FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A SAT Approach to Clique-Width of a Digraph and an Application on Model Counting Problems

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Aykut Parlak
Matrikelnummer 1225435

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Stefan Szeider
Mitwirkung: Dr. Simone Bova

Wien, 1. Oktober 2016

Aykut Parlak

Stefan Szeider

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A SAT Approach to Clique-Width of a Digraph and an Application on Model Counting Problems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Intelligence

by

Aykut Parlak
Registration Number 1225435

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Stefan Szeider
Assistance: Dr. Simone Bova

Vienna, 1st October, 2016

Aykut Parlak

Stefan Szeider

Erklärung zur Verfassung der Arbeit

Aykut Parlak
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Oktober 2016

Aykut Parlak

Acknowledgements

First and foremost I would like to thank my supervisor, Prof. Stefan Szeider, for suggesting this cutting-edge topic, as well as his tireless and valuable support during the course of thesis. The assistance with research, combined with his patience, motivation, and immense knowledge has been invaluable. Furthermore, the scientific approaches and the methodologies I learned from him will remain with me, and always guide me beyond the scope of this thesis.

I would like also to sincerely thank Dr. Simon Bova, who assisted me with his expertise, practical tips, suggestions and methodologies. Without his participation and input, this thesis would not be of the robust quality it is.

I will always be grateful to both Prof. Stefan Szeider and Dr. Simon Bova for their guidance and assistance during this thesis. It has been a true honor to have had the chance to work with them.

I would also like to acknowledge Dr. Tomer Kotek for his guidance, and assistance with figuring out the model counting algorithm from Fischer et al.

Last but not the least, I would like to thank my partner Ania, my uncle Inan, and my parents for their unwavering support, and encouragement throughout this adventure.

Kurzfassung

Eingeführt von Courcelle, Engelfriet, und Rozenberg, ist die Cliquenweite eine fundamentale Grapheninvariante, die in der diskreten Mathematik und Informatik weitgehend untersucht wurde. Viele schwere Probleme auf Graphen (und gerichteten Graphen) werden handhabbar, wenn sie auf Graphen (und gerichteten Graphen) von kleinen Cliquenweite beschränkt sind und in linearer Zeit lösbar. Cliquenweite ist allgemeiner als Baumweite, da es Graphenklassen von beschränkter Cliquenweite und unbeschränkter Baumweite gibt, aber beschränkte Baumweite auch beschränkte Cliquenweite impliziert.

Typischerweise ist für Algorithmen für Graphen von kleinen Cliquenweite als Eingabe ein Zertifikat für kleine Cliquenweite erforderlich, das ebenfalls schwer zu berechnen ist. In einer neuen Arbeit stellten Heule und Szeider eine Methode zur Berechnung der Cliquenweite von Graphen auf der Basis einer SAT-Kodierung vor. Diese wird durch einen SAT-Solver ausgewertet, um die exakten Cliquenweite kleiner Graphen herauszufinden, die zuvor noch unbekannt war.

Ein Beitrag dieser Diplomarbeit ist eine Verallgemeinerung des Verfahrens von Heule und Szeider auf gerichteten Graphen (Digraphen). Wir erstellen und implementieren einen Algorithmus, der mit Hilfe von Aufrufen eines SAT-Solvers die Cliquenweite eines gerichteten Graphen berechnet und ein Zertifikat erstellt. Wir nutzen dieses Verfahren in zweierlei Hinsicht. Zuerst finden wir die genaue Cliquenweite von verschiedenen kleinen gerichteten Graphen. Zweitens haben wir einen Algorithmus von Fischer, Makowsky, und Ravve implementiert, der die Modelle von aussagenlogischen CNF Formeln mit gerichteten Inzidenzgraphen kleiner Cliquenweite zählt.

Abstract

Introduced by Courcelle, Engelfriet, and Rozenberg, clique-width is a fundamental graph invariant that has been widely studied in discrete mathematics and computer science. Many hard problems on graphs and digraphs become tractable when restricted to graphs and digraphs of small clique-width, indeed solvable in linear time when restricted to classes of bounded clique-width. Clique-width is more general than treewidth, in the sense that algorithms parameterized by clique-width are effective on larger classes of instances than algorithms parameterized by treewidth (as there are graph classes of bounded clique-width where treewidth is unbounded, whereas small treewidth implies small clique-width).

Typically algorithms for graphs of small clique-width require as input a certificate for small clique-width, which is already computationally hard to compute. In recent work Heule and Szeider presented a method for computing the clique-width of graphs based on an encoding to propositional satisfiability (SAT), which is then evaluated by a SAT solver, managing to discover the exact clique-width of various small graphs, previously unknown.

Our main contribution is a generalization of the method by Heule and Szeider to directed graphs. Namely we present and implement an algorithm that, by invoking a SAT solver on a suitable instance, certifies the clique-width of a given directed graph. We exploit this implementation in two ways. First, we find the exact clique-width of various small directed graphs. Second, we implement an algorithm by Fischer, Makowsky, and Ravve and combine this and the aforementioned to an algorithm that counts models of CNF formulas of small directed incidence clique-width.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The algorithmic method of dynamic programming is based on the idea of decomposing an instance of a problem into subinstances, solving the problem on the subinstances, and finally combining the solutions to the subinstances into a solution to the original instance. Roughly, the complexity of the method is dominated by the complexity of the overlaps between the subinstances.

In the realm of graph algorithms, the basic example of this approach is represented by algorithms based on tree decompositions of graphs, whose complexity is dominated by the width of the tree decomposition [CMR01]. Several computationally hard problems on graphs become feasible on graphs of small treewidth (that is, graphs having tree decompositions of small width) and linear-time computable on classes of graphs of bounded treewidth.

Clique-width is a generalization of treewidth, devised to algorithmically deal with dense graph classes. It exhibits the same behaviour of treewidth on many relevant graph problems that are computationally hard in general: it warrants feasible time on graphs of small clique-width, and linear-time feasibility on graph classes of bounded clique-width. However, it is algorithmically strictly more powerful than treewidth, as there are graph classes of bounded clique-width where treewidth is unbounded, whereas small treewidth implies small clique-width [CO00].

Clique-width is defined via a graph construction process involving some fixed graph operations that introduce vertex labels in such a way that vertices that share the same label at a certain point of the construction process must be treated uniformly in subsequent steps. The minimum number of vertex labels introduced along any construction of a graph obeying the rules of the above process is called the clique-width of the graph, which is intuitively a measure of its structural complexity. A graph G whose construction uses k

vertex labels naturally corresponds to a decomposition of the graph, called k-expression, that certifies that G has clique-width k. This construction process applies to undirected and directed graphs.

