## ABSTRACT

Computer Aided Design software allows mechanical engineers and architects to design complicated systems by specifying geometric constraints on small building blocks. One goal of CAD software is to give users feedback on whether the specified constraints are consistent or not. To approach this problem, we model a design as a *body-and-cad framework*, turn the framework into a *primitive cad graph*, and then use a combinatorial property called *[a,b]-sparsity* to analyze body-and-cad rigidity. We work with an efficient algorithm called the *[a,b]-pebble game* to check $[a, b]$-sparsity using Knuth's algorithm for matroid partitioning in the pebble game. The $[a, b]$-pebble game additionally finds *[a,b]-circuits* which indicate minimal inconsistencies in the constraints. A categorization of the structures of these circuits can help in giving more intuitive feedback to CAD users.

This thesis studies the structures of circuits from a combinatorial perspective. Three categories of circuits were identified previously, but it remained open if the categorization was complete. By systematic enumeration of [a,b]-sparse graphs, we find circuits not in the known categories. We present both statistical results on the enumeration and categorization of $[a, b]$-circuits and case studies on selected uncategorized circuits. We also offer theoretical analysis on the relationship between the structures of circuits and properties of an associated graph called the *Knuth graph*.

Mount Holyoke College

Senior Thesis

# Combinatorial Analysis for CAD

*Xilin Yu*

Supervised by

Prof. Audrey St. John

June 26, 2016

# ACKNOWLEDGEMENTS

This thesis is not a work only by myself. I have been lucky enough to have had so much help and support from my advisers, my professors, friends, and families.

I joined Prof. Audrey St. John, Prof. Jessica Sidman, and Dr. Louis Theran to work on their project two years ago and an open question led to this thesis. None of this would have happened without their previous work and ongoing help.

I want to first thank my thesis adviser Prof. Audrey St. John, for introducing me to this exciting topic. Her constant guidance, support, and encouragement are what make this thesis possible.

I also want to thank Prof. Jessica Sidman as a mentor in mathematics. Her help gives me the mathematical foundation that is so crucial for understanding the core concepts of this project.

My summer internship with Dr. Louis Theran has been nothing but enlightening for me. Discussions with him initiated the main research method for this project. I am very grateful for the opportunity to work with him, which is supported by the Aalto Science Institute (AScI) internship.

This research project is also partially supported by NSF IIS-1253146.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1

# INTRODUCTION

Computer Aided Design software such as Solidworks and AutoCAD are widely used in the fields of engineering and architecture. Users input various components and geometric constraints to create precise 2D drawings or 3D models. A function of CAD software is that it can determine if the system given by the user is consistent and, in the case of a detected inconsistency, provides the user with a set of constraints to check to remove the inconsistency. However, current software often lists not only the constraints causing the inconsistency, but also other constraints whose removal will not actually resolve the inconsistency, causing potential confusion for users. Therefore, effective detection and analysis of the set of minimally inconsistent constraints would enable more useful feedback. We study these constraints from a purely combinatorial perspective using matroid theory. The set of minimally inconsistent constraints corresponds to a *circuit*, which is an object in a *matroid* defined as a minimally dependent set and which will be the main subject of this thesis.

## 1.1 RESEARCH QUESTION

To improve user feedback for Computer Aided Design software, we study the structures of circuits. The main research question we ask is whether we can categorize the structures of circuits using their

combinatorial properties. Previously, three (non-exclusive) types of circuits were identified by [2]. However, whether they comprise a complete categorization is undetermined. To answer the question, we want to check if all circuits necessarily fall under one of the three categories.

## 1.2 RELATED WORK

Related work of this thesis comes from two main communities: the CAD community [3] [4], and the rigidity theory community [8]. White and Whiteley consider bar-and-joint systems categorizing the infinitesimal rigidity of systems using a notion called a *stress* [9] and later extend their work to bar-and-body frameworks [10]. In comparison, we consider a wider range of geometric constraints by studying body-and-cad frameworks. While Owen and Jackson analyze 2D point-line frameworks and use Edmond's algorithm for matroid partitioning as a black box, our approach works with cad-frameworks in arbitrary dimensions and improves the running time complexity by adapting Knuth's algorithm into the $[a, b]$-pebble game algorithm [2].

## 1.3 CONTRIBUTIONS

This thesis project has three main contributions. First, it proves that the $[a, b]$-pebble game correctly detects circuits. Second, it provides the infrastructure needed for understanding structures of circuits. One program enumerates all possible circuit-generating bi-colored graphs given a set of parameters and checks one-by-one if the circuits generated belong to any of the three known categories. Another program produces a graph associated with each circuit, called the Knuth graph, to analyze the combinatorial properties of the struc-

tures of circuits. Third, with this infrastructure and one particular set of parameters, we are able to find a few thousand circuits that do not belong to any of the three existing categories, thus showing that the previous conjectured categorization is not complete. By studying some sample circuits and their associated Knuth graphs, we are able to make observations and come up with a new conjecture about how to categorize these newly discovered circuits. We also prove results that relate some structures of circuits to properties of their associated Knuth graphs.

## 1.4 STRUCTURE OF THESIS

Chapter 2 reproduces definitions of some fundamental concepts of matroid theory and graph theory from [1] and [8]. Chapter 3 presents previous work on rigidity and $[a, b]$-sparsity which set the foundation for this thesis. Chapter 4 focuses on the theoretical analysis of structures of $[a, b]$-circuits that are done before we use the enumeration and categorization tool to study them. Then in Chapters 5 and 6, we show experiments to test some conjectures not proven in Chapter 4. Chapter 5 describes in detail the approach to build the programs to enumerate and categorize circuits and to generate Knuth graphs, while Chapter 6 gives statistical results on the enumeration of some sets of parameters and presents a few case studies of selected circuits that do not fall under known categories. We conclude with Chapter 7, which presents open questions that arose during the project and future work that we want to do to answer those questions.

# 2

# PRELIMINARIES

This chapter reproduces definitions of relevant concepts in matroid theory and graph theory from [1] and introduces notation.

## 2.1 GRAPH THEORY

A graph is a widely used data structure to represent pairwise relations between objects. This section introduces a few common types of graphs: directed graphs, undirected graphs, multigraphs, and bipartite graphs.

**Definition 1.** *A **directed graph** $G$ is a pair $(V, E)$, where $V$ is a finite set and $E \subseteq V \times V$ is a binary relation on $V$. $V$ is the vertex set of $G$ and $E$ is the edge set of $G$. For $u, v \in V$, if $(u, v) \in E$, then there is an edge from $u$ to $v$.*

**Definition 2.** *An **undirected graph** $G$ is a pair $(V, E)$, where the finite set $V$ is the vertex set of $G$ and the edge set $E$ consists of unordered pairs of vertices. For $u, v \in V$, if there is an edge between $u$ and $v$, then $\{u, v\} \in E$.*

By convention, we use $(u, v)$ instead of $\{u, v\}$ to represent an edge in an undirected graph.

**Definition 3.** *A **multigraph** $G$ is a pair $(V, E)$, where $V$ is a finite set of vertices and $E$ is a multiset of pairs of vertices. $E$ is the edge set of $G$, and there can be multiple edges between one pair of nodes.*

**Definition 4.** *A **bipartite graph** is an undirected graph $G = (V, E)$, where $V$ can be partitioned into two disjoint vertex sets $V_1$ and $V_2$ such that for $u, v \in V$, $(u, v) \in E$ implies that either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$.*

In a bipartite graph, there is no edge between vertices from the same vertex set.

**Definition 5.** *A **subgraph** of $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$, $E' \subseteq E$.*

The following definition provides shorthand notation from [2].

**Definition 6.** *If $H$ is a subgraph of $G$, $G/H$ denotes the graph obtained by contracting edges in $H$ and identifying the vertices of every contracted edge wit one vertex.*

## 2.2 MATROID THEORY

Intuitively, a matroid is a mathematical object that generalizes the idea of linear independence of vectors.

**Definition 7.** *A **matroid** is an ordered pair $M = (S, \mathcal{I})$, satisfying the following conditions.*

1. *$S$ is a finite set.*
2. *$\mathcal{I}$ is a nonempty family of subsets of $S$ such that if $B \in \mathcal{I}$ and $A \subset B$, then $A \in \mathcal{I}$. Note that the empty set $\emptyset$ is necessarily a member of $\mathcal{I}$.*

3. *If* $A \in \mathcal{I}$, $B \in \mathcal{I}$ , *and* $|A| < |B|$, *then there exists some element* $x \in B \setminus A$ *such that* $A \cup x \in \mathcal{I}$.

In a matroid $M = (S, \mathcal{I})$, $S$ is the ground set of $M$ and $\mathcal{I}$ is the set of independent subsets of $S$.

**Definition 8.** *A subset* $A$ *of the ground set* $S$ *is* ***dependent*** *if* $A \notin \mathcal{I}$.

**Definition 9.** *A* ***circuit*** *in a matroid* $M = (S, \mathcal{I})$ *is a minimal dependent set* $C$ *such that* $C \notin \mathcal{I}$, *but for every* $x \in C$, $C \setminus \{x\} \in \mathcal{I}$.

**Definition 10.** *A* ***basis*** *of a matroid* $M = (S, \mathcal{I})$ *is a maximal independent set* $B$ *such that* $B \in \mathcal{I}$, *but* $B \cup \{x\} \notin \mathcal{I}$ *for every* $x \in S \setminus B$.

# 3

# BACKGROUND AND PREVIOUS WORK

This thesis relies heavily on previous work and the relevant background is presented in this chapter, including the following concepts: *body-and-cad frameworks* that we use to model CAD user input, *cad graphs* and *primitive cad graphs* that further model the combinatorics of the frameworks, the *[a,b]-pebble game* algorithm which takes a primitive cad graph as input and decides whether it is rigid or flexible and independent or dependent, and the *Knuth graph*, a bipartite graph underlying the implementation of the pebble game. Figure 1 displays an overview of how the above concepts fit into the project.

## 3.1 BODY-AND-CAD FRAMEWORKS AND DEGREES OF FREEDOM

To tackle the challenge of providing more insightful user feedback for CAD users, we first model the CAD structures given by users using *body-and-cad frameworks*.

**Definition 11.** *A **body-and-cad framework** consists of rigid bodies with pairwise **cad** (**c**oincidence, **a**ngular, and **d**istance) **constraints** between geometric elements.*

Each component in a CAD user input becomes a rigid body in the framework, and each geometric constraint becomes a cad constraint.

