eman ta zabal zazu

Universidad del País Vasco / Euskal Herriko Unibertsitatea

Informatics Engineering Degree

Computation

Bachelor Thesis

About Tree-Depth

Author

Asier Mujika

Supervisor

Dr. Hubert Chen

informatika fakultatea / facultad de informática

2015

Acknowledgements

I would like to express my gratitude to my supervisor Hubie Chen for the useful comments, engagement through the learning process and teaching me the rigor that goes into doing good theoretical work. I would also like to thank Montserrat Hermo for introducing me to Hubie and this project and always being willing to offer her support. I would also like to take this chance to thank all the professors that got out of their way to help and motivate me in the last four years. Specially thanks to everyone that made attending the SWERC possible the last three years, it has been an awesome experience. I am also thankful for all the amazing classmates I have had the chance to meet. Finally, I would like to mention Iván Matellanes from whom I have learnt the most in this years and has always been willing to give me advice.

Abstract

In this work I present recent scientific papers related to the concept of tree-depth: different characterizations, a game theoretic approach to it and recently discovered applications. The focus in this work is presenting all the ideas in a self-contained way, such that they can be easily understood with little previous knowledge. Apart from that all the ideas are presented in a homogeneous way with clear examples and all the lemmas, some of which didn't have proofs in the papers, are presented with rigorous proofs.

Contents

1. CHAPTER

Introduction to Graph Theory

1.1 Undirected graphs

An undirected graph is defined as a pair of sets G = (V, E), such that E $\subseteq \{\{a,b\} \mid a \neq b \wedge a,b \in V\}$. The members of V are called vertices or nodes and the ones of E edges. Take into account, that the vertices can be anything, they can even be sets themselves. The usual way to draw a graph is by representing the vertices as individual points and for each edge, draw a link between both elements of that edge. The shape in which a graph is drawn is irrelevant, it will contain the same information.

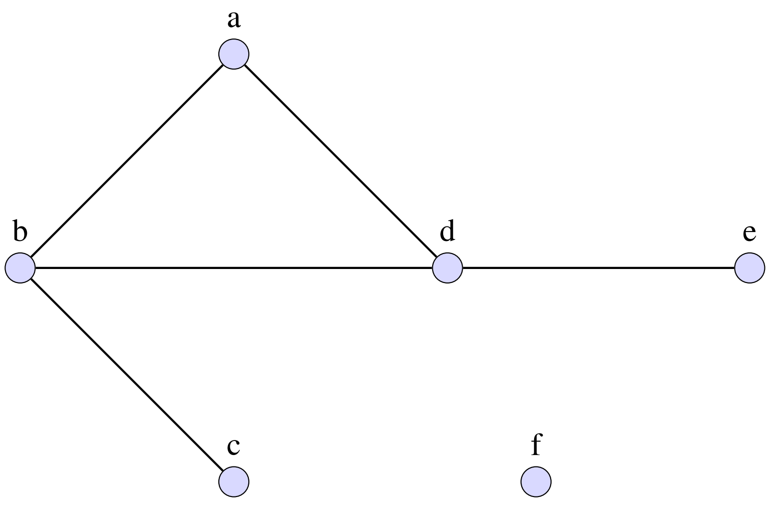

Figure 1.1: An undirected graph with $V = \{a,b,c,d,e,f\}$ and $E = \{\{a,b\},\{a,d\},\{b,d\},\{b,c\},\{d,e\}\}$

1.2 Definitions for undirected graphs

For a graph G, $V(G)$ is its vertex set and $E(G)$ its edge set. Here are some basic concepts in undirected graphs that we will later on need to present more complex ideas.

Adjacency

$a, b \in V(G)$ are said to be adjacent in G if $\{a, b\} \in E(G)$.

Path

A path $a_1, a_2, \ldots, a_n$ is a series of pairwise distinct vertices in $V(G)$ such that if $2 \leq i \leq n$, then a_{i-1} is adjacent to a_i in G. n is the length of such a path.

Cycle

A cycle is a path of the form $a, \ldots, a$ of length greater than 1.

Subgraph

G' is a subgraph of G, expressed as $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. $G' \subset G$ means that $G' \subseteq G$ but $V(G') \neq V(G)$ or $E(G') \neq E(G)$.

Connected component

A connected component G' of G is a subgraph of G such that a path exists between any two vertices of G' and no H exists such that $G' \subset H$ and H is a connected component.

Tree

A tree is a graph with a single connected component that contains no cycle.

Forest

A forest is a graph such that every connected component is a tree.

Rooted Tree

A rooted tree is a tree with a special node that is called the root.

Rooted Forest

A rooted forest is a graph such that every connected component is a rooted tree.

Ancestor

Node x is said to be the ancestor of y in a rooted forest F, if x belongs to the only path between y and the root of the component to which y belongs.

Height of a node

The height of a node in a rooted tree is the length of the path from that node to the root. The height of the root itself is 1. In a rooted forest, the height of a node is its height in the rooted tree it belongs to.

Height of a rooted forest

The height of a rooted forest is the maximum height of any of its nodes.

1.3 Directed graphs

Similar to the notion of undirected graphs, we have directed ones too. In these, the edges are ordered pairs instead of sets and the edges are drawn with arrows from the first element to the second one. Here is an example of one.

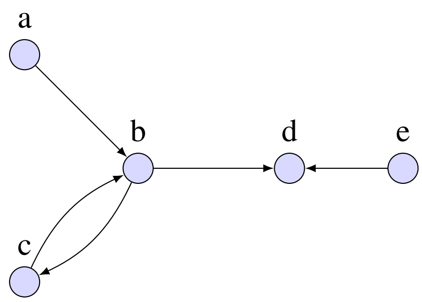

Figure 1.2: A directed graph with $V = \{a,b,c,d,e\}$ and $E = \{(a,b),(b,c),(c,b),(b,d),(e,d)\}$

1.4 Definitions for directed graphs

Just like in the undirected case, we will also need some basic definitions.

Subgraph

The subgraph relation is defined in the same way that in the undirected case.

Path

A path $a_1, a_2, \dots, a_n$ is a series of pairwise distinct vertices in $V(G)$ such that if $2 \leq i \leq n$, then $(a_{i-1}, a_i) \in E(G)$. n is the length of such a path.

Strongly connected component

A strongly connected component G' of G is a subgraph of G such that a path exists between any two vertices of G' and no H exists such that $G' \subset H$ and H is a strongly connected component.

Successor-closed

A subgraph, H, of G is successor-closed in G if there are no edges from H to $G \setminus H$.

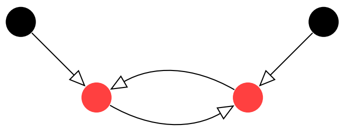

Figure 1.3: The red nodes are both a strongly connected component and a successor-closed subgraph.

2. CHAPTER

Introduction to Tree-Depth

2.1 Tree-Depth

In the following two chapters we will present the ideas from *Sparsity, Algorithms and Combinatorics* [1].

