**T. C.**
**BAHÇEŞEHİR ÜNİVERSİTESİ**
**The Graduate School of Natural and Applied Sciences**
**Applied Mathematics**

# COUNTING AND LISTING A SPECIAL CLASS OF DIRECTED GRAPHS

**Master of Science Thesis**

**Mehmet Emin GÖNEN**

**Supervisor: Assoc. Prof. Atabey KAYGUN**

**Istanbul, 2013**

T. C.
BAHÇEŞEHİR ÜNİVERSİTESİ
The Graduate School of Natural and Applied Sciences
Applied Mathematics

| | | |
|---|---|---|
| Title of the Master's Thesis | : | Counting and Listing a Special Class of Directed Graphs |
| Name/Last Name of the Student | : | Mehmet Emin GÖNEN |
| Date of Thesis Defense | : | 15 August 2013 |

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. F. Tunç BOZBURA
Acting Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Prof. Nuri KURUOĞLU
Program Coordinator

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Commitee Members:

| | |
|---|---|
| Assoc. Prof. Atabey KAYGUN (Supervisor) | : |
| Asst. Prof. Süreyya AKYÜZ | : |
| Asst. Prof. Maksat ASHYRALIYEV | : |

# ACKNOWLEDGMENTS

First of all, I would like to send my special thanks to my thesis supervisor Assoc. Prof. Atabey Kaygun, for his great supervision, guidance and encouragement. I am also greatful to him for his trust in me. He has always supported me in bad and good times, never let me give up and helped me so much to keep my motivation.

I wish to express my sincere thanks to Prof. Nuri Kuruoğlu, the coordinator of master program. If he did not accept me to master program, I did not finish this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Asst. Prof. Süreyya Akyüz and Asst. Prof. Maksat Ashyraliyev, for their encouragement and insightful comments.

My very specific thanks go to Gamze Akgun, who was always there for me, making the distance shorter. And for many many other things that cannot be listed here... Her encouragement and constant support, together with her endless patience, I was able to overcome many difficulties during this study and also in general.

The huge thanks go to the my friends for tea time talk. Especially Deniz Topuz, Kübra Çetin, Nurdan Kuru, Sena Cüre, Ayşegül Aydınoğlu and Isıl Türkan.

And last but not least, I would like to thank my family for the unconditional love and support throughout my life. Without them nothing would be possible.

Finally, I am greatful to TUBITAK-BIDEB for the financial support througout my undergraduate and master studies.

15 August 2013 Mehmet Emin GÖNEN

# ABSTRACT

COUNTING AND LISTING A SPECIAL CLASS OF DIRECTED GRAPHS

Gönen, Mehmet Emin

Applied Mathematics
Supervisor: Assoc. Prof. Atabey KAYGUN

August 2013, 50 Pages

In this thesis, we count and list a special class of directed graphs. We consider directed graphs which are transitively reduced and do not contain cycles. These directed graphs have also unique sources and unique sinks. In the text, we called such directed graphs as "admissible digraphs". Such directed graphs find applications in bioinformatics and network flow theory. We constructed explicit algorithms to count and list admissible digraphs with a specific number of vertices. Our counting algorithm is not exact, it gives us an upper bound on the number of admissible digraphs with a certain number of vertices. On the other hand, our listing algorithm is exact and list all admissible digraphs of certain vertex set size.

**Keywords:** Graph Theory, combinatorics, network flow, bioinformatics, algorithms

# ÖZET

YÖNLENDİRİLMİŞ ÇİZGELERİN ÖZEL BİR SINIFININ SAYILMASI VE LİSTELENMESİ

Gönen, Mehmet Emin

Uygulamalı Matematik
Tez Danışmanı: Doç. Dr. Atabey KAYGUN

Ağustos 2013, 50 Sayfa

Biz bu tezde yönlendirilmiş çizgelerin özel bir sınıfını saymaya ve listelemeye çalıştık. Göz önünde bulundurduğumuz bu yönlendirilmiş çizgeler geçişli indirgenmişlerdir ve çevre içermemektedirler. Ayrıca bu yönlendirilmiş çizgeler tek giriş düğümü ve tek çıkış düğümüne sahiptirler. Bahsettiğimiz yönlendirilmiş çizgeleri metinde "geçerli yönlendirilmiş çizgeler" olarak adlandırdık. Bunların biyoinformatik ve ağ akışı kuramı alanlarında uygulamaları bulunmaktadır. Düğüm sayıları belirli olan geçerli yönlendirilmiş çizgeleri saymak ve listelemek için aşikar algoritmalar oluşturduk. Sayma algoritmamız bize kesin sonuç değil, düğüm sayıları belli olan geçerli yönlendirilmiş çizgelerin sayıları üstünden üst sınır vermektedir. Buna karşılık, liste algoritmamız bize kesin sonuç vermektedir ve bütün geçerli yönlendirilmiş çizgeleri düğüm kümesinin boyutu üzerinden listelemektedir.

**Anahtar Kelimeler:** Çizge Kuramı, Kombinatorik, Ağ akışı, Biyoinformatik, Algoritmalar

# TABLE OF CONTENTS

# 1. INTRODUCTION

In this study, we count and list a special set of directed graphs. Our directed graphs contain finitely many vertices and edges. They have the following properties:

1. They are transitively reduced
2. They do not contain cycles
3. They have unique sources and unique sinks

We called our graphs as "admissible digraphs". We build up algorithms for counting and listing such graphs. Our algorithms use two specific composition operations which we called "vertical" (Definition 3.8) and "horizontal" (Definition 3.5) composition.

The literature for listing directed graphs is rather meek on the subject. On the other hand, the literature for counting is quite deep (1), (2), (3), (6), (8). However, most of these studies do their counting using generating functions (Especially (4), (7), (9)). But we do not follow this route in this study.

In the paper (9), Robinson counts digraphs with a source and a sink. His strategy is to enumarate configurations in which there is a designated out-component and a seperate designated in-component. He uses generating functions for counting which he calles $G(x)$ and $C(x)$. $C(x)$ is an exponential generating function for the set of labeled strongly connected digraphs and $G(x)$ is an exponential geenerating function for the set of all labeled digraphs. The number of such graphs is exponential ingrowth so is ours.

In the paper (7), Liskovec study a similar subject. He gives two new formulas for the number of rooted digraphs with a source. One of the formulas contains the sum of the cyclic indexes of the automorphism groups of connected rooted graphs. The paper also contains the numerical values of rooted digraphs with a source on $n$ vertices.

In the book (4), Harary and Palmer enumarate different types of graphs. Blocks and acyclic digraphs are class of graphs which contain admissible digraphs. A Block is a graph which is connected, nontrivial and has no articulations. In our thesis, we call such graph as 1-connected digraphs. 1-connected digraphs have no articulations. Harry and

Palmer enumarate blocks by using generating functions (on page 10). This function is also exponential. In another chapter of their book (on page 19), they count acyclic digraphs. They use recursive formula

$$a_p = \sum_{k=1}^{p} a_{p,k}$$

where $a_p$ is the number of labeled acyclic digraphs with $p$ vertices and $a_{p,k}$ is the number of labeled acyclic digraph with $p$ vertices which have exactly $k \geq 1$ vertices of in-degree zero. While we are generating admissible digraphs, we use a similar recursive formula (on page 19).

Studies which we mentioned above are about counting graphs. There are also some studies about composition and decomposition of graphs. In the paper (1), Cunningham and Edmans defined such decompositions on graphs. There is a similarity between their decomposition and our decomposition. They defined non-separable graphs (on page 734). According to definition, we realized that a graph is 1-connected if and only if it is non-separable. 1-connected digraphs are prime with recpect to horizontal decomposition (Proposition 3.6). But 1-connected digraphs are vertically decomposible. Therefore horizontal decomposition and decomposition in that study have similarities. They also defined prime graphs but our definition of prime graphs (Definition 3.19) is different from their definition (on page 735).

Finally, in the paper (3), Gessel uses digraph decomposition which are defined quite different from our definition of decomposition. In Gessel's paper, a simple decomposition for graphs yield generating fuctions for counting graphs by edges and connected components. In their decomposition, they choose a vertex and remove it and its incident edges. By applying this decomposition to connected graphs, they recover some known formulas for counting connected graphs by edges and for counting trees by inversion. And they get Tutte polynomial of the complete graphs by using such decomposition.

Graph decompositions are also closely related to graph searching problems. In the paper (5), Kreutzer and Ordyniak said that an important concept in the theory of graph searching games is monotonicity. The importance of monotonicity in the context of graph decompositions results from the observation that many decompositions, like tree- and path-decompositions, can be defined in terms of monotone winning strategies for the searcher (Definitions on page 4689).

Admissible digraphs are algebraic objects. The set of isomorphism class of admissible digraphs with horizontal composition is monoid. Here, our identity element is single vertex without edge. However, if we choose the operation as vertical composition then there is no identity element. Hence, the set of isomorphism class is semi-group not monoid. Our prime elements are 1-connected admissible digraphs and admissible digraphs with articulations. 1-connected admissible digraphs are prime with respect to horizontal decomposition and admissible digraphs with articulations are prime with recpect to vertical decomposition. We can decompose admissible digraphs along articulations. Using horizontal and vertical compositions, we find recursive formulas and we adapt these formulas to our algorithms. We developed explicit counting algorithms without appealing to generating functions in this thesis (Chapter 4). However, our counting algorithms give us an upper bound not an exact result. As we explaine in Chapter 4 vertical composition operation is commutative. Therefore there is a double counting problem in our counting algorithms. We defined graph ordering (Definition 5.2) to overcome this double counting problem. We also developed explicit listing algorithms using graph ordering as well (Chapter 6).

# 2. PRELIMINARY DEFINITIONS

**Definition 2.1.** *A* directed graph *(or a* digraph *in short) is a pair* $(V,E)$ *where* $V$ *is a set and* $E \subseteq (V \times V) \setminus \Delta(V)$ *where* $\Delta(V) = \{(v,v) |\ v \in V\}$. *The elements of* $V$ *are called* vertices *and the elements of* $E$ *are called* edges. *The opposite graph of a graph* $G = (V,E)$ *is a pair* $G^{op} = (V,E^{op})$ *where*

$$E^{op} = \{(v,v') \in V \times V |\ (v',v) \in E\}$$

*Let* $U \subseteq V$ *be a subset. The* full subgraph of $G$ on $U$ *is the graph*

$$G(U) := (U,\ (U \times U) \cap E)$$

**Definition 2.2.** *Let* $G = (V,E)$ *be a digraph. A vertex* $v \in V$ *is called a* source *if there exists* $w \in V$ *such that* $(v,w) \in E$ *and there is no* $v' \in V$ *with the property that* $(v',v) \in E$. *A vertex* $v \in V$ *is called a sink if* $v$ *is a source in* $G^{op}$.