A practical issue with decomposition algorithms is that they typically need the actual decomposition as an input, which is in turn hard to compute itself [FRRS09]. Whereas treewidth-based algorithms are manageable, as tree decompositions are linear-time computable on graph classes of bounded treewidth using the algorithm of Bodlaender [Bod96], the case of clique-width is much wilder; for fixed k, we can approximate in polynomial time the clique-width of graphs of clique-width at most k with an exponential error [Oum08] It is not known whether this exponential error can be avoided. There is an algorithm with exponential time $(2k+1)^n n^{O(1)}$ proposed by Wahlström such that it can decide whether the clique-width of a graph with n vertices is at most k [Wah11]. Because of this intricacy of clique-width, the exact clique-width of even very small graphs was not known until Heule and Szeider presented their approach.

As far as exact clique-width computation is concerned, Heule and Szeider managed to compute decompositions for undirected graphs of small clique-width by first encoding the question into a propositional formula and then evaluating the formula by a SAT solver. The key idea underlying the success of their method is to avoid the direct computation of a k-expression for a graph of clique-width k, which involves a number of steps quadratic in the size of the graph, in favour of the computation of an equivalent certificate, called k-derivation, involving a number of steps that is only linear in the size of the graph.

1.2 Contribution

We present a theoretical and practical investigation of the problem of certifying the clique-width of directed graphs (digraphs). One application motivating us is the model counting algorithm of Fischer, Makowsky, and Ravve [FMR08], that receives as input a k-expression for a digraph of clique-width k, corresponding to a CNF formula F, and returns in output the number of models of F. In order to make this algorithm practical, the input k-expression has to be precomputed.

We approached the problem of certifying the clique-width of digraphs by revisiting and extending the methods by Heule and Szeider from graphs to digraphs. Our main contributions can be summarized as follows.

From the theoretical side, we extend the notion of k-derivation to the digraph setting, show that it characterizes clique-width k for digraphs, and encode it to SAT (Chapters 3–4). Based on this notion, we present several contributions on the practical side, as follows:

1. We implement an algorithm that receives as input a directed graph G and returns a k-derivation, where k is the clique-width of G.

2. We implement an algorithm that receives as input a k-derivation of a directed graph G and returns in output a k-expression of G.

3. We implement the algorithm of Fischer, Makowsky, and Ravve [FMR08], and test it on small clique-width instances using the implementations mentioned in Items 1–2 to precompute the input k-expression (Chapter 5).

4. We use the implementation mentioned in Item 1 to find the exact clique-width of various small directed graphs (Chapter 6).

CHAPTER 2

Preliminaries

2.1 Formulas and Satisfiability

A *boolean variable* is a variable that can assume only two values 0 and 1. A *literal* is a boolean variable or a negated boolean variable. A *clause* is a finite set of literals such that do not contain a complementary pair. A conjunctive normal form or CNF propositional formula F is a finite set of clauses. $Var(C)$ denotes the set of variables that occur in clause C. $Var(F) = \bigcup_{C \in F} Var(C)$.

For a CNF formula F, a *truth assignment* is a mapping $\tau : var(F) \rightarrow \{1, 0\}$ where $\tau(\bar{x})$ is defined as $1 - \tau(x)$ for $x \in var(F)$. A $truth\ assignment\ \tau\ satisfies$ a clause C if there is at least one literal $x \in C$ such that $\tau(x) = 1$. An assignment τ satisfies a CNF formula F if τ satisfies all clauses in F. The *satisfiability* (SAT) problem is that of testing whether a given formula (CNF) is satisfiable. A SAT problem is NP-Complete [Coo71]. The *propositional model counting* (#SAT) problems ask for the number of truth assignments for a given (CNF) formula. #SAT is #P-complete [Val79].

2.2 Graphs and Clique-width

An undirected graph $G = (V, E)$ consists of a non-empty finite set $V = V(G)$ of vertices and a finite set $E = E(G)$ of edges. An edge is unordered pair of distinct vertices and is denoted by $\{x, y\}$, $x, y \in V(G)$. An edge between vertices x and y is also denoted by xy or yx. A directed graph or digraph $D = (V, A)$ consists of a non-empty finite set $V = V(D)$ of vertices and a finite set $A = A(D)$ of arcs. An arc is an ordered pair of distinct vertices and is denoted by (x, y) where $x, y \in V(D)$. For an arc (x, y), the first vertex is its head and its second vertex its tail. If there is an edge or arc between two vertices x and y, then the vertices are *adjacent*. If two vertices are adjacent, then they are neighbors of each other [BJG97].

All graphs considered in this thesis are finite, without self-loops and without parallel edges or arcs. In digraphs we allow two arcs between two vertices if the arcs have opposite orientation.

For a positive integer k, a *k-graph* is a graph whose vertices are labeled with integers from $\{1, \ldots, k\}$. An *initial k-graph* is a k-graph consisting of exactly one labeled vertex, denoted by $i(v)$ for a vertex v and a label i. The *clique-width* of graph G, denoted $\mathrm{cwd}(G)$, is the smallest number k of labels required to construct a graph G from initial k-graphs by using following three operations:

1. Disjoint union, denoted by the binary symbol $\oplus$;
2. Relabeling, denoted by the unary symbol $\rho_{i\to j}$;
3. a) Edge creation, denoted by the unary symbol $\eta_{i,j}$;
   b) Arc creation, denoted by the unary symbol $\alpha_{i,j}$.

Intuitively the disjoint union operation brings subgraphs into the same scope so that the other two operations can operate on them. The edge/arc creation, from label a to b, create edges/arcs between every vertex of the vertex set labeled with a and every vertex of the vertex set labeled with b. A construction of an undirected graph uses edge creation and a construction of a directed graph uses arc creation. The relabeling operation changes the label of vertices, mainly merging two different labeled subgraphs in to one labeled subgraph. If k is an integer, then a *k-graph* is a graph whose vertices are labeled with elements of {1, ..., k}. These three operations are used repeatedly until original graphs are constructed from *initial k-graphs*. This construction forms an algebraic term consisting of the operations $\oplus$, $\rho_{i\to j}$, $\eta_{i,j}/\alpha_{i,j}$ with $i, j \in \{1, \ldots, k\}$ and $i \neq j$. Such a term is called *k-expression* (Example 1).

The parse tree of a k-expression with additional information is called *k-expression tree*. Let ϕ be a k-expression of a graph $G = (V, E)$. Let Q be a k-expression tree of ϕ with a root node r. Q contains a node for each operation $\oplus$, $\rho_{i\to j}$, $\eta_{i,j}$, $\alpha_{i,j}$ occurring in ϕ and for each initial operation $i(v)$ in ϕ for $\forall v \in V$ and $i \in \{1, \ldots, k\}$. For a node $q \in Q$, ϕ_q defines the subexpression of ϕ whose k-expression tree is the tree is the subtree of Q rooted by q. Then each node $q \in Q$ is labeled by G_q which is a k-graph constructed by ϕ_q.

A k-expression is called *succinct*, when an $\oplus$-node does not have a child $\oplus$-node. Evidently, we can effectively transform a k-expression into a succinct k-expression and vice versa [HS15].

