THESIS

DIRECTED ACYCLIC CONSTRAINED CONNECTIVITY: COMPLEXITY AND APPLICATIONS

Submitted by

Brian Tan

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2025

Master's Committee:

Advisor: Ewan Davies

Sanjay Rajopadhye
Nikhil Krishnaswamy
Emily King

ABSTRACT

DIRECTED ACYCLIC CONSTRAINED CONNECTIVITY:
COMPLEXITY AND APPLICATIONS

In this thesis we will explore a new problem on Graph Theory named Directed Acyclic Constrained Connectivity. Directed Acyclic Constrained Connectivity is a modification of (s, t)-Connectivity where we introduce constraints that restrict edge use.

We show that Directed Acyclic Constrained Connectivity is NP-Complete. We reduce Graph Coloring problems such as the 3-Coloring Problem to Directed Acyclic Constrained Connectivity. We also show that Directed Acyclic Constrained Connectivity can be related to the enumeration of Maximal Independent Sets. This means that some cases of Directed Acyclic Constrained Connectivity are solvable in polynomial time using advanced algorithms for enumerating Maximal Independent Sets.

Finally, we apply our findings to a problem in the field of Access Control and Cyber Security. We show that the Safety Problem for the Next Generational Access Control (NGAC) Model is NP-Complete based on our findings for Directed Acyclic Constrained Connectivity. The NGAC Model is a new model for Access Control that is based on the Role Based Access Control (RBAC) Model.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Ewan Davies, for being a great mentor in research and in life. I am very grateful for all that you have taught me. To do research in theory and mathematics is one of the greatest privilege of my life, and I am also very grateful to learn some very important life lessons along the way while doing our research. I am also grateful for your help with this project which could not have been finished without your help on tying up some loose ends. I am excited for what's to come in the next few years!

I would like to thank Indrakshi Ray and Mahmoud Abdelgawad for your help and support on this project. This project started out as an Access Control project, and I was fortunate to receive direct help on understanding access control, which I had no previous experience on. I am very grateful for your help and this project could not have been finished without you both.

Finally, I would like to thank my parents, family, and friends for their immense support from all the way across the world. I would like to thank both my parents, Jimmy Kurnia and Yohana Melian Susanti, for always supporting me in all forms. I would not be where I am now without your help and support. I love you both and I owe every bit of my success to you.

DEDICATION

I would like to dedicate this Thesis to my parents, Jimmy Kurnia and Yohana Melian Susanti.

Everything I have achieved is thanks to the both of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction and Related Works

Directed Acyclic Constrained Connectivity is a new problem in Graph Theory that is a modification of the classic $(s, t)-$Connectivity Problem. We created this problem to serve as a graph-theoretical view of the Safety Problem in Access Control, which we talk about more in Chapter 3. In this thesis, we will discuss our findings from a purely theoretical point of view in Chapter 2. Then, we will discuss our findings from an Access Control point of view in Chapter 3.

In Section 1.1, we give a detailed introduction to necessary Graph Theory concepts. We first give definitions and notations for graphs. Then, we talk about decision problems in Graph Theory related to our work.

In Section 1.2, we will give an introduction to Computational Complexity. We will first give definitions to various complexity classes. Then, we will discuss how a problem falls under a certain class. This section contains important preliminary knowledge for Chapter 2 when we discuss about the complexity of Directed Acyclic Constrained Connectivity.

In Section 1.3, we will give an introduction to the Next Generation Access Control (NGAC) Model. We will give definitions to the model and also the *safety* problem which we will apply our findings from directed acylic constrained connectivity to. In Section 1.4, we will give an introduction to the Directed Acyclic Constrained Connectivity Problem. We will give definitions and the problem statement.

In Section 1.5, we will give related works on problems relating to directed acyclic constrained connectivity, access control, and independent sets. We will discuss past papers and well known results from each respective fields and how it applies to our problem.

1.1 Graph Theory

Graphs are a very important combinatorial object that shows up everywhere in our daily life. Graphs are able to model many things such as social connections, maps, and many more.

Definition 1: Graph

A graph $G = (V, E)$ is an ordered pair consisting of two sets V and E. V is called the set of Vertices. E is called the set of Edges. There are no restrictions to V as they are just labels to nodes in the Graph. Each element in E contains an ordered pair of vertices, where $(a, b) \in E$ represents an *edge* or a connection from a to b.

A graph can either be directed or undirected. An undirected graph can be modeled as a bidirectional graph, with an edge set $E \subseteq V \times V$, such that $(a, b) \in E \implies (b, a) \in E$. A directed graph does not necessarily have that property. From this point onwards, we assume every graph is undirected, unless said otherwise.

Definition 2: Paths

Given a graph $G = (V, E)$, and a pair of vertices $s, t \in V$. A sequence of distinct nodes $P = (v_1, v_2, \cdots, v_k)$ is a path from s to t if and only if

$$\forall i < k \leq n, v_i \in V \wedge (v_i, v_{i+1}) \in E$$

A graph is *connected* if and only if there exists a path for all possible s, t pairs.

Definition 3: $(s, t)-$Connectivity (ST-CON) Problem

Given a graph $G = (V, E)$ and a pair of vertices $s, t \in V$. Is there a path between s to t?

ST-CON is a classic graph theory problem that falls under the **P** Complexity Class. Constrained Connectivity is a modification of the ST-CON Problem where we add constraints. The modification will make the problem fall into a different complexity class unless **P** = **NP**.

Definition 4: Independent Set

For a given graph $G = (V, E)$, an independent set $I \subseteq V$, is a set of vertices such that

$$\forall u, v \in I,\ (u, v) \notin E$$

An Independent Sets is maximum if its cardinality is bigger than any other independent sets. A *maximal* independent set is a type of independent set where it is not a subset of another independent set. In simple words, an independent set is maximal when adding another vertex in the set does not make it independent anymore. Figure 1.1 is an example of a non-maximal vs a maximal independent set. The first independent set is not maximal since you can add "B" in there and it will still be an independent set. The second independent set is maximal since no other vertex can be added.

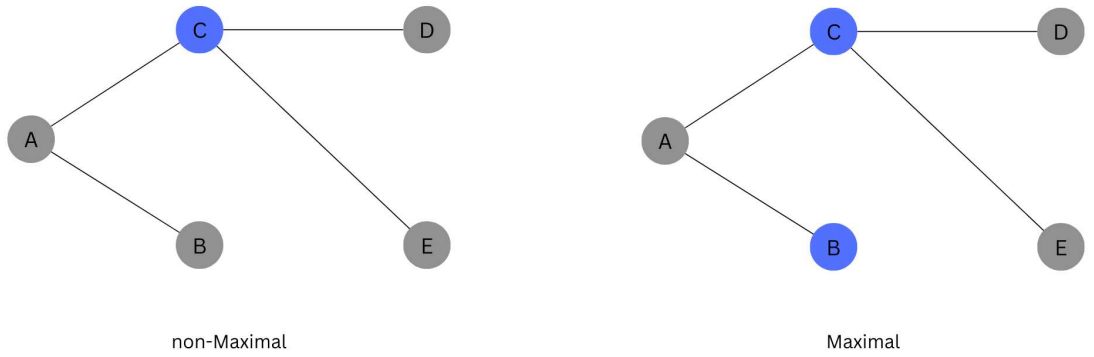

Figure 1.1: Non-maximal vs Maximal Independent Set

Definition 5: Graph Coloring

Given graph $G = (V, E)$, and a set of "colors" Q. A coloring of G is a map $\varphi : V \to Q$ such that

$$\forall u, v \in V,\ uv \in E \implies \varphi(u) \neq \varphi(v)$$

Graph Coloring is a very well known problem in Combinatorics. There are many variations of this problem such as optimal colorings (finding the minimal number of "colors" needed to color the whole graph), enumeration of colorings (finding all possible assignments of colors), existence of colorings (to check whether it is possible to color a graph given its structure), and more. We are particularly interested in $3-$Coloring for our work.