**Definition 2.3.** *Let* $G = (V,E)$ *be a digraph. A sequence of vertices* $(v_0,\ldots,v_n)$ *is called a path if* $(v_i,v_{i+1}) \in E$ *for every* $i = 0,\ldots,n-1$. *A path* $(v_0,\ldots,v_n)$ *is called* a cycle *if* $v_0 = v_n$.

**Definition 2.4.** *A digraph* $G = (V,E)$ *is called* reduced digraph *if there is a path from* $v_i$ *to* $v_j$ *where* $v_i, v_j \in V$ *then* $(v_i,v_j) \notin E$.

## 2.1 HASSE SUBGRAPHS AND TRANSITIVE CLOUSURE

**Definition 2.5.** *Assume* $G = (V,E)$ *is a directed graph and* $H$ *is a subgraph of* $G$. *A subgraph* $H$ *is called* Hasse subgraph *of* $G$ *if*

1. *Vertex set of* $H$ *is equal to vertex set of* $G$.
2. *Let* $(a,b)$ *is an edge in* $H$. *If* $(a,b)$ *is removed from* $H$ *then there is no path in* $H$ *connecting* $a$ *to* $b$.
3. *Let* $a,b \in V$. *There is a path from* $a$ *to* $b$ *in* $H$ *if and only if there is a path from* $a$ *to* $b$ *in* $G$.

**Algorithm 1** An algorithm finding a Hasse subgraph of directed graphs.

```
function HASSE(G)
    for all x in edge list of G do
        delete the edge x from G
        u ← shortest path from the source of x to the target of x
        if the length of u is equal to 0 then
            add the edge x to G
        end if
    end for
    return G
end function
```

**Definition 2.6.** *Let $G = (V,E)$ be a directed graph. $cl(G)$ is called the* transitive closure *of $G$ if*

1. *Vertex set of $cl(G)$ is equal to vertex set of $G$.*
2. *Assume $a,b \in V$. If there is path between $a$ and $b$ in $G$ then there must be an edge between $a$ and $b$ in $cl(G)$.*
3. *Any transitive subgraph of $(V,V \times V)$ is contained by $cl(G)$.*

**Algorithm 2** An algorithm finding the transitive closure of directed graphs.

```
function TRANSITIVECLOSURE(G)
    for all x in vertex set of G do
        for all y in vertex set of G do
            u ← the shortest path between x and y
            if the length of u is bigger than 2 then
                add the edge (x,y) to G
            end if
        end for
    end for
    return G
end function
```

**Remark 2.1.** *Transitive closure operator behaves like the topological closure operator. Hence;*

1. *If $G$ is transitive then $cl(G) = G$.*
2. *If $H \subseteq G$ then $cl(H) \subseteq cl(G)$.*

3. $cl(cl(G)) = cl(G)$.

**Proposition 2.1.** *Assume G is transitive and H is a Hasse subgraph of G then transitive closure of H is G.*

*Proof.* Assume $cl(H) \neq G$. Then $cl(H) \subset G$ since $G = cl(G)$ and closure operator respects set inclusion. If $cl(H) \subset G$ then there exists an edge $(a,b)$ in $G$ which is not in $H$. If there is no edge between $a$ and $b$ in $H$ then there is no path between them in $H$. Therefore, $a$ and $b$ are connected in $G$ and they are not connected in $H$ which is a contradiction. Hence $cl(H) = G$. □

**Proposition 2.2.** *Let G be a digraph which contains $K_3$(complete graph of 3 vertices). Then Hasse subgraph of G is not unique.*

*Proof.* Proof by counter example:

Let $G = (V,E) = K_3$ be a digraph and $E = \{(0,1),(1,0),(1,2),(2,1),(0,2),(2,0)\}$, $V = \{0,1,2\}$ . Then $E_1 = \{(0,2),(2,0),(1,2),(2,1)\}$ and $E_2 = \{(0,1),(1,2),(2,0)\}$ are Hasse subgraph of $G$. Therefore Hasse subgraph of $G$ is not unique. □

**Proposition 2.3.** *Every directed acyclic graph has a unique Hasse subgraph.*

*Proof.* Assume $G$ is a directed acyclic graph and $H_1$,$H_2$ are hasse subgraphs of $G$. Since $H_1$ and $H_2$ are different graphs, there exists an edge $(a,b) \in H_1$ and $(a,b) \notin H_2$. If $(a,b) \in H_1$ then $a$ and $b$ are connected in $G$ hence they must be connected in $H_2$. Then there exist a vertex $c$ and a path $\alpha = (a,\ldots,c,\ldots,b)$ in $H_2$. If $a$ and $c$ are connected in $H_2$ then they must be connected in $H_1$, in addition if $c$ and $b$ are connected in $H_2$ then they must be connected in $H_1$. Hence there exists a path between $a$ and $c$,and there exists a path between $c$ and $b$ in $H_1$. A path $\alpha = (a,\ldots,c,\ldots,b)$ must contain $(a,b)$ since $H_1$ is Hasse subgraph. If $(a,b) \in \alpha$ then we get a cycle in $H_1$ which is a contradiction because $G$ is acyclic. Hence $G$ has a unique Hasse subgraph. □

# 3. ADMISSIBLE DIRECTED GRAPHS

**Definition 3.1.** *We will call a digraph $G = (V, E)$ as* admissible *if*

1. *G has no cycles.*
2. *G has a unique source and a unique sink.*
3. *G is a reduced digraph.*

*The size of an admissible digraph is the number of elements in the vertex set $V$. Usually, we represent the unique source by $s$ and the unique sink by $t$.*

**Lemma 3.1.** *G is an admissible digraph if and only if $G^{op}$ is an admissible digraph.*

*Proof.* The only difference between $G$ and $G^{op}$ is the direction of edges. Hence if $G$ has no cycle neither does $G^{op}$. Assume $v_1$ and $v_2$ in $G$. If there is a path from $v_1$ to $v_2$ in $G$ then $(v_1, v_2)$ is not in $G$ since $G$ is reduced. Thus, there is a path from $v_2$ to $v_1$ in $G^{op}$ and $(v_2, v_1)$ is not in $G^{op}$. Hence $G^{op}$ is also reduced. Moreover, the source of $G$ is the sink of $G^{op}$ and the sink of $G$ is the source of $G^{op}$. Hence $G^{op}$ is admissible. □

**Lemma 3.2.** *Let G be a digraph. If G has a unique source and a unique sink, then every vertex of G is connected to the source and the sink via path.*

*Proof.* Assume $v_1$ is a vertex in $G$ and it is not the source. Then there exists a vertex $v_2$ in $G$ which is connected to $v_1$ since $v_1$ is not the source. If $v_2$ is the source then we are done, if not then we proceed by induction. We have finitely many vertices, therefore ultimately the last vertex must be the source. Hence every vertex is connected to the source. If we consider $G^{op}$, then we conclude that every vertex in $G$ is connected to the sink because the source of $G^{op}$ is the sink of $G$. □

**Definition 3.2.** *An admissible digraph $G = (V, E; s, t)$ is called $k$-*connected *if for every subset $V'$ of size $k$ of $V \setminus \{s, t\}$ the full subgraph $G(V \setminus V')$ on $V \setminus V'$ has a path connecting the source to the sink. An admissible digraph is called* disconnected *if it is not connected. A vertex $v$ in a graph $G = (V, E)$ is called an* articulation *if*

$$G \setminus v := G(V \setminus \{v\})$$

*is disconnected.*

## 3.1 DECOMPOSING ADMISSIBLE DIGRAPHS ALONG ARTICULATIONS AND HORIZONTAL COMPOSITION

In this section we assume $G = (V, E; s, t)$ is an admissible digraph of size $n$ with $k$ articulations $a_1, \ldots, a_k$.

**Proposition 3.3.** *Any path $(s, v_1, \ldots, v_m, t)$ connecting $s$ to $t$ contains all of the articulations. Moreover, any path connecting $s$ to $t$ contains the articulations in the same order.*

*Proof.* Let $\alpha = (s, v_1, \ldots, v_m, t)$ be a path connecting $s$ to $t$. Assume there exists an articulation $a_j$ such that for all $i = 1, \ldots, m$ we have $v_i \neq a_j$. Then removing $a_j$ from $G$ does not affect this path. This contradicts with the fact that $a_j$ is an articulation. Hence, every articulation appears in $\alpha$. Now, assume that we have two articulations $a$ and $a'$, and also that we have two paths $\alpha = (s, v_1, \ldots, v_m, t)$ and $\beta = (s, u_1, \ldots, u_m, t)$ where there is a subpath $(a, v_i, \ldots, v_j, a')$ connecting $a'$ to $a$ in $\alpha$ while there is another subpath $(a', u_\ell, \ldots, u_s, a)$ connecting $a$ to $a'$ in $\beta$. Then we get a cycle in $G$, which is a contradiction. So, articulations appear with a specific order in every path connecting $s$ to $t$. □

**Definition 3.3.** *Let $(a_1, \ldots, a_k)$ be the ordered sequence of articulations appearing in their natural order and $s = a_0$, $t = a_{k+1}$. We define for $0 \leq m \leq k$ let $U_m \subseteq V$*

$$U_m := \{v \in V \mid \text{ any path connecting } s \text{ to } v \text{ has the articulations } a_0, \ldots, a_m \text{ in it}\}$$

*It is clear that $U_0 \supset U_1 \supset \cdots \supset U_k$. Similarly, for $0 \leq m \leq k$ let*

$$W_m = \{v \in V \mid \text{ any path connecting } v \text{ to } t \text{ has the articulations } a_{m+1}, \ldots, a_k \text{ in it } \}$$

*and then $W_0 \subset W_1 \subset \cdots \subset W_k$.*

**Lemma 3.4.** *For $i = 1, \ldots, k-1$ we define $V_i = U_i \cap W_i$. Then for every $i \neq j$*

$$V_i \cap V_j = \begin{cases} \{a_{i+1}\} & \text{if } j = i+1 \\ \emptyset & \text{if } |i-j| > 1 \end{cases}$$

*Proof.* Assume without loss of generality that $i < j$. We see that

$$V_i \cap V_j = U_i \cap W_i \cap U_j \cap W_j = U_j \cap W_i$$

This is because $U_j \subset U_i$ and $W_i \subset W_j$. Let $v \in V_i \cap V_j = U_j \cap W_i$. Then any path from $s$ to $v$ contains the articulations $a_1, \ldots, a_i, \ldots, a_j$ in order, and since $v \in W_i$ any path from $v$ to $t$ contains the articulations $a_{i+1}, \ldots, a_j, \ldots, a_k$ also in order. But then we have a cycle since there is a path from $a_j$ to $v$ and another path from $v$ to $a_j$. This is a contradiction unless $j = i+1$ and $v = a_j$. □