Definition 2.1.1. *The closure of a rooted forest F, expressed as $C = clos(F)$, is defined as follows:*

- $V(C) = V(F)$
- $E(C) = \{\{x, y\} : x \neq y$ *and* x *is an ancestor of* y *in* $F\}$

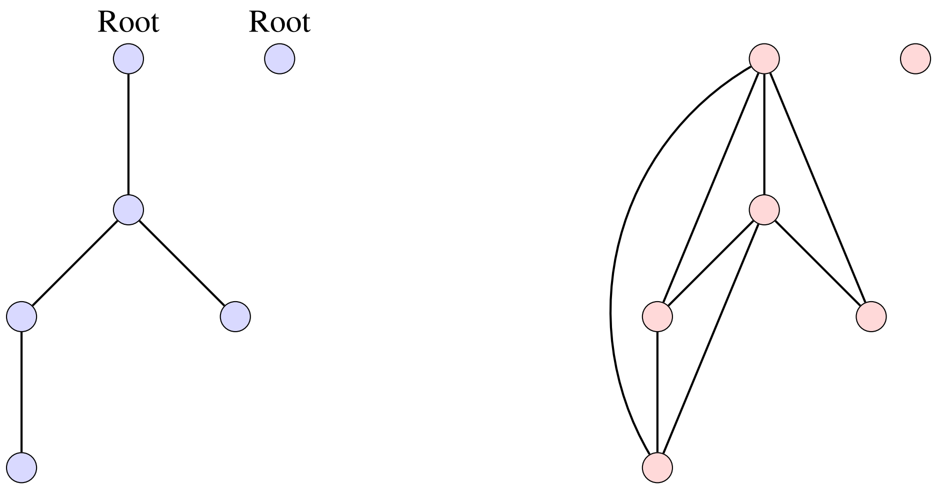

Figure 2.1: The blue graph at the left is a rooted forest F, the red graph at the right represents clos(F).

Definition 2.1.2. *The tree-depth of a graph G, expressed as* $td(G)$*, is the minimum height of a rooted forest* F *such that* $G \subseteq clos(F)$*.*

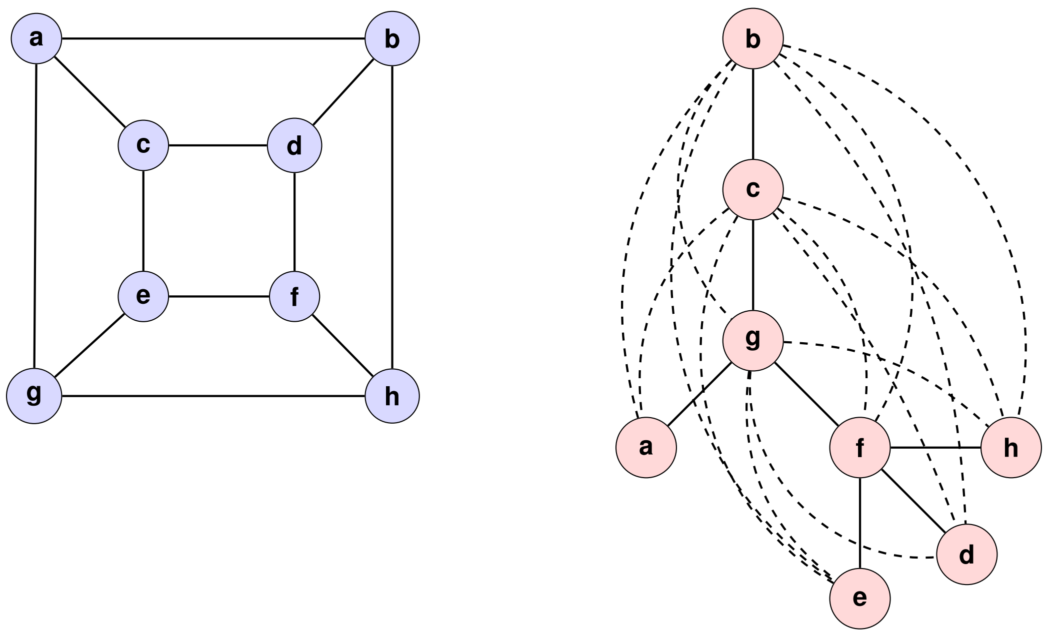

Figure 2.2: The graph G and tree T are in the left and right respectively. The dotted edges in T, represent the $clos(T)$. Because $G \subseteq clos(T)$, $height(T) = 5$ and the definition of tree-depth we just gave, we know that $td(G)$ is at most 5.

The tree-depth of a graph G is a numerical invariant of a graph. In other words, the tree-depth is a property that depends only on the abstract structure of a graph, not on its representation.

2.2 Elimination Forest

Apart from the previous definition we can also characterize tree-depth in a different way by using elimination forests.

Definition 2.2.1. *An elimination forest F of a graph G is defined recursively as follows:*

- *If V(G)={v} then F is just {v}.*
- *If G is not connected, then F is the union of the elimination forests of each component of G.*
- *Otherwise, $r \in V(G)$ is chosen as the root of F and an elimination forest is created for $G - r$. The roots of this elimination forest will be the children of r in F.*

The tree T in Figure 2.2 is an elimination forest for the graph G.

Lemma 2.2.2. *Let G be a graph and F a rooted forest such that $G \subseteq clos(F)$. Then, there exists an elimination forest Y such that $height(Y) \leq height(F)$.*

Proof.

Base case: If $V(G) = \{v\}$, then $V(Y) = \{v\}$ and $height(Y) = 1$. Beware that F can have nodes that are not in G but it must contain v, so $height(Y) \leq height(F)$.

Induction: If G is connected, set the root of F, v, as the root of Y. Clearly, $G - v \subseteq clos(F - v)$, so by induction an elimination forest Y' exists such that $G - v \subseteq clos(Y')$ and $height(Y') \leq height(F - v)$. The roots of Y' will be the children of v in Y and as $G - v \subseteq clos(Y')$, then $G \subseteq clos(Y)$ and Y is an elimination forest. With that we can prove the lemma like this: $height(Y) = 1 + height(Y') \leq 1 + height(\text{F-v}) = height(F)$, so $height(Y) \leq height(F)$.

If G is not connected, then every component G_i in G is contained in the closure of a component F_i in F. Otherwise, the edge between two adjacent nodes in G that are both in G_i but in two different components of F wouldn't be in $clos(F)$ and that can't happen. By induction we can assume that for every component G_i, there exists an elimination forest Y_i such that $G_i \subseteq clos(Y_i)$ and $height(Y_i) \leq height(F_i)$. Y will be the union of all these Y_i which is clearly an elimination forest and because for every component of Y there exists a component in F with higher or equal height, then $height(Y) \leq height(F)$. $\square$

From this lemma it is easy to see that the tree-depth of a graph will be the minimal height of an elimination forest for that graph. Taking that into account, we can use the definition of elimination forest to get this recursive formula.

Lemma 2.2.3. *The tree-depth of a graph G with $G_1, \ldots, G_k$ components is the following:*

$$td(G) = \begin{cases} 1 & \textit{if } |G| = 1 \\ max_{i=1}^{p} td(G_i) & \textit{if } G \textit{ is not connected} \\ 1 + min_{v \in V(G)} td(G - v) & \textit{otherwise} \end{cases}$$

3. CHAPTER

Game Theoretic approach to Tree-Depth

3.1 Defining the game

We will now define a pebble game on a graph G and prove that it is closely related to the concept of tree-depth[1]. Using this different approach we will be able to prove lower bounds for the tree-depth of G and it will also be useful to prove lemmas in the last chapter of this work.

For k $\geq$ 0, the k-step selection-deletion game is played by Alice and Bob on a graph. The game is played by turns as follows:

- First, Alice selects a connected component of the graph, and the rest of the components are deleted.
- Then, Bob deletes a node from the remaining graph and the next round is played with this graph.

If Bob deletes the last node at the k-th round or earlier, he is said to win. Otherwise, Alice wins. Obviously, Alice always wins the 0-step selection-deletion game.

Definition 3.1.1. *A strategy for the game played on graph G is a function from $\mathcal{P}(V(G))$ to $\mathcal{P}(V(G))$. In Alice's case, given the current subgraph the game is being played on the strategy outputs the component Alice will select. In Bob's case, it returns which vertex to* *remove. A strategy is said to be winning in the k-step selection-deletion game if no matter which moves the other player makes you are guaranteed to win in k rounds.*

From this definition we can observe that if Bob has a strategy to win in k rounds that strategy will also guaranty a win in any game that lasts more than k rounds. Conversely, if Alice has a winning strategy in k-rounds, that same strategy will also win any game with less than k rounds.