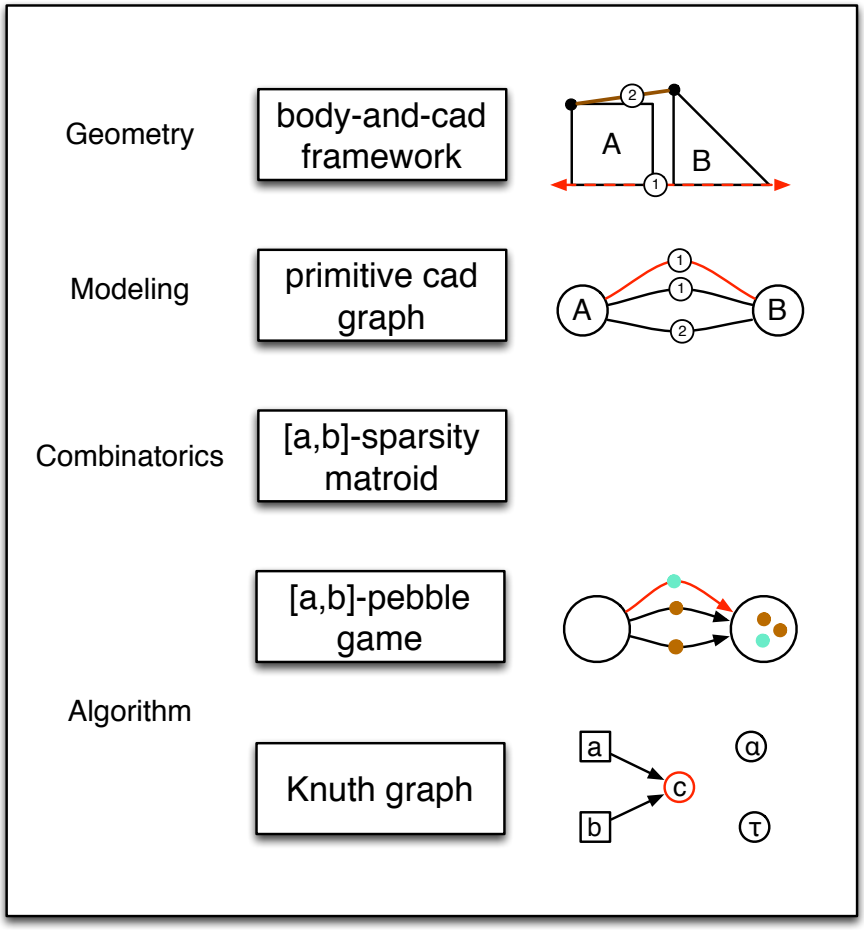


Figure 1: Overview diagram of important concepts in body-and-cad rigidity theory

In 2D, the pairwise constraints can be line-line, point-line, and point-point cad constraints. In 3D, we can additionally have plane-plane, plane-line, and plane-point cad constraints [7]. See Figure 2 for a concrete example.

Given a body-and-cad framework, the main questions we consider are whether the framework is rigid or flexible and whether the constraints are independent or dependent. Associated to a body-and-cad framework is its rigidity matrix which allows us to define the independence and dependence of constraints of a framework.

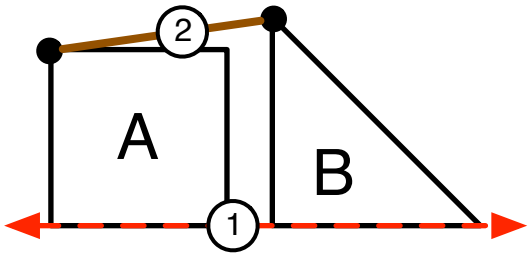


Figure 2: A body-and-cad framework in 2D. Square A and triangle B are two bodies while (1) and (2) are two constraints. (1) is a line-line coincidence constraint that requires lines on A and B to coincide, and (2) is a point-point distance constraint that fixes the distance between the two specified points on A and B.

**Definition 12.** *The constraints of a framework are* ***independent*** *if the rows of the rigidity matrix corresponding to that framework are linearly independent. Otherwise, the constraints of a framework are* ***dependent****.*

Refer to [7] for more details about the development of the rigidity matrix. For this work, we will not work with this algebraic representation of the body-and-cad framework. Instead we will consider the matroid associated to the rigidity matrix, called the rigidity matroid, which captures the independence information of the body-and-cad framework.

To provide some intuition, we discuss the ***degrees of freedom*** that count the dimension of the space of motions allowed to a body. There are two kinds of degrees of freedom for a body: ***rotational*** and ***translational***. We use $a$ and $b$ as general parameters to represent the rotational and translational degrees of freedom of a body, respectively. The space we work in determines the values of a and b. For example, in 2D, $a = 1$ and $b = 2$, while in 3D, $a = 3$ and $b = 3$; refer to Figure 3. However, our approach extends to other values of a and b. For example, we can analyze a space where $a = 2$ and $b = 2$, even though this does not correspond to any Euclidean space.

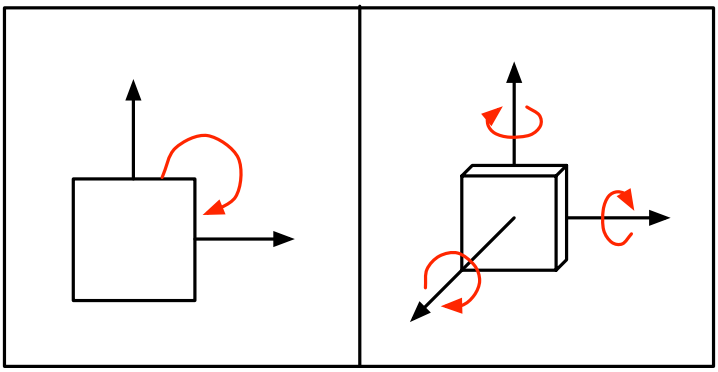

Figure 3: Diagrams of degrees of freedom in both 2D and 3D. The left diagram shows that a body in 2D has 2 translational degrees of freedom and 1 rotational degree of freedom. The right diagram shows that a body in 3D has 3 translational degrees of freedom and 3 rotational degrees of freedom as it can move in three directions and rotate around three axes.

Intuitively, each body in a framework adds $a + b$ degrees of freedom while constraints block degrees of freedom. A framework as a whole will always have $a$ rotational degrees of freedom and $b$ translational degrees of freedom in the space. We call these $a + b$ degrees of freedom the ***trivial degrees of freedom***. A body-and-cad framework consisting of $n$ bodies and no constraints has $n \cdot (a + b)$ degrees of freedom. Besides the $a + b$ trivial degrees of freedom, the remaining $n \cdot (a + b) - (a + b)$ degrees of freedom of the framework are called the ***internal degrees of freedom***. Only the internal degrees of freedom can be blocked by constraints.

**Definition 13.** *If a framework has no internal degrees of freedom, it is* ***rigid****; otherwise, it is* ***flexible****. If a framework is rigid but becomes flexible if we remove any constraint, then it is* ***minimally rigid****.*

Note that rigidity and independence describe different concepts. Rigidity is a characteristic of the framework while independence is a characteristic of the set of constraints. A framework can be flexible but have some constraints that are dependent; it can also be rigid and

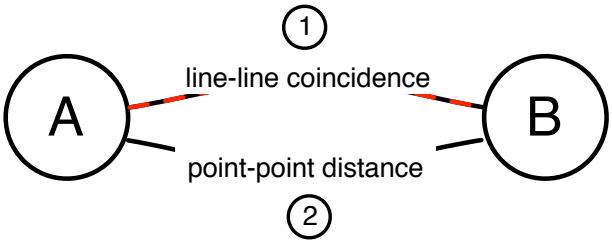


Figure 4: The cad graph of the framework in Figure 2. The square A and the triangle B from the framework both become vertices. The line-line coincidence constraint (1) and the point-point distance constraint (2) both become labeled edges.

have all constraints independent of each other, which makes it minimally rigid.

## 3.2 CAD GRAPH AND PRIMITIVE CAD GRAPH

Body-and-cad frameworks, and their associated rigidity matrices, model the structures given by the user and maintain their geometry. However, as we will see, we can use combinatorics alone to study the independence of generic constraints of a framework, and thus the consistency of constraints in a given structure. To do that, we associate a ***cad graph*** to each framework.

**Definition 14.** *A **cad graph** is an undirected multigraph with labeled edges, where each vertex represents a body in the corresponding framework, and each edge represents a cad constraint in the framework.*

The cad graph ignores the geometry of the cad constraints, and only stores the combinatorial properties of the framework. For example, it ignores the specific angle between two lines on which a line-line angular constraint is applied or the specific distance of a point-point distance constraint. Figure 4 gives an example of cad graph.

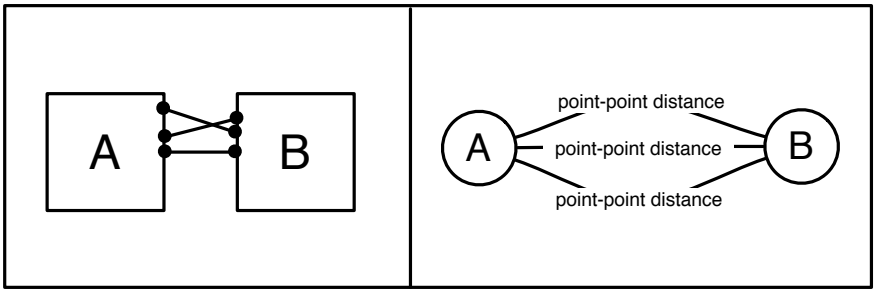


Figure 5: A two-body-three-bar framework in 2D and its corresponding cad graph. The framework is rigid. The three point-point distance constraints correspond to three blind constraints, two of which block the two translational degrees of freedom and one of which block the one rotational degree of freedom.

Different kinds of cad constraints exert different effects on the system. In observing these distinct algebraic behaviors of cad constraints, Haller et al. decomposed each cad constraint into a combination of ***primitive constraints***, called ***angular*** and ***blind*** constraints [7]. An ***angular constraint*** can block a rotational degree of freedom while a ***blind constraint*** can block either a rotational or a translational degree of freedom. Therefore, we can quantify the abilities that different cad constraints possess to constrain the system by decomposing them into primitive cad constraints. Figure 5 gives an example where a blind constraint can be seen blocking a rotational degree of freedom.

We further associate a primitive cad graph to each cad graph. Each labeled edge which represents a cad constraint becomes a combination of colored edges, which represent primitive constraints.

