + |E|)$ 時間で可能である。

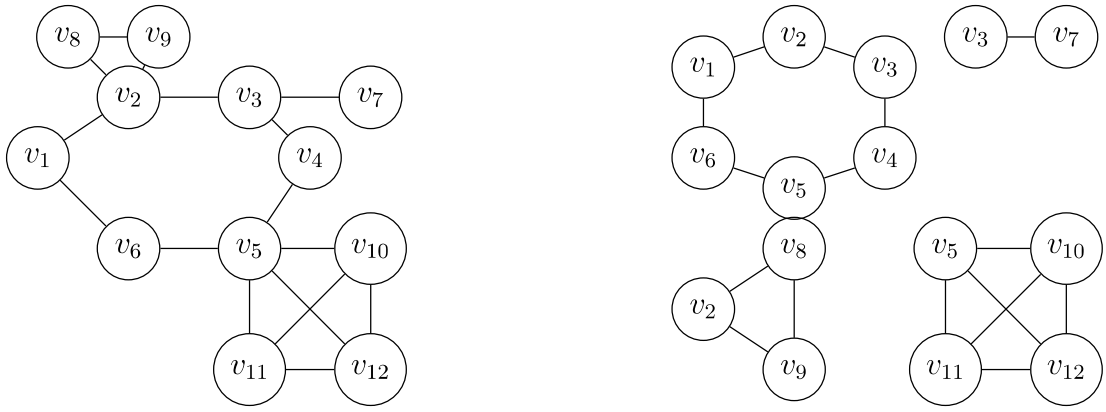


図 3.1: ブロックカクタスグラフ (左) とその二重頂点連結成分への分解 (右)

3.1.2 ブロックスターツリー

連結グラフ G が与えられた場合、グラフ H を G の**ブロックカットツリー** [6] として定義する。G のブロックの集合を $\{B_i\}$、カット頂点の集合を $\{c_j\}$ とする。B_i は、G においてはブロックである頂点集合を表すが、ブロックカットツリーにおいては一つの頂点である点に注意する。e_{i_j} は G において $c_j \in B_i$ である場合、ブロックカットツリー H における辺 $e_{i_j} = \{B_i, c_j\}$ とする。$H = (\{B_i\} \cup \{c_j\}, \{e_{i_j}\})$ である。ブロックカットツリーにおける B_i をブロック頂点と呼び、B_i 以外の頂点を間接頂点と呼ぶ。ブロックカットツリーは連結グラフのブロックに対応するブロック頂点とカット頂点に対応する間接頂点からなる二部グラフである。また全ての連結グラフは、あるブロックカットツリーに対応するため、ブロックカットツリーは連結グラフのブロックとカット頂点の集合を木構造に対応させたグラフである。

定理 3.1. 連結グラフ G が与えられた場合、G のブロックカットツリーは木である *[6]*。

ブロックカクタスグラフは二重頂点連結成分分解を用いてブロックカットツリーに変換できるが、この変換は元のブロックカクタスグラフのブロックが 1 つのブ

ロック頂点に変換されているため、頂点の全単射が存在せずグラフ同型性を失っている。この問題に対応するため、ブロックカクタスグラフのブロックの頂点集合を残しつつ、各ブロック頂点を中心頂点とするスターグラフに変換して、スターグラフをカット頂点で結合した木構造に変換する。この木構造を**ブロックスターツリー**と呼ぶ。

定義 3.2 (ブロックスターツリー)**.** 連結グラフ $G = (V, E)$ が与えられたとき，G のブロック集合を $\{\mathcal{B}_i\}$，カット頂点の集合を $\{c_j\}$ とする。各ブロック $\mathcal{B}_i$ に対して新しい頂点であるブロック頂点を B_i と表し、ブロック $\mathcal{B}_i$ に含まれるすべての頂点 $v \in V(\mathcal{B}_i)$ をブロック頂点 B_i と辺で結ぶ。これにより得られるスターグラフを

$$S_i = \big(\{B_i\} \cup V(\mathcal{B}_i),\ \{(B_i, v) \mid v \in V(\mathcal{B}_i)\}\big)$$

と定める。またスターグラフ S_i に対して、ブロック頂点 B_i において、元の G におけるブロック $\mathcal{B}_i$ がクリークグラフの場合、ブロック頂点 B_i のラベルタイプを P タイプ、ブロック集合 $\mathcal{B}_i$ がサイクルグラフの場合、ブロック頂点 B_i のラベルタイプを Q タイプとする。その他の頂点に対してはラベルタイプを P タイプとする。G のすべてのブロックに対して構成したスターグラフ S_i を、カット頂点 c_j で結合したグラフ

$$H = \left(\bigcup_i V(S_i),\ \bigcup_i E(S_i) \right)$$

を G の ブロックスターツリー と呼ぶ。

ブロックスターツリーはブロックカクタスグラフの頂点を保持しつつ、ブロックとカット頂点の集合を木構造に対応させたグラフである。ブロックカットツリーと同様にブロック頂点以外の頂点は間接頂点と呼ぶ。よってブロックスターツリーはブロック頂点と間接頂点の二部グラフである。また各ブロック頂点にラベルを付与したことで、サイクルグラフとクリークグラフの違いを区別できるようになる。ここまでの内容を整理してブロックカクタスグラフをブロックスターツリーに変換するステップをまとめると以下になる。

定義 3.3 (ブロックスターツリー変換)**.** 以下の手順からなるアルゴリズムをブロックスターツリー変換という

1. ブロックカクタスグラフ G が与えられた場合、G を二重頂点連結成分分解によりブロックの集合 $\{\mathcal{B}_i\}$ とカット頂点 $\{c_j\}$ に分解する。

2. ブロックの集合 $\{\mathcal{B}_i\}$ とカット頂点 $\{c_j\}$ からブロックスターツリー T を作成する。

3. ブロックスターツリー T において、ブロック頂点 B_i が元のブロック $\mathcal{B}_i$ においてクリークグラフである場合、B_i のラベルに P タイプを付与する。　ブ

ロックスターツリー T において、ブロック頂点 B_i が元のブロック $\mathcal{B}_i$ においてサイクルグラフである場合、B_i のラベルに Q タイプを付与する。元のブロック $\mathcal{B}_i$ の頂点数が 3 つである場合、クリークグラフかつサイクルグラフであるが、クリークグラフの定義を満たすため B_i のラベルに P タイプを付与する。他の全ての間接頂点 $\{V(T) \setminus \{B_i\}\}$ に P タイプを付与する。