Example 1. Let graph $G = (\{a, b, c, d\}, \{ab, ac, bc, cd\})$ be defined by 3-expression.

$$\eta_{2,3}(\eta_{1,2}((\rho_{2\to 1}(\eta_{1,2}(1(a) \oplus 2(b))) \oplus 2(c)) \oplus 3(d)))$$

Since G does not have 2-expression, $\text{cwd}(G) = 3$. Figure 2.1 shows a 3-expression tree of the G. ⊣

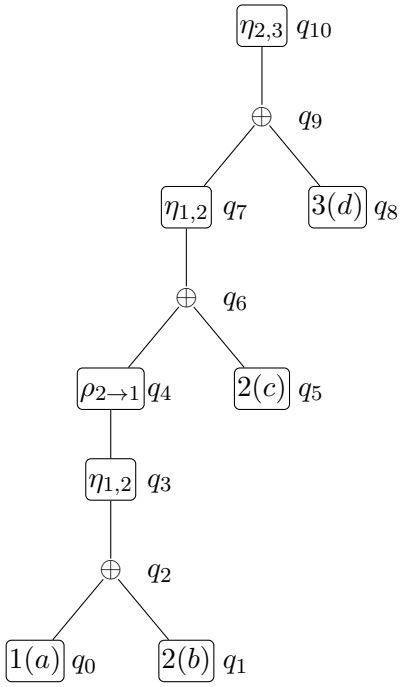


Figure 2.1: 3-expression tree of Example 1.

2.3 Partition

Set partitions play an important role in the reformulation of clique-width. Let S be a set and $P = \{S_1, S_2, \ldots, S_t\}$ be a set of nonempty subsets of S. P is called a *partition* of S if $S_i \cap S_j = \emptyset$ and $\cup_{i=1}^{t} S_i = S$ for all $1 \leq i \neq j \leq t$. The elements of the partition P are called *equivalence classes*. Let P, P' be the partition of a set S . P' is the refinement of P, if for any two elements $x, y \in S$ in the same equivalence class of P' and also in the same equivalence class of P.

CHAPTER 3

SAT Approach to Clique-width

As was mentioned in previous chapters, there was no practical algorithm known to calculate the exact clique-width of graphs until Heule and Szeider published their work. They proposed a new formulation of k-expression, called k-derivation. The new formulation is appropriate to be encoded into a SAT instance. Intuitively they coded a k-derivation of a graph with a parameter k, k is a clique-width candidate, into a SAT instance and iteratively checked whether the satisfiability of formula was amended by changing value of k. The smallest parameter k, by which the formula is satisfiable, is the *cwd* of the graph.

3.1 Undirected Graphs

3.1.1 Reformulation of Clique-Width

Intuitively a k-derivation of a graph G is the set of snap-shoots of the states of union and labels of k-expression without edge creations. A k-derivation can present more than one different graph since it does not include the edge creations. Therefore, we need the original graph in order to obtain its k-expression tree from its k-derivation.

Consider a graph $G(V, E)$, in which the finite set V of vertices is called *universe*. A *template* T consists of two partitions of V, a component of V $\mathrm{cmp}(T)$ represents an induced subgraph of V and a group of V $\mathrm{grp}(T)$ represents a set of vertices with the same label with respect to their component. A $\mathrm{grp}(T)$ of a template T is a refinement of $\mathrm{cmp}(T)$. A *derivation* of length t is a sequence of templates $D = \{T_0, \ldots, T_t\}$ with the four conditions:

- D1: $|\mathrm{cmp}(T_0)| = |V|$ and $|\mathrm{cmp}(T_t)| = 1$
- D2: $\mathrm{grp}(T_i)$ is a refinement of $\mathrm{cmp}(T_i)$, $0 \leq i \leq t$

- D3: $\text{cmp}(T_{i-1})$ is a refinement of $\text{cmp}(T_i)$, $0 \leq i \leq t$
- D4: $\text{grp}(T_{i-1})$ is a refinement of $\text{grp}(T_i)$, $0 \leq i \leq t$

D1 and D2 imply that $|\text{grp}(T_0)| = |V|$. In the first template T_0 , the $\text{cmp}(T_0)$ and $\text{grp}(T_0)$ consist of singletons. The derivation $\mathcal{D} = (T_0, \ldots, T_t)$ starts with template T_0 and the set elements are merged until the component of last template T_t has V as a single element.

The *width* of a component $c \in \text{cmp}(T)$ is the number of groups $g \in \text{grp}(T)$ such that $g \subseteq c$. The *width* of template T is the maximum width of its components. The *width* of the derivation is the maximum *width* of its templates. A k-derivation is a derivation with width being at most k. A derivation does not present the edges of graphs. To ensure that a derivation is the derivation of a graph $G = (V, E)$, there are three conditions that need to hold for all $1 \leq i \leq t$ [HS15].

1. *Edge property:* For any vertices $u, v \in V$ such that $uv \in E$, if u, v are in the same group in T_i, then u, v are in the same component in T_{i-1}.
2. *Neighborhood property:* For any three vertices $u, v, w \in V$ such that $uv \in E$ and $u, w \notin E$, if v, w are in the same group in T_i , then u, v are in the same component in T_{i-1}.
3. *Path property*: For any four vertices $u, v, w, x \in V$, such that $uv, uw, vx \in E$ and $wx \notin E$, if u, x are in the same group in T_i and v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.

The edge property ensures that an edge can be inserted before its vertices have same label, because it is not possible to insert an edge between two same labeled vertices in the same component. The neighborhood property and path properties make sure that a nonexistence edge is not inserted.

A derivation $\mathcal{D} = (T_0, \ldots, T_t)$ is *strict*, if $|\text{cmp}(T_i - 1)| > |\text{cmp}(T_i)|$ holds for all $1 \leq i \leq t$.

Lemma 1 ([HS15])**.** *Every k-derivation of a graph G contains as subsequence a strict k-derivation of G.*

Proof. Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a k-derivation of G. Since $|\text{cmp}(T_i - 1)| \geq |\text{cmp}(T_i)|$, the only case we need to be sure that we do not have is $|\text{cmp}(T_{i-1})| = |\text{cmp}(T_i)|$ in the $\mathcal{D} = (T_0, \ldots, T_t)$. We show that if we have such a template, either T_{i-1} or T_i can be eliminated. If the $\text{grp}(T_{i-1}) = \text{grp}(T_i)$, then we can remove T_i from $\mathcal{D} = (T_0, \ldots, T_t)$. If $\text{grp}(T_{i-1}) \neq \text{grp}(T_i)$, then, since $i \neq 1$, there are two cases, $1 < i < t$ or $i = t$. If $i = t$, then T_i is the last template and we can safely remove it from $\mathcal{D} = (T_0, \ldots, T_t)$. If $1 < i < t$, then since $\text{grp}(T_{i-1})$ is a refinement of $\text{grp}(T_i)$, we can safely remove T_i.
When we apply the operation above iteratively on the templates of the k-derivation, then we can obtain a strict k-derivation of G. □

Lemma 2 ([HS15]). *Every strict k-derivation of a graph with n vertices has length at most $n-1$.*

Proof. Since a k-expression is a strict, the number of components in templates has to be decreased for each step and since $|\text{cmp}(T_0)| = n$ and $|\text{cmp}(T_t) = 1|$, it follows that $t \leq n-1$. □

Lemma 3 ([HS15]). *From a k-expression of a graph G we can obtain a k-derivation of G in polynomial time.*

Proof. Let ϕ be a k-expression of a graph $G(V, E)$ and Q is a (succinct) k-expression tree of ϕ with root r. Let $R(q)$ be the number of $\oplus$-nodes appearing on the path from r to $q \in Q(V)$. Let $U, L \subset Q$ such that U is a set of $\oplus$-nodes and L is a set of leaves in Q and $t := max_{q \in L} R(q)$. We define $U_i = \{q \in U : R(q) = t - i + 1\}$ and $L_i = \{q \in L : R(q) < t - i + 1\}$.