**Definition 15.** *A **primitive cad graph** $G = (V, E = R \sqcup B)$ is a bi-colored undirected multigraph, where the edge set $E$ is partitioned into a set $R$ of red edges and a set $B$ of black edges.*

As in a cad graph, each vertex in a primitive cad graph represents a rigid body. Each red edge in $R$ represents an angular constraint and

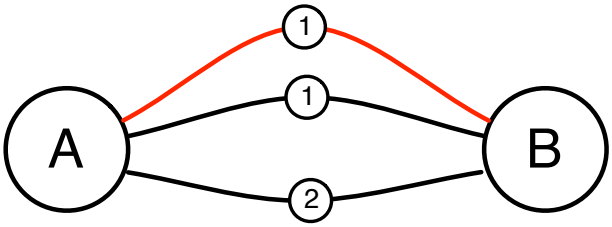


Figure 6: The primitive cad graph of the cad graph in Figure 4. A line-line coincidence constraint is associated to one angular constraint and one blind constraint, so (1) corresponds to one red edge and one black edge. A point-point distance constraint is associated to just one blind constraint, so (2) corresponds to one black edge.

each black edge in $B$ represents a blind constraint. The number of red and black edges associated to each labeled cad edge depends on the specific cad constraint. Refer to Figure 6 as an example.

Recall that a framework with $n$ bodies has $n*(a+b)-(a+b)$ internal degree of freedom, $(n*a-a)$ of which are angular degrees of freedom and $(n*b-b)$ are translational degrees of freedom. Intuitively the framework is independent if, and only if, there are no excess primitive constraints, i.e., if the number of red edges does not exceed $(n*a-a)$ and the number of total edges does not exceed $n*(a+b)-(a+b)$. In fact, this property needs to be true not only for the whole framework, but also for any sub-framework. Furthermore, not only is this condition necessary, but also it is related to sufficiency under certain conditions. The following two theorems characterize the independence of constraints of a body-and-cad framework in terms of $[a,b]$-sparsity, the definition of which will be given after the theorem.

**Theorem 1.** *The constraints of a 2D body-and-cad framework are independent if and only if its corresponding primitive cad graph is $[1,2]$-sparse [2].*

**Theorem 2.** *The constraints of a 3D body-and-cad framework (not containing point-point coincidence constraint) are independent if and only if its corresponding primitive cad graph is [3, 3]-sparse [2].*

Note that we use a property called $[a, b]$-sparsity to decide whether the constraints of a body-and-cad framework are independent in a space with $a$ rotational degrees of freedom and $b$ translational degrees of freedom. However, $[a, b]$-sparsity is not a property specific to a primitive cad graph. It can be applied, as we will see in the following definition, in any undirected bi-colored multigraph.

**Definition 16.** *Let $a, b \in \mathbb{Z}^+$. A undirected bi-colored multigraph $G = (V, E = R \sqcup B)$ is **[a,b]-sparse** if $E$ can be partitioned into two disjoint sets $A$ and $T$ such that $R \subseteq A$, $A$ is $(a, a)$-sparse, and $T$ is $(b, b)$-sparse. Additionally, $G$ is **[a,b]-tight** if $G$ is [a,b]-sparse but $G \cup \{e\}$ is not [a,b]-sparse for any $e \in V \times V \setminus E$ [2].*

In the above definition, we define $[a, b]$-sparsity in terms of $(a, a)$-sparsity and $(b, b)$-sparsity. Notice that we use a square racket around the parameters in the first sparsity property and parentheses in the latter two to differentiate two different sparsity properties. In general, we define $(k, k)$-sparsity in the following way.

**Definition 17.** *Let $k \in \mathbb{Z}^+$. A graph $G = (V, E)$ is **(k, k)-sparse** if for every subgraph $G'$ of $G$, $m' \leq k * n' - k$, where $m' = |E(G')|$, and $n' = |V(G')|$. $G$ is **(k,k)-tight** if $G$ is (k,k)-sparse but $G \cup \{e\}$ is not (k,k)-sparse for any $e \in V \times V \setminus E$.*

From now on, we may also refer to a set of edges $E$ as $[a, b]$-sparse or $(k, k)$-sparse. It means that the graph, which consists of the vertex set spanned by $E$ and the edge set $E$, is $[a, b]$-sparse or $(k, k)$-sparse.

These sparsity conditions are actually matroidal, allowing us to study a family of $[a, b]$-sparsity matroids and $(k, k)$-sparsity matroids [2].

**Definition 18.** *Let $k \in \mathbb{Z}^+$. The $(k, k)$-sparsity matroid for a graph $G = (V, E)$ is $M = (E, \mathcal{I})$ defined as follows:*

- *The ground set for $M$ is the edge set $E$ of $G$.*
- *For any set $F \subseteq E$, $F \in \mathcal{I}$ if and only if $(V, F)$ is $(k, k)$-sparse.*

Recall that a basis for a matroid is a maximal independent subset of the ground set, so if $G = (V, E)$ is $(k, k)$-tight, $E$ is a basis for the $(k, k)$-matroid. We call $E$ a **$(k,k)$-block**.

**Definition 19.** *Let $a, b \in \mathbb{Z}^+$. The $[a, b]$-sparsity matroid for a undirected bi-colored multigraph $G = (V, E = R \sqcup B)$ is $M = (E, \mathcal{I})$ defined as follows:*

- *The ground set for $M$ is the edge set $E$ of $G$.*
- *For any set $F \subseteq E$, $F \in \mathcal{I}$ if and only if $(V, F)$ is $[a, b]$-sparse.*

Note that the $[a, b]$-sparsity matroids are not identical to the corresponding rigidity matroids (discussed in Section 3.1) due to the requirement of omitting point-point coincidence constraints in 3D. An example can be find in [2] to show how the inclusion of point-point coincidence constraints in 3D creates the discrepancy between the two matroids.

Recall that determining the $[a, b]$-sparsity of a graph depends on finding an appropriate partition that satisfy a few conditions. We define such a partition as an *$[a, b]$-partition*.

**Definition 20.** *Let $a, b \in \mathbb{Z}^+$. Let $G = (V, E = R \sqcup B)$ be an undirected bi-colored multigraph. $P = (A, T)$ is an **[a,b]-partition** of $E$ if $E = A \sqcup T$, $R \subseteq A$, $A$ is $(a, a)$-sparse, and $T$ is $(b, b)$-sparse.*

**Lemma 1.** *An undirected bi-colored multigraph $G = (V, E = R \sqcup B)$ is [a,b]-sparse if and only if there exists an $[a, b]$-partition $P$ of $E$.*

## 3.3 THE $[a, b]$-PEBBLE GAME ALGORITHM

Theorem 1 and Theorem 2 can be generalized to characterize all body-and-cad frameworks in terms of $[a, b]$-sparsity (with a few exceptions). This generalization gives us a counting condition that we can use to determine if the framework is rigid and if the constraints of a framework are independent. However, a naive approach would check every partition and further rely on checking all $O(2^n)$ subgraphs. Therefore, checking the counting condition by brute force takes running time of $O(2^n)$. In order to check this counting condition more efficiently, we devise a polynomial time combinatorial algorithm for sparsity called the ***[a,b]-pebble game algorithm*** [2].

The $[a, b]$-pebble game takes in a bi-colored graph $G = (V, E = R \sqcup B)$ and determines $[a, b]$-sparsity properties by trying to insert the edges of $G$ one by one into a directed bi-colored multigraph $\overrightarrow{H} = (V, \overrightarrow{F} = \overrightarrow{A} \sqcup \overrightarrow{T})$ with aqua and tan pebbles on vertices. We call the second graph a ***pebble game configuration***.

We will give the formal definition of a pebble game configuration later, but to understand it intuitively, we can think of pebbles as representing degrees of freedom. Each aqua pebble represents a rotational degree of freedom and each tan pebble represents a translational degree of freedom. Therefore, before inserting edges, each vertex has $a$

aqua pebbles and $b$ tan pebbles. When we insert an edge, we use a pebble to cover the edge; this can be thought of as representing the primitive constraint blocking a degree of freedom. Before each insertion, we check if we have enough pebbles on the vertices to make sure that the counting condition will be maintained after insertion.

In essence, the pebble game is a way of trying to find an [a,b]-partition for $E$. The colors of the pebbles mark which edges end up in partitions $A$ and $T$; the steps of the algorithm make sure that the partitions $A$ and $T$ are $(a, a)$-sparse and $(b, b)$-sparse as we proceed. Therefore, if all edges can be inserted, there exists an $[a, b]$-partition of $E$, certifying that the input graph is $[a, b]$-sparse. Now we give the formal definition of a pebble game configuration.

**Definition 21.** *Let $a, b \in \mathbb{Z}^+$ and let $G = (V, E = R \sqcup B)$ be the input to the [a,b]-pebble game. A **pebble game configuration** is a directed bi-colored multigraph $\overrightarrow{H} = (V, \overrightarrow{F} = \overrightarrow{A} \sqcup \overrightarrow{T})$, where $F$ is the set of inserted edges that corresponds to a subset of $E$, $\overrightarrow{A}$ is a set of edges $\overrightarrow{(u, v)}$ covered by aqua pebbles, and $\overrightarrow{T}$ is a set of edges $\overrightarrow{(u, v)}$ covered by tan pebbles.*

For efficiency of notation, we will also use $H = (V, F = A \sqcup T)$ to denote the undirected version of a pebble game configuration $\overrightarrow{H}$.

Let us describe how the pebble game run on a graph $G$ informally. Each vertex in $H$ has $a$ aqua pebbles and $b$ tan pebbles at the beginning. For any pebble game configuration $H$, the number of aqua pebbles on each vertex $v$ is $a$ minus the number of edges going out of $v$ covered by aqua pebbles. The number of tan pebbles on each vertex $v$ is $b$ minus the number of edges going out of $v$ covered by tan pebbles. We can collect pebbles on vertices by flipping the direction of an existing edge if the original end vertex has an appropriate pebble to cover the edge. After the edge is flipped, the pebble on the edge

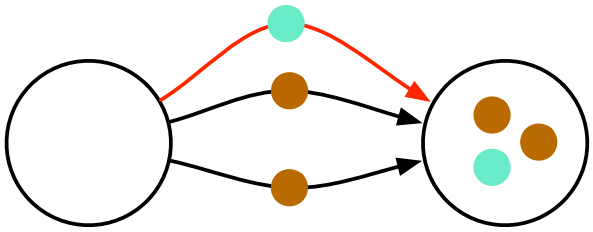

Figure 7: The [a,b]-pebble game configuration after successfully inserting each edge in the primitive cad graph from Figure 6. The algorithm decides that the input is [a,b]-tight.

is returned to the original start vertex and a pebble from the original end vertex is used to cover the edge.

The complete pseudo-code for the [a,b]-pebble game algorithm and the proof of its correctness can be found in [2].