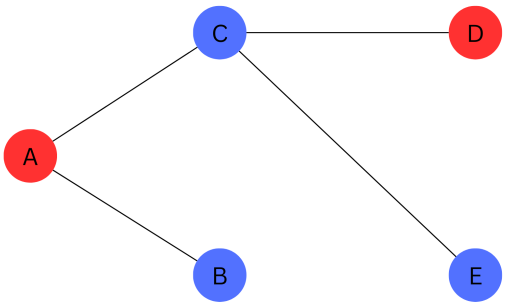

Figure 1.2: Example of a Proper Graph Coloring

Definition 6: 3-COL Problem
Given a graph $G = (V, E)$, and a set of colors Q such that $\|Q\| = 3$. Does there exist a map $\varphi : V \to Q$ such that $$\forall v \in V, \varphi(v) \notin \{\varphi(u) \,\|\, u \in N(v)\}$$

This is a well known Constraint Satisfiability Problem (CSP) in Graph Theory. 3-COL is also well known in complexity theory as this problem can be used as a reduction target for many NP-Complete Problems such as 3-SAT and more. We will also reduce 3-COL to Constrained Connectivity.

1.2 Computational Complexity

In this section, we will give a detailed introduction to computational complexity. We will give definitions to several complexity classes and notations in this section which will be solely based on [1].

In complexity theory, it is common practice to represent objects as bit strings. An object $x \in \{0,1\}^*$ where $\{0,1\}^*$ is the set of all bit strings.

Definition 7: Languages

For a decision problem, its corresponding language is the set of all objects that will return a "Yes" to that problem.

For example, for a decision problem that asks whether an integer is even, has a language $L = \{x \in \{0,1\}^* : x \text{ is an integer divisible by } 2\}$. A language is used to represent decision problems in complexity theory.

Definition 8: NP Class

A language (decision problem) $L \subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time algorithm M such that for every $x \in L$

$$\exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

u is called a certificate for x.

A certificate u is an object of polynomial length, with respect to the length of the input x, where it serves as a "proof" that x will return a "Yes". We can think of M will as a verifier algorithm to make sure that u does actually show x is a "Yes" answer.

Definition 9: Reductions

A language A is polynomial-time reducible to a language B, or $A \leq_p B$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A \iff f(x) \in B$.

To build a reduction function, we must map "Yes" inputs of A to "Yes" inputs of B, and "No" inputs of A to "No" inputs of B. We have to build a *computational* function that transforms an input of A to an input of B. We also have to make sure the transformation process takes polynomial time.

Definition 10: **NP-Hard** Class

A language B is **NP-Hard** if and only if for every language $A \in$ **NP**, $A \leq_p B$.

It is sufficient to only reduce one (proven) **NP-Hard** problem, instead of having to reduce all **NP** Problems, to show your problem is also **NP-Hard**.

Definition 11: **NP-Complete**

A language A is **NP-Complete** iff $A \in$ **NP** and $A \in$ **NP-Hard**

Definition 12: **P** Class

A language $L \subseteq \{0,1\}^*$ is in **P** if there exists a polynomial time algorithm M such that if $x \in L$, $M(x) = 1$ and if $x \notin L, M(x) = 0$.

Simply put, a problem in **P** can be solved in polynomial time. Unless **P** $=$ **NP**, **NP-Complete** problems cannot be solved in polynomial time.

1.3 Next Generation Access Control (NGAC) Model

An Access Control model is a mathematical object used to model the rights a user has to an object. There are many variants of access control models, but the current widely accepted model is known as the Harrison-Ruzzo-Ullman (HRU) Model [2]. The model that we will talk about most is the Next Generation Access Control (NGAC) Model [3].

The NGAC Model uses graphs as its representation. Formally, an NGAC Model M consists of an 11-tuple

$$M = (U, UA, R, RA, R_\psi, A_U, A_R, ASC, P, V, COM)$$

where

- U is a set of users,
- UA is a set of user attributes,
- R is a set of resources,
- RA is a set of resource attributes,
- R_ψ is a set of access rights,
- A_U is a set of assignment edges,
- A_R is a set of assignment edges
- ASC is a set of (labeled) association edges,
- P is a set of (labeled) prohibition edges,
- V is a "universe" set of entities that can be in the model,
- COM is a set of commands.

We require that the sets U, UA, R, RA and R_ψ are pairwise-disjoint and $U \cup UA \cup R \cup RA \subseteq V$. The set $U \cup UA \cup R \cup RA$ forms the vertex set of the digraph G representing the state of the model. The access rights R_ψ are used as labels for edges in ASC and P, and the sets A_U, A_R, ASC and P form edges of G. We define G piece-by-piece as follows.

The sets A_U and A_R correspond to unlabeled, directed edges in G on the vertex sets $U \cup UA$ and $R \cup RA$ such that

$$A_U \subseteq (U \times UA) \cup (UA \times UA)$$
$$A_R \subseteq (RA \times R) \cup (RA \times RA).$$

It is standard (e.g. [4]) to assume that the subgraphs of G given by $(U \cup UA, A_U)$ and $(R \cup RA, A_R)$ are acyclic and to refer to them as the *user DAG* and the *resource DAG* respectively. The set ASC is a set of labeled edges satisfying $ASC \subseteq UA \times RA \times R_\psi$, where we think of $a = (ua, rsa, r)$ as an edge in G from the user attribute ua to the resource attribute rsa labeled with the access right r. The prohibition edges P satisfy $P \subseteq UA \times RA \times R_\psi$ and are also thought of as labeled edges of G in the same way.

The universe set V represents the collection of all possible users, user attributes, resources, and resource attributes that may be added to the state of the model. That is, we can think of V as a "reservoir" of vertices that do not yet exist in G but that may be added by commands. We restrict R_ψ and V to be finite.

Definition 13: Commands
A command is a state-changing function that can create or destroy edges and nodes in the model given some conditions regarding the existing model.

Each command would have a format of

command $\alpha(X_1, X_2, \ldots, X_k)$

if $cond_1$ **and** $cond_2$ **and** $\cdots$ **and** $cond_m$ **then**

op_1

op_2

$\ldots$

op_n

end if

end command

where $X_1, \ldots, X_k$ are the formal parameters of the command, $cond_i$ is a condition and op_j is one of the primitive operations detailed in Table 1.1. Each primitive operation consists of a single addition or deletion of an element of the sets U, UA, R, RA, A_U, A_R, ASC, and P. Thus, there are 16 primitive operations.

Table 1.1: List of Primitive Operations

Operation	**Conditions**	**Action**
create user u	$u \notin U \wedge u \in V$	$U \mapsto U \cup \{u\}$
create user attr. ua	$ua \notin UA \wedge ua \in V$	$UA \mapsto UA \cup \{ua\}$
create res. rs	$rs \notin R \wedge rs \in V$	$R \mapsto R \cup \{rs\}$
create res. attr. rsa	$rsa \notin RA \wedge rsa \in V$	$RA \mapsto RA \cup \{rsa\}$
create user assign. au	$au \notin A_U \wedge au \in (U \times UA) \cup (UA \times UA)$	$A_U \mapsto A_U \cup \{au\}$
create res. assign. ar	$ar \notin A_R \wedge ar \in (RA \times R) \cup (RA \times RA)$	$A_R \mapsto A_R \cup \{ar\}$
create assoc. a	$a \notin ASC \wedge a \in UA \times RA \times R_\psi$	$ASC \mapsto ASC \cup \{a\}$
create prohib. p	$p \notin P \wedge p \in UA \times RA \times R_\psi$	$P \mapsto P \cup \{p\}$
destroy user u	$u \in U$	$U \mapsto U \setminus \{u\}$
destroy user attr. ua	$ua \in UA$	$UA \mapsto UA \setminus \{ua\}$
destroy res. rs	$rs \in R$	$R \mapsto R \setminus \{rs\}$
destroy res. attr. rsa	$rsa \in RA$	$RA \mapsto RA \setminus \{rsa\}$
destroy user assign. au	$au \in A_U$	$A_U \mapsto A_U \setminus \{au\}$
destroy res. assign. ar	$ar \in A_R$	$A_R \mapsto A_R \setminus \{ar\}$
destroy assoc. a	$a \in ASC$	$ASC \mapsto ASC \setminus \{a\}$
destroy prohib. p	$p \in P$	$P \mapsto P \setminus \{p\}$

There are multiple assumptions we have to make, to make the problem simpler.