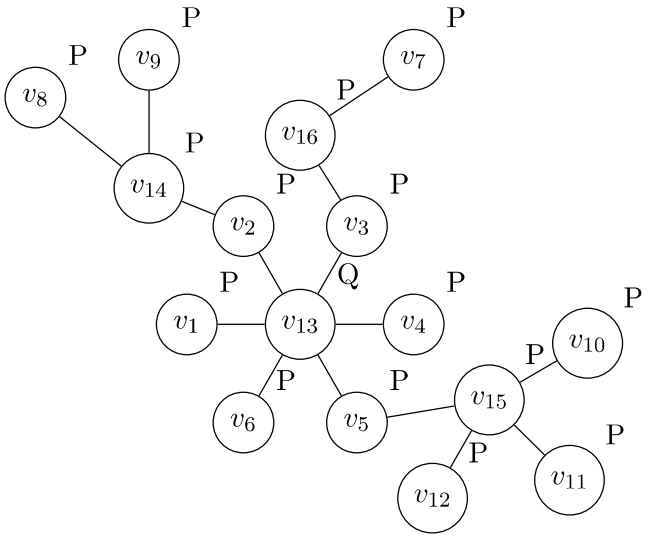


図 3.2: 図 3.1 のブロックカクタスグラフを変換したブロックスターツリー

定理 3.4. ブロックカクタスグラフ $G = (V, E)$ が与えられた時に G をブロックスターツリーに $O(|V| + |E|)$ 時間で変換できる。

Proof. ブロックスターツリー変換の手順 1 の時間計算量は G の二重頂点連結成分分解の時間計算量より $O(|V| + |E|)$ 時間で実行できる。手順 2 において全てのブロック B_i をスターグラフに変換する計算量は $O(|V|)$ 時間である。また全てのカット頂点 c_j に対して、c_j を共有する相異なるブロック $\mathcal{B}_i, \mathcal{B}_k (c_j \in \mathcal{B}_i, c_j \in \mathcal{B}_k)$ を c_j で結合してブロックスターツリー T を作成する処理の時間計算量は $O(|V|)$ である。よって手順 2 の時間計算量は $O(|V|)$ である。ここまでのステップでできたブロックスターツリー T の頂点数は $O(|V|)$ である。手順 3 の時間計算量は T の全ての頂点にラベルを付与する処理の時間計算量より $O(|V|)$ である。よってブロックスターツリー変換全体の時間計算量は $O(|V| + |E|)$ である。 □

3.2 ブロックスターツリーの表現上の課題

ブロックスターツリーはブロックカクタスグラフのグラフ同型性を失わない木構造への変換として不十分である。この課題は、ブロックカクタスグラフを構成

する 2 つのブロックが持つ根本的な頂点の並べ替えの制約の違いに起因する。具体的には、クリークは頂点を任意の並び順に並べ替えたクリークグラフがグラフ同型なグラフを与えるのに対し、サイクルは頂点をサイクルの巡回順か、その逆順に並べ替えたサイクルグラフのみがグラフ同型なグラフを与える。つまりブロックカクタスグラフのクリークにおける頂点の並べ替えは自由であるのに対し、サイクルブロックにおける頂点の並べ替えはサイクルの巡回順か、その逆順にすることのみ可能である。つまり、ブロックスターツリーにおいてサイクルを変換した頂点の並び順を元のサイクルの逆順以外に並べ替えるとグラフ同型性を失ってしまう。この問題はブロックスターツリーのスターグラフがサイクルが持つ頂点の順序の並べ替えの制約を表現できないことに起因する。したがって、ブロックカクタスグラフを正しくグラフ同型性を失わずに木構造に変換するには、この頂点の並べ替えの制約を木構造表現に組み込む必要がある。

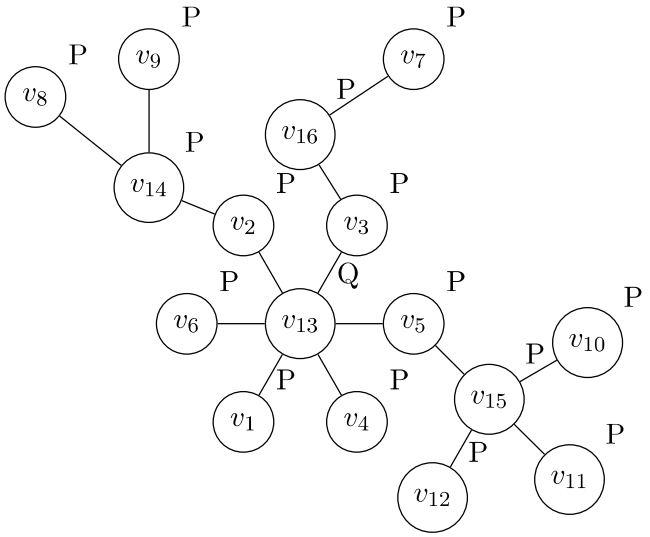


図 3.3: 図 3.2 とグラフ同型でないブロックスターツリー

3.3 ブロックスターツリーにおける PQ-Tree の利用

前節では、ブロックスターツリーがサイクルブロックの頂点の並べ替えの制約を表現できないという課題を明らかにした。本節では、この課題を解決するために木構造の頂点の子頂点に並べ替えの制約を与えられる *PQ-Tree*[3] というデータ構造を導入し、ブロックスターツリー上でこれらの制約を表現する方法を説明する。具体的には、ブロックスターツリーを PQ-Tree で表現して、サイクルのブロック頂点に PQ-Tree の Q ノードを割り当てると、サイクルのブロック頂点の子頂点は元のサイクルの逆順のみへの並べ替えしかできないという制約を組み込むことが

できる。これにより、ブロックカクタスグラフのグラフ同型性を失わないグラフ同型性が判定可能な木構造のデータ構造が作成できる。

3.3.1 PQ-Tree

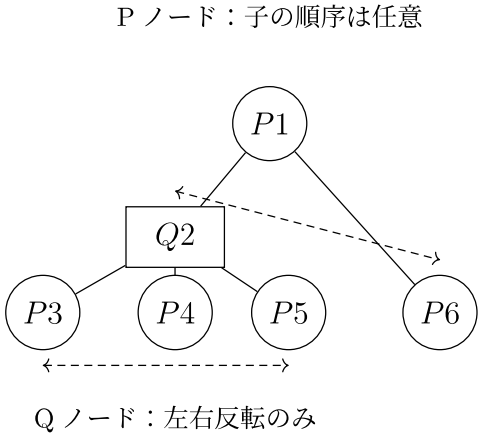


図 3.4: PQ-Tree（P は任意並べ替え，Q は反転のみ）

PQ-Tree は Booth,Lueker が導入した根付き順序木であり、葉頂点以外の頂点を *P* **ノード**あるいは *Q* **ノード**という特別なノードで表現する。2 つのノードの違いは、ある頂点における子頂点の列の並べ替えに対する制約の違いである。P ノードは子頂点の順序に任意の並べ替えを許可し、Q ノードは子頂点の順序に正順か逆順の並べ替えのみを許可する。

定義 3.5 (P ノード)**.** *P ノードとは、PQ-Tree の頂点であり子頂点の列を任意の列に並べ替え可能な頂点のことである [3]。*

定義 3.6 (Q ノード)**.** *Q ノードとは、PQ-Tree の頂点であり子頂点の列を正順か逆順のみに並べ替え可能な頂点のことである [3]。*

本稿では根付き木のある頂点の子頂点の列の並べ替えの制約を表現するために PQ-Tree を用いるため、根付き木の葉頂点は P ノードとする。また PQ-Tree において子頂点を 2 つ以下しか持たない頂点が存在するときは、その頂点は P ノードとする。