As said, the $[a, b]$-pebble game efficiently checks if a given undirected bi-colored graph satisfies the counting condition of $[a, b]$-sparsity by maintaining the counting conditions using pebbles and this is formally stated in the following theorem and lemma, which follows from the correctness of the $[a, b]$-pebble game.

**Theorem 3.** *Let $a, b \in \mathbb{Z}^+$. An undirected bi-colored graph $G = (V, E = R \sqcup B)$ is $[a, b]$-sparse if and only if every edge in $G$ can be inserted into a pebble game configuration by the [a,b]-pebble game [2].*

**Lemma 2.** *In an $[a, b]$-pebble game on $G = (V, E = R \sqcup B)$, for any pebble game configuration $H = (V, F = A \sqcup T)$, the partition $(A, T)$ is always an $[a, b]$-partition for $F$, which is a subset of $E$.*

The [a,b]-pebble game additionally finds the [a,b]-circuits of the associated sparsity matroid using searches on a corresponding bipartite graph called the *Knuth graph* which we will define and discuss in the next section.

## 3.4 KNUTH GRAPH

During the $[a, b]$-pebble game, the algorithm uses searches on a bipartite graph associated to the current pebble game configuration to determine if we can collect enough pebbles to insert an edge. We call this bipartite graph a Knuth graph because it is a special case of the kind of graphs built for Knuth's algorithm for matroid partitioning [5]. We formally define a Knuth graph as follows.

**Definition 22.** *Let $a, b \in \mathbb{Z}^+$. Let $G = (V, E = R \sqcup B)$ be an undirected bi-colored multigraph. Let $P = (A, T)$ be an [a, b]-partition of E. Then a* ***Knuth graph*** *$\Gamma = (X \sqcup Y, S)$ of G under the partition P is a bipartite graph with vertex sets X and Y and edge set S defined as follows :*

$$
\begin{aligned}
X &= A \cup \{\alpha\}, \\
Y &= T \cup \{\tau\}, \\
S = \{\overrightarrow{(x,y)} \mid x \in A \cap B, y \in T, &\text{ and } T \setminus \{y\} \cup \{x\} \text{ is (b,b)-sparse}\} \\
\cup \{\overrightarrow{(y,x)} \mid y \in T, x \in A, &\text{ and } A \setminus \{x\} \cup \{y\} \text{ is (a,a)-sparse}\} \\
\cup \{\overrightarrow{(x,\tau)} \mid x \in A \cap B, &\text{ and } T \cup \{x\} \text{ is (b,b)-sparse}\} \\
\cup \{\overrightarrow{(y,\alpha)} \mid y \in T, &\text{ and } A \cup \{y\} \text{ is (a,a)-sparse}\}
\end{aligned}
$$

Note that every edge in $G$ corresponds to a vertex in $\Gamma$, and $\Gamma$ has two extra vertices: two terminal vertices $\alpha$ and $\tau$ [5]. Refer to Figure 8 for an example.

The meaning of a directed edge in the Knuth graph is as follows: if $\overrightarrow{(x,c)} \in S$, where $c$ is $\alpha$ or $\tau$, then $a$ can be added into the partition $C$ ($C$ being $A$ or $T$ depending on $c$), without destroying graph sparsity of $G$. If $\overrightarrow{(x,y)} \in S$, where $x$ and $y$ are two vertices other than $\alpha$ or $\tau$ in different partitions, then we can take $y$ out of its partition and put $x$ into $y$'s original partition without destroying graph sparsity of $G$.

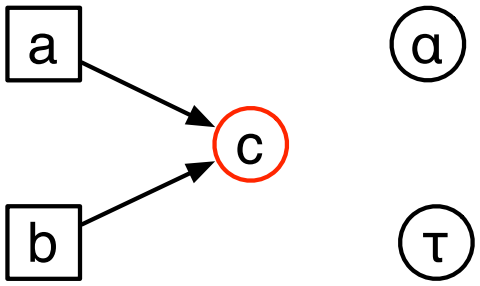


Figure 8: The Knuth graph of the [1, 2]-pebble game configuration in Figure 7. The two black edges become the vertices $a$ and $b$ and the red edge becomes the vertex $c$.

Intuitively, this Knuth graph thus gives us a way to redo partitioning assignment of $G$ by redirecting edges and changing the color of pebbles covering the edges along a path starting from a terminal vertex $\alpha$ or $\tau$ and going backwards to any Knuth vertex (which is an edge in $G$), provided that the path is the shortest from the vertex to the terminal [2].

For our purpose, we will introduce the theorem that describe how an $[a, b]$-pebble game determines if an edge can be inserted at a given stage of the pebble game without proving correctness. The proof can be found in [2]. Before the theorem can be presented, we need a definition of an *augmented Knuth graph*.

**Definition 23.** *Let $a$, $b$, $G = (V, E = R \sqcup B)$, $P = (A \sqcup T)$, and $\Gamma = (X \sqcup Y, S)$ denote the same objects as in the definition of a Knuth graph. Let $e \in V \times V$ be an edge not in $P$. Let $c \in \{1, 2\}$. An **augmented Knuth graph** $\Gamma_c(e) = (X' \sqcup Y', S')$ is an Knuth graph defined in the following way: if $c = 1$,*

$$X' = X \cup \{e\},$$
$$Y' = Y,$$
$$\text{if } e \in R,\ S' = S,$$
$$\text{if } e \in B,\ S' = S \cup \{\overrightarrow{(e, y)} \mid y \in T, \text{ and } T \setminus \{y\} \cup \{e\} \text{ is (b,b)-sparse}\}$$

$$\cup \{\overrightarrow{(e,\tau)} \mid T \cup \{e\} \text{ is (b,b)-sparse}\}$$

*and if* $c = 2$*:*

$$X' = X,$$
$$Y' = Y \cup \{e\},$$
$$S' = S \cup \{\overrightarrow{(e,x)} \mid x \in A, \text{ and } A \setminus \{x\} \cup \{e\} \text{ is (a,a)-sparse}\}$$
$$\cup \{\overrightarrow{(e,\alpha)} \mid A \cup \{e\} \text{ is (a,a)-sparse}\}.$$

Figure 9 gives an example of an augmented Knuth graph. For brevity, we will abuse terminology and simply refer to an augmented Knuth graph as a Knuth graph.

Let $a, b \in \mathbb{Z}^+$. Let $H = (V, F = A \sqcup T)$ be the undirected version of a pebble game configuration.

**Theorem 4.** *Let $e \in V \times V$ be an edge not inserted by the [a,b]-pebble game. Let $\Gamma_1(e)$ and $\Gamma_2(e)$ be the two associated Knuth graphs. The $[a, b]$-pebble game can insert $e$ if and only if there is a path from $e$ to a terminal $\alpha$ or $\tau$ in either $\Gamma_1(e)$ or $\Gamma_2(e)$.*

This follows from Theorem 10 of [2], where the graph used is the union of $\Gamma_1(e)$ and $\Gamma_2(e)$, and the observation that any shortest path in the union is contained in one of the $\Gamma_i(e)$.

If there is no path from $e$ to any terminal, then $e$ cannot be inserted by the $[a, b]$-pebble game, indicating that $G$ is not $[a, b]$-sparse. For an $[a, b]$-dependent input, the $[a, b]$-pebble game additionally finds the circuits in it using searches on the Knuth graph, which we prove in the next chapter.

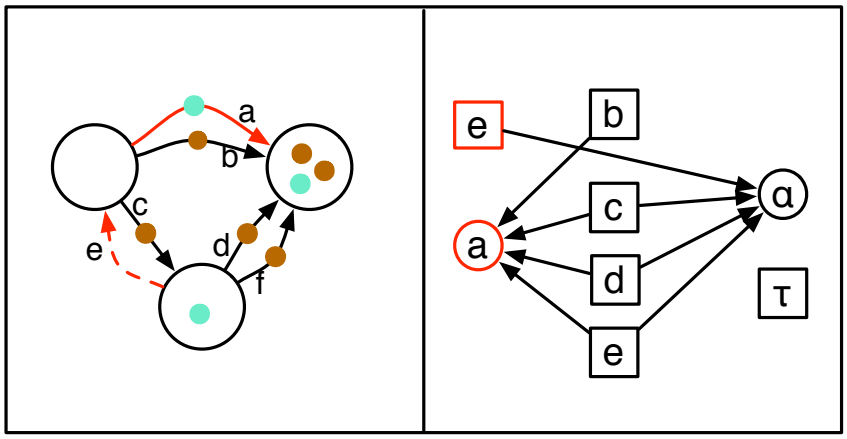


Figure 9: A pebble game configuration with an extra edge $e$ and its corresponding Knuth graph. Since there is a path from $e$ to $\alpha$ in the Knuth graph, the edge will be successfully inserted.

# 4

# [A,B]-CIRCUITS

In this chapter we turn to study the circuits of the $[a, b]$-sparsity matroid. Recall that a circuit of a matroid is a minimally dependent set, therefore an ***[a,b]-circuit*** is a minimally dependent set of the $[a, b]$-sparsity matroid. Similarly, a ***(k,k)-circuit*** is a minimally dependent set of the $(k, k)$-sparsity matroid.

To fix some notation, let $a, b \in \mathbb{Z}^+$. Let $G = (V, E = R \sqcup B)$ be an undirected bi-colored graph. Let $H = (V, F = A \sqcup T)$ represent the undirected version of a pebble game configuration at some stage of the pebble game run on $G$. From Lemma 2, we know that the partition $P = (A, T)$ is always an $[a, b]$-partition of $F$ and $F \subseteq E$. Let $\Gamma = (X \sqcup Y, S)$ be the corresponding Knuth graph of $G$ under the partition $P$. Let $x \in X \sqcup Y$ be any vertex of $\Gamma$. The reachable area from $x$ in $\Gamma$ is the set of all vertices $y \in X \sqcup Y$ such that $x = y$ or there is a path from $x$ to $y$.

## 4.1 CIRCUIT IDENTIFICATION

If the $[a, b]$-pebble game cannot insert every edge of $G$ into $H$, it determines that $G$ is not [a,b]-sparse and $E$ is dependent. For every edge $e$ that cannot be inserted, there is a circuit of $e$ in $E$. The way the pebble game finds the circuit for a given failed edge $e$ is to search on the

associated augmented Knuth graphs of $G$ under the partition given by the current pebble game configuration $H = (V, F = A \sqcup T)$. In order to present the proofs of the theorems that summarize the above information, we first look at Lemmas 3 and 4 and their proofs.