1. All commands are mono-operational, which means only one action can be done at a time (i.e. $n = 1$).

2. All edges and nodes must have a destroy function that is non-conditional.

3. Conditions must be in form "if objects are NOT assigned to attributes".

4. Conditions only exist to enforce Mutual Exclusiveness, which implies conditions only appearing in edge creating commands.

Figure 1.3 is an example of an NGAC Model, without the command set. In this example, we have that Bob is able write on Contracts. We also have that Alice is not allowed to write on Contracts.

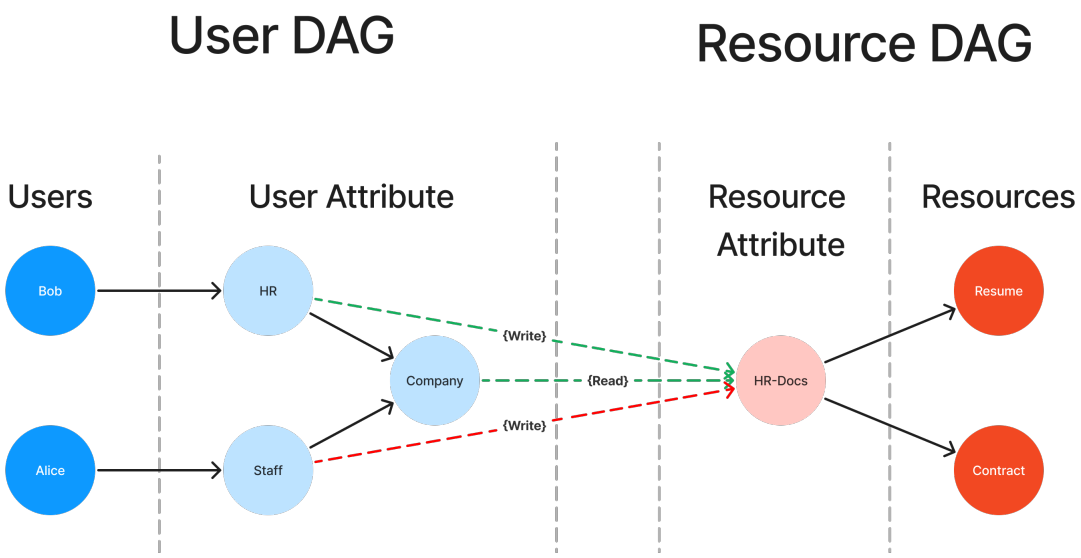

Figure 1.3: An Example of an NGAC Model

Definition 14: Access

Given a state digraph, we define for each $r \in R_\psi$ the *access relation* to be binary relation $\rightarrow^r$ (formally, a subset of $U \times R$) such that $u \rightarrow^r rs$ if and only if there exists a path in the state digraph from u to rs that goes through an association edge labeled with r.

Definition 15: Leak Safety

Given an NGAC Model with initial state G_0, user u, a resource rs, and a right r. The model is (u, rs, r)-leak-unsafe iff after going a sequence of commands of length $k \in \mathbb{N}$ such that

$$\text{Non-Access}((u, rs, r), G_0) \wedge \text{Access}((u, rs, r), G_k)$$

With the leak safety defined above, we can now define the safety problem.

Definition 16: Safety Problem

Given an NGAC Model, and a set of commands, are all possible tuples $(u, rs, r) \in U \times R \times R_\Psi$ safe from leaks?

Definition 17: Seperation of Duty

Separation of Duty (SOD) are sets of constraints, represented by pairs of attributes, that does not allow a user to be given rights to both attributes in the each pair [5].

For example, a person cannot be a student and a faculty at the same time.

SOD constraints can be defined explicitly, or they can be extracted from the conditional statements in the commands. A command can come with conditional statements, specifically conditions that does not allow giving rights to an attribute if a user already has rights to another attribute. For example, a command could look like : if not a student, then user can be a faculty. Semantics aside, this is exactly an example of an implicit SOD constraint.

For this problem, we will assume that all SOD constraints are associated with a conditional command, and the other way around. From this, we can see that most of the complexity in this problem comes from the command set.

Another convenient way to define the SOD constraints is through a constraint graph. The vertex set of the constraint graph is the set of all edges in the NGAC model graph. The edges in the constraint are the set of pairs of edges in the NGAC graph that are constrained by SOD. Since each edge in the NGAC graph is an assignment of attribute, then the SOD constraints just makes sure that the assignments is not to both attributes in the SOD constraints.

Separation of Duty is the bridge that connects the safety problem into directed acyclic constrained connectivity problem. Without SOD, the safety problem is simply a regular path-finding problem. The constraint that does not allow pairs of edges, which are pairs of assignments, to exist at the same time is what makes both problems complex.

1.4 Directed Acyclic Constrained Connectivity Problem

Directed Acyclic Constrained connectivity is a modification of the classic ST-CON problem. We will introduce some constraints which will turn this problem into a subgraph search and path-finding problem. First, we will define the constraints, which are represented by a graph.

Definition 18: Constraint Graphs
Given a Directed Acylic Graph $G = (V, E)$, we define $C = (E(G), E')$, where $E' \subseteq \binom{E(G)}{2}$, to be our constraint graph such that $(e_1, e_2) \in E' \iff e_1$ and e_2 cannot exist at the same time

Unless $E' = \emptyset$, G always violates this constraint. The only *possible* valid graphs must be subgraphs of G. However, not all subgraphs are valid.

Definition 19: Family of Valid Graphs

$\mathcal{G}_C$ is a family of valid graphs under a Constraint graph C if and only if each graph in $\mathcal{G}_C$ follows the Constraint graph C. In other words,

$$\mathcal{G}_C = \{\, \Gamma = (V, E') \mid E' \subset E \text{ and } \forall\, e_1, e_2 \in E',\ e_1 \not\sim e_2 \text{ in } C \,\}$$

Definition 20: Directed Acyclic Constrained Connectivity Problem Statement

Given a Directed Acyclic Graph $G = (V, E)$, a source vertex $s \in V$, a target vertex $t \in V$, and a constraint graph C. Does there exist a graph $\Gamma \in \mathcal{G}_C$ such that there is a path from s to t in Γ?

As said before, DACC is a modification of the classic STCON Problem. Instead of simply finding an $s-t$ path in a given graph, we have to find a subgraph that does not violate the constraints given that connects s and t. In fact, if $E(C)$, which are the constraints, is empty, then DACC is exactly the same as STCON. Since the constraints are represented through a graph, this problem becomes naturally combinatorial. We will prove later that it is connected to Independent Sets.

1.5 Related Works

This paper [6] talks about a problem similar to directed acyclic constrained connectivity. Instead of having the constraints be on edges not allowed to exist at the same time, they constrained nodes to not be on the same path from s to t. Both problems are trying to find a path with constraints. Our algorithm to solve directed acyclic constrained connectivity searches through subgraphs that connects s and t instead of looking through all (s, t) paths that satisfies their constraints. Our proof of NP-Completeness is inspired by this paper's proof. They reduced 3-SAT to their problem, and we reduced 3-COL to directed acyclic constrained connectivity. We were inspired by their sequential assignment method for the reduction.

These papers [2, 7] define the complexity of the safety problem for the predecessor model of the NGAC Model called the Harrison-Ruzzo-Ullman (HRU) Model. Instead of using a graph to model access rights like the NGAC Model, the HRU Model uses a matrix with users as rows and objects as columns. Each cell lists what rights a user has to an object. The results for thr HRU Model are that the safety problem (for the mono-operational case) is NP-Complete. In this thesis, we show that is also true for the NGAC Model.

The Moon and Moser theorem [8] is an old and popular result in extremal combinatorics. They gave an upper bound for the number of maximal cliques in a graph, which we will incorporate as an upper bound for directed acyclic constrained connectivity problem also. The papers [9–11] are a critical part to our algorithm for solving constrained connectivity. They show that enumerating maximal cliques or maximal independent sets can be done with an output sensitive algorithm with a polynomial time delay. Output sensitive means that runtime of the algorithm depends on the size of the set we are trying to enumerate. A polynomial time delay is the computation time needed in the algorithm per one element in the output set. Therefore, the runtime for an output sensitive algorithm with a polynomial time delay is the size of the output times the polynomial time delay. This means that there are some cases where the algorithm is polynomial time. Enumerating in-

dependent sets dominates the running time of our algorithm to solve directed acyclic constrained connectivity.