**Lemma 3.5.** *Let $u \in V_i$ and $v \in V_j$ for some $i, j$. If $(u,v)$ is an edge then either $i = j$, or $j = i+1$ and $v = a_{i+1}$.*

*Proof.* Assume $i > j$. Since all paths from $s$ to $u$ contain the articulations $a_1, \ldots, a_j, \ldots, a_i$ and all paths from $v$ to $t$ contain the articulations $a_{j+1}, \ldots, a_i, \ldots, a_k$ we get a cycle on $a_i$. This is a contradiction. Now, assume $i < j$. Let $\alpha = (s, u_1, \ldots, u_\ell, u)$ be a path from $s$ to $u$ which necessarily contains the articulations $a_1, \ldots, a_i$, and let $\beta = (v, v_1, \ldots, v_m, t)$ be a path from $v$ to $t$ which necessarily contains the articulations $a_{j+1}, \ldots, a_k$. Then the path

$$(s, u_1, \ldots, u_\ell, u, v, v_1, \ldots, v_m, t)$$

is a path from $s$ to $t$ which skips the articulations between $a_i$ and $a_j$. This is also a contradiction unless $j = i+1$ and $v = a_{i+1}$. □

**Proposition 3.6.** *Let $a_0 := s$ and $a_{k+1} := t$. Then for each $i = 0, \ldots, k$ the full subgraph $G(V_i)$ is an admissible 1-connected digraph in which $a_i$ is the unique source and $a_{i+1}$ is the unique sink.*

*Proof.* Assume we have an element $w \in V_i$. Since all paths connecting $s$ to $w \in V_i \subseteq U_i$ pass through $a_i$, the element $w$ can not be a source unless $w = a_i$. Similarly, all paths connecting $w$ to $t$ must pass through $a_{i+1}$ since $w \in V_i \subseteq W_i$. Thus $w$ can not be sink unless $w = a_{i+1}$. Assume $G(V_i)$ has an articulation $a \in V_i$. Then this articulation is also an articulation of $G$ and $a$ must appear between $a_i$ and $a_{i+1}$ in order because of Proposition 3.3. This is a contradiction. Thus $G(V_i)$ is a 1-connected admissible digraph. □

**Definition 3.4.** *Let $G = (V, E, s, t)$ and $G' = (V', E', s', t')$ be two directed admissible digraphs. A morphism of admissible digraphs $\phi : G \to G'$ is a map $\phi : V \to V'$ such that*

1. $\phi(s) = s'$ *and* $\phi(t) = t'$.

2. *When extended to a map* $\phi \times \phi \colon V \times V \to V' \times V'$ *it restricts to a map of edges* $(\phi \times \phi)|_E \colon E \to E'$.

*We will use to* $\phi_V$ *and* $\phi_E$ *to denote repectively* $\phi \colon V \to V'$ *and* $(\phi \times \phi)|_E \colon E \to E'$. *A morphism of admissible digraphs* $\phi$ *is called a monomorphism (resp. epimorphism or isomorphism) if* $\phi_V$ *is injective (resp. surjective or bijective.)*

**Lemma 3.7.** *Assume* $G$ *and* $G'$ *are two admissible digraphs and let* $(a_q, \ldots, a_k)$ *and* $(a'_1, \ldots, a'_\ell)$ *be the sequences of articulations appearing in* $G$ *and* $G'$ *in their natural orders. If* $\phi \colon G \to G'$ *is an isomorphism of admissible digraphs then* $k = \ell$ *and* $\phi(a_i) = a'_i$ *for every* $i = 1, \ldots, k$.

*Proof.* If $(s, v_1, \ldots, v_{n-1}, t)$ is a path of length $n$ from $s$ to $t$ in $G$ then $(s', \phi(v_1), \ldots, \phi(v_{n-1}), t')$ is a path from $s'$ to $t'$ in $G'$ of length $n$ in $G'$. Hence $a$ is an articulation in $G$ if and only if $\phi(a)$ is an articulation in $G'$. This way we conclude that the number of articulations for $G$ and $G'$ are equal. Since any path in $G$ from $s$ to $t$ contains all articulations in their natural order, and the same is true for $G'$, we conclude that the natural order for the images of the articulations is $(\phi(a_1), \ldots, \phi(a_k))$. □

**Proposition 3.8.** *Assume* $G$ *is an admissible digraph and let* $(a_1, \ldots, a_k)$ *be its sequence of articulations. Let* $G(V_i)$ *be the full subgraph of* $G$ *with whose source and target is* $a_i$ *and* $a_{i+1}$ *respectively as we constructed in Proposition 3.6 for* $i = 0, \ldots, k$. *Then* $\phi$ *restricts to graph automorphisms of the form* $\phi_i \colon G(V_i) \to G(V_i)$ *for* $i = 0, \ldots .k$.

*Proof.* One can think of $V_i$ as the subset of vertices which appear in some path from $a_i$ to $a_{i+1}$. Then the result follows after using Lemma 3.7. □

**Definition 3.5.** *Let* $G_1$ *and* $G_2$ *be two admissible digraphs. We define the* horizontal composition $G_1 \circ_h G_2$ *as the new graph obtained from* $G_1$ *and* $G_2$ *taking the union of vertices and edges, then identifying the unique sink* $t_1$ *and the unique source* $s_2$.

**Proposition 3.9.** *Let* $G_1$ *and* $G_2$ *be two admissible digraphs, and* $G$ *be a digraph that is formed by horizontal composition of* $G_1$ *and* $G_2$. *Then* $G$ *is an admissible digraph.*

*Proof.* In this proof we represent that $G_1 = (V_1, E_1)$ , $G_2 = (V_2, E_2)$ , $s$ is the source of $G_1$ , $m$ is the sink of $G_1$(source of $G_2$) and $t$ is the sink of $G_2$.

1. Assume that $G$ has a cycle. Then the vertices of cycle are in $V_1$ or in $V_2$ or in $V_1 \cup V_2$. If vertices are in $V_1$ or in $V_2$ then we get a conradiction since $G_1$ and $G_2$ are admissible digraphs, therefore they do not have any cycle. Now suppose that vertices are in $V_1 \cup V_2$ and we represent cycle as $\alpha = (v_0, \ldots, v_n, \ldots, v_0)$. If the cycle start and finish the vertex in $G_1$(or $G_2$) then it must pass through the vertex $m$ at least two times, if not then there must exist another common vertex of $G_1$ and $G_2$. However, this is a conradiction since the only common vertex of $G_1$ and $G_2$ is $m$. If the cycle includes the vertex $m$ at least two times then there exist a cycle in $G_1$ and there exist another cycle in $G_2$ which is also a conradiction. Hence $G$ has no cycle.

2. If we compose $G_1$ and $G_2$ horizontally, then the only vertex whose degree changes is $m$. However $m$ cannot be source or sink of $G$. Therefore the unique source of $G$ is $s$ which is the source of $G_1$ and the unique sink of $G$ is $t$ which is the sink of $G_2$.

3. Assume $v_1$ and $v_2$ are vertices of $G_1 \circ_h G_2$ and assume there is a path from $v_1$ to $v_2$. If $v_1, v_2 \in V_1$ or $v_1, v_2 \in V_2$, then there is no edge between $v_1$ and $v_2$ since $G_1$ and $G_2$ are reduced. Without loss of generality, $v_1 \in V_1$ and $v_2 \in V_2$. In the horizontal composition, we do not add edges hence there is no edges between vertices of $G_1$ and $G_2$. For this reason, there is also no edge between $v_1$ and $v_2$ in this case. Therefore $G_1 \circ_h G_2$ is reduced.

□

**Definition 3.6.** *One can form a monoid out of isomorphism classes of admissible digraphs. Given two admissible digraphs $G_1$ and $G_2$, their product is the horizontal composition of $G_1$ and $G_2$. In order to make this set into a monoid, we must add a trivial admissible digraph $\varepsilon$ which consists of a single vertex (thus we have a graph where $s = t$) and no edges.*

**Definition 3.7.** *Assume $(M, \cdot, e)$ is a monoid. For every $p, x \in M$ we will write $p|x$ if there exists $u, v \in M$ such that $x = upv$. An element $p \in M$ is called* prime *when for every $x, y \in M$ if $p = xy$ then $p = x$ or $p = y$.*

**Theorem 3.10.** *The set of isomorphism classes of admissible digraphs is a free monoid on the set of 1-connected admissible digraphs and 1-connected admissible digraphs are* prime *with respect to horizontal composition.*

*Proof.* Let $S$ be the set of isomorphism classes of admissible digraphs. For an admissible graph $X$, we will use $[X]$ to denote the isomorphism class of $X$ in $S$. Then the trivial

admissible digraph $\varepsilon$ which consists of a single vertex is the identity element since for every admissible graph $G$ we can easily see that

$$G \circ_h \varepsilon = \varepsilon \circ_h G = G$$

This proves $\varepsilon$ is the identy element with respect to $\circ_h$ in $S$. Assume $P$ is a 1-connected admissible graph, and let $X = (V_X, E_X, s_X, t_X)$ and $Y = (V_Y, E_Y, s_Y, t_Y)$ be two admissible graphs such that $P = X \circ_h Y$. Note that $t_X = s_Y$ in $X \circ_h Y$ and it becomes an articulation point if $|V_X| > 1$ and $|V_Y| > 1$. This would be a contradiction since $P$ contains no articulation points. This means $|V_X| = 1$ or $|V_Y| = 1$ which is equivalent to $[X] = \varepsilon$ or $[Y] = \varepsilon$ which in turn equivalent to $[Y] = [P]$ or $[X] = [P]$. □

**Proposition 3.11.** *Assume G is an admissible digraph. If G contains m articulations then G is a unique horizontal product of m prime digraphs.*

*Proof.* We will give the proof by induction on the number of articulation points. If $G$ has 1 articulation then $G$ is prime so it is unique. Assume $G'$ and $G''$ has $m$ articulations, $G$ has $m+1$ articulations and $G = G' \circ_h P_1 = G'' \circ_h P_2$ where $P_1$ and $P_2$ are prime. Consider the sink of $G$. It is obvious that the sink of $G$ is the sink of $P_1$ and $P_2$. Moreover, the edges related to the sink are same in $P_1$ and $P_2$. If we delete the sink and the edges connected to the sink, then the remaining graphs are still identical. Now we have finitely many sinks. If we continue this process until we obtain a single sink which is the sink of $G'$ and $G''$, we realize that $P_1$ and $P_2$ are identical. By induction hypothesis $G' = G''$ is written uniquely as a product of $m$ prime digraphs. □

## 3.2 VERTICAL COMPOSITION AND DECOMPOSING 1-CONNECTED ADMISSIBLE DIGRAPHS

**Definition 3.8.** *Let $G_1$ and $G_2$ be two admissible digraphs. We define the vertical composition $G_1 \circ_v G_2$ as the new graph obtained from $G_1$ and $G_2$ taking the union of vertices and edges, then identifying the unique sources $s_1$ and $s_2$, and then identifying the unique sinks $t_1$ and $t_2$.*

**Remark 3.1.** *No* cycle *passes through the sink or the source, because the source has only out-arrows and the sink has only in-arrows.*

**Proposition 3.12.** *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two admissible digraphs, and $G$ be a digraph formed by vertical composition of $G_1$ and $G_2$. Then $G$ is an admissible 1-connected digraph.*

*Proof.* 1. In vertical composition, we identify the unique sources $s_1$ and $s_2$ (represented by $s$), and we identify the unique sinks $t_1$ and $t_2$ (represented by $t$). Therefore $G$ has a unique source $s$ and a unique sink $t$.