Let $e \in V \times V$ be the edge that the $[a, b]$-pebble game fails to insert. Let $H = (V, F = A \sqcup T)$ be the current pebble game configuration.

**Lemma 3.** *Let $\Gamma_1(e) = (X_1 \sqcup Y_1, S_1)$, $\Gamma_2(e) = (X_2 \sqcup Y_2, S_2)$ be the two associated Knuth graphs. Let $Z$ denote the union of the two sets of vertices reachable from $e$ in $\Gamma_1(e)$ and $\Gamma_2(e)$, not including $e$. Then $Z \cup \{e\}$ is not $[a, b]$-sparse.*

*Proof.* Assume for contradiction that $Z \cup \{e\}$ is $[a, b]$-sparse, where $Z_1$ and $Z_2$ denote the reachable area from $e$ in $\Gamma_1(e)$ and $\Gamma_2(e)$, respectively, and $Z = Z_1 \cup Z_2$. Let $A_Z = A \cap Z$ and $T_Z = T \cap Z$. Since $(A \sqcup T)$ is an $[a, b]$-partition, $(A_Z \sqcup T_Z)$ is also an $[a, b]$-partition. Let $\Gamma_1(e)'$ and $\Gamma_2(e)'$ be the two associated augmented Knuth graphs under the partition $(A_Z \sqcup T_Z)$.

Since $Z \cup \{e\}$ is $[a, b]$-sparse, by Theorems 3 and 4, there is a path from $e$ to a terminal $\alpha$ or $\tau$ in either $\Gamma_1(e)'$ or $\Gamma_2(e)'$.

Let $\Gamma_c(e)'$ be the Knuth graph in which there is a path from $e$ to a terminal $\alpha$ or $\tau$. If $e \in R$, then by definition there is no edge out of $e$ in $\Gamma_1(e)'$, so $Z_1 = \varnothing$, and $c = 2$. If $e \in B$, $c$ can be 1 or 2.

Without loss of generality, say the terminal is $\alpha$. Let $x$ be the last vertex on the path from $e$ to $\alpha$ in $\Gamma_c(e)'$. Then $\overrightarrow{(x, \alpha)}$ indicates that $x \notin A_Z$ and $A_Z \cup \{x\}$ is $(a, a)$-sparse.

Since the pebble game fails to insert $e$ , there is no path from $e$ to any terminal in $\Gamma_c(e)$. Since $x \in Z$, $x$ is reachable from $e$ in $\Gamma_c(e)$, so there is no path from $x$ to a terminal in $\Gamma_c(e)$, in particular, $\overrightarrow{(x, \alpha)} \notin S$. Since

$x \in Z$ but $x \notin A_Z$, then $x \notin A$. Then, by definition of the absence of $\overrightarrow{(x, \alpha)}$, $A \cup \{x\}$ is not $(a, a)$-sparse. However, $A$ is $(a, a)$-sparse. Then there must exist an $(a, a)$-circuit in $A \cup \{x\}$ that contains $x$. We use $D \cup \{x\}$ to denote that circuit. By definition of an $(a, a)$-circuit, for every $y \in D$, $D \cup \{x\} \setminus \{y\}$ is $(a, a)$-sparse. So there is an edge $\overrightarrow{(x, y)}$, for all $y \in D$.

Recall that $x$ is reachable from $e$, then $y$ is reachable from $e$, for all $y \in D$. Therefore, $D \subseteq Z$. Since $D \cup \{x\} \subseteq Z$ and $D \cup \{x\} \subseteq A \cup \{x\}$, we know $D \cup \{x\} \subseteq A_Z \cup \{x\}$. Since $D \cup \{x\}$ is an $(a, a)$-circuit, $A_Z \cup \{x\}$ is not $(a, a)$-sparse. This is a contradiction to our earlier deduction that $A_Z \cup \{x\}$ is $(a, a)$-sparse.

Therefore, the assumption is wrong. So we have proved that $Z \cup \{e\}$ is not $[a, b]$-sparse. □

**Lemma 4.** *Let $\Gamma_1(e) = (X_1 \sqcup Y_1, S_1)$, $\Gamma_2(e) = (X_2 \sqcup Y_2, S_2)$ be the two associated Knuth graphs. For any vertex $f$ reachable from $e$ in either $\Gamma_1(e)$ or $\Gamma_2(e)$, $F \setminus \{f\} \cup \{e\}$ is $[a, b]$-sparse.*

*Proof.* Let $Z_1$ and $Z_2$ denote the set of vertices reachable from $e$ in $\Gamma_1(e)$ and $\Gamma_2(e)$, respectively, not including $e$. Let $Z = Z_1 \cup Z_2 \cup \{e\}$. We want to show that for $f \in Z$, $F \setminus \{f\} \cup \{e\}$ is $[a, b]$-sparse.

If $f = e$, then $F \setminus \{f\} \cup \{e\} = F$. By Theorem 3, since $(V, F)$ is a valid pebble game configuration, it is $[a, b]$-sparse.

Now let $f \in Z$ and $f \neq e$. Let $\Gamma_c(e)$ denote the Knuth graph $f$ is in. Let $g \in F \cup \{e\}$ be a vertex such that $\overrightarrow{(g, f)} \in S_c$ and that either $g = e$ or there is a path from $e$ to $g$. Since $f$ is reachable from $e$, there must exist such a vertex $g$.

If $g = e$, then $\overrightarrow{(e, f)} \in S_c$. If $c = 1$, then $f \in T$ and $T \setminus \{f\} \cup \{e\}$ is $(b, b)$-sparse. If $c = 2$, then $f \in A$ and $A \setminus \{f\} \cup \{e\}$ is $(a, a)$-sparse. Now

consider removing $f$ from the original graph. That is, if $c = 1$, let $A' = A$, $T' = T \setminus \{f\}$, and $F' = A' \sqcup T'$, then $T' \cup \{e\}$ is $(b, b)$-sparse. If $c = 2$, let $A' = A \setminus \{f\}$, $T' = T$, and $F' = A' \sqcup T'$, then $A' \cup \{e\}$ is $(a, a)$-sparse. Therefore in the associated Knuth graph $\Gamma_c(e)'$ of $(V, F')$, there is an edge $\overrightarrow{(e, \tau)}$ if $c = 1$, or $\overrightarrow{(e, \alpha)}$ if $c = 2$. Then by Theorem 4, we can insert $e$ when the pebble game configuration is $(V, F')$. Thus by Theorem 3, $F' \cup \{e\} = F \setminus \{f\} \cup \{e\}$ is $[a, b]$-sparse.

If $g \neq e$, then similarly we can show that if we remove $f$ from the original graph and define $A'$, $T'$, $F'$, and $\Gamma_c(e)'$ the same way, there is an edge $\overrightarrow{(g, \alpha)}$ or $\overrightarrow{(g, \tau)}$ in $\Gamma_c(e)'$. Since removing $f$ does not make any currently sparse graph non-sparse, all edges that are neither from nor to $f$ in $\Gamma_c(e)$ are still present in $\Gamma_c(e)'$. Therefore, there is a path from $e$ to $g$ in $\Gamma_c(e)'$. This together with the edge $\overrightarrow{(g, \alpha)}$ or $\overrightarrow{(g, \tau)}$ gives rise to a path from $e$ to a terminal. Now we can apply Theorems 3 and 4, and we have $F' \cup \{e\} = F \setminus \{f\} \cup \{e\}$ is $[a, b]$-sparse.

So we have shown that for any $f \in Z$, i.e., for any $f$ reachable from $e$ in either $\Gamma_1(e)$ or $\Gamma_2(e)$, $F \setminus \{f\} \cup \{e\}$ is $[a, b]$-sparse. □

Now, we are ready to present the theorem of how to detect an $[a, b]$-circuit of the failed edge $e$ using searches on the Knuth graphs.

**Theorem 5.** *Let $\Gamma_1(e) = (X_1 \sqcup Y_1, S_1)$ and $\Gamma_2(e) = (X_2 \sqcup Y_2, S_2)$ be the two associated Knuth graphs. The circuit containing $e$ is defined by the union of the reachable area from $e$ in $\Gamma_1(e)$ and in $\Gamma_2(e)$.*

*Proof.* Let $Z_1$ and $Z_2$ denote the set of vertices reachable from $e$ in $\Gamma_1(e)$ and $\Gamma_2(e)$, not including $e$, respectively. Let $Z = Z_1 \cup Z_2$.

To show that the union of the reachable area, i.e., $Z \cup \{e\}$, defines the circuit containing $e$, we want to show two things. First, we want to show that $Z \cup \{e\}$ is not $[a, b]$-sparse; this follows from Lemma 3.

Second we want to show that for any $f$ in $Z \cup \{e\}$, $Z \cup \{e\} \setminus \{f\}$ is $[a, b]$-sparse; this follows from Lemma 4. □

## 4.2 CIRCUIT CATEGORIZATION

With the algorithm to find the [a,b]-circuits, we try to analyze their structures from a combonatorial perspective. Previously, three main categories of $[a, b]$-circuits were established [2]. Two of the three categories can also have a sub-variation, which depends on the following concept of *expansion*.

**Definition 24.** *Let $G$ and $H$ be undirected bi-colored graphs, and let $H$ be an $[a, b]$-tight subgraph. Then $G'$ is an expansion of $G$ if $G = G'/H$.*

**Circuit Categories**:

TYPE 1 $(k, k)$-circuits where $k = a + b$.

TYPE 2 $(a, a)$-circuits of red edges.

TYPE 2' expansion of a type 2 circuit.

TYPE 3 $(b, b)$-circuits in a spanning $(a, a)$-block of red edges.

TYPE 3' expansion of a type 3 circuit.

Examples of each type of circuit are shown in Figures 10, 11, and 12.

Note that these types are not exclusive. Consider the $[1, 2]$-circuit in Figure 13. First, the circuit shown in the example is a $(k, k)$-circuit where $k = a + b$, i.e., it is a $(3, 3)$-circuit because it has $3 \times (2 - 1) + 1 = 4$ edges. Second, it is also a $(b, b)$-circuit in a spanning $(a, a)$-block of red edges, i.e., it is a $(2, 2)$-circuit in a spanning $(1, 1)$-block of red edges as the only red edge forms the spanning $(1, 1)$-block of red edges and the three black edges form the $(2, 2)$-circuit.