3.2 Bob's winning strategy

Lemma 3.2.1. *Let G be a graph and let F be a rooted forest of height t such that $G \subseteq clos(F)$. Then, Bob has a winning strategy for the t-step selection-deletion game.*

Proof. Because of lemma 2.2.2 we know an elimination forest Y exists such that $height(Y) \leq height(F)$. Consider $h = height(Y)$, we will prove that a winning strategy exists in h rounds which is also a winning strategy in the t-step selection-deletion game because $h \leq t$.

- **Base case**: If $h = 1$, then every component of G will have a single vertex, so it's clear that Bob will win the 1-step selection-deletion game.
- **Induction**: Let $G_i \subseteq G$ be the component Alice chooses, then Y_i exists such that Y_i is an elimination forest belonging to Y, $G_i \subseteq clos(Y_i)$ and obviously $height(Y_i) \leq h$. Bob will delete v, the root of Y_i. This will leave us with $G' = G_i - v$ as the new graph. If we consider the children of v the new roots in $Y' = Y_i - v$, then $G' \subseteq clos(Y')$ because of how the elimination forests are built. As $height(Y') \leq h - 1$, we can assume by induction that Bob has a winning strategy in $h - 1$ rounds for G', which together with the strategy for the first round we have just defined makes a winning strategy for Bob in the h-step selection-deletion game on the graph G.

□

3.3 Alice's winning strategy

Definition 3.3.1. *A shelter S in a graph G is a set of graphs with the next properties:*

- *$\forall H \in S$, $H \subseteq G$ and H is connected.*
- *H is said to be minimal if no H' exists in S such that $H' \subset H$.*
- *H is said to be maximal if no H' exists in S such that $H \subset H'$.*
- *If $H \in S$ and H is not minimal, then $\forall v \in V(H)$, there exists $H' \subseteq H - v$ such that H covers H'. We will say that $a \in S$ covers $b \in S$ if and only if $b \subset a$, and no $c \in S$ exists such that $b \subset c \subset a$.*

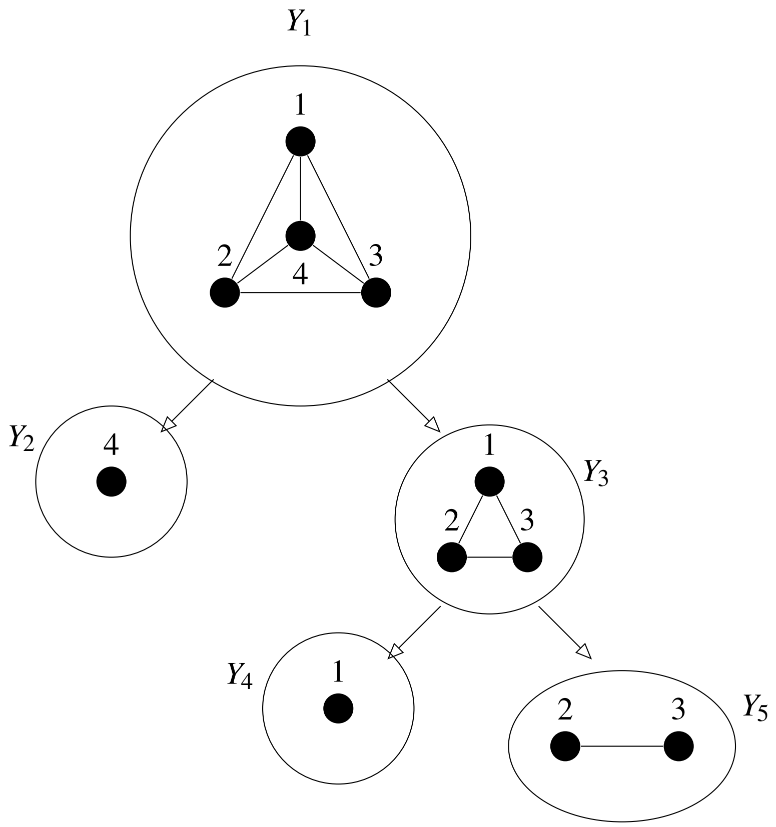

Figure 3.1: An example of a shelter. The arrows represent the covering relation.

The thickness of a shelter S is the shortest sequence of elements in S of the form $a_1, \ldots, a_n$ such that a_1 is maximal and a_n is minimal and if $2 \le i \le n$, then a_{i-1} covers a_i. The length of a chain is defined as the number of elements in it. The thickness of the shelter in figure 3.1 is 2, because of the sequence Y_1, Y_2.

Lemma 3.3.2. *Let G be a graph, S a shelter in G, and t the thickness of S. Then, there exists a winning strategy for Alice in the $(t-1)$-step selection-deletion game.*

Proof. We will proof this by induction over t.

- **Base case**: If $t = 1$, then clearly Alice wins the 0-step selection-deletion game.
- **Induction**: Let H be a maximal element in S. Then, Alice picks the connected component G_i of G, such that $H \subseteq G_i$. Because $t > 0$, H is not minimal, so for any vertex v that Bob removes, if $v \in H$ there exists $H' \in S$ that is covered by H and $v \notin H'$. Otherwise, H is still a subgraph of $G_i - v$. Let $S' = \{X \mid X \in S \wedge X \subseteq G_i - v\}$. It is clear that S' is a shelter for $G_i - v$ and that the thickness of S' is greater than or equal to $t-1$. By induction we can assume Alice has a winning strategy in $t-2$ steps in $G_i - v$, which together with the strategy for the first round we have just defined is a winning strategy for the $(t-1)$-step selection-deletion game.

□

3.4 Relation to Tree-Depth

It is clear that if Alice has a winning strategy in the t-step selection deletion game, Bob can't have a winning strategy in that same game. Because of this and lemmas 3.2.1 and 3.3.2 we can state the following:

Theorem 3.4.1. *Let G be a graph, S a shelter in G of thickness x and F a rooted forest of height y such that $G \subseteq clos(F)$. Then the following is true.*

1. *Alice has a winning strategy in the $(t-1)$-step selection-deletion game, for any t smaller than or equal to x.*
2. *Bob has a winning strategy in the t-step selection-deletion game, for any t greater than or equal to y.*
3. *Every rooted forest who's closure contains G has an height higher than or equal to x. Otherwise, Bob would have a winning strategy in the $(x-1)$-step selection-deletion game, which contradicts statement* 1.
4. *Every shelter in G has a thickness smaller than or equal to y. Otherwise, Alice would have a winning strategy in the y-step selection-deletion game, which contradicts statement* 2.

5. *Because we have F, $td(G) \leq y$. Also, from statement 3 it is clear that $x \leq td(G)$. So we can say that $x \leq td(G) \leq y$.*

With this theorem we can now prove upper-bounds and lower-bounds to a graphs tree depth.

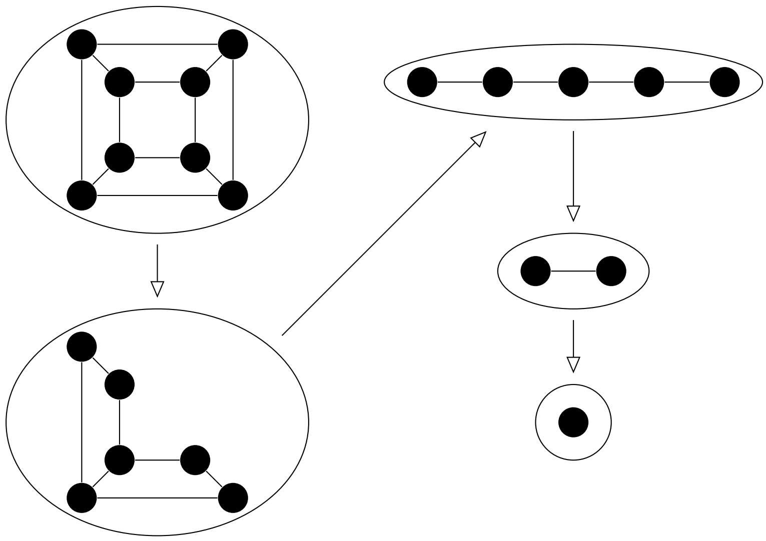

Figure 3.2: This is a shelter of thickness 5 for the graph in Figure 2.2. Beware that not all graphs in the shelter are drawn, but every graph in the shelter is isomorphic to these. With this and the rooted forest from Figure 2.2 we can say that $td(G) = 5$.