We can define a derivation $\mathcal{D} = (T_0, \ldots, T_t)$ as follows. For each template $T_i \in \mathcal{D}$, $\text{cmp}(T_i) = \{V(G_q) : q \in L_i \cup U_i\}$ and $\text{grp}(T_i) = \{\cup_{q \in U_i \cup L_i} \text{grp}(G_q)\}$ where $\text{grp}(G_q)$ is the partition of $V(G_q)$ with respect to labels of vertices. Since $|\text{grp}(G_q)| \leq k$ for all nodes $q \in Q$, $\mathcal{D}$ is a k-derivation. It remains to be shown that $\mathcal{D}$ is a k-derivation of G. For this we need to ensure that all properties of a k-derivation of G hold.

To show that the *edge property* holds, any two vertices $u, v \in V$ such that $uv \in E$ have to be in the same component in T_{i-1} if they are in the same group in T_i. We can proof this by contradiction. Assume that $u, v \in V$, $uv \in E$ and u, v are in the same group but in different components c_1, c_2 in T_{i-1}. Since u, v are in the same group of T_i, u, v are in the same component of T_i. Hence, there is an $\oplus$-node $q \in U_i$ with $u, v \in V(G_q) \in \text{cmp}(T_i)$. Let q_1, q_2 be children of q such that $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. But neither $uv \in E(G_{q_1})$ nor $uv \in E(G_{q_2})$, and there is not a η-operation that inserts edge uv . So $uv \notin E(G_r) = E$, which is a contradiction. Hence, the edge property holds.

To show that the *neighborhood property* holds, we use contradiction as well. Assume that there are three vertices $u, v, w \in V$, $uv \in E$, $uw \notin E$ and v, w are in the same group of T_i but u, v are in the different components in T_{i-1}, respectively in components c_1 and c_2. Since v, w are in the same group in T_i then they have to in the same component c in T_i. Let $q \in U_i$ be an $\oplus$-node such that $v, w \in V(G_q) = c$, let q_1, q_2 be children of q such that $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. Hence, $uv \notin E(G_{q_1}) \cup E(G_{q_2})$, so we need to introduce the edge uv in some node between q and r. Since v, w are in the same label in G_q, they have to have the same labels in any node between q and r. Hence, an edge creation between the label of v and the label of u would also introduce edge uw which is in contradiction with $uw \notin E$. Hence, the neighborhood property holds.

To show that the *path property* holds, we use contradiction as before. Assume that four vertices $u, v, w, x \in V$, such that $uv, uw, vx \in E$ and $xw \notin E$, u, x are in the same group in T_i and v, w are in the same group in T_i, but u, v are in the different components c_1 and c_2 in T_{i-1}, respectively. According to neighborhood property, u, w are in the same

component in T_{i-1} and v, x are in the same component in T_{i-1}. Since u, x are in the same group of T_i, then they have to be in a same component c in T_i. Since u, w are in the same component in T_{i-1}, then they have to be in the same component in T_i. Since u, x are in c in T_i and u, w are in the same component in T_i, then $w \in c$. In a similar way we can see that $v \in c$ in the template T_i. Let $q \in U_i$ be an $\oplus$-node such that $v, w \in V(G_q) = c$. Let q_1, q_2 be children of q such that $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. Since u and v are in different components in T_{i-1} then $u, v \notin E(G_{q_1}) \cup E(G_{q_2})$. Therefore, we need to introduce the edge uv in some node between q and r. Since v, w are in the same label in G_q and u, x are in the same label in G_q, they have to have the same labels in any node between q and r. Hence, an edge creation between the label of v, w and the label of u, x would also introduce an edge xw which is in contradiction with $xw \notin E$. Hence, the path property holds as well. We can conclude that $\mathcal{D}$ is a k-derivation of G.

The procedure above can be clearly carried out in polynomial time. □

Lemma 4 ([HS15])**.** *From a k-derivation of a graph G we can obtain a k-expression of G in polynomial time.*

Proof. Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a k-derivation of graph $G(V, E)$ with the label set $\{1 \ldots k\}$. By Lemma 2, we can assume that $\mathcal{D} = (T_0, \ldots, T_t)$ is a strict k-derivation of G. We are going to construct a k-expression tree for G from k-derivation in polynomial time.

The construction consists of three steps.

In the first step we construct a k-expression tree $Q_\oplus$ that consists of leaves and $\oplus$-node. For each component $c = \{v\} \in \text{cmp}(T_0)$, where $v \in V$, we introduce a node $q(c, 0)$ with the label $1(v)$.These are the leaves of the k-expression tree. Then for each $c \in \text{cmp}(T_i)$, we introduce an $\oplus$-node $q(c, i)$ and add edge with children nodes if there is a node $q(c', i-1)$ such that $c' \subseteq c$. D1 and D3 properties ensure that the $Q_\oplus$ is a tree. $Q_\oplus$ does not have to be succinct.

In the second step we are going to obtain a k-expression tree $Q_{\oplus,\rho}$ by adding the ρ operation on $Q_\oplus$. By depth first ordering, we can visit each node $q(c, i)$ and inset at most k ρ-node between $q(c, i)$ and $q'(c', i-1)$ such that $\text{grp}(G_{q'}) = \{g \in \text{grp}(G_{q(c,i)}) : g \subseteq c\} \subseteq \text{grp}(T_i)$. Since any partition can be obtained by its refinements, properties D2 and D3 ensure that such a node insertion is possible.

The last step is obtaining the k-expression tree $Q_{\oplus,\rho,\eta}$ by adding η-nodes on $Q_{\oplus,\rho}$.We are going to show that for each edge $uv \in E$, there is an added η-node q above an $\oplus$-node q in $Q_{\oplus,\rho,\eta}$ such that q is a child of p and it introduces the edge uv but does not introduce an edge not found in E.