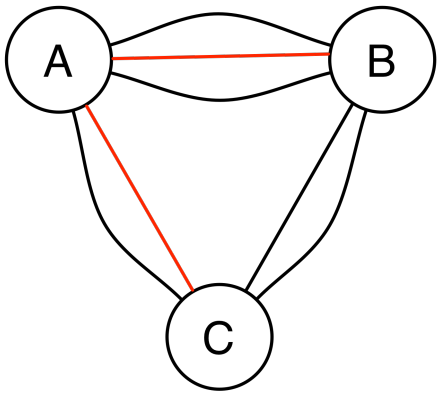


Figure 10: A $[1,2]$-circuit of Type 1, i.e., a $(3,3)$-circuit.

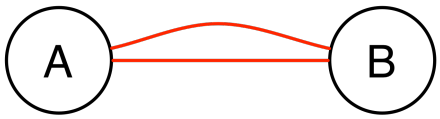


Figure 11: A $[1,2]$-circuit of Type 2, i.e., a $(1,1)$-circuit of red edges.

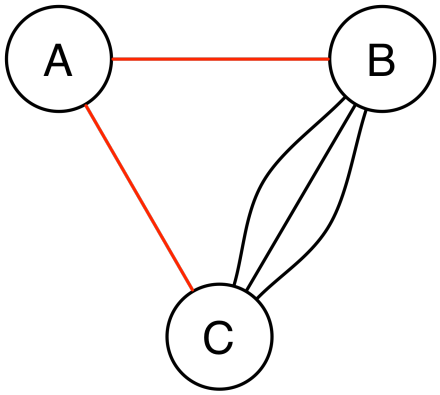


Figure 12: A $[1,2]$-circuit of Type 3, i.e., a $(2,2)$-circuit in a red spanning $(1,1)$-block. The two red edges form a red spanning $(1,1)$-block and the three balck edges covering two vertices form a $(2,2)$-circuit.

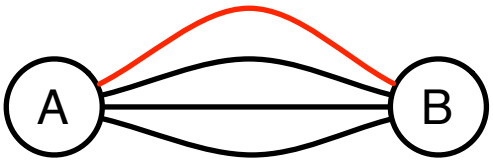


Figure 13: A [1, 2]-circuit of both Type 1 and Type 3.

## 4.3 CIRCUIT STRUCTURES AND KNUTH GRAPHS

Since we can detect an $[a, b]$-circuit by searching on the Knuth graph associated to the graph of the circuit under an $[a, b]$-partition, we surmise that the structures of $[a, b]$-circuits are related to the properties of their Knuth graphs. Here we present some claims we derive from observation and their proofs.

**Claim 1.** *For any $e \in E$, if $e$ is a red edge, then $e$ is a sink in its corresponding Knuth graph $\Gamma = (X \sqcup Y, S)$ under any $[a, b]$-partition $P = (A, T)$.*

*Proof.* Let $e$ be a red edge. For any edge $\overrightarrow{(u, v)} \in S$, $u \in A \cap B$ or $u \in T$. Since $e \in R \subseteq A$, $R \sqcup B$, and $A \sqcup T$, then $e \notin A \cap B$ and $e \notin T$. So no edge directed out of $e$ is defined and the claim is trivially true. □

**Claim 2.** *For any $e \in E$, if $e$ is a black edge, then $e$ cannot be a sink in its corresponding Knuth graph $\Gamma = (X \sqcup Y, S)$ under any $[a, b]$-partition $P = (A, T)$.*

*Proof.* Let $e$ be a black edge. Without loss of generality, assume that $e \in A$. If we change the partition of $e$, i.e., take it from $A$ and put it into $T$, the new partition is $(A \setminus \{e\}) \sqcup (T \cup \{e\})$. $T \cup \{e\}$ is either $(b, b)$-sparse or not $(b, b)$-sparse. If $T \cup \{e\}$ is $(b, b)$-sparse, then by the construction of the Knuth graph, there is an edge from $e$ to a terminal vertex $\tau$ in $\Gamma$. Then $e$ is not a sink. If $T \cup \{e\}$ is not $(b, b)$-sparse, there is a $(b, b)$-circuit in $T \cup \{e\}$ that contains $e$. Then for every edge $f$ that

is not $e$ in the circuit, $T \setminus \{f\} \cup \{e\}$ is $(b, b)$-sparse. Therefore, there is an edge from $e$ to every such an $f$ in $\Gamma$. We conclude that in any Knuth graph under any given $[a, b]$-partition, a black edge must have an out-edge and therefore cannot be a sink. □

**Claim 3.** *For an proper [a,b]-tight subgraph, its corresponding component in the Knuth graph has no out-edge.*

*Proof.* Let $H = (V, F = A \sqcup T)$ be a pebble game configuration and $\Gamma$ be its Knuth graph. Let $H' = (V', F' = A' \sqcup T')$ be a proper [a,b]-tight subgraph and let $\Gamma'$ be its corresponding component in the Knuth graph. Suppose $\Gamma'$ has an out-edge $e \to f$ where $e$ is an edge in $F'$ and $f$ is an edge not in $F'$. Assume that $e \in C$ ($C$ being either the partition $A$ or $T$). Let $D$ be the other partition. Therefore $f \in D$. Since $\overrightarrow{(e, f)}$, $D \setminus \{f\} \cup \{e\}$ is independent ($(a, a)$-sparse, or $(b, b)$-sparse depending on $D$). Since $f$ is not in $F'$, $(D \setminus \{f\} \cup \{e\}) \cap F' = (D \cup \{e\}) \cap F' = \{e\} \cup (D \cap F') = \{e\} \cup D'$, where $D'$ is either $A'$ or $T'$. So $\{e\} \cup D'$ is a subset of $D \setminus \{f\} \cup \{e\}$ and therefore it is independent. But we know $H'$ is [a,b]-tight, so $D'$ is $(a, a)$-tight (if $D' = A'$) or $(b, b)$-tight (if $D'$ is $T'$). Since $e \notin D'$, then $\{e\} \cup D'$ must be dependent. Thus there is a contradiction, which means the assumption that $\Gamma'$ has such an out-edge is wrong. Therefore we proved that in any Knuth graph there is no edge out of the corresponding component of an [a,b]-tight subgraph. □

The converse of the above claim is obviously not true. We know all red edges correspond to sinks in a Knuth graph. Therefore a component in a Knuth graph consisting of only nodes which correspond to red edges necessarily has not out-edges, but it does not correspond to an $[a, b]$-tight subgraph. However, we might think that if we exclude this case, the converse of the above claim is true.

**Question 1.** *If a component in the Knuth graph has no out-edge and does not correspond to a set of edges that are all red, does it correspond to an proper [a,b]-tight subgraph in the pebble graph?*

It turns out that the answer is no because we are able to find counterexamples through the circuit enumeration. More will be presented in Section 6.2.

**Claim 4.** *A circuit is of type 2 (a red (a,a)-circuit) if and only if its Knuth graph has only sinks.*

*Proof.* We prove the forward direction first. Let C be a circuit of type 2. Then C only has red edges. Since each Knuth node corresponding to a red edge is a sink by Claim 1, the Knuth graph of C has only sinks.

We then show the backward direction. Let C be a circuit whose Knuth graph has only sinks. Since a Knuth node corresponding to a black edge cannot be a sink, then C does not have black edges. Then C consists of only red edges, thus C is of type 2. □

# 5

# METHODOLOGY FOR CIRCUIT ENUMERATION AND CATEGORIZATION

The main goal of this thesis is to study the structures of the circuits for [a,b]-sparsity matroids. To investigate circuits, given a set of parameters we enumerate all possible undirected bi-colored graphs that can generate non-trivial circuits and try to categorize them into the three known categories or identify the circuits as not categorized. Since a Knuth graph is the underlying graph that we search on in the $[a, b]$-pebble game to identify the circuits, we want to explore potential connections between the structures of circuits and properties of their corresponding Knuth graphs. Thus, we explicitly generate the associated Knuth graphs for the uncategorized circuits to make observations on the relationship between the two.

In this section we present the methods and algorithms used to enumerate and categorize circuits and to generate the Knuth graphs. Refer to Figure 14 to see how these algorithms together give an infrastructure to study $[a, b]$-circuits.

## 5.1 GRAPH AND CIRCUIT ENUMERATION

Given the parameters $a$, $b$, and the number of nodes in a graph, which is denoted as $n$, we devise a systematic way of enumerating all possi-

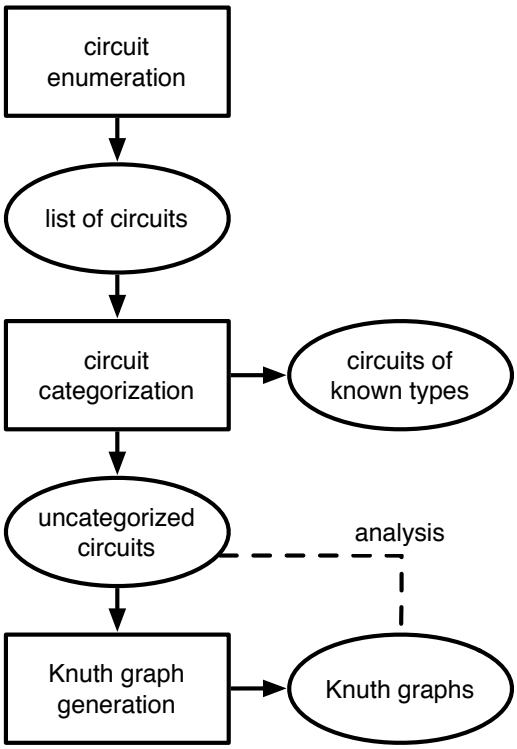


Figure 14: Overview of how the algorithms that we will present form an infrastructure to study $[a, b]$-circuits

ble undirected bi-colored graphs without trivial dependency.

To avoid some trivial dependencies, we allow at most $a$ red edges and $a + b$ black edges between any pair of nodes because any extra edge is guaranteed to cause a trivial (k,k)-circuit between that pair of nodes. We also only allow $(a + b)(n - 1)$ total edges in the graph because any extra edge is guaranteed to cause a trivial (k,k)-circuit. These two types of dependencies are not interesting for our study, thus we do not allow them during the enumeration in order to reduce the number of graphs to be enumerated.

Now, we do some algebra to show how many graphs we need to enumerate with our current setup and why it quickly becomes computationally infeasible to enumerate them all. With $n$ nodes, we have $\binom{n}{2} = n(n-1)/2$ possible pairs of nodes and between each pair of nodes we can have at most $a$ red edges and $a + b$ black edges.