定義 3.7 (PQ-Tree)**.** *PQ-Tree とは、全ての頂点が P ノードあるいは Q ノードである根付き順序木のことである [3]。*

3.3.2　根頂点以外に対する PQ-Tree の適用

PQ-Tree への適用はブロックスターツリーを根付き木に変換し、各頂点に以下のルールに基づいて PQ-Tree のノードを適用する。根頂点以外の頂点には以下のルールに基づいてノードを適用する。
Algorithm PQ-Tree の根頂点以外のノード適用ルール

1. **P タイプ頂点への適用ルール**: P ノードを適用する。

2. **Q タイプ頂点への適用ルール**: Q ノードを適用する。

P タイプ頂点はクリークのブロック頂点、間接頂点に対応する頂点である。これらの頂点はブロックスターツリー上で子頂点を自由に並べ替えてもグラフ同型性を失わない。よって、子頂点の列を任意の列に並べ替え可能とする P ノードを適用することで、PQ-Tree でその性質を表現できる。
Q タイプ頂点はサイクルのブロック頂点であり、ブロックスターツリーで親頂点に該当するサイクルの頂点を固定すると、子頂点はサイクルの頂点は裏返すように並べ替えてもグラフ同型性を失わない。よって子頂点の列を正順か逆順のみに並べ替えることができる Q ノードを適用することで、PQ-Tree でその性質を表現できる。ここで Q ノードの親頂点はサイクルを切り開いて残りの頂点を Q ノードの子頂点にしていることに注意する。

3.3.3　根の正規化

PQ-Tree 表現の問題

本章では、ブロックスターツリーの PQ-Tree 表現における最も核心的な課題とその解決策を詳述する。具体的には、ブロックスターツリーの根頂点が Q タイプであるノードの適用において問題が発生する。根頂点が Q タイプの場合、根頂点はサイクルのブロック頂点となる。しかしこのサイクルのブロック頂点には親頂点が存在しない。そのためサイクルを切り開いて子頂点の列を作り Q ノードを割り当てることができないという問題が発生する。仮に、サイクルをある頂点から切り開いて子頂点の列を作ったとしても、子頂点の列の作り方によって異なる PQ-Tree 表現ができてしまう。Q ノードの子頂点の列の作り方が異なる場合、Q ノードは子頂点を正順か逆順にしか並べ替えができないため、並べ替えにより同一の子頂点の列を作ることができるとは限らない。その結果、グラフ同型なブロックスターツリーであっても異なる PQ-Tree 表現が生成されてしまう場合が存在してしまう。これでは、グラフ同型性判定に用いる一意な表現とは言えない。つまり最も核心的な課題とはブロックスターツリーの根頂点が Q ノードの場合、PQ-Tree 表現が一意に定まらないということである。一方で、根頂点が P タイプの場合、根頂点はクリークのブロック頂点、間接頂点となる。これらの頂点は子頂点を自由に並

べ替えてもグラフ同型性を失わない。よって根頂点にPノードを適用する。Pノードの根頂点は子頂点の列の作り方によらず、子頂点の列を並べ替えて同一の列にすることができため一意なPQ-Tree表現ができる。ブロックスターツリーの根頂点の候補が2つになる場合、ブロックスターツリーはブロック頂点、間接頂点の二部グラフであるため、ブロック頂点と間接頂点が根頂点の候補となる。この場合は間接頂点を根頂点とする。

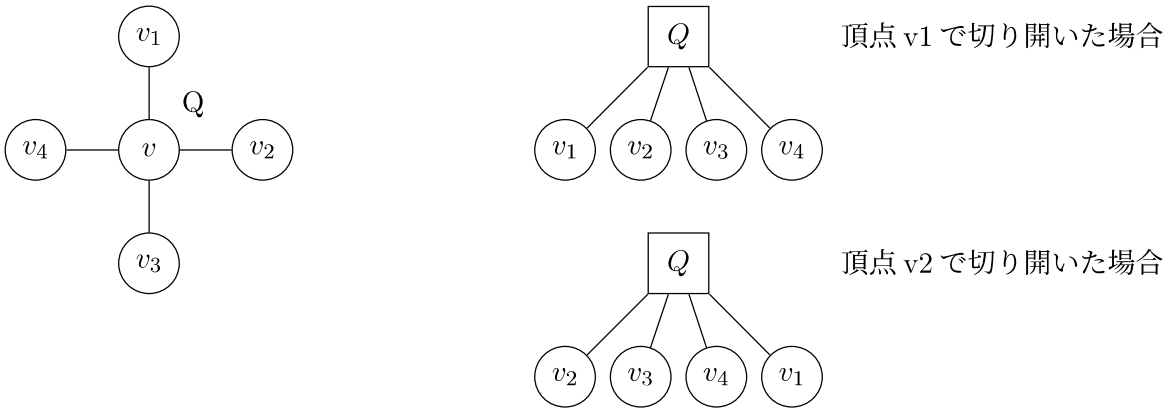


図 3.5: ブロックスターツリーの根がQタイプの場合に生じる表現の非一意性

サイクル頂点の標準表現

ブロックスターツリーから一意なPQ-Treeを作るために、Qノードである根頂点のサイクル頂点から一意な子頂点の列を作る必要がある。このため、サイクルの循環的な性質を扱えるようにサイクル頂点は頂点の循環列であるとする。頂点の循環列から一意な列を選ぶことができれば、サイクル頂点から一意な子頂点の列を作ることができる。一意な列を選ぶために、頂点の列を文字列として表現する。文字列は辞書順の比較ができるため、文字列の循環列から最も辞書順が小さい文字列を選ぶ。この選んだ文字列の元の頂点列をサイクル頂点の一意な列とする。この手法によりQノードの根頂点にサイクル頂点から一意な子頂点の列を与えることを **根の正規化** と呼ぶ。よって根の正規化を含めることで根頂点のノード適用ルールは以下のようになる。

Algorithm PQ-Treeの根頂点のノード適用ルール

1. **Pタイプ頂点への適用ルール**: Pノードを適用する。
2. **Qタイプ頂点への適用ルール**: 前処理として根の正規化を行い、Qノードを適用する。

以上のルールに基づいてPQ-Treeのノードを適用することで、ブロックスターツリーの元のブロックカクタスグラフが持つ頂点順序の制約を完全に保持したPQ-

Tree 表現ができる。その結果、この PQ-Tree に対するグラフ同型性判定は、元のブロックカクタスグラフのグラフ同型性判定と等価になる。

第4章　ブロックカクタスグラフのグラフ同型性文字列

本章では、ブロックカクタスグラフのグラフ同型性判定問題を文字列比較の問題に帰着させることで解決するアルゴリズムを紹介する。本稿の主な貢献は、グラフ同型性判定アルゴリズムそのものを新たに開発することではなく、与えられたブロックカクタスグラフの構造的特徴を完全に保持する一意な文字列表現を定義し、その構築手法を提案することにある。この文字列表現は、2つのグラフがグラフ同型である場合、またその場合に限り、同一の文字列となるように設計されている。したがって、本稿が提案するグラフ同型性判定アルゴリズムは、各グラフに対応するこの文字列を構築したのち、最終的に2つの文字列が等しいかどうかを比較するだけの非常にシンプルなものとなる。