Let $q(c, i)$ be the $\oplus$-node with the smallest i in $Q_{\oplus,\eta}$ such that $u, v \in V(G_q)$. Therefore, there are two child nodes $q_1(c_1, i-1)$, $q_2(c_2, i-1)$ of q in $Q_{\oplus,\eta}$ such that $u \in V(G_{q_1})$ and $v \in V(G_{q_2})$. It follows that $c_1, c_2 \in T_{i-1}$ are distinct components with $u \in c_1$ and $v \in c_2$. According to the edge property, if u and v are in different components in T_i then u and v are in different groups of T_i. This means that u and v have different labels in

G_q, l_u and l_v respectively. We can add a η-node p with operation η_{l_u,l_v} above q as its parent node. The node p introduces the edge uv. It remains to be shown that it does not introduce an edge not found in E. We can show it by each vertex couple $u', v' \in c$ with labels l_u and l_v respectively. We need to consider four cases.

Case 1: $u = u'$, $v = v'$. Then trivially $u'v' = uv \in E$.

Case 2: $u = u'$, $v \neq v'$. Assume on the contrary that $u'v' \notin E$. Since v and v' have the same label in G_q that means they are in the same group of T_i. According to the neighborhood property u and v have to be in a same component T_{i-1} which is a contradiction to the smallest choice of i.

Case 3: $u \neq u'$, $v = v'$. This is the symmetric to Case 2.

Case 4: $u \neq u'$, $v \neq v'$. We can proof once more by contradiction. Assume that $u'v' \notin E$. From Case 2 and Case 3, we know that $uv', u'v \in E$. According to the path property, it follows that u and v belong to the same component in T_{i-1}. Then it is a contradiction with smallest choice of i.

Since η-nodes do not introduce a non-existing edge, we can apply the procedure above to obtain a k-expression tree of G in polynomial time. □

Lemma 5 ([HS15])**.** *Let $1 \leq k \leq n$. If a graph with n vertices has a k-derivation, then it has a k-derivation of length $n - k + 1$.*

Proof. For the proof, we need to use the *k-length* of a derivation. The *k-length* of a derivation for a fixed $k \geq 1$ is the number of templates that contain a component of a larger size than k. Let $l(n, k)$ be the largest k-length of a strict derivation with universe size n. Before we show the proof, we need to present 3 claims.

Claim 1: $l(n, k) < l(n + 1, k)$.

To see the claim, let $\mathcal{D} = (T_0, \ldots, T_t)$ be a strict derivation over a universe V of size n with length $l(k, n)$. Consider a strict derivation $\mathcal{D}' = \{T'_0 \ldots T'_{t+1}\}$ over universe $V \cup \{a\}$ such that $\text{cmp}(T'_i) = \text{cmp}(T_i) \cup \{a\}$ for $0 \leq i \leq t$ and $\text{cpm}(T'_{t+1}) = V \cup \{a\}$. The derivation $\mathcal{D}'$ has k-length $l(k, n) + 1$.

Claim 2: Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a strict derivation over a universe V of size n of k-length $l(k, n)$. Then, template $T_{t-l(k,n)+1}$ has exactly one component the size of $k + 1$ and all other components are singleton.

To show the claim, consider the template T_j where $j = t - l(k, n)$. Since there are $l(k, n)$ templates which have a component size of larger than k in $\mathcal{D}$ and $\mathcal{D}$ is a strict derivation, j is the largest index that all components of T_j have a size of at most k. Let $c_1 \ldots c_r$ be the components of T_{j+1} such that $|c_1| \geq |c_1| \geq \cdots \geq |c_r| > 1$, therefore $|c_1| > k$. We show that $r = 1$. We are going to show this by contradiction. Assume that $r > 1$. It holds that $|c_i| \geq 2$ for $2 \geq i \geq r$. Then we pick an element $a_i \in c_i$, $2 \geq i \geq r$, and obtain set $X = \cup_{i=2}^{r} c_i \setminus \{a_i\}$. Then we induce $\mathcal{D}$ to a strict derivation $\mathcal{D}'$ over the universe

$V' = V \setminus X$. $l(n', k) = l(n, k)$ since $n = |V| > |V'| = n$ and $\mathcal{D}'$ have same k-length. This contradicts Claim 1. Hence $r = 1$, therefore c_1 is the only component that $|c_1| > k$ and all other components are singleton. It remains to be shown that $|c_1| = k + 1$. We use contradiction again in order to show it. Assume that $|c_1| > k + 1$. We can induce the $\mathcal{D}$ to a strict derivation $\mathcal{D}''$ over universe V'', where $\{b_1 \dots b_{k+1}\} \in c_1$, $X = c_1 \setminus \{b_1 \dots b_{k+1}\}$ and $V'' = V \setminus X$. For $n = |V|$ and $n'' = |V''|$, $l(n'', k) = l(n, k)$. It contradicts Claim 1 since $n = |V| > |V''| = n''$. Hence Claim 2 holds.

Claim 3: $l(n, k) < n - k$.

To show the claim, let $\mathcal{D} = (T_0, \dots, T_t)$ be a strict derivation over a universe V of size n with length $l(k, n)$ and $j = t - l(n, k)$. By Claim 2, we know that T_{j+1} has exactly one component with size $k + 1$ and $n - k - 1$ singleton components. By this we can conclude that $|\mathrm{cmp}(T_{j+1})| = n - k$. While $\mathcal{D}$ is strict derivation, we have $n - k = |\mathrm{cmp}(T_{j+1})| > |\mathrm{cmp}(T_{j+2})| > \dots > |\mathrm{cmp}(T_t)| = 1$. Therefore $l(n, k) = t - j \leq n - k$. Hence Claim 3 holds.

Now we can proof the lemma. Let $\mathcal{D} = (T_0, \dots, T_t)$ be a strict derivation of a graph $G(V, E)$ with $|V| = n$. We can assume that $\mathcal{D}$ is a strict derivation according to Lemma 1. Let l be k-length of $\mathcal{D}$ and let $j = t - l$. We define a derivation $\mathcal{D}' = (T_0, T'_j, T_{j+1} \dots, T_t)$ where $T_0, T_{j+1}, \dots T_t$ are templates of $\mathcal{D}$ and $\mathrm{cmp}(T'_j) = \mathrm{cmp}(T_j)$, $\mathrm{grp}(T'_j) = \mathrm{grp}(T_0)$. We need to show that $\mathcal{D}'$ is a k-derivation of G. For this we need to consider the edge, neighborhood and path properties for T'_j and T_{j+1}. T'_j only has singleton groups, then T'_j satisfies the properties. Also the properties hold for T_{j+1}, because T'_j has exactly the same components as T_j. The length of derivation $\mathcal{D}'$ is $t - j + 1$. When we substitute j with $t - l$, then the length of $\mathcal{D}'$ is $l + 1$. By Claim 3 we know that $l \leq n - k$. Hence $l + 1 \leq n - k + 1$. □

With the result of Lemmas 3, 4, and 5, we can conclude with the Proposition 1.