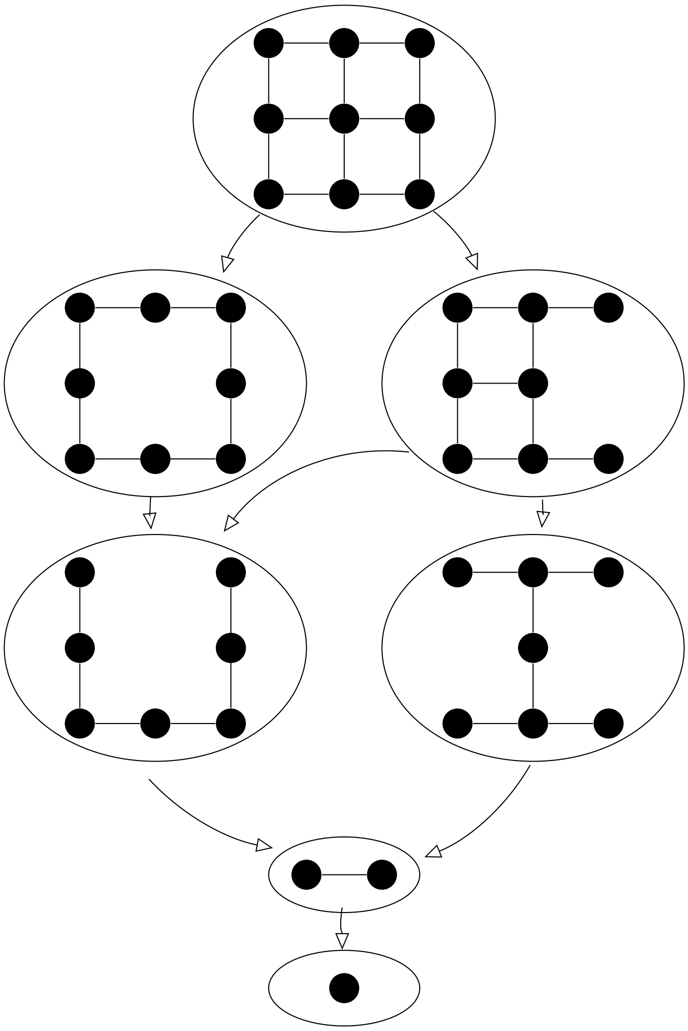

Figure 3.3: Here we see a more complex shelter that proofs that the tree-depth of the graph is at least 5.

4. CHAPTER

Cycle rank

4.1 Defining cycle rank

Cycle rank is a numerical invariant in a directed graph which is closely related to the tree-depth of an undirected graph. In this chapter and the next one we will present the main ideas of the paper LIFO-search [2].

Definition 4.1.1. *The cycle rank of a digraph $G = (V, E)$, denoted by $r(G)$ is defined as follows:*

- *If $|V| = 1$, then $r(G) = 0$.*
- *If G is strongly connected and $|V| > 1$, then $r(G) = 1 + min_{v \in V} \{r(G - v)\}$.*
- *If G is not strongly connected, then $r(G)$ is the maximum cycle rank among all strongly connected components of G.*

4.2 Directed elimination forest

Similar to the notion of elimination forests in undirected graphs, we have directed elimination forests on digraphs.

Definition 4.2.1. *A directed elimination forest for a digraph G is a rooted forest F. F can be defined recursively as follows:*

- *For the $k \geq 0$ strongly connected components of G with size strictly greater than 1, $Y_1, \ldots, Y_k$, (v_i, Y_i) are the roots in F, where $v_i \in Y_i$ and $1 \leq i \leq k$.*
- *For each (v_i, Y_i), a directed elimination forest is created for $G[Y_i]$ - v_i and the roots of that forest are the children of (v_i, Y_i) in F.*

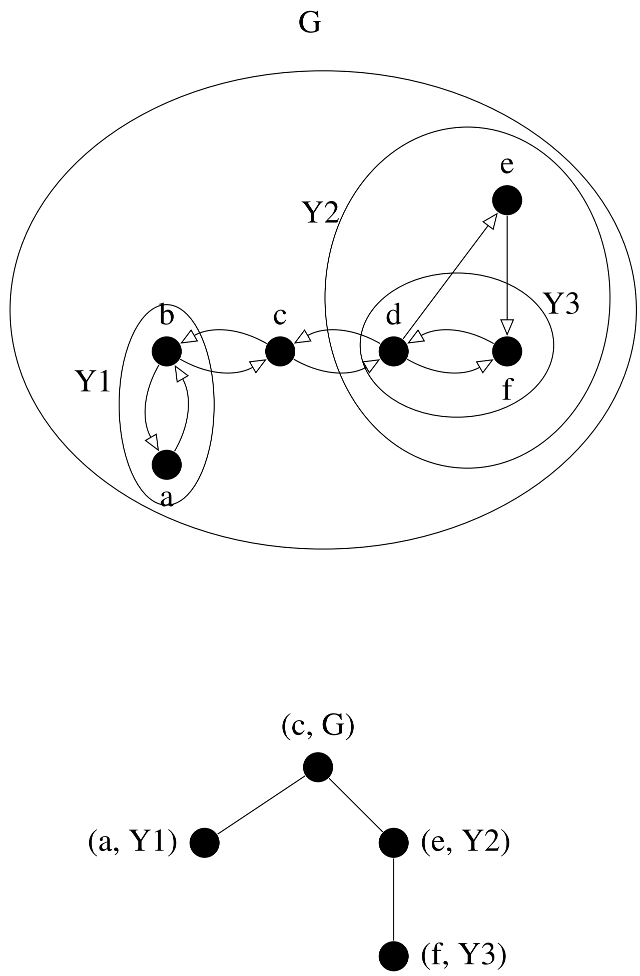

Figure 4.1: This is an elimination forest for the graph G. The circles represent the subgraphs that are used for the nodes of the elimination forest.

Lemma 4.2.2. *Let F be directed elimination forest of minimum height for a digraph G = (V, E). Then, r(G) = height(F).*

Proof. We will proof this by induction on the number of vertices of G.

- **Base case**: If $|V|$ = 1, then r(G) = 0. height(F) is 0 because we assume that the height of the empty tree is 0.

- **Induction**: If G is strongly connected and $|V| > 1$, then $v \in V$ exists, such that r(G) = 1 + r($G - v$). Let (v, V) be the root of F, then height(F) = 1 + height(F') where F' is any directed elimination forest of $G - v$ because of definition 4.2.1. If we consider F' to be the directed elimination forest of $G - v$ of minimum height, by induction we can assume that r($G - v$) = 1 + height(F'). So, r(G) = 1 + r($G - v$) = 1 + height(F') = height(F).

  If G is not strongly connected but it has at least a cycle, then, for every X that is a strongly connected component of G by induction we can assume that r(G[X]) = height(F_X) where F_X is the directed elimination tree of minimum height for G[X]. Because r(G) is the maximum among all r(G[X]) and the height(F) is the maximum among all height(F_X), r(G) = height(F).