2. Assume $G$ has a cycle. Then this cycle cannot be in $G_1$ or $G_2$, since they are admissible digraphs. Therefore, vertices of cycle must be in $V_1 \cup V_2$. Without loss of generality, assume cycle start in $G_1$ and it passes through a vertex in $G_2$. Then it must go through source or sink, because the only common vertices after the identification are the source and the sink. However cycle cannot include the source or the sink by Remark 3.1, we get a contradiction. Therefore $G$ has no cycle.

3. Suppose there is a path from $v_1$ to $v_2$ in $G_1 \circ_v G_2$. Then we have three cases;

   (a) $v_1, v_2 \in V_1$ or

   (b) $v_1, v_2 \in V_2$ or

   (c) $v_1 \in V_1$ and $v_2 \in V_2$(or vice versa)

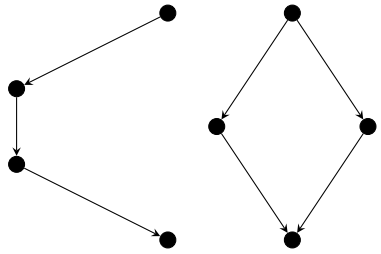
   $G_1$ and $G_2$ are reduced therefore if $(a)$ or $(b)$ is true then there is no edge between $v_1$ and $v_2$. We do not add edges in the vertical composition. So in this case too, there is no edge between $v_1$ and $v_2$. Hence $G_1 \circ_v G_2$ is reduced.

What remains to be shown is that $G$ is 1-connected. If we remove one vertex in $G_1$ then $G_2$ has path connecting $s$ to $t$ or if we remove one vertex in $G_2$ then $G_1$ has path connecting $s$ to $t$. Therefore $G$ is an admissible 1-connected digraph. □

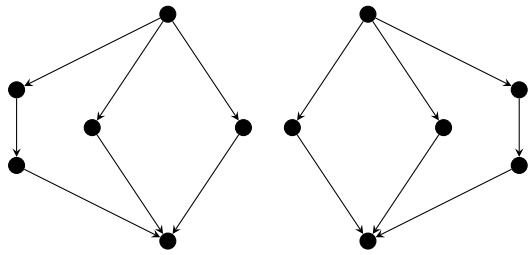
**Lemma 3.13.** *The operation of vertical composition $\circ_v$ is commutative that means if $G_1$ and $G_2$ are admissible digraphs then $G_1 \circ_v G_2 = G_2 \circ_v G_1$.*

*Proof.* In the vertical composition, we just identify the unique sources of digraphs and also we identify the unique sinks of digraphs. Therefore, there is no difference between $G_1 \circ_v G_2$ and $G_2 \circ_v G_1$. □

**Example 3.1.** *Assume the following graphs are $G_1$ and $G_2$ which are admissible digraph*



*We can easily see that $G_1 \circ_v G_2 = G_2 \circ_v G_1$*



**Proposition 3.14.** *Suppose that $G_1$ and $G_2$ are admissible digraphs. If $G_1 \circ_v G_2$ is reduced then $G_1 \neq A_1$ or $G_2 \neq A_1$.*

*Proof.* We will prove the contrapositive. Without loss of generality, assume $G_1 = A_1$. Then there is an edge between the source of $G_1 \circ_v G_2$ and the sink of $G_1 \circ_v G_2$, because $G_1 = A_1$. Also there is a path from the source of $G_1 \circ_v G_2$ to the sink of $G_1 \circ_v G_2$, since $G_2$ is admissible. Then $G_1 \circ_v G_2$ is not reduced. □

**Remark 3.2.** *Vertical composition $\circ_v$ has no unit element. Therefore the set of isomorphism class with operation $\circ_v$ is not monoid but semi-group. Moreover, we have to redefine prime digraphs with respect to $\circ_v$.*

**Definition 3.9.** *Assume G is an admissible digraph. G is decomposed by the following way;*

1. *Delete the source and the sink of G,*
2. *Find connected components of G,*
3. *Add one source and one sink to each components back seperately.*

*Decomposition operation of G is represented by* decomp(G). decomp(G) *is a multi-set valued function.*

**Definition 3.10.** *(**Vertically prime**) G is vertically prime if for all $A, B$ $A \neq G$ and $B \neq G \Rightarrow G \neq A \circ_v B$ where A and B are admissible digraphs. That means if G is not prime then we can write G as the composition of two admissible digraphs.*

**Proposition 3.15.** *Assume P is an admissible digraph. P is vertically prime if and only if $decomp(P) = \{P\}$.*

*Proof.* ($\Leftarrow$) By definition of vertically prime.

($\Rightarrow$) If we delete the source and the sink of $P$ then we have one connected component since $P$ is prime with respect to the vertical composition. Then adding one source and one sink to this component brings about reobtaining $P$. Hence $decomp(P) = \{P\}$. □

**Corollary 3.16.** *Assume G is an admissible digraph and $decomp(G) = \{A_1, \cdots, A_n\}$. Then each $A_i$'s are prime admissible digraph.*

*Proof.* We know that vertical decomposition operation cannot decompose vertically primes. Hence all elements in the multi-set of decomp(G) are vertically prime. □

**Proposition 3.17.** *If G and H are admissible digraphs then the decomposition of $G \circ_v H$ is same as the disjoint union of the decomposition of G and the decomposition of H i.e ;*

$$decomp(G \circ_v H) = decomp(G) \sqcup decomp(H)$$

*We take the disjoint union in the sense of multi-set disjoint union.*

*Proof.* If $G$ and $H$ are prime digraphs with respect to vertical composition then the proof is done. Now assume we decompose $G \circ_v H$ and we take a digraph, say $A$, from this decomposition set. Suppose this digraph neither in $decomp(G)$ nor in $decomp(H)$. Therefore some part of $A$ is in $decomp(G)$ and the other part of $A$ is in $decomp(H)$. It means that $A$ can be decomposed but this is a contradiction since $A$ is prime. Hence if we decompose $G \circ_v H$, then all digraphs in $decomp(G \circ_v H)$ are either in $decomp(G)$ or in $decomp(H)$. □

**Proposition 3.18.** *If $A_1, \cdots, A_n$ are prime admissible digraphs then $decomp(A_1 \circ_v \cdots \circ_v A_n) = \{A_1, \cdots, A_n\}$.*

*Proof.* $decomp(A_1 \circ_v \cdots \circ_v A_n) = decomp(A_1) \sqcup \cdots \sqcup decomp(A_n) = \{A_1\} \sqcup \cdots \sqcup \{A_n\} = \{A_1, \cdots, A_n\}$ □

**Lemma 3.19.** *Assume $A_i$'s and $B_j$'s are prime and $G = A_1 \circ_v \cdots \circ_v A_n = B_1 \circ_v \cdots \circ_v B_m$. Then n=m and $\{A_1, \cdots, A_n\} = \{B_1, \cdots, B_n\}$.*

*Proof.* $decomp(A_1 \circ_v \cdots \circ_v A_n) = \{A_1, \cdots, A_n\}$ and $decomp(B_1 \circ_v \cdots \circ_v B_m) = \{B_1, \cdots, B_m\}$. Hence $\{A_1, \cdots, A_n\} = \{B_1, \cdots, B_m\}$. However if two multiset are equal then the number of elements must be equal. Hence $n = m$ and $\{A_1, \cdots, A_n\} = \{B_1, \cdots, B_n\}$. □

**Lemma 3.20.** *Every 1-connected admissible digraph is a unique product of prime digraphs (w.r.t $\circ_v$).*

*Proof.* We know that vertical composition of admissible digraphs is 1-connected admissible digraph. Now, assume $G$ is 1-connected admissible digraph and

$$G = A_1 \circ_v \cdots \circ_v A_n = B_1 \circ_v \cdots \circ_v B_m$$

where $A_i$'s and $B_j$'s are vertically prime. Then by Lemma 3.19 $n = m$ and $\{A_1, \cdots, A_n\} = \{B_1, \cdots, B_n\}$. But if two multi-sets are equal then for all $i$ there exists $j$ such that $A_i = B_j$ where $i, j = 1, \cdots, n$. Moreover $\circ_v$ is commutative, hence $G = A_1 \circ_v \cdots \circ_v A_n$ is the unique product for $G$. □

# 4. ALGORITHMS

In this chapter, we are going to find the number of admissible digraphs and the list of all admissible digraphs by using specific algorithms which we will develop. We assume $S(n,k)$ denotes the set of all admissible digraphs of size $n$ with $k$ articulations, $G(n)$ denotes the set of all admissible 1-connected digraphs of size $n$ and $U(n)$ denotes the set of all admissible digraphs of size $n$. First of all, we will write algorithms which count the number of admissible digraphs.

**Algorithm 3** An algorithm counting all admissible digraphs of size $n$ with $k$ articulations.

**Input:** $n > 0$ **and** $k > 0$
  **function** COUNTS($n,k$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** 0
    **else if** $n \leqslant k+1$ **then**
      **return** 0
    **else if** $k = 0$ **then**
      **return** COUNTG($n$)
    **else if** $n = k+2$ **then**
      **return** 1
    **end if**
    $Sum = 0$
    **for all** $i$ in (2,n-1) **do**
      $Sum \leftarrow Sum + \text{COUNTG}(i) \cdot \text{COUNTS}(n+1-i, k-1)$
    **end for**
    **return** *Sum*
  **end function**

In **Algorithm 3** , we consider admissible digraph of size $n$ with $k$ articulations as the horizontal composition of admissible 1-connected digraph and admissible digraph with $k-1$ articulations. In other words, we break admissible digraph into two pieces such that one part is admissible 1-connected digraph and the other part is admissible digraph with $k-1$ articulations. Therefore, in computation we use recursive formula

$$CountS(n,k) = \sum_{i=2}^{n-1} CountG(i)CountS(n+1-i,k-1)$$

Our algorithms are recursive, therefore we must specify the base cases seperately. If the graph has $n$ vertices and $n-2$ articulations, then this graph is straight line graph in which

all vertices except the source and the sink are articulations. Therefore, in the algorithm we return 1 for this graph in case of $n = k+2$. Moreover, if the graph has no articulation which means $k = 0$, then we return the number of 1-connected admissible digraphs with size $n$.