Chapter 2

Directed Acyclic Constrained Connectivity Problem

In this chapter, we will determine which complexity class Directed AcyclicConstrained Connectivity fall under. We will show that Directed Acyclic Constrained Connectivity is **NP-Complete**. We will also show some relations of famous graph theory problems to Directed Acyclic Constrained Connectivity. We will then show our state-of-the-art algorithms for Directed Acyclic Constrained Connectivity.

2.1 Computational Complexity

We will first prove that Directed Acyclic Constrained Connectivity is in **NP**. To prove that a problem is in **NP**, we must show that

1. There exists a certificate of polynomial length with respect to the input length.
2. The certificate is verifiable in polynomial time.

For inputs of the problem that returns a Yes, a certificate is proof that those input will actually return a Yes. For example, we can think about the Independent Sets Problem : Given a graph $G = (V, E)$ and a natural number k, does there exist an independent set of size k in G? Say we have a graph Γ and $k = 2$, that will return a Yes to this problem. Then, the certificate for this input would be the independent set of size 2 of Γ. The verification process will make sure that the certificate is a valid independent set of Γ, which one can easily do in quadratic time.

For Directed Acyclic Constrained Connectivity, our certificate will be a subgraph of G, let Γ be the certificate. Then, we will have to verify that

1. The certificate is an actual subgraph of G.
2. The certificate does not violate the constraints.
3. The certificate contains a path from s to t.

Finally, we also have to ensure that the verification process runs in polynomial time.

Algorithm 1 Subgraph Verification

```
1: procedure VERIFY-SUBGRAPH(Γ = (V', E'), G = (V, E))
2:
3:     for e ∈ E' do
4:         if e ∉ E then
5:             return False
6:         end if
7:     end for
8:
9:     return True
10: end procedure
```

Lemma 1. Algorithm 1 verifies that Γ is a subgraph of G.

Proof. Assuming that the input is not of malicious type, to check if Γ is a subgraph of G, we just have to make sure that the edge set of Γ is a subset of the edge set of G. A simple for loop check that all elements in $E(\Gamma)$ is also in $E(G)$ will take care of that. □

Algorithm 2 Constraint Verification

```
1: procedure VERIFY-CONSTRAINT(Γ, C)
2:
3:     for e_1, e_2 ∈ (E(Γ) choose 2) do
4:         if (e_1, e_2) ∈ E(C) then
5:             return False
6:         end if
7:     end for
8:
9:     return True
10: end procedure
```

Lemma 2. Algorithm 2 verifies that Γ does not violate C.

Proof. Let $G = (V, E)$ be our graph, Γ be a subgraph of G verified through Algorithm 1, and C be our constraint graph.

The constraints requires that pairs of edges connected in C, cannot exist at the same time. Therefore, if all pairs of edges of Γ is not in $E(C)$, this does not violate the constraint.

The for loop iterates through all possible pairs of edges of Γ. If at some point the loop fails, that means there must exist a pair of edges $(e_1, e_2) \in \binom{E(\Gamma)}{2}$ that violates the constraints of C that tells us that e_1 and e_2 cannot exist at the same time. Then, we can directly return False since there exists one pair of edges that violates the constraints already.

If the loop succeeds, that means no pairs of edges violates the constraints, which means that Γ does not violate C. □

Lemma 3. We can verify that Γ has an $s - t$ path.

Proof. Let $G = (V, E)$ be our graph, Γ be a subgraph of G, that does not violate C, verified through Algorithm 1 and 2, and $s, t \in V$.

Since Γ is unweighted, we can simply use breadth first search [12] to find a path from s to t. □

Lemma 4. The certificate verification process is done in polynomial time.

Proof. Algorithm 1 is a simple for loop that runs $|E(\Gamma)|$ time. For an $n-$vertex graph, the number of edges is at most $O(n^2)$, thus algorithm 1 runs in $O(n^2)$ time.

Algorithm 2 is also a simple for loop that runs $\binom{|E(\Gamma)|}{2}$ time. As said before $|E| \leq O(n^2)$, thus $\binom{O(n^2)}{2} = O(n^4)$. So, algorithm 2 runs in $O(n^4)$ time.

The final step of our verification process uses BFS. BFS is known to have a runtime of $O(|V|+|E|)$ [12]. Thus, this step can be done in $O(n^2)$ time.

Finally, the total runtime of the verification process takes $O(n^4)$ time. □

Theorem 5. DIRECTED ACYCLIC CONSTRAINED CONNECTIVITY $\in$ **NP**

Proof. Let Γ be the certificate for an input G, C, s, t.
Through Algorithms 1, 2, and BFS, we can verify that Γ is a valid certificate. The runtime of the combined algorithms will be $O(n^4)$ which means that it is verifiable in polynomial time.
Thus since for every input, there exists a polynomial time verifiable certificate, Directed Acyclic Constrained Connectivity is in **NP**. □

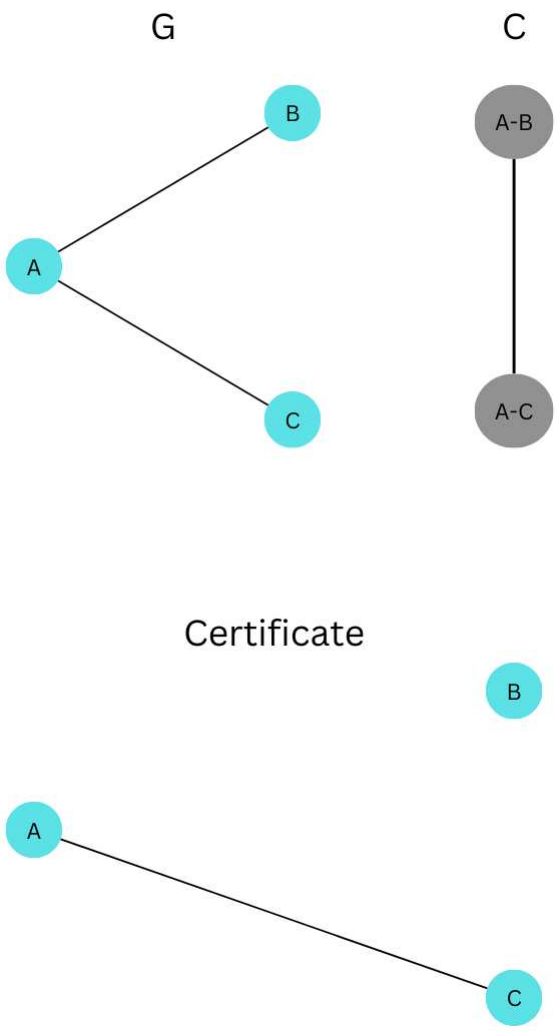

Figure 2.1: An Example of a Certificate for a Given Input

Figure 2.1 is a simple example of finding a certificate for a given input. We are given a graph G, which is a P_3 graph, and a constraint graph C which disallows the simultaneous existence of the edge AB and AC, a source node A, and a target node C. The certificate for this input will be the graph $(\{A, B, C\}, \{(A, C)\})$. The certificate is a subgraph which does not violate the constraints from C, and has a path from A to C.

Now, we want to show that Directed Acyclic Constrained Connectivity is in **NP-Hard**. To prove that a problem is **NP-Hard**, we need the following steps

1. Pick a (proven) **NP-Hard** Problem.
2. Construct a function that maps inputs of the known **NP-Hard** problem, named A, to the inputs of your problem, named B, such for all $a \in$ inputs of A, a returns "Yes" in problem A if and only if $f(a)$ return "Yes" in problem B.
3. Prove that the construction runs in polynomial time.

This process is called reduction. If successful, we can say that the known **NP-Hard** Problem is polynomial-time reducible to your problem [1].

The known **NP-Hard** Problem that we will pick is the 3-Coloring Problem [13]. We will show that 3-COL is polynomial-time reducible to Directed Acyclic Constrained Connectivity through a series of construction algorithms. We will first explain what the algorithm is doing, and then show that it satisfies step 2.