4.1　根付き木のグラフ同型性判定アルゴリズム

本章ではグラフ同型性判定問題の基本的な考え方となる根付き木のグラフ同型性判定問題を考える。

4.1.1　木の中心頂点

根付き木のグラフ同型性判定問題を考える最初のステップは木からグラフ同型性を保持したまま一意な根付き木を得ることである。**木の中心頂点**とは、木 $T = (V, E)$ においてある頂点 $v_c \in V$ を根頂点とした場合、根付き木の高さが最小となる頂点 v_c のことである。木の根頂点を得るために、木の中心頂点 v_c を計算して、v_c を木の根頂点とする。木の中心頂点を得る Bulterman[1] らによる木の直径を求めるアルゴリズムを利用する。木の中心頂点を取得するアルゴリズムは以下である。

Algorithm 木の中心頂点を取得するアルゴリズム (T : 木)

1. 適当な頂点 $v \in V(T)$ を選ぶ
2. v から深さ優先探索を行い、v から全ての頂点に対する距離 $\mathrm{dist}(v, v_i), v_i \in V$ を計算する。v からの距離が最も大きい頂点を v_0 とする。

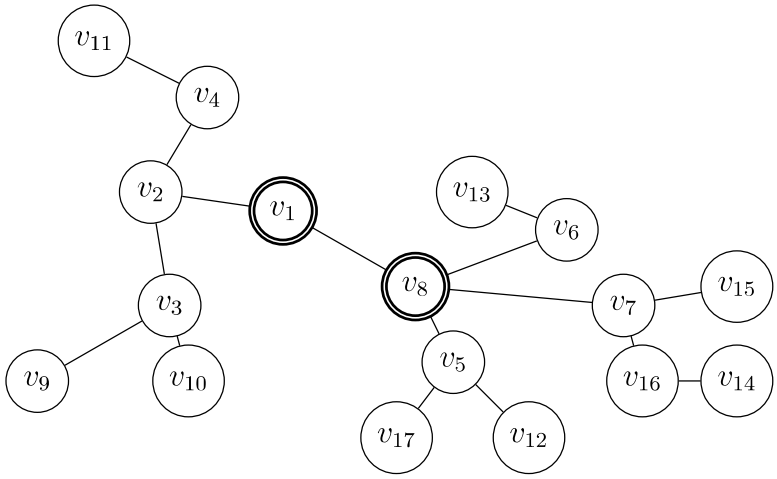


図 4.1: 木と中心頂点

3. v_0 から深さ優先探索を行い、v_0 からの全ての頂点に対する距離 $\mathrm{dist}(v_0, v_i), v_i \in V$ を計算する。
4. $R = \max_{v_i \in V} (\mathrm{dist}(v_0, v_i))$ は木の直径である。木の半径を r とすると、木の中心頂点 v_c は $\mathrm{dist}(v_0, v_c) = r$ となる頂点 v_c である。

図 4.1 は $R = 7, r = 3$ の場合であり、中心頂点は v_1, v_8 である。木の中心頂点は 1 つまたは隣接する 2 つの頂点になる。

定理 4.1. 木の中心頂点 c は線形時間で計算可能である *[1]*。

Proof. 木を $T = (V, E)$ とする。木の中心頂点を計算するアルゴリズムは手順 2,3 の深さ優先探索の時間計算量が $O(|V| + |E|)$、手順 1,4 は $O(1)$ より、全体で $O(|V| + |E|)$ の時間計算量で実行できる。よって、木の中心頂点は線形時間で計算可能である。 □

4.1.2 *GI - Canonical* 表現

次に根付き木をグラフ同型性判定問題で扱いやすくするために木の *GI - Canonical* 表現を提案する。木の *GI - Canonical* 表現はグラフ同型な根付き木に対して同じ文字列となる長さ $O(|V|)$ の文字列表現である。

定義 4.2. 根付き木 Tが与えられた時、Tのグラフ同型性を表現する一意な文字列表現を GI - Canonical 表現と定義する。T の GI - Canonical 表現を $S_{can}(T)$ として表記する。文字列集合 S を辞書順にソートして結合した文字列を $lexsort(S)$ とする。演算子 $+$ は文字列の連結を表すものとする。T の子頂点を根頂点とする部分木の集合を $\{T_1 \dots T_k\}$ とする。$S_{can}(T)$ は以下で定義される文字列である。

$$S_{can}(T) = \begin{cases} \text{“()”} & T \text{が葉頂点の場合} \\ \text{‘(’} + lexsort(\{S_{can}(T_i) \mid 1 \leq i \leq k\}) + \text{‘)’} & \text{それ以外の場合} \end{cases}$$

木の *GI - Canonical* は文字集合 {‘(’,‘)’} からなる ($O(|V|)$ の括弧文字列表現である。この文字列表現の導入により、グラフ同型成判定問題はより単純になる。なぜなら 2 つの根付き木 T_1,T_2 が与えられた時、$S_{can}(T_1)$ と $S_{can}(T_2)$ の文字列の等しさは、T_1 と T_2 のグラフ同型性判定問題の答えと等しくなるためである。つまり、根付き木の *GI - Canonical* 表現の導入は根付き木のグラフ同型性判定問題を文字列比較問題へ帰着できる。木の *GI - Canonical* 表現作成方法は木のソートと文字列作成処理の 2 ステップからなる。このアルゴリズムにおける木のソート処理を**木のグラフ同型ソート**と呼ぶ。木のグラフ同型ソートは木の全ての頂点の子頂点の並び順を正規化したソート処理である。グラフ同型ソートは Alfred らによる木のグラフ同型性判定アルゴリズム [2] を利用する。

定義 4.3 (木のグラフ同型ソート)**.** n 頂点の木 T において木のグラフ同型性判定アルゴリズムを実行する *[2]*。T の全ての頂点 v に付与されたラベルを $l(v)$ とする。ラベルを $l(v)$ の値は v を根とする部分木 T_v のグラフ同型性を一意に識別する整数ラベルである。v の子頂点のラベルをソートしたリストを $r(v)$ とする。v の子頂点の並び順を $r(v)$ におけるラベルの値を持つ子頂点の並び順にソートした木を T' とする。T から T' を得る処理を木のグラフ同型ソートと呼ぶ。

補題 4.4. *n 頂点の木のグラフ同型ソートは $O(n)$ 時間で実行可能である [2]。*

木のグラフ同型性判定アルゴリズムを実行した木の頂点 v の子頂点 c_1, c_2 のラベルの順序関係 $l(c_1) < l(c_2)$ と、c_1, c_2 を根頂点とする部分木 T_1, T_2 の GI-Canonical 表現の順序関係 $S_{\text{can}}(T_1) < S_{\text{can}}(T_2)$ は同値関係にある。そのため木のグラフ同型ソートを行った木に対して、深さ優先探索により文字列を作成することで GI - Canonical 表現が作成できる。

Algorithm GI-Canonical 表現を作成するアルゴリズム

1. 木をグラフ同型ソートする。
2. 作成する文字列 S を初期化する。ソートされた木に対して、深さ優先探索を実行する。葉頂点に到達した場合、文字列 S に “()” を追加する。親頂点に到達した場合、S に ‘(’ を追加する。次に子頂点の探索を行い、探索が全て終了した後、S に ‘)’ を追加する。