**Algorithm 4** An algorithm counting all admissible 1-connected digraphs of size $n$.

**Input:** $n > 0$
  **function** COUNTG($n$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** 0
    **else if** $n = 2$ **then**
      **return** 1
    **else if** $n = 3$ **then**
      **return** 0
    **else if** $n = 4$ **then**
      **return** 1
    **end if**
    $Sum = 0$
    **for all** $i$ in (3,n-1) **do**
      **for all** $j$ in (1,i-2) **do**
        $Sum \leftarrow Sum + \text{COUNTS}(i,j) \cdot \text{COUNTU}(n+2-i)$
      **end for**
    **end for**
    **return** *Sum*
  **end function**

Vertical composition is quite helpful for **Algorithm 4**. We write admissible 1-connected digraphs as the vertical composition of admissible digraphs. For this computation we use the recursive formula

$$CountG(n) = \sum_{i=3}^{n-1} \sum_{j=1}^{i-2} CountS(i,j) CountU(n+2-i)$$

In vertical composition, first we take admissible digraph with at least one articulation and the other graph is any admissible digraph. This is the reason why $j$ must through from 1 to $i-2$ in the sum. Also in algorithm, if $n = 2$ we take the digraph which has a just one edge, if $n = 4$ we take $C_4$ which has a unique source and a unique sink. Hence the algorithm return number 1 for these base cases.

For listing all admissible digraphs in **Algorithm 5** , we collect admissible digraphs with articulations from 0 to $n-2$. Thus, we use formula

**Algorithm 5** An algorithm counting all admissible digraphs of size $n$.

**Input:** $n > 0$
  **function** COUNTU($n$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** 0
    **end if**
    $Sum = 0$
    **for all** $k$ in (0,n-2) **do**
      $Sum \leftarrow Sum + \text{COUNTS}(n,k)$
    **end for**
    **return** *Sum*
  **end function**

$$CountU(n) = \sum_{k=0}^{n-2} CountS(n,k)$$

After writing algorithms which count the number of admissible digraphs, now we are going to generate algorithms which list the elements of admissible digraphs. For writing the list algorithms, we will use a modified version of the recursive formulas we applied in counting algorithms.

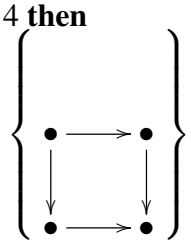
**Algorithm 6** An algorithm listing all admissible digraphs of size $n$ with $k$ articulations.

**Input:** $n > 0$ **and** $k > 0$
  **function** LISTS($n,k$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** $\emptyset$
    **else if** $n \leqslant k+1$ **then**
      **return** $\emptyset$
    **else if** $k = 0$ **then**
      **return** $\text{LISTG}(n)$
    **else if** $n = k+2$ **then**
      **return** $\{\bullet \rightarrow \bullet \cdots \bullet \rightarrow \bullet\}$
    **end if**
    $Set = \emptyset$
    **for all** $i$ in (2,n-1) **do**
      $Set \leftarrow Set \cup \text{LISTG}(i) \circ_h \text{LISTS}(n+1-i, k-1)$
    **end for**
    **return** *Set*
  **end function**

In **Algorithm 6**, we see the horizontal composition $X \circ_h Y$ (Definition 3.5) in the last **for** loop where $X$ and $Y$ are sets of admissible digraphs. We extend $\circ_h$ to sets by taking all

possible horizontal composition $\alpha \circ_h \beta$ for all $\alpha \in X$ and $\beta \in Y$. Special cases are similar with cases in the count algorithms.

**Algorithm 7** An algorithm listing all admissible 1-connected digraphs of size $n$.

**Input:** $n > 0$
  **function** LISTG($n$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** $\emptyset$
    **else if** $n = 2$ **then**
      **return** $\{\bullet \to \bullet\}$
    **else if** $n = 3$ **then**
      **return** $\emptyset$
    **else if** $n = 4$ **then**
      **return** 
    **end if**
    $Set = \emptyset$
    **for all** $i$ in (3,n-1) **do**
      **for all** $j$ in (1,i-2) **do**
        $Set \leftarrow Set \cup \text{LISTS}(i, j) \circ_v \text{LISTU}(n+2-i)$
      **end for**
    **end for**
    **return** $Set$
  **end function**

In the **Algorithm 7**, procedure in the algorithm 6 for the horizontal composition is same as the vertical composition $X \circ_v Y$ (Definition 3.8) in the **for** loop. We extend $\circ_v$ to sets by taking all possible vertical composition $\alpha \circ_v \beta$ for all $\alpha \in X$ and $\beta \in Y$.

The **Algorithm 8** is just modified version of the **Algorithm 5**. Instead of counting, this algorithm find the list of all admissible digraphs with size $n$.

## 4.1 CONCLUSION

After writing these algorithms, we observe that they do not count the number of admissible digraphs exactly. There are over counts. Actually the numbers that our algorithms

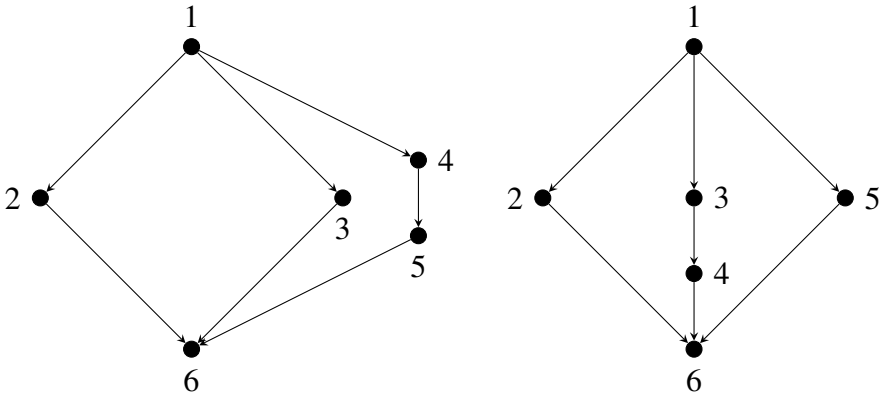
**Algorithm 8** An algorithm listing all admissible digraphs of size $n$.

**Input:** $n > 0$

```
function ListU(n)
    if n = 0 or n = 1 then
        return ∅
    end if
    Set = ∅
    for all k in (0,n-2) do
        Set ← Set ∪ ListS(n,k)
    end for
    return Set
end function
```

find are upper bounds, because our algorithms count isomorphic digraphs more than once in some cases.

For example, consider the admissible digraphs with 6 vertices below;




When our algorithms find the list of $G(6)$, these graphs are in the list as seperate digraphs. Hence they are counted as different digraphs according to the algorithms. However, these digraphs are isomorphic. Therefore, it is a mistake that they are counted seperately. Actually, not the horizontal composition but the vertical composition leads to such a mistake since vertical composition is commutative but our algorithm does not take this into account. We have to do something to avoid these overcounts.

**Remark 4.1.** *In order to prevent double counting, we will define well-ordering over admissible digraphs. We draw inspiration from ordered partitions of natural numbers. Let n and k be natural numbers. $p(n,k)$ is the number of ordered k-partition of n such that*

$$n = a_1 + a_2 + \cdots + a_k$$

*where* $a_i \leq 1$ *and* $a_1 \leq a_2 \leq \cdots \leq a_k$. *For example,* $p(5,3) = 2$ *since* 3*- partition of* 5 *are* $5 = 1+1+3$ *and* $5 = 1+2+2$. *If we do not order the partition then* $5 = 1+1+3$ *is counted three times as* $5 = 1+1+3$, $5 = 1+3+1$, $5 = 3+1+1$

# 5. GRAPH ORDERING

**Definition 5.1.** *Assume that we have two pairs $\alpha = (x_1, y_1)$ and $\beta = (x_2, y_2)$. We say that $\alpha \leq \beta$ if;*

1. *$x_1 < x_2$ or*
2. *If $x_1 = x_2$ then we ask $y_1 < y_2$ or*
3. *If $x_1 = x_2$ and $y_1 = y_2$ then $\alpha = \beta$*

**Definition 5.2.** *Assume $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are digraphs. We say that $G_1 \leq G_2$ if*

1. *$|V_1| < |V_2|$ or*
2. *If $|V_1| = |V_2|$ then we require $|E_1| < |E_2|$ or*
3. *If $|V_1| = |V_2|$ and $|E_1| = |E_2|$ then we define the ordering as fallows: Suppose that $G_1$ and $G_2$ are two admissible digraphs and $|V_1| = |V_2|$ , $|E_1| = |E_2|$. Assume we order the edge-lists of two graphs $G_1 = \{(v_1, u_1), (v_2, u_2), \cdots, (v_n, u_n)\}$ where $(v_i, u_i) \leq (v_{i+1}, u_{i+1})$ and $G_2 = \{(v'_1, u'_1), (v'_2, u'_2), \cdots, (v'_n, u'_n)\}$ where $(v'_i, u'_i) \leq (v'_{i+1}, u'_{i+1})$. We say $G_1 < G_2$ if there exists $i \in \{1, 2, \cdots, n\}$ such that $(v_j, u_j) = (v'_j, u'_j)$ for $j = 1, 2, \cdots, i-1$ and $(v_i, u_i) < (v'_i, u'_i)$. This is a generalization of lexicographical ordering.*

**Example 5.1.** *Let $G_1 = \{(0,1), (0,2), (1,3), (2,3)\}$ and*

*$G_2 = \{(0,1), (1,2), (1,3), (2,4), (3,4)\}$. The number of vertices of $G_1$ is less than the number of vertices of $G_2$, thus $G_1 < G_2$.*

**Example 5.2.** *Let $G_1 = \{(0,1), (1,2), (2,3), (3,4)\}$ and*

*$G_2 = \{(0,1), (1,2), (1,3), (2,4), (3,4)\}$. The number of vertices of $G_1$ and $G_2$ are equal. However the number of edges of $G_1$ is less than the number of edges of $G_2$, thus $G_1 < G_2$.*

**Example 5.3.** *Let $G_1 = \{(0,1), (0,2), (1,3), (2,3), (3,4)\}$ and*

*$G_2 = \{(0,1), (1,2), (1,3), (2,4), (3,4)\}$. The number of edges and vertices of $G_1$ and $G_2$ are equal. However $G_1 < G_2$, since in the edge-list $(0,1) = (0,1)$ but $(0,2) < (1,2)$.*

**Remark 5.1.** *Horizontal and vertical composition do not change the total number of edges. In other words, if G has n edges and H has m edges then either $G \circ_h H$ or $G \circ_v H$ has $n+m$ edges.*

**Lemma 5.1.** *Let G be an admissible digraph with k vertices and H be an admissible digraph with l vertices. Then $G \circ_h H$ has $k+l-1$ vertices and $G \circ_v H$ has $k+l-2$ vertices.*

*Proof.* In the horizontal composition, we identify the sink of $G$ and the source of $H$ and in the vertical composition we identify the source of $G$ and the source of $H$, the sink of $G$ and the sink of $H$. Hence $G \circ_h H$ has $k+l-1$ vertices and $G \circ_v H$ has $k+l-2$ vertices.