Let $G = (V, E)$ be a graph, $Q = \{R, G, B\}$ be the set of Colors. We need to find a graph Γ, a constraint graph C, a source node s, and a target node t to feed into our problem. We will first explain how to construct Γ for a given G.

Algorithm 3 Graph Construction

procedure CONSTRUCT-GRAPH(V)
    $\Gamma \leftarrow (\{s, t\} \cup V, \{(s, V_1)\})$

    for $i \in [n]$ **do**
        for $q \in \{R, G, B\}$ **do**
            $V(\Gamma) \leftarrow V(\Gamma) \cup \{q_i\}$
            $E(\Gamma) \leftarrow E(\Gamma) \cup \{(V_i, q_i)\}$

            if $i \leq n$ **then**
                $E(\Gamma) \leftarrow E(\Gamma) \cup \{(q_i, V_{i+1})\}$
            else
                $E(\Gamma) \leftarrow E(\Gamma) \cup \{(q_i, t)\}$
            end if

        end for
    end for

    return Γ
end procedure

Given a graph G with a vertex set V, we can construct Γ using Algorithm 3. First, let V be ordered in any arbitrary way. Then, we want to construct a fully connected $1, 3, 1, 3, \cdots, 1, 3$ "network" with $2n$ layers. At each even layer $2i$ for $i \in [0, n-1]$, which will be the single vertexed layer, label the vertex as V_i. At each of the odd layer $2k + 1$ for $k \in [0, n - 1]$, label each vertex with a color plus the index k, only to make sure each color vertices are distinct per layer. Create a new vertex S and connect to V_0. Create a new vertex T and connect the last layer consisting of R_n, G_n, B_n to it. Make sure that the edges are directed. Figure 2.2 is an example of Γ.

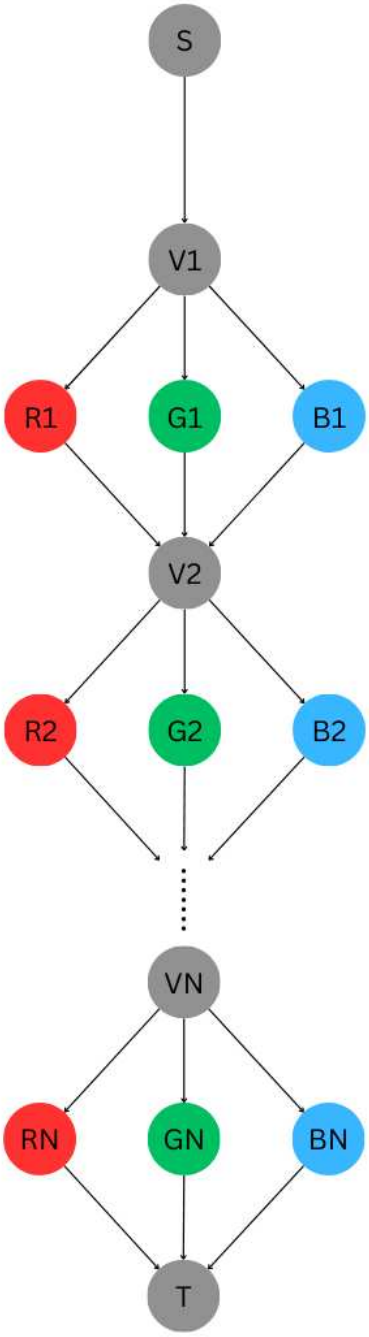

Figure 2.2: An Example of Γ

Now, we want to construct a constraint graph C when given a graph G. To find C, we must have a vertex set and an edge set. Since the vertex set is simply the set of edges of G, then we do not need to do anything more for the vertex set. To find the edge set, we need to rely on the structure of G and constraints of coloring. We will break up the constraints into two parts for easy intuition.

Before we construct the constraints, we will first explain the intuition on why this specific set of constraint will work. For every vertex, it's connected by its respective choice edges to three colors. For the first part of the constraints, we want to make sure that only one of the three choice edges must exist at a time. Once each vertex has picked an edge (as they are not allowed to pick more than one), that edge can be thought of as an assignment for that vertex to that color. We can call this set of constraints as single assignment constraint set. The name implies that the constraints will make sure that each vertex can only pick at most one color.

Now, we want to make the second part of the constraints. Once a vertex at iteration i picks a color, we must make sure that any vertex onwards that is a neighbor of the $i-$th vertex does not pick the same color. We can replicate this by using a similar method from before. Let u be the $i-$th vertex. An edge (or assignment) from u to a color, cannot exist at the same time as an edge connecting any of u's neighbors to the same color. We will have to do this for all vertices, and all possible colors (three only). We can call this set of constraints as coloring constraint set. The name implies that the constraints will make sure that a vertex and its neighbors cannot pick the same color.

One can think of this process as sequential assignment. We start with the first vertex and pick a color. Once the first vertex picked a color, the first constraint makes sure that it will be the one and only color picked, and the second constraint makes sure that none of its neighbors picked the same color in the future when it is their turn. Notice that if the current vertex has zero choices of edge to pick, that means that it is constrained by its past neighbors. In graph coloring terms, lack of choice edge is equivalent to lack of colors to choose from since its neighbors have already used it.

Part 1: Single Assignment Constraint Set.

For the first constraint set, we have to make sure that every set of choice edges are pairwise constrained to exist at the same time. This will ensure that only one choice edge can exist at a time. From a constraint graph point of view, this will result in a triangle between of each cluster of choice edges. For a vertex at iteration i, the set would look like the following

$$\{((v_i, R_i), (v_i, B_i)),\ ((v_i, R_i), (v_i, G_i)),\ ((v_i, B_i), (v_i, G_i))\}$$

Remember that each edge / pair of edges means that the pair cannot exist at the same time. Then, we have to do this for every vertex. Visually, this will lead to having a constraint graph with unions of disjoint triangles, and some singleton nodes.

Part 2: Coloring Constraint Set.

For the second constraint set, we have to make sure that a vertex does not pick a same color as a neighbor. We will do that by pairing up every choice edge from a vertex u to a color c, with from all of u's neighbors to c. We also have to do this for all possible c. Say we have a vertex v_i, that is *only* connected to v_k, then the set would look like

$$\bigcup_{v_k \in N(v_i)} \{((v_i, R_i), (v_k, R_k)),\ ((v_i, B_i), (v_k, B_k)),\ ((v_i, G_i), (v_k, G_k))\}$$

It is clear that this set heavily relies on the neighborhood structure of the graph. We have to do this process for each vertex, each color, and all neighbors of the current vertex. This constraint will ensure that a commitment of assignment from a vertex to a color forbids its neighbors to pick the same color.

Algorithm 4 Single Assignment Constraint Construction

procedure CONSTRUCT-ASSIGNMENT-CONSTRAINT(V)
    $C_1 \leftarrow \{\}$
    $Q \leftarrow \{R, G, B\}$

    for $v \in V$ **do**
        for $c_1, c_2 \in \binom{Q}{2}$ **do**
            $C_1 \leftarrow C_1 \cup \{((v, c_1), (v, c_2))\}$
        end for
    end for

    return C_1
end procedure

Algorithm 5 Coloring Constraint Construction

procedure CONSTRUCT-COLORING-CONSTRAINT(G)
    $C_2 \leftarrow \{\}$
    $Q \leftarrow \{R, G, B\}$

    for $v \in V(G)$ **do**
        for $u \in N(v)$ **do**
            for $c \in Q$ **do**
                $C_2 \leftarrow C_2 \cup \{((v, c), (u, c))\}$
            end for
        end for
    end for

    return C_2
end procedure

Finally the edge set will be the union of the single assignment constraint set and the coloring constraint set. Now that we have the vertex set and the edge set, we can say that

$$C = (E(G),\, C_1 \cup C_2)$$

Finally, we can fix $s = S$ and $t = T$ where $S, T \in V(\Gamma)$

We have now finished with constructing Γ, C, s, t. To be exact, we can create a new algorithm named f combining algorithms 3 - 5 with parameter G. Then,

$$f : G \to \Gamma \times C \times s \times t$$

is the function we needed in step 2 of the **NP-Hard** proof process. Now, we will show that when $f(G)$ is fed to Directed Acyclic Constrained Connectivity will return true if and only if G is $3-$Colorable.