Thus in total we have $(2a+b)n(n-1)/2$ edges as candidates to choose from. Because we only allow $(a+b)(n-1)$ total edges in the graph, the number of graphs we will generate can be calculated as $\binom{(2a+b)n(n-1)/2}{(a+b)(n-1)}$. Refer to Table 1 to see the of graphs that we will need to enumerate if no other restrictions are imposed during the enumeration to reduce the number of graphs.

| $a$ | $b$ | $n$ | Number of Graphs |
|---|---|---|---|
| 1 | 1 | 4 | $1.85 \times 10^4$ |
| 1 | 1 | 5 | $5.85 \times 10^6$ |
| 1 | 2 | 4 | $1.31 \times 10^6$ |
| 1 | 2 | 5 | $5.59 \times 10^9$ |
| 2 | 2 | 4 | $1.25 \times 10^9$ |
| 2 | 2 | 5 | $1.50 \times 10^{14}$ |
| 2 | 3 | 4 | $9.87 \times 10^{10}$ |
| 2 | 3 | 5 | $1.62 \times 10^{17}$ |
| 3 | 3 | 4 | $9.69 \times 10^{13}$ |
| 3 | 3 | 5 | $4.40 \times 10^{21}$ |

Table 1: Number of graphs with different parameters of $a$, $b$, and $n$

As we can see, the number of graphs, i.e., $\binom{(2a+b)n(n-1)/2}{(a+b)(n-1)}$, has a factorial increase with respect to the parameters, so that for any interesting combination of parameters it is impossible to enumerate all graphs within reasonable time. Since all graphs we generate are relatively small, we assume that the time to generate one graph is constant. Therefore, the running time increases roughly linearly with respect to the number of graphs. The time used to enumerate $1.85 \times 10^4$ graphs (where $a = 1$, $b = 1$, $n = 4$) is around 6 minutes. Then, enumerating $1.25 \times 10^9$ graphs (where $a = 2$, $b = 2$, $n = 4$) would take approximately 280 days if we assume linear running time.

Considering the time limitation of this project, we further scale down the number of graphs to enumerate by introducing two more param-

eters $\boldsymbol{p}$ and $\boldsymbol{q}$. Instead of allowing at most $a$ red edges and $a+b$ black edges between each pair of nodes, we allow at most $p$ red edges and $q$ black edges. Obviously, $p \leq a$ and $q \leq a+b$. Thus we have a total of $(p+q)n(n-1)/2$ candidates of edges instead of $(2a+b)n(n-1)/2$. Additionally, the number of candidates we allow has to be no less than the number of edges we need to form a circuit under given $a$, $b$, and $n$. Therefore, $(p+q)n(n-1)/2 \geq (a+b)(n-1)$, which simplifies to $(p+q) \geq 2(a+b)/n$.

Depending on the values of $p$ and $q$, we may be unable to generate some non-trivial circuits. In this way, it is not guaranteed that we will find uncategorized circuits, but it greatly reduces the number of graphs to enumerate. Table 2 shows the number of graphs to be enumerated with given $a$, $b$, $n$ and possible $p$ and $q$. If we compare this table to Table 1, we will see that the magnitude of the number of graphs to enumerate is greatly reduced.

| $a$ | $b$ | $n$ | $p$ | $q$ | Number of Graphs |
|---|---|---|---|---|---|
| 1 | 1 | 5 | 1 | 1 | $1.26 \times 10^5$ |
| 1 | 2 | 4 | 1 | 2 | $4.86 \times 10^4$ |
| 1 | 2 | 5 | 1 | 2 | $8.65 \times 10^7$ |
| 2 | 2 | 4 | 1 | 3 | $2.70 \times 10^6$ |
| 2 | 2 | 4 | 2 | 1 | $1.86 \times 10^4$ |
| 2 | 2 | 4 | 2 | 2 | $2.70 \times 10^6$ |
| 2 | 2 | 5 | 1 | 2 | $1.45 \times 10^8$ |
| 2 | 3 | 4 | 1 | 3 | $1.31 \times 10^6$ |
| 2 | 3 | 4 | 2 | 2 | $1.31 \times 10^6$ |
| 2 | 3 | 4 | 2 | 3 | $1.55 \times 10^8$ |

Table 2: Number of graphs with different parameters $a$, $b$, $n$, $p$, and $q$

Our whole approach of enumerating the circuits given a set of parameters as described above i captured concisely in algorithm 1.

**Algorithm 1** Circuit Enumeration Algorithm

**Input**: $a$, $b$, $n$, $p$, and $q$.
**Output**: A list of $[a, b]$-circuits on $n$ vertices.

1. Initialize $V = \{v_1, v_2, ..., v_n\}$, $L = \emptyset$.
2. For every pair $(v_i, v_j)$, create $p$ red undirected edges and $q$ black undirected edges between $(v_i, v_j)$. Uniquely index each edge with an integer starting from 0.
3. In lexicographic order, generate all combinations of $(a + b)(n - 1)$ integers from the set of integers including 0 to $((p + q)n(n - 1)/2) - 1$ using the Lexicographic Generation Algorithm from [6].
4. For each combination, let $E$ be the set of edges whose index correspond to the integers in the combination. Run the $[a, b]$-pebble game algorithm on the input $G = (V, E)$ to retrieve a list of failed edges. For each failed edge, use the pebble game to identify the circuit $C$ containing that edge and add $C$ to $L$.
5. Output $L$.

## 5.2 CIRCUIT CATEGORIZATION

Now with all the circuits generated by the circuit generation algorithm, we try to see if each circuit can be categorized into any one of the three types we know. Algorithm 2 demonstrates how we use the $[a, b]$-pebble game with special parameters to help identify the circuit categories. Note that in this algorithm, we contract all the $[a, b]$-tight subgraphs before checking if the circuit belongs to Type 2 or Type 3 or none of the two. Therefore, Type 2' and Type 3' circuits are counted as Type 2 and Type 3 respectively. Checking circuit types in the order of Type 1, 2, and 3, this algorithm also only assigns one type to any circuit, even if the circuit may belong to more than one types.

If some circuits are identified as not belonging to any of the three known categories, the graph of the circuit will be saved into an XML

**Algorithm 2** Circuit Categorization Algorithm

**Input**: an undirected bi-colored graph $G = (V, E = R \sqcup B)$, where $E$ is an $[a, b]$-circuit.
**Output**: "$(k, k)$-circuit", "$(a, a)$-circuit of red edges", "$(b, b)$-circuit in a red spanning $(a, a)$-block" or "uncategorized".

1. Check if $E$ is a $(k, k)$-circuit. If $|E| = (a + b)(|V| - 1) + 1$, output "$(k, k)$-circuit"; otherwise, continue.

2. Contract $[a, b]$-tight subgraphs. For every subset $V'$ of $V$, let $E'$ be the set of edges spanned by $V'$. If $|E'| = (a + b)(|V'| - 1)$, contract the subgraph $G' = (V', E')$ into one vertex by reassigning any edge coming out of or into $G'$ to one vertex in $V'$ and then deleting all other vertices and edges in $G'$ from $G$. Repeat until there is no such subset.

3. Check if $E$ is an $(a, a)$-circuit of red edges. If every edge in $E$ is red and $|E| = a(|V| - 1) + 1$, output "$(a, a)$-circuit of red edges"; otherwise, continue.

4. Check if $E$ has a red spanning $(a, a)$-block. Let $G_R = (V, R)$ be the graph with only red edges and run the $[a, 0]$-pebble game on $G_R$. If $G_R$ is not $[a, 0]$-tight, output "uncategorized"; otherwise continue.

5. Check if $E$ is a $(b, b)$-circuit in a red spanning $(a, a)$-block. Let $G_B = (V, B)$ be the graph with only black edges and run the $[0, b]$-pebble game on $G_B$. If $G_B$ is not $[0, b]$-sparse and has exactly one failed edge, output "$(b, b)$-circuit in a red spanning $(a, a)$-block"; otherwise, output "uncategorized".

file that can be easily read into the $[a, b]$-pebble game and the Knuth graph generation program for future analysis.

## 5.3 KNUTH GRAPH GENERATION

For each unclassified circuit, we generate its Knuth graph with $e$ representing the failed edge that causes us to detect the $[a, b]$-circuit.

In Algorithm 3, we give a general algorithm for generating both the Knuth graph for any undirected bi-colored graph with an $[a, b]$-partition of the edges and the special case where we have an failed edge $e$ that doesn't fit into the $[a, b]$-partition.

**Algorithm 3** Knuth Graph Generation Algorithm

**Input** an undirected bi-colored graph $G = (V, E = R \sqcup B)$ and an $[a, b]$-partition $(A, T)$ of $E$. Optionally, a failed edge $e$ with an index $c \in \{1, 2\}$.
**Output** a Knuth graph $\Gamma = (X \sqcup Y, S)$, or $\Gamma_c(e) = (X \sqcup Y, S)$ if $e$ is specified in input.

1. Initialize the vertex and edge sets. $X = A \cup \{\alpha\}$. $Y = T \cup \{\tau\}$. $S = \emptyset$
2. Let $G_T$ be the graph consisting of all the edges in $T$ and the vertices spanned by $T$. Similarly, let $G_A$ be the graph consisting of all the edge in $A$ and the vertices spanned by $A$.
3. For each $e \in B$,
   - If $e \in X$, try to obtain a pebble game configuration where there are $(b + 1)$ tan pebbles on the endpoints of $e$ in the $[0, b]$-pebble game on $G_T$. If this succeeds, add an edge $\overrightarrow{(e, \tau)}$ to $S$.
   - If $e \in Y$, try to obtain a pebble game configuration where there are $(a + 1)$ aqua pebbles on the endpoints of $e$ in the $[a, 0]$-pebble game on $G_A$. If this succeeds, add an edge $\overrightarrow{(e, \alpha)}$ to $S$.
   - In either of the above case, if it fails to obtain the desired pebble game configuration, let $D$ denotes the set of edges visited in the pebble game. Add an edge $\overrightarrow{(e, x)}$ to $S$ for all $x \in D$.
4. If there is an additional edge $e$,
   - If $c = 1$, let $X = X \cup \{e\}$ and follow the procedure in step 3.
   - If $c = 2$, let $Y = Y \cup \{e\}$ and follow the procedure in step 3.
5. Output the Knuth graph $(X \sqcup Y, S)$.

# 6

# RESULTS OF CIRCUIT ENUMERATION AND CATEGORIZATION

With the circuit enumeration and circuit categorization algorithms, we are able to enumerate and categorize circuits ranging from $[1,1]$-circuits on three nodes to $[3,3]$-circuits on five nodes. Not all enumerations yield interesting categorization results. In this chapter, we present part of the statistical results and case studies of some circuits that do not fit into our established categories.

## 6.1 STATISTICAL RESULTS

Table 3 presents part of the statistical results of our circuit enumeration and categorization. Note in this table, type 2 and type 3 circuits includes type 2′ and type 3′ respectively.