$\square$

5. CHAPTER

Game Theoretic approach to Cycle Rank

5.1 Definitions

For this section, we will assume all our graphs are directed and contain no self loops. We will also need some basic definitions about strings before we can start talking about the games we will use to define cycle rank.

String

A string is a sequencce of elements $a_1, \ldots, a_n$ such that all a_i belong to the same set. That set is called the alphabet.

Length

The lenght of a string A = $a_1, \ldots, a_n$, denoted by $|A|$ is n. The length of the empty string is 0.

Concatenation

The concatenation of two string A = $a_1, \ldots, a_n$ and B = $b_1, \ldots, b_k$, denoted by A·B is $a_1, \ldots, a_n, b_1, \ldots, b_k$

V*

V* is the set of all possible finite words over the set V, including the empty word.

Prefix

X $\in$ V* is a prefix of Y $\in$ V*, denoted by X$\preceq$Y if Z $\in$ V* exists such that Y = X·Z.

String to set

For a string S = $a_1, \ldots, a_n$, $\{|S|\}$ denotes the set $\{a_1, \ldots, a_n\}$.

5.2 Game description

As the game is quite complicated and has a few variations we will first present it in an informal way so that the reader can get an intuitive idea of what is going on. Later on we will make a formal definition of the game.

We will use a cops and robbers game played on a graph G, where the cops will try to catch a robber. In each step of the game the cops can either place a cop on a node or remove only the most recently placed one. This is why it's called a LIFO search. The cops win if the manage to place a cop in the same node where the robber is.

There are four variants of the game depending on how the robber moves and which information do the cops have.

Invisible - i

The cops don't know the position where the robber is located and he can move along directed paths in G that contain no cops.

Visible - v

The cops know the position where the robber is located and he can move along directed paths in G that contain no cops.

Invisible strongly connected - isc

The cops don't know the position where the robber is located and he can only move inside the same strongly connected component of G that contain no cops.

Visible strongly connected - isc

The cops know the position where the robber is located and he can only move inside the same strongly connected component of G that contain no cops.

We will now proceed to formalize the game we have presented.

For a digraph G, the state of the game is described by a pair (X,R). $X \in V*$ is the position of the cops and the order in which they were added. R is an induced subgraph of $G\backslash\{|X|\}$. In the invisible variants, R represents where the robber may be, while in the visible variants its means which nodes can the robber reach. We will define the valid states for each game variant.

i-state

R is successor closed in $G\backslash\{|X|\}$. If R wouldn't be successor closed the robber would have an edge without cops which he could use to scape R and R wouldn't represent all possible positions of the robber.

v-state

R is successor closed in $G\backslash\{|X|\}$ and $v \in V(R)$ exists such that a directed path exist from v to any other node in $V(R)$.

isc-state

R is a union of strongly connected components of $G \setminus \{|X|\}$.

vsc-state

R is a single strongly connected component of $G \setminus \{|X|\}$.

Let (X,R) be the current state of the game and (X',R') a valid successor (a possible next state). Then, $|\{|X|\} \cap \{|X'|\}| = 1$ and $|X| \preceq |X'|$ or $|X'| \preceq |X|$. R is defined differently for different game variants.

- In the i and v variants, for every $v' \in V(R')$ there exists a $v \in V(R)$ such that a path exists from v to v' in $G \setminus (\{|X|\} \cap \{|X'|\}|)$.
- In the isc and vsc variants, for every $v' \in V(R')$ there exists a $v \in V(R)$ such that v and v' are contained in the same strongly connected component of $G \setminus (\{|X|\} \cap \{|X'|\}|)$.

The initial state of a game in the invisible variants is clearly (ε, G). In the visible variants this is not necessarily a valid state, so the initial state will be any valid position of the form (ε, R). A strategy for the cops is a function that given a game state (X,R) returns X', the position the cops will take in the next state. A strategy is said to be a winning strategy if no matter which moves the robber makes the strategy reaches a state of the form $(X,\emptyset)$ from any possible initial state.

For every previously mentioned game variants we can create a new monotone variant (mi, mv, misc, mvsc). The monotone variant of each game is equal to the non monotone one, except that for every position (X_i, R_i) and its successor (X_{i+1}, R_{i+1}), the cops strategy must ensure that R_{i+1} is a subgraph of R_i no matter what the robber does.

We are interested in the minimum number of cops necessary to capture the robber. For any game variant, gv $\in$ {i, v, isc, vsc, mi, mv, misc, mvsc}, we will call LIFO$^{gv}(G)$ the minimum number of cops needed to capture a robber in G in that game variant. We will also define one more game called searcher stationary vsc, which is equal to the LIFO vsc but for every X_i,$X_i \prec X_{i+1}$, i.e, cops can only be added, not removed. SS^{vsc} will be the minimum number of cops needed in this strategy.

Theorem 5.2.1. *For any digraph G the same number of cops are needed to capture a robber in every game variant and that number is equal to the cycle rank of G plus 1:*

$1 + r(G) = LIFO^{mi}(G) = LIFO^{i}(G) = LIFO^{misc}(G) = LIFO^{isc}(G) = LIFO^{mv}(G) = LIFO^{v}(G) = LIFO^{mvsc}(G) = LIFO^{vsc}(G) = SS^{vsc}(G)$.

Observation 5.2.2. *There are some trivial relations between these games*

- *Every monotone winning strategy is also a winning strategy in the non monote variant of that same game.*

- *Every winning strategy for an invisible game variant is also a winning strategy for the visible variant of that same game.*

- *Every winning strategy for when the robber is not restricted to only move in strongly connected components is also winning when the robber is restricted to only move in strongly connected components.*

With this observation we can build the following figure.

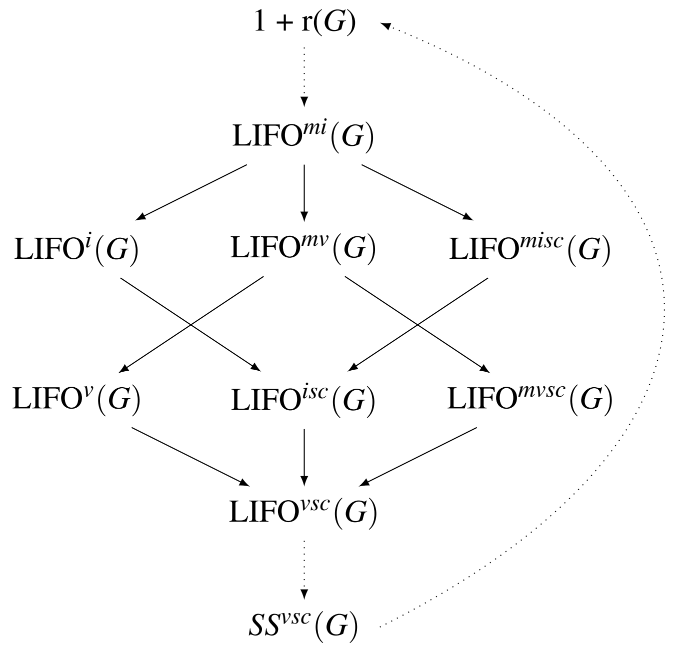

Figure 5.1: The arrows go from the bigger values to the smaller ones. We have the normal arrows from the previous observation. We will prove the doted arrows and that will prove that the lemma holds.

Lemma 5.2.3. *For any digraph G, $LIFO^{vsc}(G) \geq SS^{vsc}(G)$.*

Proof. We will proof this by contradiction. We will assume a LIFO strategy exists for the vsc game that is not searcher stationary. Let (X_2, R_2) be the first state where a cop will be removed and (X_1, R_1) and (X_3, R_3) the previous and next state respectively. Based on this we can conclude the following:

- Because it is a LIFO game, we will remove the most recently added cop, so it's easy to see that $X_1 = X_3$.
- Using the fact that (X_2, R_2) is a valid successor of (X_1, R_1) every node in R_2 is in the same strongly connected component of a node in R_1.