Proposition 1 ([HS15])**.** *Let $1 \leq k \leq n$. A graph G with n vertices has clique-width at most k if and only if G has a k-derivation of length at most $n - k + 1$.*

3.2 Directed Graphs

This section is an important part where this thesis goes beyond the study from Heule and Szeider [HS15]. We extend their approach to digraphs. The differences between the clique-width of an undirected and a directed graph is the edge and arc creations. The clique-width of directed graphs use an arc creation instead of an edge creation, also is called directed clique-width. Let G be a undirected graph and G' be a directed graph obtained by orienting the edges of G then $cwd(G) \leq cwd(G')$. Then we can obtain a k-expression of G from a k-expression of G' by replacing the arc creations with edge creations [CO00]. We can get the k-expression of an undirected graph G from the k-expression of its directed version with an equal or fewer numbers of labels. In Example 2, we can see clearly that $cwd(G) \leq cwd(G')$ holds.

Example 2. Consider the digraph $G' = (\{a, b, c\}, \{(a, b), (b, c)\})$ (Figure 3.1a) and its undirected version $G = (\{a, b, c\}, \{ab, cb\})$ (Figure 3.1b). The 3-expression of G' is

$$\alpha_{2,3}(\alpha_{1,3}(1(a) \oplus 2(b) \oplus 3(c)))$$

and it has 3 labels ($cwd(G) = 3$). We can obtain a 3-expression for G by replacing arc(α) operations with edge(η) operations, such that :

$$\eta_{2,3}(\eta_{1,3}(1(a) \oplus 2(b) \oplus 3(c))).$$

On the other hand, there exists a 2-expression for G :

$$\eta_{1,2}(1(a) \oplus 2(b) \oplus 1(c)).$$

Hence $cwd(G) = 2$. ⊣

(a) Graph G' (b) Digraph G

Figure 3.1: G is an undirected graph by omitting the direction of edges of G' (Figure 3.1a). $cwd(G) = 2$ and $cwd(G') = 3$.

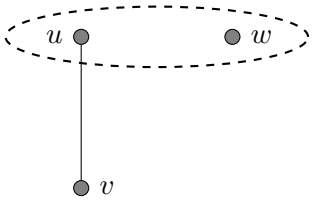
3.2.1 Reformulation of clique-width

The reformulation of the k-expression of a digraph into a k-derivation is quite similar with undirected graphs. Since the arc creations are not considered in a k-derivation, the derivation conditions D1-D4, stay the same for digraphs as well. The differences occur through the derivation of a graph's conditions. To ensure that a derivation is derivation of a digraph $D = (V, A)$, we formulate the following three conditions that hold for all $1 \leq i \leq t$.

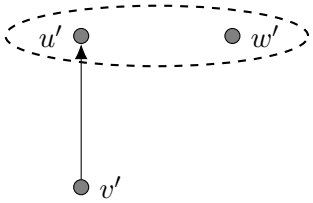
> *Arc Property*: For any two vertices $u, v \in V$ such that $(u, v) \in A$, if u, v are in the same group in T_i, then u, v are in the same component in T_{i-1}.
>
> *Neighborhood Property*:
>
> > For any three vertices $u, v, w \in V$ such that $(u, v) \in A$ and $(u, w) \notin A$, if v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.
> > For any three vertices $u, v, w \in V$ such that $(v, u) \in A$ and $(w, u) \notin A$, if v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.
>
> *Path Property*: For any four vertices $u, v, w, x \in V$, such that $uv, uw, xv \in A$ and $xw \notin A$, if u, x are in the same group in T_i and v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.

(a) Edge property (b) Arc property

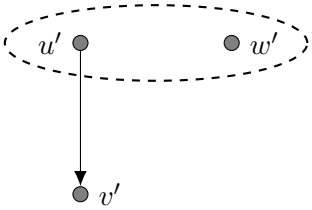
Figure 3.2: Visualization of edge and arc properties. The vertices in the dashed area are in the same group in a template of a derivation.



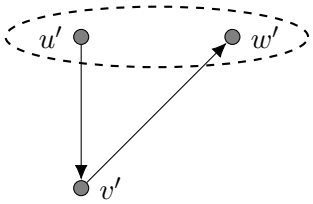
(a) Only case for undirected graphs.



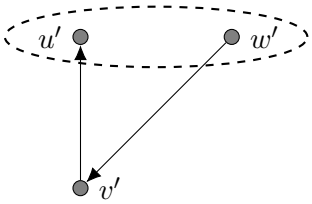
(b) Case 1 for directed graphs



(c) Case 2 for directed graphs



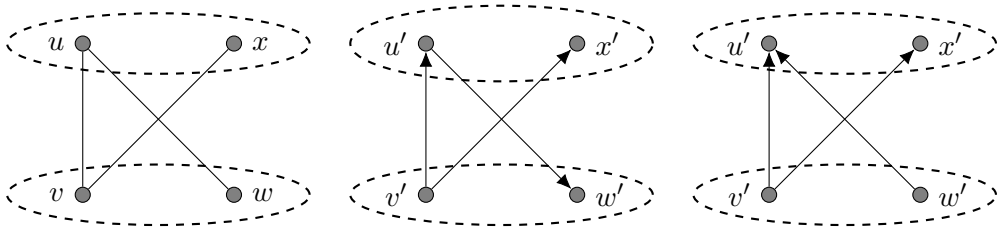
(d) Case 3 for directed graphs



(e) Case 4 for directed graphs

Figure 3.3: Visualization of neighborhood property for undirected and directed graphs. The vertices in the dashed area are in the same group in a template of a derivation.

It is clear that the edge and arc creations stay relatively the same for undirected and directed graphs (see Figure 3.2.). On the other hand, the neighborhood property is a bit different for directed graphs. It is essentially the same as for undirected graphs but it considers the direction of arcs. Therefore, the neighborhood property covers four cases for directed graphs, while it covers one case for undirected graphs (see Figure 3.3). Hence, the neighborhood property is more powerful for directed graphs. It has an effect on the path property so that the path property also stays relatively the same for undirected graphs. Consider Figure 3.4b. It is a directed version of the path property of



(a) For undirected graphs. (b) For a directed graph. (c) For a directed graph.

Figure 3.4: Visualization of path property for undirected and directed graphs.The vertices in the dashed area are in the same group in a template of a derivation.

the undirected case (Figure 3.4a). According to the neighborhood property for directed graph, we can be sure that all arcs of the case are inserted. But even so we need the path property for directed graphs. Consider the case in Figure 3.4c. According to the neighborhood property we can not be sure that arc (v', u') is inserted. The arc can be inserted by the path property. The path property stay relatively same for directed graph through the neighborhood property for directed graphs.

Since the k-derivation of graphs does not consider the edge/arc creation, Lemmas 1 and 2 preserve their validity for digraphs. On the other hand, Lemmas 3, 4 and 5 need to be reconsidered for digraphs. The directed versions of lemmas are Lemma 6 for Lemma 3, Lemma 7 for Lemma 4 and Lemma 8 for Lemma 5, respectively. The neighborhood and path property used in the following proofs are the new properties which we redefined for directed graphs.