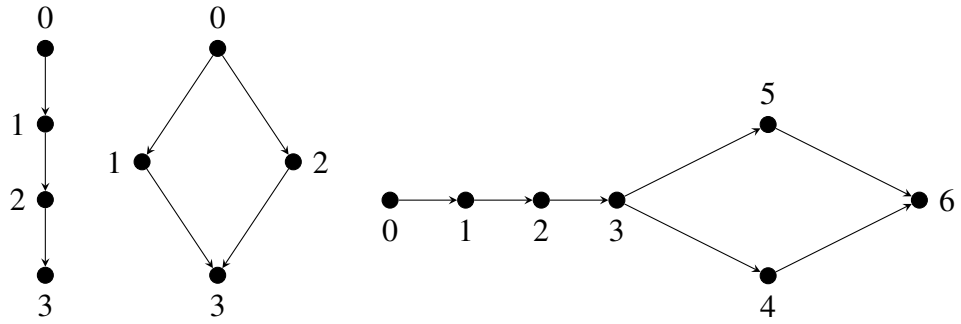
□

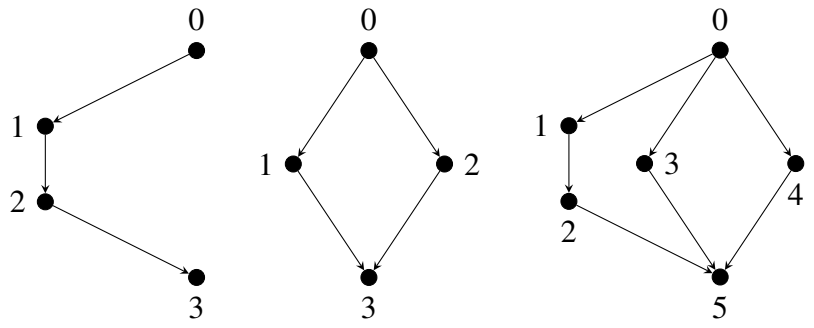
**Remark 5.2.** *In our algorithms, every graph has a specific representation. Vertices are subset of $\mathbb{N}$ without gaps. Assume $G_1 = (V_1, E_1, s_1, t_1)$ and $G_2 = (V_2, E_2, s_2, t_2)$ are admissible digraphs. In the horizontal and vertical composition, we define labelling as follows;*

*In the horizontal composition $G_1 \circ_h G_2$, vertex labels of $G_1$ do not change and labels of vertices of $G_2$ increase by $t_1$.*

*After the vertical composition $G_1 \circ_v G_2$, every vertex in $G_1$ (except its sink) has same labelling as in $G_1$, and we increase the label of the vertices of $G_2$ (except the source of $G_2$) by $t_1 - 1$. In the vertical composition, we identify the source of $G_1$ and the source of $G_2$ and also we identify the sink of $G_1$ and the sink of $G_2$. The label of $s_1$ and $s_2$ does not change and the label of $t_1$ and $t_2$ is equal $(t_1 + t_2 - 1)$ since we use specific representation.*

**Example 5.4.** *The following graphs $G_1$ and $G_2$ are admissible digraphs. We easily understand labelling of horizontal and vertical composition in figures;*






**Proposition 5.2.** *Suppose that $G = (V,E)$ and $H = (V',E')$ are two admissible digraphs. If $G < H$ then for every other admissible digraphs A and B we have;*

1. $A \circ_h G \circ_h B < A \circ_h H \circ_h B$
2. $A \circ_v G \circ_v B < A \circ_v H \circ_v B$

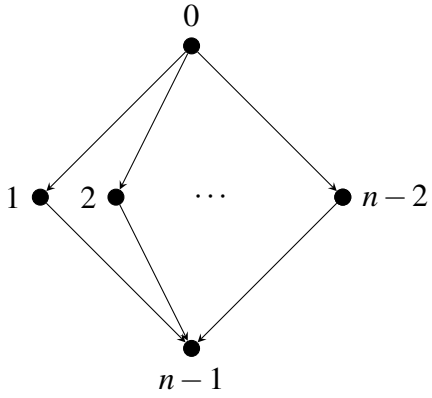
*Proof.* For the horizontal composition, the proof can be reduced to $A \circ_h G < A \circ_h H$ and $G \circ_h B < H \circ_h B$. For the vertical composition, it is enough to prove that $A \circ_v G < A \circ_v H$ since the vertical composition is commutative. Before we begin to the proof, we define $A = (V_1,E_1)$, $B = (V_2,E_2)$. If $G < H$ then we have three cases;

1. $|V| < |V'|$ or
2. $|E| < |E'|$ or
3. $G$ is less than $H$ with respect to the well-ordering we defined item by Definition 5.2

If $|E| < |E'|$ then $|E_1| + |E| < |E_1| + |E'|$ and $|E_2| + |E| < |E_2| + |E'|$. Moreover if $|V| < |V'|$ then $|V_1| + |V| - 1 < |V_1| + |V'| - 1$ , $|V_2| + |V| - 1 < |V_2| + |V'| - 1$ and $|V_1| + |V| - 2 < |V_1| + |V'| - 2$. Therefore the proof is done for cases $(1)$ and $(2)$ by Remark 5.1 and Lemma 5.1. Now, assume $|V| = |V'|, |E| = |E'|$ and $G < H$. Therefore we have to take into account their edge-lists. In the horizontal composition, labels of first graph do not change and labels of second graph change by Remark 5.2. Thus if $G < H$ then $A \circ_h G < A \circ_h H$ and $G \circ_h B < H \circ_h B$. In the vertical composition, if $G < H$ then $A \circ_v G < A \circ_v H$ again by Remark 5.2.

□

**Definition 5.3.** *$\Phi(n)$ is the unique admissible digraph with n vertices such that there is an edge from the source to every vertex except the sink and there is an edge from every vertex to the sink. However, there are no edges between other vertices. It is easily seen that $\Phi(n)$ has $2(n-2)$ edges.*




**Proposition 5.3.** $\Phi(n_1) \circ_v \Phi(n_2) = \Phi(n_1 + n + 2 - 2)$

*Proof.* From the definition of vertical composition. □

**Proposition 5.4.** $\Phi(n_1) \circ_h \Phi(n_2) < \Phi(n_1 + n_2 - 1)$

*Proof.* $\Phi(n_1 + n_2 - 1)$ has $n_1 + n_2 - 1$ vertices and $2[(n_1 + n_2 - 1) - 2]$ edges. If we compose $\Phi(n_1)$ and $\Phi(n_2)$ horizontally then $\Phi(n_1) \circ_h \Phi(n_2)$ has $n_1 + n_2 - 1$ vertices and $2(n_1 - 2) + 2(n_2 - 2)$ edges. $2n_1 + 2n_2 - 8 < 2n_1 + 2n_2 - 6$, therefore $\Phi(n_1) \circ_h \Phi(n_2) < \Phi(n_1 + n_2 - 1)$. □

**Proposition 5.5.** *$\Phi(n)$ is the largest admissible digraph with n vertices with respect to the order we defined earlier.*

*Proof.* Assume $G$ is an admissible digraph with $n$ vertices. We want to prove that forall $G$, $G < \Phi(n)$. We do the proof by induction;

Basis step: $n = 3$. $\Phi(3)$ is the only admissible digraph with 3 vertices, so it is the largest.

Now assume $G$ has articulations and $G_i$ has $n_i$ vertices for $i = 1,2,\cdots,k$ then we can write;

$$G = G_1 \circ_h G_2 \cdots \circ_h G_k < \Phi(n_1) \circ_h \Phi(n_2) \circ_h \cdots \circ_h \Phi(n_k) \quad by \quad Proposition\ 5.2$$

and also for $n = n_1 + n_2 + \cdots + n_k - k - 1$ we can write;

$$\Phi(n_1) \circ_h \Phi(n_2) \circ_h \cdots \Phi(n_k) < \Phi(n_1 + n_2 + \cdots + n_k - k - 1) = \Phi(n) \quad by \quad Proposition\ 5.4$$

If $G$ has no articulation then;

$$G = G_1 \circ_v G_2 \cdots \circ_v G_j < \Phi(n_1) \circ_v \Phi(n_2) \circ_v \cdots \circ_v \Phi(n_j) \quad by \quad Proposition\ 5.2$$

and for $n = n_1 + n_2 + \cdots + n_j - 2(j-1)$

$$\Phi(n_1) \circ_v \Phi(n_2) \circ_v \cdots \circ_v \Phi(n_j) < \Phi(n_1 + n_2 + \cdots + n_j - 2(j-1)) = \Phi(n) \quad by \quad Proposition\ 5.3$$

Therefore $G < \Phi(n)$ for all $n$. That means $\Phi(n)$ is the largest admissible digraph with $n$ vertices. □

**Remark 5.3.** *$A_n$ is the line digraph with n vertices.*

**Proposition 5.6.** $A_{n_1+n_2-2} < A_{n_1} \circ_v A_{n_2}$

*Proof.* $A_{n_1+n_2-2}$ has $n_1 + n_2 - 2$ vertices and $n_1 + n_2 - 3$ edges. $A_{n_1} \circ_v A_{n_2}$ has $n_1 + n_2 - 2$ vertices and $n_1 + n_2 - 2$ edges. $n_1 + n_2 - 3 < n_1 + n_2 - 2$ therefore $A_{n_1+n_2-2} < A_{n_1} \circ_v A_{n_2}$. □

**Proposition 5.7.** $A_{n_1+n_2-1} = A_{n_1} \circ_h A_{n_2}$

*Proof.* By definition of horizontal composition. □

**Proposition 5.8.** *$A_n$ is the smallest admissible digraph with n vertices (with respect to order we defined).*

*Proof.* Assume $G$ is an admissible digraph with $n$ vertices. We want to prove that forall $G$, $A_n < G$. We do the proof by induction;

Basis step: $n = 2$. $A_2$ is the only admissible digraph with 2 vertices, so it is the smallest.