Taking advantage of this the robber can move back to R_1 in the last state, so the robber can force a situation where $(X_1, R_1) = (X_3, R_3)$. In this way the cops strategy will loop and it will never catch the robber, a contradiction to the initial assumption. □

The proof for the other two lemmas were specially hard to follow so we present them using the method for structured proofs presented by Leslie Lamport [3]. This made them much easier to understand and check for correctness.

Lemma 5.2.4. *For any digraph G, $SS^{vsc}(G) \geq 1 + r(G)$.*

Proof. We will prove this by induction over the number of vertices of G.

1. If $|V(G)| = 1$, $SS^{vsc}(G) = 1 + r(G) = 1$.
   PROOF: A cop in the single node of G will always capture the robber and $|V(G)| = 1$, so $r(G) = 0$ by definition.

2. Assume that for every G' such that $|V(G')| < |V(G)|$, $SS^{vsc}(G') \geq 1 + r(G')$.
   PROOF: Induction hypothesis, we can assume it because 1.

3. If G is not strongly connected, then $SS^{vsc}(G) \geq 1 + r(G)$

   3.1. G has $k > 1$ strongly connected components $H_1, \ldots, H_k$ and for every H_i, $|V(H_I)| < |V(G)|$.
   PROOF: In 3 we assume G is not strongly connected.

   3.2. $SS^{vsc}(G) \geq SS^{vsc}(H_i)$ such that H_i is a strongly connected component of G.
   PROOF: $SS^{vsc}(G)$ must have a winning strategy for any strongly connected component the robber may start in.

   3.3. $\max_{H_i} SS^{vsc}(H_i) \geq \max_{H_i} (1 + r(H_i))$.
   PROOF: By 2 and 3.1.

   3.4. $SS^{vsc}(G) \geq \max_{H_i} SS^{vsc}(H_i) \geq \max_{H_i} (1 + r(H_i)) = 1 + r(G)$.
   PROOF: The first equality by 3.2. The second inequality by 3.3. The last one by definition of cycle rank.

4. If G is strongly connected, then $SS^{vsc}(G) \geq 1 + r(G)$

   4.1. Let ϕ be a minimal strategy, that uses $SS^{vsc}(G)$ cops and $v = \{|\phi(\varepsilon, G)|\}$.
   PROOF: By 4, (ε, G) is the initial state, so v exists.

   4.2. $SS^{vsc}(G) = 1 + SS^{vsc}(G - v)$.
   PROOF: By 4.1 ϕ induces a winning strategy for $SS^{vsc}(G - v)$ using $SS^{vsc}(G)$-1 cops.

4.3. SS^{vsc}(G) = 1 + SS^{vsc}(G - v) $\geq$ 2 + r(G-v) $\geq$ 1 + r(G).
PROOF: The first inequality by 4.2. The second inequality by 2. The last one is by the definition of cycle rank, as r(G) is the smallest r(G-u) + 1, then 1+ r(G-u) $\geq$ r(G) for any u $\in$ V(G).

5. Q.E.D.
PROOF: By 3 and 4.

□

Lemma 5.2.5. *1 + r(G) $\geq$ LIFO^{mi}(G)*

Proof. We will prove this by induction over the number of vertices of G.

1. If $|$V(G)$|$ = 1, LIFO^{mi}(G) = 1 + r(G) = 1.
PROOF: A cop in the single node of G will always capture the robber and $|$V(G)$|$ = 1, so r(G) = 0 by definition.

2. Assume that for every G' such that $|$V(G')$|$ $<$ $|$V(G)$|$, 1 + r(G') $\geq$ LIFO^{mi}(G').
PROOF: Induction hypothesis.

3. If G is strongly connected, 1 + r(G) $\geq$ LIFO^{mi}(G).

3.1. v $\in$ V(G) exists, such that r(G) = 1+ r(G-v).
PROOF: By definition of cycle rank.

3.2. 1 + LIFO^{mi}(G-v) $\geq$ LIFO^{mi}(G).
PROOF: We can place a cop in v in the first step of the game, never remove it and win the game in G using 1 + LIFO^{mi}(G-v) cops.

3.3. 1 + r(G) = 2 + r(G-v) $\geq$ 1 + LIFO^{mi}(G-v) $\geq$ LIFO^{mi}(G).
PROOF: By 3.1, 2 and 3.2

4. If G is not strongly connected, 1 + r(G) $\geq$ LIFO^{mi}(G).

4.1. G has k $>$ 1 strongly connected components $H_1, \ldots, H_k$ and for every H_i, $|$V(H_i)$|$ $<$ $|$V(G)$|$.
PROOF: In 4 we assume G is not strongly connected.

4.2. A strongly connected component H_i exists such that there is no edge from $G \setminus H_i$ to H_i.
PROOF: By 4.1 G is not strongly connected, so H_i must exist.

4.3. LIFO^{mi}(G) $\geq$ max(LIFO^{mi}($G \setminus H_i$), LIFO^{mi}(H_i)).
PROOF: The cops can search the robber only in H_i. Then, they can remove every cop in H_i and search only in $G \setminus H_i$. The robber will never be able to go back to H_i because of 4.2

4.4. LIFO^{mi}(G) $\geq$ max(LIFO^{mi}($G \setminus H_i$), LIFO^{mi}(H_i)) $\geq$ max(1+r($G \setminus H_i$), $1 +$ r(H_i)) = 1 + r(G).
PROOF: The first inequality by 4.3. The second one by 2. The last equality by the definition of cycle rank and the assumption that G is not strongly connected.

5. Q.E.D.
PROOF: By 3 and 4.

□

6. CHAPTER

Isomorphism

6.1 Problem definition

In the isomorphism problem we have to determine for two given graphs G and H if there exists a bijection ϕ from $V(G)$ to $V(H)$ such that $\forall u, v \in V(G),\ \{u, v\} \in E(G) \iff \{\phi(u), \phi(v)\} \in E(H)$.

Unless stated otherwise, in this chapter we will deal only with connected graphs. At the end we will show how to extend such an algorithm to arbitrary graphs. The lemmas and algorithms presented in this chapter are an adaptation of the ones presented in the paper *On tractable Parameterizations of Graph Isomorphism* [4].

6.2 Parameterized complexity

In classical complexity we only take into account the length of the input to study the complexity of a problem. Unfortunately, many problems become intractable under this measure with the best algorithms we know. This is the case of both the famous NP-complete problems and the graph isomorphism problem. The fastest know exact algorithms for these run in exponential and sub-exponential time respectively.

In parameterized complexity on the other hand, we don't only take into account the size of the input, but also a parameter about the input, e.g the tree-depth of the input graph in the isomorphism problem. An interesting complexity class is the Fixed Parameter Tractable. We consider a problem to be in this class, if we can find a solution in time $\mathcal{O}(f(d) * n)$, where n is the size of the input, d is the parameter and f a computable function. Beware, that such a function can be exponential or even worse.

In this chapter we will show that such an algorithm exists for the isomorphism problem parameterized by the tree-depth of the graphs. This means that for any class of graphs with a bounded tree-depth, we can consider $f(d)$ to be a constant and consequently, solve the problem in polynomial time with respect to their size.

6.3 Bounded roots

Definition 6.3.1. *Let G be a connected graph with tree-depth d. Then,* $root(G) = \{v \in G : td(G-v) = d-1\}$ *is the set of roots of G.*

For the algorithm we will need to proof that $|root(G)|$ is bounded by a function of the tree-depth of G. Still, this is not a simple proof and we will need to introduce new concepts and lemmas.