We notice that for every set of parameters, there are many more type 1 and type 2 circuit than type 3 circuit (if there is any). Among all the trials, we are only able to find uncategorized circuits under one particular set of parameters, i.e., when $a = 2$, $b = 2$, $n = 4$, $p = 2$, and $q = 2$.

| $a$ | $b$ | $n$ | $p$ | $q$ | Type 1 | Type 2 | Type 3 | Unknown |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 4 | 1 | 2 | 5064 | 3371 | 32 | 0 |
| 1 | 2 | 4 | 1 | 2 | 3373 | 23274 | 177 | 0 |
| 2 | 2 | 3 | 2 | 3 | 355 | 731 | 18 | 0 |
| 2 | 2 | 4 | 1 | 2 | 334 | 222 | 0 | 0 |
| 2 | 2 | 4 | 2 | 2 | 110606 | 1527310 | 34545 | 2491 |
| 2 | 3 | 3 | 2 | 4 | 22764 | 27642 | 768 | 0 |
| 2 | 3 | 4 | 1 | 3 | 25278 | 10182 | 0 | 0 |

Table 3: Number of different types of circuits generated in enumerations under different parameters $a$, $b$, $n$, $p$, and $q$

## 6.2 CASE STUDIES

Since the only setup in which we are able to find circuits that do not belong to the known categories is where $a = 2$, $b = 2$, $n = 4$, $p = 2$, $q = 2$, in all the following case studies, circuits are with respect to the $[2,2]$-sparsity matroid. We will first present four different $[2,2]$-circuits, then show why they do not belong to the known three categories. Since similar arguments can be made for each of the circuit, we will argue in general after all cases are presented to avoid repetition.

Consider the following four graphs $G = (V, E = R \sqcup B)$ in Figure 15. In each example, the set of all edges $E$ is the circuit we find not belonging to the three previous categorizations.

In each of the cases, we know $E$ is an $[a,b]$-circuit because it is detected by the $[a,b]$-pebble game and the correctness of the $[a,b]$-pebble game to detect circuits is proven. Now we show that $E$ is not in any of the three established categories.

First, $E$ is not a type 1 circuit, i.e., a $(k,k)$-circuit where $k = a + b = 4$. It is easy to check that for every subgraph $G = (V, E)$ of $G$, $|E| <= k * |V| - k$, where $k = (a + b) = 4$. Therefore $G$ is $(4,4)$-sparse, so $E$ is

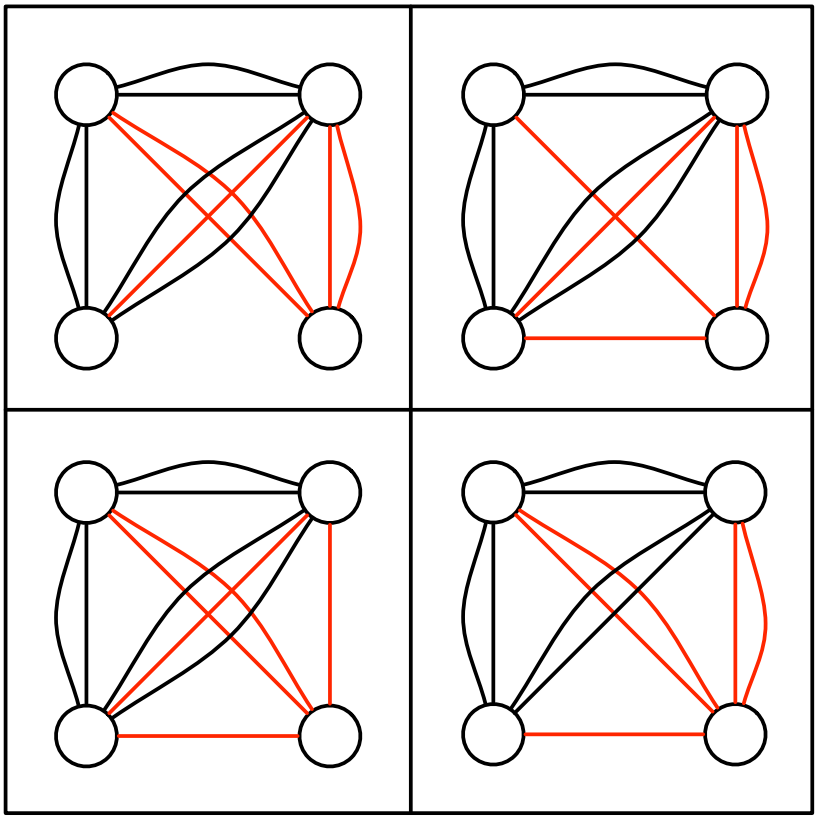

Figure 15: Four undirected bi-colored graphs each of whose edge set forms a circuit not in the known categories. Each graph has five red edges and six black edges.

not a $(4,4)$-circuit.

$E$ is also not a type 2 circuit, i.e., an $(a,a)$-circuit of red edges, where $a = 2$. The reason is that $E$ clearly contains black edges. We continue to demonstrate why $E$ is not a type 3 circuit, i.e., a $(b,b)$-circuit in a spanning $(a,a)$-block of red edges, where $a = 2$ and $b = 2$. This can be shown by simply counting the number of red edges. Since $|V| = 4$, a spanning red $(2,2)$-block would require $4 * 2 - 2 = 6$ red edges. Since we only have five red edges, $E$ cannot be a type 3 circuit.

Through observation, we find that the above four circuits share some common properties of edge counts. They all contain 6 black edges that span 3 of the vertices, which is one edge more than a $(b,b)$-circuit where $b = 2$. They also all contain 5 red edges that span the whole vertex set, which is one edge less than a $(a,a)$-block where $a = 2$. We intuitively categorize them as a $(b,b)$-circuit plus one edge in a red spanning $(a,a)$-block minus one edge.

Formally, for a undirected bi-colored graph $G = (V,E)$, if $E$ consists of only black edges and $|E| = b * (|V| - 1) + 2$ such that there exists an edge $e \in E$ such that $E \setminus \{e\}$ is a $(b,b)$-circuit, then we day $E$ is a $(b,b)$-circuit plus one edge.

For a undirected bi-colored graph $G = (V,E)$, if $E$ consists of only red edges and $|E| = a * (|V| - 1) - 1$ such that there exists an edge $e \in V \times V \setminus E$ such that $E \cup \{e\}$ is a red spanning $(a,a)$-block, then we say $E$ is a red spanning $(a,a)$-block minus one edge.

Now we state the natural resulting conjecture.

**Conjecture 1.** *A $(b,b)$-circuit plus one edge in a red spanning $(a,a)$-block minus one edge is an $[a,b]$-circuit.*

The intuitive idea behind a $(b,b)$-circuit plus one in a red spanning $(a,a)$-block minus one is that we have too many black edges, i.e., blind

constraints, that one must be used to block a rotational degree of freedom. Therefore, we almost have a spanning $(a, a)$-block of red edges. Even though one edge in the block is black, together with all red edges, i.e., angular constraints, the system is angular rigid.

Following the intuitive idea of too many black edges, we ask whether an $[a, b]$-circuit can be a $(b, b)$-circuit plus two in a red spanning $(a, a)$-block minus two, or in general, a $(b, b)$-circuit plus k in a red spanning $(a, a)$-block minus k. We leave this as an open question for now.

# 7

# FUTURE WORK

This thesis project develops an infrastructure to enumerate and categorize $[a, b]$-circuits and to generate their associated Knuth graphs. With this infrastructure, we can study the structures of circuits in hope to provide more helpful feedback to the CAD software users. We successfully generated circuits not belonging to any of the three known categories. However, due to the limited time, we are unable to answer all the questions that arise during the project. There are several things we wold like to do next.

First, even though we came up with a formal description for the circuits in our case studies, we have not been able to prove the conjecture that edge sets constructed according to the description are necessarily $[a, b]$-circuits. We would like to study more examples of circuits and their associated Knuth graphs snd hope to find relations between the two that help to prove the conjecture.

Second, it remains an open question whether the description can be generalized into a $(b, b)$-circuit plus $c$ edges in a red spanning $(a, a)$-block minus $c$ edges where $c \in \mathbb{Z}^+$. If we can prove that edge sets constructed according to the generalized description are indeed $[a, b]$-circuits, it can define a new type of $[a, b]$-circuits. Then our original type 3 circuits will be a special case of this type where $c = 0$. More enumerations with larger parameters and less repetition of graphs

may also generate more examples to help answer the question.

Third, with larger parameters of $a$, $b$, or $n$, the number of graphs to enumerate increases exponentially, thus making the enumeration too long for this project. Other sets of parameters may produce more circuits that do not fit into the three established categories. However, to run the enumeration under larger parameters, we have to find a way to reduce the number of graphs we generate. Currently, our circuit enumeration algorithm does not take graph isomorphism into account. For each graph, our algorithm may generate hundreds or thousands of isomorphic ones throughout the enumeration process, which means that if we have some way to detect obvious isomorphic graphs, we can reduce the number of graphs to enumerate by a magnitude of two or three.

# 8

# BIBLIOGRAPHY


[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* Cambridge, Mass. : MIT Press, c2009., 2009.

[2] James Farre, Helena Kleinschmidt, Audrey Lee-St.John, Jessica Sidman, Stephanie Stark, Louis Theran, and Xilin Yu. Algorithms for detecting dependencies and rigid subsystems for cad. *Submitted to Computer Aided Geometric Design*, 2015.

[3] Sebti Foufou and Dominique Michelucci. Interrogating witnesses for geometric constraint solving. *Information and Computation*, 216:24 – 38, 2012. Special Issue: 8th Conference on Real Numbers and Computers.

[4] Jack Graver, Brigitte Servatius, and Herman Servatius. *Combinatorial Rigidity*, volume 2 of *Graduate Studies in Mathematics*. American Mathematical Society, 1993.

[5] Donald Knuth. Matroid partitioning. Technical report, Standford University, March 1973.

[6] Donald Ervin Knuth. *The art of computer programming*, volume 4. Addison-Wesley, third edition, 2005.

[7] Audrey Lee-St.John and Jessica Sidman. Combinatorics and the rigidity of cad systems. *Computer-Aided Design*, 45(2):473–482, 2013.

[8] James G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[9] Neil White and Walter Whiteley. The algebraic geometry of stresses in frameworks. *SIAM Journal on Algebraic Discrete Methods*, 4(4):481–511, 1983.

[10] Neil White and Walter Whiteley. The algebraic geometry of motions of bar-and-body frameworks. *SIAM Journal of Algebraic Discrete Methods*, 8:1–32, 1987.