Now assume $G$ has articulations and $G_i$ has $n_i$ vertices for $i = 1, 2, \cdots, k$ then we can write;

$$G = G_1 \circ_h G_2 \cdots \circ_h G_k > A_{n_1} \circ_h A_{n_2} \circ_h \cdots \circ_h A_{n_k} \quad by \quad Proposition\ 5.2$$

and also for $n = n_1 + n_2 + \cdots + n_k - k - 1$ we can write;

$$A_{n_1} \circ_h A_{n_2} \circ_h \cdots A_{n_k} = A_{n_1+n_2+\cdots+n_k-k-1} = A_n \quad by \quad Proposition\ 5.7$$

If $G$ has no articulation then;

$$G = G_1 \circ_v G_2 \cdots \circ_v G_j > A_{n_1} \circ_v A_{n_2} \circ_v \cdots \circ_v A_{n_j} \quad by \quad Proposition\ 5.2$$

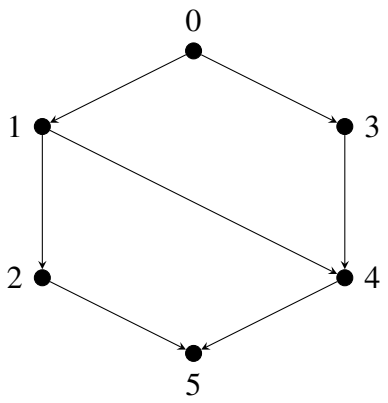
and for $n = n_1 + n_2 + \cdots + n_j - 2(j-1)$

$$A_{n_1} \circ_v A_{n_2} \circ_v \cdots \circ_v A_{n_j} > A_{n_1+n_2+\cdots+n_j-2(j-1)} = A_n \quad by \quad Proposition\ 5.6$$

Therefore $G > A_n$ for all $n$. That means $A_n$ is the smallest admissible digraph with $n$ vertices. □

# 6. ALGORITHMS WITH GRAPH ORDERING

As we mentioned before, our algorithms do not count all admissible digraphs due to several reasons. As we explained earlier, our counting algorithm depends on decomposing our graphs both vertically and horizontally until we reach vertically and horizontally indecomposable graphs. If we have an admissible digraph $G$, firstly we horizontally decompose $G$, then we vertically decompose all connected components of $G$. We continue this process until we just have the specific digraph with two vertices called $A_1$. Hence all admissible digraphs we want to count are generated by $A_1$ using horizontal and vertical composition even though there other graphs which are irreducible in this sense such as the graph below.




Moreover, our counting algorithm will over-count those graphs which can be reduced to $A_1$ due to the fact that vertical composition is commutative and the algorithm counts different permutations of the vertical compositions as distinct even though it should not. In this chapter we would like to correct this-over count, and develop precise algorithms to list all admissible graphs which under recursive horizontal and vertical decompositions all reduce to the irreducible graph $A_1$. We will solve this problem using the graph-ordering we developed in Chapter 5.

In these algorithms, we use graph ordering to overcome the isomorphism problem we mentioned above. Our algorithms are going to give us the list of admissible digraphs which are less than or equal to a given admissible digraph. All of our algorithms take two values: a digraph and the number of vertices. We assume,

- $ListArt(G,n)$ denotes the set of all admissible digraphs (less than or equal to a given digraph) of size $n$ with at least one articulation
- $ListOne(G,n)$ denotes the set of all 1-connected admissible digraphs of size $n$ which are less than or equal to a given digraph
- $List(G,n)$ denotes the set of all admissible digraphs of size $n$ which are less than or equal to a given digraph

If we want to find the list of digraphs with $n$ vertices, we have to choose the given digraph as $\Phi(n)$ since $\Phi(n)$ is the largest admissible digraph with $n$ vertices (from Proposition 5.5).

**Algorithm 9** An algorithm listing all admissible digraphs $H \leq G$ of size $n$ with at least 1 articulation.

**Input:** $n > 1$
**function** LISTART($G,n$)
**if** $n = 2$ **then**
**return** $\emptyset$
**else if** $n = 3$ **then**
**return** $\{\bullet \to \bullet \to \bullet\}$
**end if**
$Set \leftarrow \emptyset$
**for all** $i$ in (2,n-1) **do**
$A = ListOne(\Phi(i), i)$
$B = List(\Phi(n-i+1), n-i+1)$
**for all** $(x,y)$ in $A \times B$ **do**
**if** $x \circ_h y \leq G$
$Set \leftarrow Set \cup \{x \circ_h y\}$
**end for**
**end for**
**return** $Set$
**end function**

In **Algorithm 9**, we list all admissible digraphs with at least one articulation which are less than or equal to $G$. We consider admissible digraph with at least one articulation as the horizontal composition of 1-connected admissible digraph and any admissible digraph of appropriate sizes. Therefore in our computation we use the recursive algorithm.

In mathematical form, the algorithm above describes the following set decomposition

$$ListArt(G,n) = \bigcup_{i=2}^{n-1} ListOne(\Phi(i), i) \circ_h List(\Phi(n+1-i), n+1-i)$$

**Algorithm 10** An algorithm listing all one-connected admissible digraphs $H \leq G$ such that $|H| = n$.

**Input:** $n > 1$

**function** LISTONE$(G, n)$

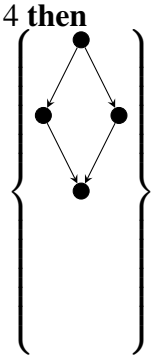
**if** $n = 2$ **then**

**return** $\{\bullet \to \bullet\}$

**else if** $n = 3$ **then**

**return** $\emptyset$

**else if** $n = 4$ **then**

**return** 

**end if**

$Set \leftarrow \emptyset$

**for all** $i$ in (3,n-1) **do**

$A = ListArt(\Phi(i), i)$

**for all** $x$ in $A$ **do**

$B = List(x, n - i + 2)$

**for all** $y$ in $B$ **do**

**if** $x \circ_v y \leq G$

$Set \leftarrow Set \cup \{x \circ_v y\}$

**end for**

**end for**

**end for**

**return** $Set$

**end function**

**Algorithm 11** An algorithm listing all admissible digraphs $H \leq G$ with size $n$.

**Input:** $n > 0$
  **function** LIST($G,n$)
    **if** $n = 0$ **or** $n = 1$ **then**
      **return** $\emptyset$
    **end if**
    $Set \leftarrow \emptyset$
    $A = ListArt(G,n)$
    $B = ListOne(G,n)$
    $Set \leftarrow A \cup B$
    **return** *Set*
  **end function**

We extend $\circ_h$ to sets by taking all possible horizontal composition (Definition 3.8) $x \circ_h y$ for all $x \in ListOne(\Phi(i),i)$ and $y \in List(\Phi(n+1-i),n+1-i)$. The only restriction is horizontal composition of $x$ and $y$ must be less than or equal to $G$. We have to write the base cases because our algorithm is recursive. It is easy to see that there is no admissible digraph of size 2 with at least one articulation. Therefore, for $n \leq 2$ we return the empty set. Also, the only admissible digraph with at least one articulation which has three vertices is line digraph of size three.

We use the vertical composition in **Algorithm 10**. For this algorithm, we consider any 1-connected admissible digraph as the vertical composition of admissible digraph with at least one articulation (such digraphs are indicomposable vertically) and any admissible digraph. Unlike the horizontal composition, it is diffucult to write the set inclusion for the vertical composition. As we mentioned before, vertical composition is commutative operation. Therefore, we have to use graph-ordering to get correct list in this algorithm. The idea comes from the partition of natural numbers. For example, 2-partition of 3 without order are $1+2$ and $2+1$. But if we use increasing order, it is just $2+1$. Hence in this algorithm when we compose two admissible digraphs vertically, the first digraph is always bigger then the second digraph. We first take a digraph, say $x$, from the set of $ListArt(\Phi(i),i)$ and then we choose the second digraph, say $y$, from the set of $List(x,n-i+2)$. Admissible digraphs in the set $List(x,n-i+2)$ are less than or equal to $x$. Therefore, $x$ is always bigger than or equal to $y$. For base cases, if $n = 2$ we take the digraph which has a just one edge, if $n = 4$ we take $C_4$ which has a unique source and a unique sink.

In **Algorithm 11**, we just take the union of one-connected admissible digraphs and admissible digraphs with at least one articulation points. Therefore, we find the list of all admissible digraphs.

# BIBLIOGRAPHY


[1] Cunningham, W.H., Edmonds, J., "A Combinatorial Decomposition Theory," *Canadian Journal of Mathematics*, pp. 734-765, no. 3, 1980.

[2] Enomoto H., "Graph Decomposition without Isolated Vertices", *Theoretical Computer Science*, pp. 111-124, no. 1, 1995.

[3] Gessel Ira M., "Enumerative Applications of a Decomposition for Graphs and Digraphs", *Discrete Mathematics*, pp. 257-271, no. 1-3, 1992.

[4] Harary F., Palmer E.M.,"Graphical enumaration", *Academic Press*, 1973.

[5] Kreutzer S., Ordyniak S.,"Digraph Decompositions and Monotonicity in Digraph Searching", *Theoretical Computer Science*, pp. 4688-4703, no. 35, 2011.

[6] Linial N., "Graph Decomposition without Isolates", *Theoretical Computer Science*, pp. 16-25, no. 1, 1984.

[7] Liscovec V.A., "Enumeration of Rooted Digraphs With a Source", *Vesci Akad. Navuk BSSR Ser. Fiz.-Mat. Navuk*, pp. 35-44, no. 3, 1973.

[8] Robinson R.W., "Counting Labeled Acyclic Digraphs", *New directions in the theory of graphs*, pp. 239-273, no. 1-3, 1973.

[9] Robinson R.W., "Counting Digraphs with Restrictions on the Strong Components", *Combinatorics and graph theory '95*, ,Vol. 1, pp. 343-354, 1995.

# APPENDICES

## A.The python codes for hasse diagrams and transitive closure

We use **Python** in all codes as our programming language.

```
import igraph
```

Moreover, we will use `igraph`, a python package for basic operations on graphs.

```
def hasse(G):
    for x in G.get_edgelist():
        G.delete_edges([x])
        u = G.get_shortest_paths(x[0],x[1])
        if len(u[0]) == 0:
           G.add_edges([x])
    return G
```

This code finds the hasse diagram of a given digraph. The command `get_edgelist()` gives us the edge list of a digraph as the tuple of vertices. `G.get_shortest_paths(`$v_1, v_2$`)` finds the shortest path between two vertices. First of all, we select an edge from the digraph, we mark the source and the sink of this edge and we delete this edge. Then we find the shortest path between such source and such sink. If this path has length zero which means that there is no path between these vertices, we add the deleted edge. If there is a path between these vertices, we do not add that edge.

```
def tranClosure(G):
    for i in G.vs():
        for j in G.vs():
            x= G.vs()[i]
            y= G.vs()[j]
```

```
            u = G.get_shortest_paths(x,y)
            if len(u[0]) > 2:
                G.add_edges([(x,y)])
    return G
```

We find the transitive closure of a given digraph by using above code. `G.vs()` gives as the vertex set of $G$. We consider all paths between any two vertices by means of a double `for` loop. If the shortest path between two vertices is greater than 1 which means that there is no edge between these vertices but there is a path, then we add an edge between such vertices.