Lemma 6. *From a k-expression of a digraph $D = (V, A)$ we can obtain a k-derivation of D in polynomial time.*

Proof. The proof is quite similar to the undirected version. The main differences are that, instead of edges and edge creations, we have arcs and arc creations.

Let ϕ be a k-expression of a digraph $D(V, A)$ and Q is a (succinct) k-expression tree of ϕ with root r. Let $R(q)$ be the number of $\oplus$-nodes appearing on the path from r to $q \in Q(V)$. Let $U, L \subset Q$ such that U is a set of $\oplus$-nodes, L is a set of leaves in Q and let $t := max_{q \in L} R(q)$. We define $U_i = \{q \in U : R(q) = t - i + 1\}$ and $L_i = \{q \in L : R(q) < t - i + 1\}$.

We can define a derivation $\mathcal{D} = (T_0, \ldots, T_t)$ as follows. For each template $T_i \in \mathcal{D}$, $\text{cmp}(T_i) = \{V(D_q) : q \in L_i \cup U_i\}$ and $\text{grp}(T_i) = \{\cup_{q \in U_i \cup L_i} \text{grp}(D_q)\}$ where $\text{grp}(D_q)$ is the partition of $V(D_q)$ with respect to labels of vertices. Since $|\text{grp}(D_q)| \leq k$ for all nodes $q \in Q$, $\mathcal{D}$ is a k-derivation. It remains to be shown that $\mathcal{D}$ is a k-derivation of D. For this we need to ensure that all properties of the k-derivation of D hold.

To show that the *arc property* holds, we need to show for any two distinct vertices $u, v \in V$ such that $(u, v) \in A$ have to be in the same component in T_{i-1} if they are in the same group in T_i. We can proof this by contradiction. Assume that $u, v \in V$, $(u, v) \in A$, u, v are in the same group but in different components c_1, c_2 in T_{i-1}. Since u, v are in the same group of T_i, therefore u, v are in the same component of T_i. Hence there is an $\oplus$-node $q \in U_i$ with $u, v \in V(D_q) \in \text{cmp}(T_i)$. Let q_1, q_2 be children of q such that $V(D_{q_1}) = c_1$ and $V(D_{q_2}) = c_2$. But neither $(u, v) \in A(D_{q_1})$ nor $(u, v) \in A(D_{q_2})$, therefore there is not a η-operation that inserts an (u, v) arc. So $(u, v) \notin A(D_r) = A$ which is a contradiction. Hence, the arc property holds.

To show that the *neighborhood property* holds, we use contradiction as well. Assume that there are three vertices $u, v, w \in V$, $(u, v) \in A$, $(u, w) \notin A$ and v, w are in the same group of T_i but u, v are in different components in T_{i-1}, respectively in components c_1 and c_2. Since v, w in the same group in T_i then they have to in the same component c in T_i. Let $q \in U_i$ be an $\oplus$-node such that $v, w \in V(D_q) = c$, let q_1, q_2 be children of q such that $V(D_{q_1}) = c_1$ and $V(D_{q_2}) = c_2$. Hence $(u, v) \notin A(D_{q_1}) \cup A(D_{q_2})$, so we need to introduce arc (u, v) in some node between q and r. Since v, w are in the same label in D_q, they have to have the same labels in any node between q and r. Hence, an arc creation between label of v and label of u would also introduce an arc (u, w) which is in contradiction with $(u, w) \notin A$. Hence, the path property holds.

We also need to show the second case of the *neighborhood property.* Assume that there are three vertices $u, v, w \in V$, $(v, u) \in A$, $(w, u) \notin A$ and v, w are in the same group of T_i but u, v are in different components in T_{i-1}, respectively in components c_1 and c_2. Since v, w are in the same group in T_i then they have to in the same component c in T_i. Let $q \in U_i$ be an $\oplus$-node such that $v, w \in V(D_q) = c$, let q_1, q_2 be children of q such that $V(D_{q_1}) = c_1$ and $V(D_{q_2}) = c_2$. Hence $(v, u) \notin A(D_{q_1}) \cup A(D_{q_2})$, so we need to introduce the arc (v, u) in some node between q and r. Since v, w are in the same label in D_q, they have to have the same label in any node between q and r. Hence, an arc creation between label of v and label of u would also introduce an arc (w, u) which is in contradiction with $(w, u) \notin A$. Hence, the neighborhood property holds.

To show that the *path property* holds, we use contradiction as before. Assume that four vertices $u, v, w, x \in V$, such that $(u, v), (u, w), (x, v) \in A$ and $(x, w) \notin A$, u, x are in the same group in T_i and v, w are in the same group in T_i, but u, v are in the different components c_1 and c_2 in T_{i-1}, respectively. According to the neighborhood property, u, w need to be in the same component in T_{i-1} and v, x need to be in the same component in T_{i-1}. Since u, x are in the same group of T_i, then they have to be in the same component c in T_i. Since u, w are in the same component in $Ti - 1$, then they have to be in the same component in T_i. Since u, x are in c in T_i and u, w are in the same component in T_i, then $w \in c$. Similarly we can see that $v \in c$ in the template T_i. Let $q \in U_i$ be an $\oplus$-node such that $v, w \in V(D_q) = c$, let q_1, q_2 be children of q such that $V(D_{q_1}) = c_1$ and $V(D_{q_2}) = c_2$. Since u and v are in different components in T_{i-1} then $u, v \notin E(D_{q_1}) \cup E(D_{q_2})$. Therefore, we need to introduce arc (u, v) in some node between q and r. Since v, w are in the same label in D_q and u, x are in the same label in D_q, they have to have the same labels in

any node between q and r. Hence an arc creation between label of v, w and label of u, x would also introduce an arc (x, w) which is in contradiction with $(x, w) \notin A$. Hence, the path property holds.

We can conclude that $\mathcal{D}$ is a k-derivation of G. The procedure above can be clearly carried out in polynomial time. □

Lemma 7. *From a k-derivation of a digraph $D = (V, A)$ we can obtain a k-expression of D in polynomial time.*

Proof. Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a k-derivation of digraph $D(V, A)$ with the label set $\{1 \ldots k\}$. According to Lemma 2, we can assume that $\mathcal{D} = (T_0, \ldots, T_t)$ is a strict k-derivation of D. We are going to construct a k-expression tree for D from its k-derivation in polynomial time.

The construction consists of three steps.

In the first step, we construct a k-expression tree $Q_{\oplus}$ which consists of leaves and $\oplus$-nodes. For each component $c = \{v\} \in \mathrm{cmp}(T_0)$, where $v \in V$, we introduce a node $q(c, 0)$ with the label $1(v)$. These are the leaves of the k-expression tree. Then for each $c \in \mathrm{cmp}(T_i)$, we introduce an $\oplus$-node $q(c, i)$ and add edge with children nodes if there is a node $q(c', i-1)$ such that $c' \subseteq c$. D1 and D3 properties ensure that the $Q_{\oplus}$ is a tree. $Q_{\oplus}$ does not have to be succinct.