補題 4.5. *n 頂点の木のグラフ同型性判定問題は $O(n)$ 時間で判定可能である。*

Proof. 頂点数が n の根付き木を T_1, T_2 とする。T_1, T_2 を木のグラフ同型ソートをした木を T_1', T_2' とする。T_1' の GI - Canonical 表現を得るための時間計算量は T_1' に対す

る深さ優先探索の計算量より $O(n)$ である。同様に T_2' に関して GI - Canonical 表現を得るための時間計算量は $O(n)$ である。2 つの n 頂点の木 T_1', T_2' の *GI - Canonical* 表現の長さは $O(n)$ より、文字列比較の計算量は $O(n)$ である。よって、T_1, T_2' のグラフ同型性判定問題の時間計算量は $O(n)$ である。 □

定理 4.6. 根付き木 T_1,T_2 が与えられた時、T_1 の GI - Canonical 文字列表現と T_2 の GI - Canonical 文字列表現が等しい時、T_1 と T_2 はグラフ同型である。

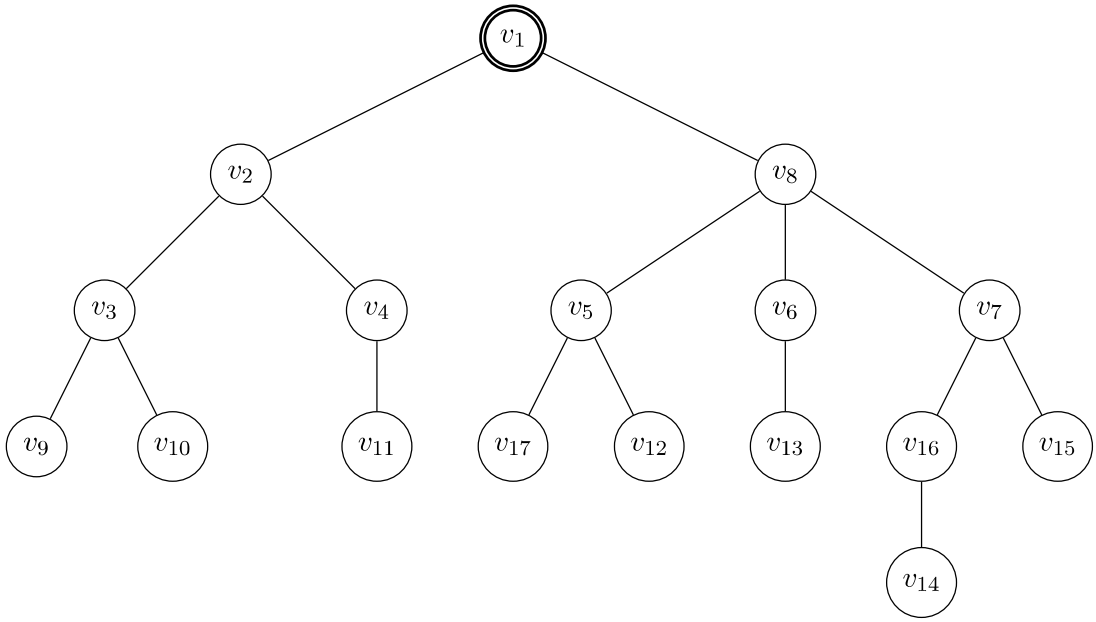
Proof. $S_{\mathbf{can}}(T_1) = S_{\mathbf{can}}(T_2) \Rightarrow T_1$ と T_2 は同型

仮定より、T_1 と T_2 の *GI - Canonical* 文字列が等しいため、$S_{\mathrm{can}}(T_1)$ と $S_{\mathrm{can}}(T_2)$ から復元したグラフは同一である。よって、T_1 と T_2 はグラフ同型である。

$S_{\mathbf{can}}(T_1) = S_{\mathbf{can}}(T_2) \Leftarrow T_1$ と T_2 は同型

T_1、T_2 の木の高さが 1 の場合、T_1、T_2 は葉頂点である。よって GI - Canonical は “()” であるため同じである。よって木の高さが 1 の場合、成り立つ。T_1、T_2 の木の高さが $k-1$ の場合、T_1, T_2 がグラフ同型であるならば、$S_{\mathrm{can}}(T_1) = S_{\mathrm{can}}(T_2)$ が成り立つと仮定する。この時、T_1、T_2 の木の高さが k の場合も成り立つことを示す。T_1、T_2 はグラフ同型より、根頂点の子頂点を根頂点とする部分木の集合は同じである。この部分木の集合は高さが $k-1$ であるため、仮定より根頂点の子頂点からできる GI-Canonical の表現の集合は同じである。また根頂点の子頂点の GI-Canonical の表現のリストは辞書順ソートされるため、同じ GI-Canonical の表現のリストになる。よって木の高さが k である T_1、T_2 の GI-Canonical の表現は等しい。よって全ての高さのグラフ同型な T_1、T_2 について GI-Canonical 表現は等しい。

□



GI-Canonical 表現: ((((())())((()))(()))((()())(())))

図 4.2: 根付き木の GI-Canonical 表現

4.2 PQ-Tree の GI-Canonical 表現構築アルゴリズム

本章ではブロックカクタスグラフを表現した PQ-Tree の GI-Canonical 文字列表現を構築することでグラフ同型性判定問題を解決するアルゴリズムを紹介する。この目的を達成するため、本稿ではブロックカクタスグラフを変換したブロックスターツリーを根付き木に変換し、葉頂点からボトムアップに PQ-Tree のルールを適用しながら GI-Canonical 表現文字列を構築する方法を紹介する。この手法により最終的に得られた GI-Canonical 表現文字列を比較するだけで、ブロックカクタスグラフのグラフの同型性を判定できるため、複雑なグラフ同型性判定問題が単純な文字列比較問題に帰着される。GI-Canonical 表現の構築はブロックスターツリーの根頂点が P タイプの場合は比較的単純であるが、根頂点が Q タイプになる場合、前処理として根の正規化が必要になる。根の正規化には GI-Canonical 表現文字列の循環列の正規化という問題が生じるため、愚直に取り組むと $O(|V|^2)$ 時間の計算量が必要になる。これでは効率的なブロックカクタスグラフのグラフ同型性判定アルゴリズムとは言えない。そのため、まず GI-Canonical 表現を構築する一般的なアルゴリズムを示し、次にこの計算量の課題を LCE というデータ構造を用いて解決する手法を紹介する。

4.2.1 GI-Canonical 表現構築アルゴリズム

本章では与えられたブロックスターツリーを表現する PQ-Tree のグラフ同型性を表現する基本的な GI-Canonical 表現文字列を構築するアルゴリズムを述べる。本アルゴリズムは、入力としてブロックスターツリーを受け取り、グラフ同型なブロックスターツリーを表現した PQ-Tree に対しては必ず同一となる GI-Canonical 文字列を出力する。アルゴリズムの処理は、以下の手順で行われる。

Algorithm GI-Canonical 表現を作成するメインアルゴリズム (T : ブロックスターツリー)