## B.The python code for counting algorithms

```
def S(n,k):

    if n <= 0 or k < 0:
        result=0
    elif n <= k+1:
        result=0
    elif n == (k+2):
        result=1
    elif k == 0:
        result=G(n)
    else:
        result=0
    for m in range(2,n):
        (result)=(result) + G(m)*S(n+1-m,k-1)
    return (result)
```

The function `S(n,k)` finds the number of admissible digraphs of $n$ vertices with $k$ articulations. You can see the details of code on page 17 in **Algorithm 3**.

The `range(2,n)` command gives us this list $2,3,\cdots,n-1$. We also use this command in the code below.

```
def G(n):

    if n <= 1:
       result=0
    elif n == 2:
       result=1
    elif n == 3:
       result=0
    elif n == 4:
       result=1
    else:
```

```
        result=0
    for m in range(3,n-1):
        for l in range(1,m-1):
            (result)=(result)+S(m,l)*U(n+2-m)
    return (result)
```

The function `G(n)` finds the number of 1-connected admissible digraphs of $n$ vertices. (For details see the page 18 in **Algorithm 4**)

```
def U(n):

    if n <= 1:
        result=0
    else:
        result=0
    for k in range(0,n-1):
        (result)=(result) + S(n,k)
    return (result)
```

The function `U(n)` finds the number all admissible digraphs of $n$ vertices. (For details see the page 18 in **Algorithm 5** )

We use memoization to count admissible digraph to speed up the calculation. In computing, memoization is an optimization technique used primarily to speed up calculation by having function calls avoid previously calculated results. `buffer` variables (which are python dictionaries) provide us to memoize the results. When we used memoization, we found the results faster.

```
bufferS = dict()
bufferG = dict()
bufferU = dict()

def S(n,k):
   try: result = bufferS[(n,k)]
   except:
      if n <= 0 or k < 0:
           result=0
      elif n <= k+1:
           result=0
      elif n == (k+2):
           result=1
      elif k == 0:
           result=G(n)
      else:
           result=0
           for m in range(2,n):
              result=result + G(m)*S(n+1-m,k-1)

   bufferS.update({(n,k):result})
   return(result)



def G(n):
   try: result = bufferG[n]
   except:
      if n <= 1:
         result=0
```

```
        elif n == 2:
            result=1
        elif n == 3:
            result=0
        elif n == 4:
            result=1
        else:
            result=0
            for m in range(3,n-1):
                for l in range(1,m-1):
                    result=result + S(m,l)*U(n+2-m)

    bufferG.update({n:result})
    return(result)



def U(n):
    try: result = bufferU[n]
    except:
        if n <= 1:
            result=0
        else:
            result=0
            for k in range(0,n-1):
                result=result + S(n,k)

    bufferU.update({n:result})
    return(result)
```

## C.The python code for listing algorithms

Now, we are going to write the code for list algorithms. We use some new funtions in this code. `product(f, A, B)` applies a function $f(x,y)$ to all $x$ in $A$ and all $y$ in $B$ and the resulting set. `hcat(G1,G2)` composes two admissible digraph horizontally as we defined in Remark 5.2 and `vcat(G1,G2)` composes two admissible digraph vertically as we defined in Remark 5.2. (`lshift(G,n)` and `rshift(G,n)` are used in the function `vcat()` in order to satisfy labelling correctly)

```
def product(f, A, B):
    return [f(x,y) for x in A for y in B]

def hcat(G1,G2):
    N = max(G1)
    return G1 + map(lambda x: x+N,G2)

def lshift(G,n):
    N = max(G)
    return map(lambda x: x if x<N else x+n-1,G)

def rshift(G,n):
    return map(lambda x: x+n-1 if x>0 else x,G)

def vcat(G1,G2):
    return lshift(G1,max(G2)) + rshift(G2,max(G1))

def listS(n,k):

    if n <= 1 or k < 0:
        return []
    elif n <= k+1:
        return []
    elif k == 0:
        return listG(n)
    elif n == k+2:
```

```
    tupe = []
    for i in range(n-1):
        tupe += [i,i+1]
    return [tupe]
    elif n == k+1:
        return []

    else:
         temp = []
         for m in range(2,n):
            temp += product(hcat,listG(m),listS(n+1-m,k-1))
         return temp


def listG(n):

    if n <= 1:
        return []
    elif n == 2:
        return [[0,1]]
    elif n == 3:
        return []
    elif n == 4:
        return [[0,1,0,2,1,3,2,3]]
    else:
        temp = []
        for m in range(3,n-1):
            for l in range(1,m-1):
                temp += product(vcat,listS(m,l),listU(n+2-m))
        return temp


def listU(n):

    if n <= 1:
        return []
```

```
else:
    temp = []
    for k in range(0,n-1):
        temp += listS(n,k)
    return temp
```

We again use memoization in the code below to get the list faster. The code here is memoized version of the code in Appendix C.

```
bufferS = dict()
bufferG = dict()
bufferU = dict()


def listS(n,k):
    try: temp = bufferS[(n,k)]
    except:
    if n <= 1 or k < 0:
        temp = []
    elif n <= k+1:
        temp = []
    elif k == 0:
        temp = listG(n)
    elif n == k+2:
    temp = []
    for i in range(n-1):
        temp += [i,i+1]
    return [temp]
    elif n == k+1:
        temp = []

    else:
         temp = []
         for m in range(2,n):
            temp += product(hcat,listG(m),listS(n+1-m,k-1))

    bufferS.update({(n,k):temp})
    return(temp)


def listG(n):
```

```
    try: temp = bufferG[n]
    except:
    if n <= 1:
        temp = []
    elif n == 2:
        temp = [[0,1]]
    elif n == 3:
        temp = []
    elif n == 4:
        temp = [[0,1,0,2,1,3,2,3]]
    else:
        temp = []
        for m in range(3,n-1):
            for l in range(1,m-1):
                temp += product(vcat,listS(m,l),listU(n+2-m))

    bufferG.update({n:temp})
    return(temp)


def listU(n):
    try: temp = bufferU[n]
    except:
    if n <= 1:
        temp = []
    else:
        temp = []
        for k in range(0,n-1):
            temp += listS(n,k)
    bufferU.update({n:temp})
    return(temp)
```

## D.The python code for listing with order algorithms

In this code, we use order relation between admissible digraphs in order to prevent double counting. The reason why we need this version is explained in conclusion on page 20. While defining order, we have to use some functions.

```
def tograph(a):
    N=len(a)
    H=igraph.Graph((N/2)+1)
    H.to_directed()
    for i in range(0,N,2):
        H.add_edge(a[i],a[i+1])
    return H
```

The function `tograph()` takes a sequence which is our notation of digraphs and turns it to a digraph.

```
def tosequence(g):
    N=2*len(g.get_edgelist())
    n=0
    A=range(N)
    for e in g.get_edgelist():
        A[n]= e[0]
        n=n+1
        A[n]= e[1]
        n=n+1
    return A
```

The function `tosequence()` takes a digraph and turns it to a sequence. There is a difference between the notation for digraphs in `igraph` and our notation for digraphs. Functions `hcat()` and `vcat()` cannot work for the notation of digraphs in `igraph`. Therefore, we need functions `tograph()` and `tosequence()` to change the notation.

```
def phi(n):
```

```
    G= igraph.Graph(n)
    G.to_directed()
    for i in range(1,n-1):
        G.add_edges([(0,i)])
        G.add_edges([(i,n-1)])
    return tosequence(G)
```

`phi(n)` is a function which gives us the largest admissibble digraph with $n$ vertices, i.e. $\Phi(n)$. (See the page 26)

```
from operator import itemgetter, attrgetter
```

We will need the command `ittemgetter` since this command help us in ordering which we defined on Definition 5.2.

```
def compare_graph(A,B):
    G = tograph(A)
    H = tograph(B)
    e1=G.ecount()
    e2=H.ecount()
    v1=G.vcount()
    v2=H.vcount()
    if v1 < v2:
       return -1
    elif v2 < v1:
       return 1
    elif v1 == v2 and e1 < e2:
       return -1
    elif v1 == v2 and e2 < e1:
       return 1
    else:
       Glist=sorted(G.get_edgelist(), key=itemgetter(0))
       Hlist=sorted(H.get_edgelist(), key=itemgetter(0))
       if Glist < Hlist:
          return -1
```

```
elif Hlist < Glist:
   return 1
elif Hlist == Glist:
   return 0
```

`compare_graph(A,B)` takes two sequences. Then, by using the fuction `tograph()`, it turns sequences to digraph and it compares them.

```
def ListArt(G,n):
    if n <= 1:
       return []
    elif n ==2:
       return []
    elif n ==3:
       return [[0,1,1,2]]
    else:
       temp = []
       for i in range(2,n):
           A=ListOne(phi(i),i)
           B=List(phi(n-i+1),n-i+1)
           for x in A:
   for y in B:
   H=hcat(x,y)
   if compare_graph(G,H)== 1:
   temp += [ H ]
       return temp
```

The function `ListArt(G,n)` finds the set of all admissible digraphs (less than or equal to a given digraph) of size $n$ with at least one articulation. Remember that we consider admissible digraph with at least one articulation as the horizontal composition of 1-connected admissible digraph and any admissible digraph of apprapriate sizes. Therefore in this fuction, we use the command `hcat()`. (For the details of algorithm see the page 30)

```
def ListOne(G,n):

    if n <= 1:
        return []
    elif n == 2:
        return [[0,1]]
    elif n == 3:
        return []
    elif tograph(G).vcount()>=4 and n == 4:
        return [[0,1,0,2,1,3,2,3]]
    else:
        temp = []
```

```
        for i in range(3,n):
            A=ListArt(phi(i),i)
            for x in A:
                B = List(x,n-i+2)
                for y in B:
                    H=vcat(x,y)
                    if compare_graph(G,H)== 1:
                       temp += [ H ]
    if G == phi(n):
    temp += [G]
    return (temp)
```

The function `ListOne(G,n)` gives us the set of all 1-connected admissible digraphs of size $n$ which are less than or equal to a given digraph. And we use `vcat()` since we consider any 1-connected admissible digraph as the vertical composition of admissible digraph with at least one articulation and any admissible digraph. (For the details of algorithm see the page 32)

```
def List(G,n):

    if n <= 1:
        return []
    else:
        temp = []
        A=ListArt(G,n)
        B=ListOne(G,n)
        temp = A + B
        return temp
```

The function `List(G,n)` lists the set of all admissible digraphs of size $n$ which are less than or equal to a given digraph. Here, + uses for union of lists. (For the details of algorithm see the page 33)