Lemma 6. G is $3-$Colorable if and only if $f(G)$ return "Yes" in Directed Acyclic Constrained Connectivity.

Proof. Let $G = (V, E)$.

Part 1: G is $3-$Colorable implies $f(G)$ returns "Yes" in DACC.

Assume G is $3-$Colorable, and let $\varphi : V \to \{R, G, B\}$ be one of the valid color maps of G.
Let $\Gamma, C, s, t = f(G)$.
We can do the same sequential assignment process we did earlier. In order, let each vertex pick the color based on φ, then through the constraints delete the edges of Γ. The deleted edges per vertex at iteration i will be all $i-$th choice edges besides $\varphi(v_i)$ and the $\varphi(v_i)$ choice edge for all its neighbors.

At iteration i, the vertex must have at least one choice edge since, as stated before, zero choice edge means that it is constrained by its neighbors to not have any choice of color. However, since ϕ is a valid color map, this can never happen.

Once every vertex has picked its colors, every layer going from a vertex to the three color options must only be connected by one choice edge. Since every vertex has a choice edge, all layers are connected, therefore there exists a path from s to t. Therefore, there exists a subgraph of Γ that does not violate C, obtained through sequential assignment, that has a path from s to t. Thus, $f(G)$ returns a "Yes" in DACC.

Part 2: $f(G)$ returns "Yes" in DACC implies G is $3-$Colorable.
We will be using proof by contrapositive for this part.
Assume G is not $3-$Colorable. We need to show that $f(G)$ will return "No" in DACC.

If G is not $3-$Colorable, then there does not exist any valid color maps. Therefore, if we do the sequential assignment, any choice of color will lead to at least one vertex to have zero choice edges when it's their turn to pick. This would mean that at some point, a vertex cannot reach the next layer due to lack of choice edges. Thus, Γ will be split into two, one component consists everything left of the vertex, and everything right of its color options. This would mean that there is now no path from s to t.

Since every possible assignment will always lead to having zero choice edges at some point, $f(G)$ will always return "No" in DACC.

Therefore, f is a valid reduction map. □

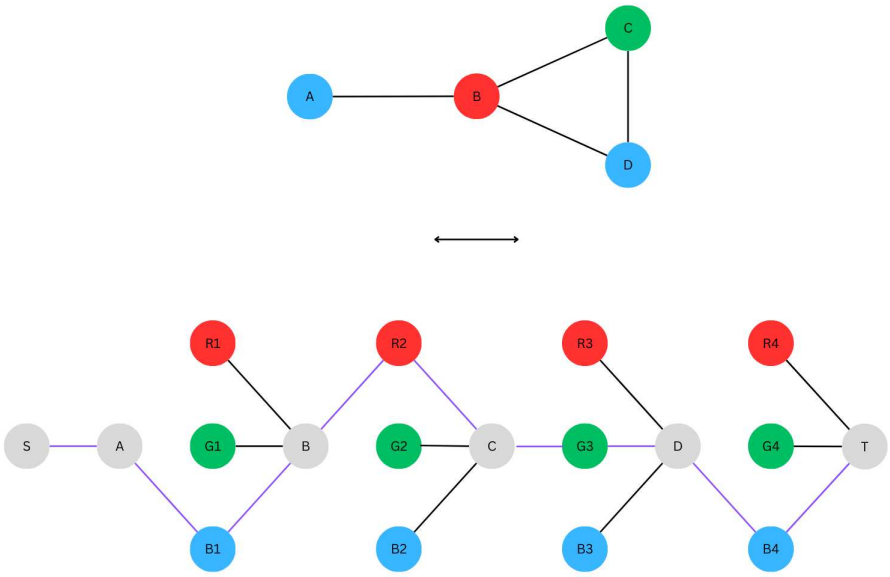

Figure 2.3: 3-COL to DACC Reduction Example

Lemma 7. Construction of Γ and C can be done in polynomial time.

Proof. As said previously, the construction combines algorithms 3 - 5. Since all the operation in algorithm 3 - 5 takes constant time, then the runtime is solely based on the nested loops. Algorithm 3 will run in $O(n)$ time. Algorithm 4 will run in $O(n)$ time. For algorithm 5, the second loop has a maximum iteration of $n - 1$ since a vertex can have at most that amount of neighbors, thus the runtime for is $O(n^2)$.

Therefore, the total runtime for the construction is $O(n^2)$. □

This concludes step 3 of our reduction process. We have now shown that a construction of a reduction map exists and such construction takes polynomial time. Figure 2.3 is a great example of how the reduction construction and process works.

Now that we have finished all three steps, we can prove that Directed Acyclic Constrained Connectivity is **NP-Hard**.

Theorem 8. DIRECTED ACYCLIC CONSTRAINED CONNECTIVITY $\in$ **NP-Hard**

Proof. Let $G = (V, E)$ be the graph we are trying to color.

Let

$$f : G \to \Gamma \times C \times s \times t$$

be the reduction mapping function proven in Lemma 6.

Through f, and Lemma 6 and 7, we know that 3-COL is polynomial-time reducible to Directed Acyclic Constrained Connectivity in $O(n^2)$ time. Since 3-COL is a known **NP-Hard** Problem, we can conclude that Directed Acyclic Constrained Connectivity is in **NP-Hard**. □

3-COL is a well known and very interesting problem both in Graph Theory and also Complexity theory due to its relation with 3-SAT. Since we are able to reduce 3-COL to Directed Acyclic Constrained Connectivity, this would mean that any progress made in our problem, is also progress for 3-COL. That would make our problem very important and interesting to work on.

Theorem 9. DIRECTED ACYCLIC CONSTRAINED CONNECTIVITY $\in$ **NP-Complete**.

Proof. Through theorems 5 and 8, we know that Directed Acyclic Constrained Connectivity is in **NP** and **NP-Hard**. Thus, constrained connectivity is in **NP-Complete**. □

Directed Acyclic Constrained connectivity now joins a prestigious class of problems. **NP-Complete** problems are known to have no efficient algorithms, unless **P = NP**. As said previously, any progress made on this problem will help the field of Computer Science and Mathematics as a whole.

2.2 Independent Sets

In this section, we will discuss how Directed Acyclic Constrained Connectivity is related to independent sets. We will also discuss known results from independent sets that will impact Directed Acyclic Constrained Connectivity too.

Lemma 10. For a given graph $G = (V, E)$, and a constraint graph $C = (E(G), E')$, there exists a bijection from $\mathcal{I}(C)$ to $\mathcal{G}_C$, where $\mathcal{I}(C)$ is the set of all independent sets of C.

Proof. Let $G = (V, E), C = (E, F), \mathcal{G}_C$ be the family of valid graphs of G under C, and $\mathcal{I}(C)$ be the set of all independent sets of C.

Recall that if a pair of edges $e_1, e_2 \in E$ are connected in C, they are not allowed to exist at the same time. Thus, if we were to build a valid graph, starting from an empty edge set, we are only allowed to pick one of the two edges. An independent set in C is a set of edges (in G) that is not connected in C. That would mean that all independent sets in C, when put as the edge set of the graph we are trying to build, does not violate any constraints.

Let $I \in \mathcal{I}(C)$, and $f : \mathcal{I}(C) \to \mathcal{G}_C$ such that $f(I) = (E, I)$. From the above reasoning, we know that $f(I) \in \mathcal{G}_C$.

Let $i, j \in \mathcal{I}(C)$ be distinct independent sets of C. Since i and j are distinct, that means there must exist at least one edge, without loss of generality, that is in $f(i)$ but not in $f(j)$. This means that $f(i) \neq f(j)$. Thus, by contrapositive, we have that $f(i) = f(j) \implies i = j$, which is the definition of f being injective.

Let $\Gamma \in \mathcal{G}_C$. We want to show that $E(\Gamma) \in \mathcal{I}(C)$ to prove that an inverse exists. Since all pairs of edges $e_1, e_2 \in E(\Gamma)$ cannot be connected in C, then it follows the definition of an independent set in C. Thus, $E(\Gamma) \in \mathcal{I}(C)$, which means f is surjective.