1. 定理 4.1 に示される方法を用いて、入力されたブロックスターツリー T の中心頂点を求める。
   (a) **中心頂点が 1 つである場合**: T を中心頂点を根頂点とする根付き木に変換する。根付き木 T から GI-Canonical 表現を作成するサブアルゴリズムを呼び出し、求めた GI-Canonical 文字列を出力する。
   (b) **中心頂点が 2 つである場合**: 2 つの中心頂点の中で間接頂点である頂点を根頂点とする。根付き木 T から GI-Canonical 表現を作成するサブアルゴリズムを呼び出し、求めた GI-Canonical 文字列を出力する。

Algorithm GI-Canonical 表現を作成するサブアルゴリズム (ブロックスターツリーの根付き木)

1. 木のグラフ同型ソート処理を行う。ただし親頂点が Q タイプの場合、子頂点のラベルのリストをソートをしない。代わりに、子頂点のラベルのリストとその逆順リストの辞書順が小さいものを子頂点のラベルのリストとする。
2. 作成する文字列 S を初期化する。根頂点から深さ優先探索を開始し、各頂点に到達した場合、PQ-Tree のノード適用ルールを適用する。
3. 葉頂点に到達した場合、S に文字列 “()” を追加する。
4. 葉頂点に以外の親頂点に到達した場合、S に ‘(’ を追加する。次に子頂点の探索を行い、探索が全て終了した後、S に ‘)’ を追加する。
5. 根頂点に到達した場合、前処理を行う。
   (a) **根が P タイプの場合**: 根頂点に P ノードを適用する。
   (b) **根が Q タイプの場合**: 根頂点に Q ノードを適用する。根の正規化のため、子頂点の探索で追加される文字列を文字列リストとする。文字列リストを循環列とみなし、その標準形となる文字列リストを求める。求めた文字列リストとその逆順の文字列リストの辞書順が小さいものを子頂点の探索で追加される文字列のリストとする。

P タイプにおけるグラフ同型ソート処理

P タイプの頂点は PQ-Tree における P ノードに対応する。P ノードの場合、子の順序を任意に並べ替え可能であるという性質を持つため、木のグラフ同型ソート処理を行う。

Q タイプにおけるグラフ同型ソート処理

Q タイプの頂点は PQ-Tree における Q ノードに対応する。Q ノードは子頂点の順序を逆順にすることのみを許容する性質を持つため、木のグラフ同型ソート処理の代わりに子のラベルリストを逆順リストとの辞書順比較を行う。

4.2.2 根の正規化アルゴリズムと LCE による高速化

根の正規化のアルゴリズム上の問題

本章では、Q ノードが根頂点となる PQ-Tree 表現が一意に定まらないという課題を解決する根の正規化のアルゴリズムを示す。GI-Canonical 表現を作成するアルゴリズムにおいて、根頂点が Q タイプの場合、単純に深さ優先探索の順序で子頂点の GI-Canonical 表現を連結するだけでは、一意な表現を定めることができない。この問題を正確に理解するため、まずアルゴリズムが根頂点を処理する直前の状態を定義する。深さ優先探索において、根頂点の子頂点の探索が全て完了し、処理が根頂点 v に戻ってきた時点において以下の根付き木と文字列集合を考える。

1. Q タイプを根頂点とする根付き木 T と根頂点の子頂点の列 $v_1, v_2, \ldots, v_k$

$$T = (V, E), \quad (v_1, v_2, \ldots, v_k)$$

2. T の根頂点の探索により得られた子頂点の GI-Canonical 表現の文字列リスト S。

$$S = (s(v_1), \ldots, s(v_k))$$

子頂点 v_i を根頂点とする部分木の GI-Canonical 表現を文字列 $S(v_i)$ とする。

根頂点以外の Q タイプは深さ優先探索において親頂点が存在する。この親頂点は元のサイクルの巡回順の開始頂点を決定しているため、探索における子頂点の探索順は元のサイクルを切り開いたサイクル上での巡回順を反映している。しかし、根頂点が Q タイプの場合は親頂点が存在しないためサイクルの巡回順の開始頂点を決定することができない。そのため深さ優先探索で探索される子頂点の順序 $(v_1, \ldots, v_k)$ は、数あるサイクルの巡回順の一つの順序であるため、このまま文字列集合 S を連結するだけではグラフ同型なブロックスターツリーからでも、処

理する子頂点の順序によって異なる GI-Canonical 文字列表現が生成されてしまう問題がある。すなわち、根の正規化処理の本質は、抽象的なサイクル上に並んだ GI-Canonical 文字列の集合から標準形となる文字列リストを生成することである。

サイクル上文字列の標準形

ブロックスターツリーの根頂点が Q タイプの子頂点から得られる GI-Canonical 表現のリストは、元のサイクル構造を反映したサイクル上の文字列集合として扱わなければならない。サイクル構造を扱いやすくするため、抽象的なサイクル上の文字列集合は文字列の循環列であるとする。文字列の循環列から一意な標準形となる文字列リストを生成するため、循環列の標準形となる要素を決定する。そのための方法として、本稿では **辞書順最小循環順列**（*Lexicographically Smallest Circular Permutation*）を求める。

定義 4.7 (辞書順最小循環順列)**.** 入力に長さ n の列 $a_1, a_2, \ldots, a_n$ が与えられたとする。この時、入力列の以下の条件を満たすある順序 $b_k, \ldots, b_0, \ldots, b_{k-1}$ を辞書順最小循環順列という *[5]*。

条件*:* 全ての $i = 0, \ldots, n-1$ に対して、$b_k, \ldots, b_0, \ldots, b_{k-1} \leq b_i, \ldots, b_0, \ldots, b_{i-1}$

なぜなら循環列において辞書順最小の要素は一意に定まるからである。辞書順最小循環順列が複数になる場合は、どれを選んでも結果は同じである。これを循環列の標準形と見なすことで、Q タイプが根頂点である場合においても一意な GI-Canonical 表現を得ることができる。よって、循環列の辞書順最小循環順列を求めることで、抽象的なサイクル上に並んだ文字列集合から、標準形となる文字列リストを作ることができる。

補題 4.8. 入力に長さ n の列 $a_1, a_2, \ldots, a_n$ が与えられたとする。入力の辞書順最小循環順列を $O(n)$ 時間で求めるアルゴリズムが存在する *[5]*。

辞書順最小循環順列を求める問題は、Booth によるアルゴリズム [5] などを利用することで、列の長さを n として $O(n)$ 時間で解けることが知られている。この事実だけを見ると、$O(n)$ 時間の根の正規化処理を行い、全体で $O(n)$ 時間で n 頂点の PQ-Tree の GI-Canonical 表現を構築できるように思われる。し かし、このアルゴリズムには時間計算量の問題がある。Booth のアルゴリズムは、列の要素同士の辞書順比較が $O(1)$ 時間で完了することを前提としている。しかし、本問題で扱う列の各要素は単一の文字ではなく、それ自体が $O(n)$ の長さになりうる GI − Canonical 表現文字列である。そのため、2 つの要素を比較するだけで $O(n)$ の計算時間が必要となり、辞書順最小循環順列を求めるアルゴリズムが $O(n^2)$ の時間計算量となってしまう。この時間計算量の問題を解決し、ブロックスターツリーの GI-Canonical 表現の作成を全体で $O(n)$ 時間で作成するためには、GI-Canonical