Definition 6.3.2. *Let G be a connected graph and B a subset of the nodes of G. For two connected components of* $G \setminus B$, C_1 *and* C_2, *we will say* C_1 *and* C_2 *are equivalent in G with respect to B if and only if the following holds:*
There exists a bijection $\Phi : C_1 \cup B \longrightarrow C_2 \cup B$ *such that* $\forall b \in B$, $\Phi(b) = b$ *and* $\forall u, v \in V(C_1) \cup V(B)$, $\{u,v\} \in E(G) \iff \{\Phi(u), \Phi(v)\} \in E(G)$.

We can visualize this equivalence relation as two components being isomorphic and connected in the same way to the set B.

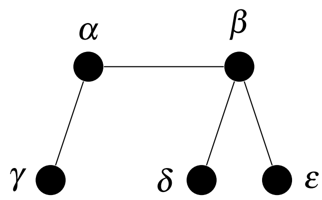

Figure 6.1: Let the graph in the image be G and $B = \{\alpha, \beta\}$. The component of the node δ in $G \setminus B$ and the one of node ε are equivalent with respect to B, but they are not equivalent to the one consisting of γ.

Lemma 6.3.3. *Let G be a connected graph with td(G) = d and B a subset of the nodes of G. Let $C_1, C_2, \ldots, C_k$ be equivalent components in G with respect to B. Let G' be the graph left after we remove all the C_i with $i > d$. Then, td(G) = td(G') and root(G) $\subseteq G'$.*

Proof. To prove this we will use the game characterization we defined for tree-depth. Using definition 6.3.2 we can see that, if a node in B is connected to a node in a C_i, then that node should be connected to a node in every C_i. Without lose of generality, we will assume that every node in B is connected to a node in a C_i. We can assume this because if we remove nodes from B that are not adjacent to any C_i, the C_i will still be equivalent in G with respect to the new B.

From theorem 3.4.1 we know that Alice will have a winning strategy in $d-1$ rounds in G. We will show that Alice can mimic this strategy in G'. It is clear that B together with all the C_i will be a connected component until every node from B is removed. Thus, Alice can always select the same component she would in G until Bob has removed every node in B. If Bob has removed every node in B, the remaining C_i's become disconnected. Because Bob will have removed $d-1$ nodes at most, if we have left d C_i's Alice will always be able to select a C_i with all the nodes. Thus, being able to mimic the strategy for G in G' and proving that $td(G') \geq td(G)$. Proving $td(G') \leq td(G)$ is trivial because G' is a subgraph of G and clearly, removing nodes won't increase the tree-depth.

Proving that $root(G) \subseteq G'$ becomes easy once we have proved the first part of the lemma. If any root of G was in a removed equivalent component, then we can create a graph, H, without that component such that $td(H) = td(G)$. This is a contradiction because if a root of G is in the removed component then $td(G)$ will be at least $1 + H$. $\square$

6.4 Bounded roots in minimal graphs

Definition 6.4.1. *We will say a graph G with td(G) = d is a minimal graph if for any subset of its nodes B, it has at most $d+1$ equivalent components in G with respect to B.*

We are interested in minimal trees because of the following lemma.

Lemma 6.4.2. *For any graph G, there exists a graph G' such that root(G) $\subseteq G'$, td(G) = td(G') and G' is minimal.*

Proof. With lemma 6.3.3 it is easy to see that if G is not minimal then, we can convert it to a minimal graph without changing it's tree-depth and root set. □

By using this lemma, we only have to proof that the number of nodes is bounded by the tree-depth in minimal graphs. This is easy to see, because for any graph, there exists a minimal graph with the same roots and tree-depth. As the roots are a subset of the nodes, proving that the set of nodes is bounded is sufficient.

Lemma 6.4.3. *Let G be a minimal graph with td(G) = d. Then, there exists $f(d,i)$ such that it returns the maximum possible size of the graph left after i rounds of the d-selection-deletion game.*

Proof. We will proof this by reverse induction on i.

- **Base case**: If $i=d$, then clearly f exists. $f(d,i) = 0$, because otherwise Alice would have winning strategy in the $d+1$-selection-deletion game and td(G) would be higher than d, a contradiction.

- **Induction**: If $i<d$, we can assume $f(d,k)$ exists for all $k>i$. Let B be the set of nodes that Bob has removed in the first i rounds of the game. We know that the size of each component of $G \setminus B$ is at most $s = f(d,i+1)$, because Alice will pick a component of $G \setminus B$ for the round $i+1$ and Bob can only remove a node. There are less than 2^{s^2} isomorphic graphs of size s. Each of this graphs can be connected to B in $2^{i \cdot s}$ different ways, because each node in the component can have an edge to each node in B. With all this we can calculate the total number of equivalent components with respect to B, $s' = 2^{s^2} \cdot 2^{i \cdot s} = 2^{s^2+i \cdot s}$. Because G is minimal, we know that each equivalent component will appear at most d times. Thus, $f(d,i) = 1 + d \cdot s'$.

$\square$

With this last lemma, we know that any minimal graph G with td(G) = d will have at most $f(d,0)$ nodes. With lemma 6.4.2 it is easy to see that this the set of roots is bounded by the tree-depth on non minimal graphs.

Theorem 6.4.4. *Let G be a graph with $td(G) = d$. Then, a function f of d exists such that $|root(G)| = f(d)$.*

6.5 An ordering on elimination trees

Definition 6.5.1. *Let G be a connected graph, $P = p_1,\dots,p_n$ a sequence of vertices of G and T an elimination tree of a single component of $G-P$. For a triple of the form (G, T, P):*

- *r_T is the root of T*
- *$T' = \{T_1,\dots,T_k\}$ is the set of trees in $T - r_T$.*
- *P' is P with the root of T appended.*
- *G_T will be the graph induced by the nodes of T.*
- *For $u,v \in V(G), E_G(u,v)$ returns 1 if $\{u,v\} \in E(G)$ and 0 otherwise.*

We will now proceed to define an ordering on such triples.

Definition 6.5.2. *Let (G,T,P), (H,Y,S) be two triples of the form we have just defined such that $|V(G)| = |V(H)|$ and $|P| = |S|$. We will say $(G,T,P) < (H,Y,S)$ if any of the following holds:*

- *$|V(G_T)| < |V(H_Y)|$.*
- *$|V(G_T)| = |V(H_Y)|$ and $|T'| < |Y'|$.*
- *$|V(G_T)| = |V(H_Y)|$, $|T'| = |Y'|$ and $(E_G(p_1,r_T),\dots,E_G(p_n,r_T))$ $< (E_H(s_1,r_Y),\dots,E_H(s_n,r_Y))$ lexicographically.*

- $|V(G_T)| = |V(H_Y)|$, $|T'| = |Y'| = k$, $\forall i = 1,\dots,n\; E_G(p_i, r_T) = E_H(s_i, r_Y)$ *and* $((G,T_1,P'),\dots,(G,T_k,P')) < ((H,Y_1,S'),\dots,(H,Y_k,S'))$ *lexicographically, where each list is ordered by this relation.*

Lemma 6.5.3. *For two triples (G,T,P), (H,Y,S), if neither $(G,T,P) < (H,Y,S)$ nor $(H,Y,S) < (G,T,P)$, then a bijection ϕ exists such that $\forall v \in P \cup V(G_P)$ and $\forall u \in V(G_P)$, $\{u, v\} \in E(G) \iff \{\phi(u), \phi(v)\} \in E(H)$ and $\phi(p_i) = s_i$.*

Proof. We will proof this by induction on the height of T.

- **Base case**: If $height(T) = height(Y) = 1$, then there is only one possible ϕ. This ϕ obviously preserves the conditions mentioned in the lemma because of the third condition of the $<$ operator.