Thus, f is bijective. □

Since we are trying to find a path, adding as many edges as we can does not hurt us. Therefore, if the independent set is not maximal, it does not hurt to put more edges (of G) in the set.

Definition 21: Family of Maximal Valid Graphs
$\tilde{\mathcal{G}}_C \subseteq \mathcal{G}_C$ is a set of valid subgraphs of G under C such that for a subgraph $\Gamma \in \tilde{\mathcal{G}}_C$, $E(\Gamma)$ is a maximal independent set of C.

It is easy to see that every graph in $\mathcal{G}_C \setminus \tilde{\mathcal{G}}_C$ is a subgraph of at least one graph in $\tilde{\mathcal{G}}_C$. Therefore, we do not need to worry that a possible answer is left out by removing "non-maximal" subgraphs.

Lemma 11. $|\tilde{\mathcal{G}}_C| \leq O(1.44^m)$ where $m = |E(G)|$.

Proof. From lemma 10, we know that # MIS of C = $|\tilde{\mathcal{G}}_C|$.
Moon and Moser [8] proved that for any k-vertex graph, the maximizer of the number of MIS, is a disjoint union of triangles. Specifically, we have

$$\#\text{ MIS } \leq O(1.44^k)$$

Since $|V(C)| = |E(G)| = m$, then

$$|\tilde{\mathcal{G}}_C| = \#\text{ MIS of } C \leq O(1.44^m)$$

□

Through this connection, we can solve this problem by searching through all possible maximal independent sets of C, which equivalently searches through all possible *maximal* valid graphs, and check if there is one with a path from s to t.

2.2.1 Enumeration

In this section, we will discuss about enumerating all possible maximal independent sets of a graph. It is well known that a clique in a graph $G = (V, E)$ is an independent set in the complement graph $G' = \left(V, \binom{V}{2} \setminus E\right)$. To find the set of all maximal independent sets is equivalent to finding the set of all maximal cliques in the complement graph.

The current best algorithm to enumerate all maximal cliques in a graph is an output sensitive algorithm [9, 11]. It has a polynomial time delay of $O(M(n))$ where $M(n)$ is the time needed to multiply two $n \times n$ matrix. The current best matrix multiplication algorithm takes $O(n^{2.372})$ time [14]. Therefore, to enumarete all maximal cliques in a graph takes time $O(n^{2.372} \cdot \mathcal{C}(G))$ where $\mathcal{C}(G)$ is the number of maximal cliques of G.

Since the algorithm depends on the structure of the graph, it is also important to note that there are some class of graphs with a polynomial number of maximal cliques. A class that is useful in our application to access control are graphs with high minimum degree. Specifically, each vertex can only be disconnected to a constant number of vertices.

2.3 Algorithm

In this section, we give the state-of-the-art algorithm to solve Directed Acyclic Constrained Connectivity. Using our connection to independent sets and enumeration techniques, we can iterate through all *maximal* valid subgraphs of G and check if there is at least one that has a path from s to t.

Algorithm 6 Directed Acyclic Constrained Connectivity Solver

```
1:  procedure SOLVE-CC(G = (V, E), C, s, t)
2:      μ ← Enumerate-MIS(C)
3:      for I ∈ μ do
4:          Γ ← (V, I)
5:          P ← BFS(Γ, s, t)
6:          if P ≠ None then
7:              return True
8:          end if
9:      end for
10:     return False
11: end procedure
```

The algorithm to enumerate all MIS in line 2, is the same algorithm explained in the previous section. We will use BFS for path finding in line 5 as the graph is unweighted. The algorithm itself is pretty simple to comprehend. The runtime of this algorithm will be

$$O((|E(C)|)^{2.372} \cdot \mathcal{C}(C) + \mathcal{C}(C) \cdot (|V| + E(\Gamma)) \leq O((n^2)^{2.372} \cdot \mathcal{C}(C) + \mathcal{C}(C) \cdot (n + n^2)) = O(n^{4.744} \cdot \mathcal{C}(C))$$

Since BFS on a $n-$vertex graph takes at most $O(n^2)$ the polynomial factor from the matrix multiplication in MIS enumeration is a dominating factor. As explained in previous sections, the runtime clearly depends on the structure of the graph. There are some cases where $\mathcal{C}(C)$ is polynomial bounded, and also some cases where it is exponential.

A standard brute force algorithm would have to pick which edge, per each constraint, to ex-

ist, then check whether the current subgraph connects s and t. This algorithm will take time $O(n^2 \cdot 2^{|E(C)|}) \leq O\left(n^2 \cdot 2^{n^4}\right)$ which is a very huge gap from our algorithm.

Chapter 3

Application to Access Control

In this chapter, we will discuss how we can apply constrained connectivity to a well-known problem in access control. We will first define the *safety* problem on access control. Finally, we will show that we can apply our results for access control.

3.1 Computational Complexity

In this section, we will prove that the safety problem is **co-NP-Complete**. We will first give a formal definition of the safety problem in set theoretic language.

Definition 22: Formal Safety Problem Statement

Given an NGAC Model

$$M = (U, UA, R, RA, R_\psi, A_U, A_R, ASC, P, V, COM)$$

We define a relation on $U \times R$ with respect to a right $r \in R_\psi$ such that $u \xrightarrow{r} rs$ if and only if there exists a path from u to rs that goes through an association edge labeled r. Each command in COM can create or destroy nodes and edges in the graph which will directly impact the relation. We denote $\underset{S}{\xrightarrow{r}}$ as the relation defined above after applying the sequence of commands S.

Is it true that for all rights $r \in R_\psi$, and all sequences of commands $S \in COM^m$ for $m \in \mathbb{N}$ that

$$\underset{S}{\xrightarrow{r}} \subseteq \xrightarrow{r}$$

Lemma 12. For a Directed Acyclic Constrained Connectivity input G, C, s, t, a set of commands can be created based off C, such that any sequence of commands can create any graph in $\mathcal{G}_C$.

Proof. For each binary constraint $(e_1, e_2) \in E(C)$, we can create two commands in the form of

1. if e_1 does not exist, then can create e_2
2. if e_2 does not exist, then can create e_1

Then, we can create a command set COM which are all destroy commands of nodes and edges, and the two commands above per binary constraint. This will make sure that only one of the two exists. Any valid sequence of commands, meaning none are rejected by the conditions, will mean that it follows the constraints. Therefore, since the command set is created such that it enforced the binary constraints, all valid sequences of command will lead to a valid subgraph. □

Now, we can start building the reduction from constrained connectivity to the safety problem. Let $U = \{u\}, UA = V(G), R = \{rs\}, RA = \{rsa\}, A_U = \{(u, s)\}, A_R = \{(rsa, rs)\}, A_S = \{(t, rsa), P = \emptyset, R_\psi = \{r\}$, and COM be the command set built from above.

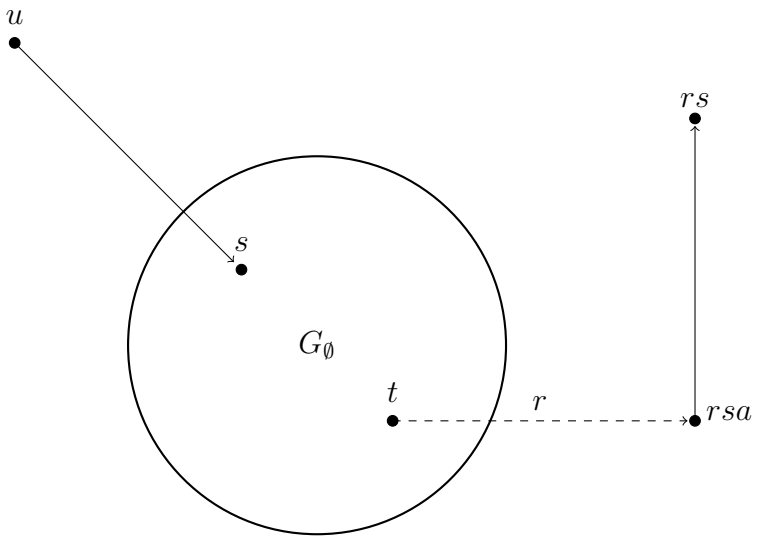

Figure 3.1: A Reduction from DACC to the Safety Problem

Figure 3.1 is how the model would look. $G_\emptyset$ is the graph G with an empty edge set. Notice that $(u, rs) \notin \xrightarrow{r}$ since there is no path from u to rs. Since all sequence of command will lead to a valid subgraph of G, then if s and t can be connected by a valid subgraph, there musts exist a sequence S such that the model will have a path from s to t. Since s and t are connnected, it is easy to see that u and rs are connected, which means that $(u, rs) \in \xrightarrow[S]{r}$.