表現文字列同士の辞書順比較を $O(1)$ 時間で行う手法が必要となる。この課題を解決する手段として、次節で LCE（Longest Common Extension）問題の解法を利用する。

LCE による循環列要素の辞書順比較の高速化

前節では、辞書順最小循環順列を標準形として採用することを定義したが、その計算過程において、GI-Canonical 表現文字列同士の比較が $O(n)$ 時間となり、根の正規化アルゴリズムが $O(n^2)$ 時間がかかるという課題が明らかになった。本節では、この時間計算量の問題を解決し、線形時間で GI-Canonical 表現文字列の循環列の辞書順最小循環順列を求めるアルゴリズムを提案する。この目的を達成するため問題を「複数の文字列の比較問題」から「一本の長い文字列に対する LCE（Longest Common Extension）問題」へと変換する。このアプローチにより、線形時間の前処理を行うことで、$O(n)$ の長さの GI-Canonical 表現文字列同士の比較を $O(1)$ 時間で実行可能になる。この結果、辞書順最小循環順列を求めるアルゴリズム全体が線形時間で動作可能となる。この辞書順最小循環順列アルゴリズムに LCE 問題を組み合わせることで、文字列の循環列から線形時間で辞書順最小循環順列を構築する手法が、本論文における最も大きな貢献である。LCE 問題は、以下のように定義される文字列に関する問題である。

定義 4.9 (Longest Common Extension 問題)**.** 長さが n となる 2 つの文字列 S_1 と S_2、S_1 のインデックス i と S_2 のインデックス j からなるクエリの組 (i, j) が与えられる。クエリ (i, j) に対し、S_1 の位置 i から始まる接尾辞 $S_1[i..]$ と、S_2 の位置 j から始まる接尾辞 $S_2[j..]$ の最長共通接頭辞の長さを求めることを *Longest Common Extension(LCE)* 問題という *[7]*。

この問題の自明な解法は、クエリに対して、2 つの接尾辞 $S_1[i..], S_2[j..]$ を先頭から一文字ずつ直接比較する方法である。しかし、この方法はクエリ毎の計算時間は接尾辞の一致部分の長さに比例してしまうため $O(n)$ である。よって、各クエリに対して定数時間で応答できるようなアルゴリズムを構築することが LCE 問題の目的である。すなわち、与えられた文字列 S_1 と S_2 に対して前処理を行い、効率的なデータ構造を構築することで、その後の各クエリに文字列長に依存しない定数時間 $O(1)$ で答えることを目的とする。長さが n の文字列 S に対する LCE 問題の効率的なアルゴリズムは Gusfield[7] によって示された S に対する接尾辞木の構築と、接尾辞木に対する最小共通祖先クエリを組み合わせた手法である。この手法により、$O(n)$ 時間の前処理で LCE 問題のクエリに対して $O(1)$ 時間で応えることができる。

補題 4.10. 長さが n である文字列 S が与えられた場合、$O(n)$ 時間の前処理を行うことで S のインデックス i, j に対するクエリの組 (i, j) に対する *LCE* 問題に $O(1)$ 時間で答えることができるデータ構造 T を作成できる *[7]*。

このLCE問題の効率的な解法を、我々のGI-Canonical表現文字列の比較に応用するアルゴリズムを以下に示す。

アルゴリズム: 文字列の循環列に対する O(1) 時間の辞書順比較辞書の構築

1. **前処理：連結文字列と LCE データ構造の構築**

   入力として与えられる文字列の循環列の任意の要素を $S = (S_1, \ldots, S_n)$ とする。ここで S_i は文字列である。次に、この列を文字列 S' に連結する。このとき各文字列 S_i の終端にそれぞれにユニークな区切り文字（デリミタ $\$_i$）を挿入する。これらのデリミタは、アルファベットのどの文字よりも辞書順で小さく、かつ互いに区別可能（例: $\$_1 < \$_2 < \ldots$）と定義する

   $$S' = S_1\$_1S_2\$_2 \ldots S_n\$_n$$

   ユニークなデリミタを挟むことで、ある文字列 S_i が別の文字列 S_j の接頭辞である場合や、隣接する文字列 S_i, S_{i+1} が等しい場合でも、LCEが文字列の境界を越えてしまうことなく、各文字列を正しく区別して比較することが可能になる。この連結文字列 S' に対して、LCE問題を解くためのデータ構造 T を補題4.10の手法により $O(n)$ 時間で構築する。このデータ構造 T がサイクル上の文字列集合に対する $O(1)$ 時間での辞書順比較を可能にする辞書である。

2. **クエリ：$O(1)$ 時間での辞書順比較の実現**

   前処理で構築したデータ構造 T を利用して、任意の2つの元の文字列 S_i と S_j を $O(1)$ 時間で辞書順比較する。S' における文字列 S_i の開始インデックスを p_i、S_j の開始インデックスを p_j とする。

   (a) S' に対して LCE クエリ (i, j) を実行し、S_i と S_j の最長共通接頭辞長 $l = \mathrm{LCE}(p_i, p_j)$ を $O(1)$ 時間で求める。

   (b) 次に、S' の $p_i + l$ 番目と $p_j + l$ 番目の文字（すなわち、2つの文字列が最初に異なる文字）を比較する。この2文字の大小関係がそのまま S_i と S_j の辞書順となる。

もし $p_i + l$ 番目の文字がデリミタであった場合、s_i は s_j の接頭辞であることを意味し、s_i の方が辞書順で小さいと判定される。両方の文字が同時にデリミタの場合、両者は同一の文字列であるためデリミタの辞書順により s_j より s_i の方が辞書順で小さいと判定される。これにより文字列長に依存しないサイクル上の文字列集合の $O(1)$ 時間での辞書順比較が実現される。

定理 4.11. 各要素が $O(k)$ の文字列である長さが n である列を S とする。$O(nk)$ 時間の前処理を行うことで、S の任意の要素である文字列 S_i, S_j の辞書順を $O(1)$ 時間で比較できる。

Proof. S の各要素にユニークなデリミタ $\$_i$ として、デリミタと各要素を結合した文字列を S' とする。

$$S' = S_1\$_1S_2\$_2 \dots S_n\$_n$$

補題 4.10 より、S' の LCE 問題に応えるデータ構造 T の作成の計算量は $O(nk)$ 時間である。S_i, S_j の S' における開始インデックスを p_i、p_j とすると、データ構造 T に対して $O(1)$ 時間で $l = LCE(pi, pj)$ を求めることができる。文字列 S' の $p_i + l$ 番目の文字と $p_j + l$ 番目の文字の辞書順比較は $O(1)$ であり、この文字の比較結果は S_i, S_j の辞書順比較の結果である。よって、$O(nk)$ 時間の前処理を行うことで、S の要素である文字列 S_i, S_j の辞書順を $O(1)$ 時間で比較できる。 □

1. 前処理：連結文字列 S' の構築

入力: $S =$ (`"((())()"`, `"()"`, `"(())"`, `"((()))"`)