- **Induction**: By induction a ϕ_i exists from each (G,T_i,P') to each (H,Y_i,S'). We can build a ϕ from (G,T,P) to (H,Y,S) that preserves the conditions mentioned in the Lemma simply by joining the different ϕ_i.

□

Definition 6.5.4. *For a connected graph G and $P = p_1,\dots,p_n$ a sequence of vertices of G, we will say an elimination tree T is minimal iff no Y exists such that $(G,Y,P) < (G,T,P)$.*

6.6 Algorithm

Lemma 6.6.1. *For two minimal (G,T,ε) and (H,Y,ε), $G \cong H \iff$ neither $(G,T,\varepsilon) < (H,Y,\varepsilon)$ nor $(H,Y,\varepsilon) < (G,T,\varepsilon)$.*

Proof. $\Longrightarrow$: If G and H are isomorphic, then the second condition holds because otherwise T or Y wouldn't be minimal.
$\Longleftarrow$: This is proven in lemma 6.5.3. □

Algorithm 1: Recursively generate a minimal elimination tree

<u>function MinET</u> (G, P, G');
Input : G is a connected graph, $P = (p_1, \ldots, p_n)$ is a sequence of nodes in $V(G)$ and G' is a connected component of $G \setminus P$.
Output: A minimal elimination tree of G' for G and P
if $td(G') == 1$ **then**
    Output the single node of G';
else
    $R \leftarrow \{r \in V(G') : td(G' - r) + 1 = td(G')\}$;
    Remove from R every $r \in R$ that doesn't have a minimal number of components in $G' - r$;
    Remove from R every $r \in R$ that doesn't have minimal values of $(E_G(p_1, r), \ldots, E_G(p_n, r))$;
    $T \leftarrow \emptyset$;
    foreach $r \in R$ **do**
        $P' \leftarrow (p_1, \ldots, p_n, r)$;
        $ET \leftarrow$ tree formed by the single node r;
        foreach *connected component* $H_i \in G' - r$ **do**
            $ET_i \leftarrow$ MinET(G, P', H_i);
            $ET \leftarrow ET_i$ connected to the root of ET;
        end
        $T \leftarrow T \cup \{ET\}$;
    end
    Output the minimal elimination tree in T for graph G and sequence P;
end

With this lemma we can now build an algorithm to check whether or not two graphs are isomorphic. We find a minimal elimination tree for each graph and then we just have to compare them.

Algorithm 2: Check isomorphism of two graphs

```
function CheckIso (G, H);
Input  : G and H are connected graphs
Output: True if and only if G is isomorphic to H
T ← MinET(G, ε, G);
T' ← MinET(H, ε, H);
if (G, T, ε) < (H, T', ε) or (H, T', ε) < (G, T, ε) then
    Output False;
else
    Output True;
end
```

6.7 Complexity analysis

To be able to make the analysis we will need the following lemma presented in *Sparsity, Algorithms and Combinatorics* [1]. Unfortunately we don't have a proof for it.

Lemma 6.7.1. *Given a graph G with a $td(G) = d$, we can find the tree-depth of G in time $\mathcal{O}(f(d)n^2)$ for some computable function f.*

We will first analyze the complexity of the algorithm $MinET(G,P,G')$. We will analyze it's complexity as a function $T(n,d,N)$, where $n = |V(G')|$, $d = td(G')$ and $N = |V(G)|$. We will calculate the time complexity step by step:

Compare two triples: This is not a decision problem, we just have to check the inequality conditions recursively, which takes time $\mathcal{O}(n^3 + N)$. A tighter bound exists but it won't change the order of the procedure and seeing that it won't take more than cubic time is trivial.

Find R (line 5): By lemma 6.7.1, we can check whether each node is in R in time $\mathcal{O}(f(d-1)n^2)$, for some function f. Thus, finding R takes time $\mathcal{O}(f(d-1)n^3)$.

Reduce R (lines 6-7): By theorem 6.4.4, we know $|R| \leq g(d)$, for some function g. Checking whether each element in R is minimal takes time $\mathcal{O}(n^2 + N^2)$, so this steps take at most time $\mathcal{O}(g(d)(n^2 + N))$.

Recursion (lines 9-17): For each $r \in R$ and each component $H_1, \ldots, H_k \in G' - r$, the complexity is $T(|H_i|, d-1, N)$ where, $1 \leq i \leq k \leq n$. We repeat this for each element in R, so the total running time is $\mathcal{O}(g(d) \sum_{i=1}^{k} T(|H_i|, d-1, N))$.

Select a minimal tree (line 18): We only have to make a comparison for each element in T. Considering that $|T| = |R| \leq g(d)$ and the previously calculated complexity for comparing two triples we get that the last step takes time $\mathcal{O}(g(d)(n^3 + N))$.

Let $T(n,d,N)$ be the upper bound of the run time of $MinET(G,P,G')$. From all the previous statements we get the following recursion:

$$T(n,d,N) = \mathcal{O}\left(n^3 + N + f(d-1)n^3 + g(d)(n^2+N) + g(d)(n^3+N)\right) + \mathcal{O}\left(g(d) \sum_{i=1}^{k} T(|H_i|, d-1, N)\right)$$

If we define a function $h(d) = f(d-1)g(d)$, then we get the following inequality if we assume that $f(d)$ and $g(d)$ are always greater than 1:

$$\mathcal{O}\left(n^3 + N + f(d-1)n^3 + g(d)(n^2+N) + g(d)(n^3+N)\right) \leq \mathcal{O}\left(h(d)(n^3+N)\right)$$

If we assume T is a convex and growing function with respect to n we know the following:

$$\sum_{i=1}^{k} T(|H_i|, d-1, N) \leq T(n, d-1, N)$$

Finally, if we assume that the function is also growing with respect to d we get the following and that it will recurse d times at most we get:

$$T(n, d-1, N) \leq d \cdot \mathcal{O}\left(h(d)(n^3+N)\right)$$

With all this, considering $h'(d) = d \cdot h(d)$ and summing both expressions we get that the final complexity is:

$$T(n,d,N) \leq \mathcal{O}\left(h'(d)(n^3+N)\right)$$

Theorem 6.7.2. *The graph isomorphism problem is Fixed Parameter Tracktable. For two graphs with n nodes and a tree-depth of d it has a time complexity of $\mathcal{O}\left(f(d)n^3\right)$ for some computable function f.*

Proof. In the first call to the $T(n,d,N)$, $n = N$, so the complexity is $\mathcal{O}\left(h'(d)(n^3+n)\right)$. □

6.8 Extending the algorithm to general graphs

In this chapter we have been dealing with connected graphs. Still, extending the algorithm to disconnected graphs takes only polynomial time. We take the first component of the first graph and compare it with each component in the second one. If we are able to find a match for each component, then we know that both graphs are isomorphic. It is easy to see that this only adds a polynomial overhead and thus, graph isomorphism in general graphs is Fixed Parameter Tractable parameterized by the tree-depth.

Bibliography

[1] J. Nesetril, P. Ossona de Mendez. *Sparsity, Algorithms and Combinatorics*, Volume 28, Springer, 2012.

[2] Archontia C. Giannopoulou, Paul Hunter, and Dimitrios M. Thilikos. *LIFO-search: A min-max theorem and a searching game for cycle-rank and tree-depth*, Discrete Applied Mathematics, 160(15):2089-2097, 2012.

[3] L. Lamport. *How to write a 21st century proof*, Journal of Fixed Point Theory and Applications, Mar. 2012. DOI: 10.1007/s11784-012-0071-6

[4] A. Bouland, A. Dawar, and E. Kopczynski. *On Tractable Parameterizations of Graph Isomorphism*, Parameterized and Exact Computation, Springer Berlin Heidelberg, 2012, pp. 218–230