If we think about the co-Safety Problem, which simply asks does there exist a right and a sequence of command such that $\xrightarrow[S]{r} \not\subseteq \xrightarrow{r}$ (a Yes in the safety problem is a No in the co-Safety Problem and vice versa), then we can reduce DACC to it. If DACC returns a Yes, then there must exist a valid subgraph such that s and t are connected. Therefore, there must exist a sequence of commands to make said subgraph. Which means that the co-Safety problem will return a Yes too. If DACC returns No, then there is no sequence of command to make up a new relation, which means the co-Safety Problem will return No too. Therefore, the co-Safety problem is **NP-Hard**.

It's also easy to see that the co-safety problem is **NP**. The certificate will be a right and a sequence of command, and it is easy to verify in polynomial time that the new relation is not a subset of the original relation. Therefore, the co-safety problem is **NP-Complete**. Which means that the safety problem is **co-NP-Complete**.

3.2 Solving the Safety Problem

In this section, we will discuss how to solve the safety problem using our algorithm to solve Directed Acyclic Constrained Connectivity. We will show the steps needed to transform an input for the safety problem to an input to constrained connectivity.

Let $M = (U, UA, R, RA, R_\psi, A_U, A_R, ASC, P, V, COM)$ be an input to the safety problem SP and that M satisfies our assumptions.

Let N be the length of the input M in a natural binary encoding. We will construct an input (Γ, C, s, t) for DACC in time polynomial in N, and use algorithm 6 to solve DACC for (Γ, C, s, t) in order to solve the safety problem for M.

Step 1 (Constructing Γ) :

We let Γ be the *supergraph* of M, meaning that Γ is the result of starting with the initial state digraph of M and executing, for all valid inputs, all possible commands in COM whose primitive operation is one of the form "**create** ...", *without* checking the condition(s) in the command. Since we assume that COM is finite, and V and hence the vertex set of Γ is also finite, and we know that the number of edges of Γ is bounded by polynomial in $|V|$ and $|R_\psi|$, it takes time polynomial in the size of the input M (which includes COM, R_ψ and V) to construct Γ. We also have the property that $G_t \subseteq \Gamma$ where G_t is the state digraph at any given time t.

Step 2 (Constructing C) :

We construct a constraint graph C for the supergraph Γ which enforces the conditions of the create commands in the model M. The goal is to do this in such a way that valid subgraphs of Γ correspond to possible state digraphs G_t of the model M.

We initialize an empty constraint graph $C = (E(\Gamma), \emptyset)$. By assumptions 3 and 4, each edge cre-

ation command has one parameter X corresponding to an edge of Γ and must have a conditional of the form

$$e == X \textbf{ and } cond_1 \textbf{ and } cond_2 \textbf{ and } \cdots \textbf{ and } cond_m,$$

where each $cond_i$ is of the form $e_i \notin G_{t-1}$ for some potential edge (element of $V \times V$ or $V \times V \times R_\psi$). We may assume that $e_i \in \Gamma$, else the command can never execute and we may remove it. Then for each such command c with this structure we add the edges

$$\{ee_i : i \in M\}$$

to C. This will make sure that in a valid subgraph $\Gamma' \subseteq \Gamma$, if e exists in Γ' then the edges in the conditions of c must not exist.

Lemma 13. Valid Subgraphs of Γ corresponds to reachable states of G_0.

Proof. Let $M = (U, UA, R, RA, R_\psi, A_U, A_R, ASC, P, V, COM)$ and G_0 be the initial state of M. Let Γ and C be the graphs constructed from step 1 and 2.

Let Γ' be a valid subgraph of Γ. First, we want to show that Γ' is a reachable state from G_0.

First, we delete every vertices and edges that are in G_0 but not in Γ'. By assumption 2, we always have an unconditional destroy command, so we can destroy these vertices and edges.

Now, we need to add every vertices and edges that are in Γ' but not in G_0. Since C is constructed in a way to enforce mutual exclusiveness constraints, and Γ' is a valid subgraph, there should not be any mutual exclusiveness constraints violated in Γ'. Therefore, we can add these vertices and edges without having to worry about violating mutual exclusiveness.

Therefore, we can say that Γ' is reachable from G_0.

Next, we want to show that any reachable state of M, present or future, G_t, is a valid subgraph of Γ. Suppose after running an arbitrary command sequence, we arrive at a state G_t. Again, by construction of C, following the mutual exclusiveness constraints is equivalent to following the constraints from C. Since G_t is a state that we have reached from G_0, there must not be any mutual exclusive constraint violation in G_t. Therefore, we can say that G_t does not violate C too. Finally, we can say that G_t is a valid subgraph of Γ. □

Step 3 (Testing for Access Paths) :
Now that we have constructed Γ and C, we want to loop over all possible access paths and check that, for each path that does not exist in the G_0, there is no sequence of commands S that results in the access path existing.

First, we fix a tuple $(u, rs, r) \in U \times R \times R_\psi$. Then, if $u \not\rightarrow^r rs$, we run. Let Γ^r be the subgraph of Γ obtained from Γ by deleting all association edges not labeled with r and run Algorithm 6 with input (Γ^r, C, u, rs). If the answer is "Yes", then there must exist a sequence of commands S such that $u \rightarrow_S^r rs$, which means we can return "No" for the safety problem with input M.

If all queries DACC(Γ^r, C, u, rs) in the loop above tuple return "No", then M is safe and we return "Yes" in the safety problem.

3.2.1 Running Time

Constructing Γ takes time at most

$$\mathcal{O}(|COM| \cdot |V|^2 \cdot |R_\psi|)$$

as the bulk of the construction occurs in a loop over each possible labeled edge of state the digraph in the model, where we check for a create command for that edge and add it to Γ.

Constructing C takes time at most

$$\mathcal{O}(|COM| \cdot m_{\max} \cdot |O|^2 \cdot |R_\psi|),$$

where $m_{\max} \leq \mathcal{O}(|V|^2|R_\psi|)$ is the maximum number of conditions in any command in COM

Testing for access paths takes time at most

$$|O|^2|R_\psi| \cdot \mathcal{O}(|V|^2) \cdot M(|V|^2) \cdot \mu(C)$$

because we execute for each tuple $(u, rs, r) \in V \times V \times R_\psi$ Algorithm 6 with the input (Γ^r, C, u, rs). This Γ^r has at most $|V|$ vertices and $|V|^2$ edges, and hence $\mu(C) \leq 1.45^{|V|^2}$

Therefore the total runtime is at most

$$\text{poly}(|M|) \cdot 1.45^{|V|^2},$$

where $|M|$ is the size of the input M and $|V|$ is the size of the set V of possible objects.

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[2] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, aug 1976.

[3] Joshua Roberts, Gopi Katwala, and David Ferraiolo. *Next Generation Access Control System and Process for Controlling Database Access*. National Institute of Standards and Technology, USA, 2024.

[4] Erzhuo Chen, Vladislav Dubrovenski, and Dianxiang Xu. Mutation Analysis of NGAC Policies. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 71–82, Virtual Event Spain, 2021. ACM.

[5] Vincent C Hu, Rick Kuhn, Dylan Yaga, et al. Verification and test methods for access control policies/models. *NIST Special Publication*, 800(192):800–192, 2017.

[6] H.N. Gabow, S.N. Maheshwari, and L.J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, SE-2(3):227–231, 1976.

[7] Mahesh V. Tripunitara and Ninghui Li. The foundational work of harrison-ruzzo-ullman revisited. *IEEE Transactions on Dependable and Secure Computing*, 10(1):28–39, 2013.

[8] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

[9] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[10] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[11] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004*, pages 260–272, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[14] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. *More Asymmetry Yields Faster Matrix Multiplication*, pages 2005–2039.