↓

$S' =$ `((())()$()$(())$((()))$`

---

2. クエリ 1: S_1 vs S_3 の比較	**3. クエリ 2: S_1 vs S_2 の比較**
$p_1 = 0$, $p_3 = 10$	$p_1 = 0$, $p_2 = 7$
(a) LCE$(0, 10) = 4$ (共通部分は `"((())"`)	(a) LCE$(0, 7) = 1$ (共通部分は `"("`)
(b) 次の文字を比較: $S'[4] \rightarrow$ `'('` $S'[14] \rightarrow$ `$`	(b) 次の文字を比較: $S'[1] \rightarrow$ `'('` $S'[8] \rightarrow$ `')'`
判定: `'('` > `$` より $\mathbf{S_1 > S_3}$	判定: `'('` < `')'` より $\mathbf{S_1 < S_2}$

図 4.3: GI-Canonical 表現文字列リストに対する LCE クエリの実行例

定理 4.12. 各要素の長さが $O(k)$ である長さが n である列を S とする。S の辞書順最小循環順列を $O(nk)$ 時間で求めることができる。

Proof. 定理 4.11 より、S の任意の要素である S_i, S_j の辞書順比較を $O(nk)$ 時間の前処理を行うことで $O(1)$ 時間で求めることができるデータ構造が作成できる。よって、補題 4.8 のアルゴリズムにおいて S の要素同士の辞書順比較を前処理で作成したデータ構造を用いることで、$O(n)$ 時間で、S の辞書順最小循環順列を求めることができる。アルゴリズム全体の時間計算量は前処理の $O(nk)$ の計算量が最大であるため、$O(nk)$ である。 □

PQ-Tree の GI-Canonical 表現文字列の作成の時間計算量

これまでの議論を統合し n 頂点のブロックスターツリーを表現した PQ-Tree に対しての GI-Canonical 表現が $O(n)$ で構築可能であることを示す。

定理 4.13. 頂点数が n であるブロックスターツリー T が与えられた場合、T を表現した *PQ-Tree* の *GI-Canonical* 表現文字列を $O(n)$ 時間で作成できる。

Proof. n 頂点のブロックスターツリー T を表現した PQ-Tree に対しての GI-Canonical 表現構築アルゴリズムの計算量は、T に対するメインアルゴリズムの計算量である。

- **メインアルゴリズムの計算量:** n 頂点のブロックスターツリー T の中心頂点を求めるアルゴリズムの計算量は定理 4.1 より $O(n)$ である。中心頂点 2 つである場合、間接頂点を中心頂点とする処理の計算量は $O(1)$ である。中心頂点を根頂点とした根付き木 T に対してサブアルゴリズムを呼び出す計算量は $O(n)$ である。よってメインアルゴリズムの計算量は $O(n)$ である

- **サブアルゴリズムの計算量:** 根付き木 T の木のグラフ同型ソート処理において親頂点が Q タイプの場合、子頂点のラベルリストの逆順リストの比較は子頂点の数を k とすると $O(k)$ である。よって、補題 4.4 より T の木のグラフ同型ソート処理は全体で $O(n)$ である。根付き木 T の全ての頂点に対して P ノードと Q ノードを適用する処理は $O(n)$ である。T に対する GI-Canonical 表現構築アルゴリズムの時間計算量は深さ優先探索で各頂点で行われる処理の時間計算量の総和である。したがって、サブアルゴリズムの計算量は $O(n)$ 時間である。

  1. **根頂点の Q ノードでの処理:** 頂点が根頂点であり Q ノードの場合は、根の正規化を行う。根の正規化の計算量は、根頂点の子頂点からなる GI-Canonical 表現文字列リストを S とする。S の各 GI-Canonical 表現文字列の長さは $O(k)$、列の長さが j とすると、S の全ての文字列の長さの総和は $O(jk) = O(n)$ である。よって、根の正規化処理の計算量は定理 4.12 より S の辞書順最小循環順列を求める計算量より $O(n)$ である。根の正規化処理で作成した文字列リストの逆順リストの比較は $O(k)$ 時間である。よって根の正規化の計算量が最大より、$O(n)$ 時間である。
  2. **それ以外頂点の処理:** 文字 '(','(' を追加する処理は $O(1)$ である。子頂点の数を k とすると、子頂点を処理する処理は $O(k)$ 時間である。よって全体で $O(k)$ 時間である。

□

第5章　ブロックカクタスグラフのグラフ同型性判定アルゴリズム

本論文の結論として、ブロックカクタスグラフのグラフ同型性判定問題を効率的に解くためのアルゴリズムを提案する。これまでの議論をまとめると以下の定理が導かれる。

定理 5.1. ブロックカクタスグラフ $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ が与えられた場合、G_1, G_2 のグラフ同型性を $O(|V_1| + |E_1| + |V_2| + |E_2|)$ 時間で判定可能である。

Proof. 本定理は、以下のアルゴリズムの手順と、その時間計算量解析によって証明される。

1. まず、定理 3.4 に基づき、与えられたグラフ G_1, G_2 を、それぞれブロックスターツリー T_1, T_2 に変換する。この処理の時間計算量は、$O(|V_1| + |E_1| + |V_2| + |E_2|)$ である。

2. 次に、定理 4.13 で示されたアルゴリズムを用いて、T_1, T_2 を表現した PQ-Tree の GI-Canonical 表現文字列 S_1, S_2 を作成する。この処理の時間計算量は、$O(|V_1| + |V_2|)$ である。

3. 最後に、文字列 S_1 と S_2 を辞書順で比較する。両者が同一であれば G_1 と G_2 はグラフ同型であり、同一でなければグラフ同型ではない。GI-Canonical 表現文字列の長さは V 頂点の PQ-Tree に対して $O(|V|)$ であるため、この比較処理の時間計算量は $O(|V_1| + |V_2|)$ である。

以上の各ステップの時間計算量を合計すると、全体の時間計算量は $O(|V_1| + |E_1| + |V_2| + |E_2|)$ となる。 □

参考文献

[1] R.W. Bulterman, F.W.van der Sommen, G. Zwaan,T.Verhoeff, A.J.M. van Gasteren, W.H.J. Feijen. On computing a longest path in a tree, Information Processing Letters, 81, pp. 93—96 (2002)

[2] Alfred V. Aho, John E. Hopcroft, Jeffrey D. UIIman. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass, pp. 76–83 (1974)

[3] K. S. Booth and G. S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-a Algorithms, Journal of Computer and System Sciences, 13, No. 3, pp. 335–379 (1976)

[4] Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms, SIAM Journal on Computing, 1, No. 2, pp. 146–160 (1972)

[5] Kellogg S. Booth. Lexicographically Least Circular Substrings, Information Processing Letters, 10, No. 4–5, pp. 240–242 (1980)

[6] Frank Harary. Graph Theory, Addison-Wesley, p. 36 (1969)

[7] Dan Gusfield. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, pp. 198–199 (1997)

[8] K. S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism, Technical Report CS-77-04, Computer Science Department, University of Waterloo, pp. 1–9 (1979)

[9] C. J. Colburn. On testing isomorphism of permutation graphs, Networks, 11, No. 1, pp. 13–21 (1981)