In the second step, we are going to obtain a k-expression tree $Q_{\oplus,\rho}$ by adding the ρ operation on $Q_{\oplus}$. By depth first ordering, we can visit each node $q(c, i)$ and insert at most k ρ-node between q, i and $q'(c', i-1)$ such that $\mathrm{grp}(A_{q'}) = \{g \in \mathrm{grp}(A_{q(c,i)}) : g \subseteq c\} \subseteq \mathrm{grp}(T_i)$. Since any partition can be obtained by its refinements, properties D2 and D3 ensure that such a node insertion is possible.

The last step is obtaining the k-expression tree $Q_{\oplus,\rho,\alpha}$ by adding α-nodes on $Q_{\oplus,\rho}$. We are going to show that for each arc $(u, v) \in A$, there is an added α-node q above an $\oplus$-node q in $Q_{\oplus,\rho,\alpha}$ such that q is a child of p and it introduces the arc (u, v) but does not introduce an arc not found in A.

Let $q(c, i)$ be the $\oplus$-node with the smallest i in $Q_{\oplus,\alpha}$ such that $u, v \in V(A_q)$. Therefore, there are two $q_1(c_1, i-1)$, $q_2(c_2, i-1)$ child nodes of q in $Q_{\oplus,\alpha}$ such that $u \in V(A_{q_1})$ and $v \in V(A_{q_2})$. It follows that $c_1, c_2 \in T_{i-1}$ are distinct components with $u \in c_1$ and $v \in c_2$. According to the arc property, if u and v in different components in T_i then u and v are in different groups of T_i. This means that u and v have different labels in G_q, l_u and l_v respectively. We can add a α-node p with operation α_{l_u,l_v} above q as it is the parent node. The node p introduces the arc (u, v). It remains that it does not introduce an arc not found in A. We can show it by each vertex couple $u', v' \in c$ with labels l_u and l_v respectively. We need to consider four cases.

Case 1: $u = u'$, $v = v'$, then trivially $(u', v') = (u, v) \in A$.

Case 2: $u = u'$, $v \neq v'$. Assume on the contrary that $(u', v') \notin A$. Since v and v' have the same label in D_q that means they are in the same group of T_i. According to the neighborhood property, u and v have to be in a same component T_{i-1} which is contradiction to smallest choice of i. Hence $(u', v') \in A$.

Case 3: $u \neq u'$, $v = v'$ Assume on the contrary that $(u', v') \notin A$. Since u and u' have the same label in D_q that means they are in the same group of T_i. According to the neighborhood property, u and v have to be in the same component T_{i-1} which is in contradiction to the smallest choice of i. Hence $(u', v') \in A$.

Case 4: $u \neq u'$, $v \neq v'$ We can proof once more by contradiction. Assume that $u'v' \notin A$. By Case 2 and Case 3, we know that $uv', u'v \in A$. According to the path property, it follows that u and v belong to the same component in T_{i-1}. Then it is a contradiction to the smallest choice of i.

Since α-nodes do not introduce a non-existing edge, we can apply the procedure above to obtain a k-expression tree of D in polynomial time. □

Lemma 8. *Let $1 \leq k \leq n$. If a directed graph with n vertices has a k-derivation, then it has a k-derivation of length $n - k + 1$.*

Proof. The proof is quite similar to the proof of Lemma 5. We also use the *k-length* of a derivation. Let $l(n, k)$ be the largest k-length of a strict derivation with the universe size n. We can use the same claims from Lemma 5, since they are based on the derivation.

Claim 1: $l(n, k) < l(n + 1, k)$.

Claim 2: Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a strict derivation over a universe V of size n of k-length $l(k, n)$. Then, template $T_{t-l(k,n)+1}$ has exactly one component the size of $k + 1$ and all other components are singleton.

Claim 3: $l(n, k) < n - k$.

We need to proof the lemma with the claims for a derivation of a directed graph.

Now we can prove the lemma. Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a strict derivation of a digraph $D(V, A)$ with $|A| = n$. We can assume that $\mathcal{D}$ is a strict derivation according to Lemma 1. Let l be k-length of $\mathcal{D}$ and let $j = t - l$. We define a derivation $\mathcal{D}' = (T_0, T'_j, T_{j+1} \ldots, T_t)$ where $T_0, T_{j+1}, \ldots T_t$ are templates of $\mathcal{D}$ and $\text{cmp}(T'_j) = \text{cmp}(T_j)$, $\text{grp}(T'_j) = \text{grp}(T_0)$. We need to show that $\mathcal{D}'$ is a k-derivation of D. For this we need to consider the arc, neighborhood and path properties for T'_j and T_{j+1}. T'_j has only singleton groups, then T'_j satisfies the properties. Also the properties hold for T_{j+1}, because T'_j has exactly the same components as T_j has. The length of derivation $\mathcal{D}'$ is $t - j + 1$. When we substitute j with $t - l$, then the length of $\mathcal{D}'$ is $l + 1$. By Claim 3 we know that $l \leq n - k$. Hence $l + 1 \leq n - k + 1$. □

Similar to the way in which Heule and Szeider concluded with Proposition 1, we can also conclude with the Proposition 2 with respect to the Lemmas 6, 7, and 8.

Proposition 2. *Let $1 \leq k \leq n$. A digraph D with n vertices has clique-width at most k if and only if D has a k-derivation of length at most $n - k + 1$.*

CHAPTER 4

Encoding a Derivation of a Graph

4.1 Undirected Graphs

Let $G = (V, E)$ be an undirected graph and $t > 0$ be an integer parameter. Heule and Szeider constructed a CNF formula $F_{der}(G, t)$ such that it is satisfiable if and only if there is a derivation of length t for G.

It is assumed that the vertices of G are given in some arbitrary but fixed linear order. For any distinct two vertices $u, v \in V(G)$, they introduce a propositional variable $c_{u,v,i}$ which indicates whether u and v are in the same *component* of template T_i. Also, in the same sense, they introduce a propositional variable $g_{u,v,i}$ for any two distinct vertices $u, v \in V(G)$ to indicate whether the vertices are in the same *group* of template T_i. We start to obtain clauses for derivation conditions D1-D4.

The formula $F_{\text{der}}(G, t)$ consists of the conjunction of all clauses generated below.

For the conditions D1–D4:

$$(\bar{c}_{u,v,0}) \wedge (c_{u,v,t}) \wedge (c_{u,v,i} \vee \bar{g}_{u,v,i}) \wedge (\bar{c}_{u,v,i-1} \vee c_{u,v,i}) \wedge (\bar{g}_{u,v,i-1} \vee g_{u,v,i}) \quad \text{for } u, v \in V,\ u < v,\ 0 \leq i \leq t.$$

The transitive properties of components and groups also need to be considered such that for the vertices u, v, w and a component c, if $u, v \in c$ and $u, w \in c$, then $v, w \in c$.

$$(\bar{c}_{u,v,i} \vee \bar{c}_{v,w,i} \vee c_{u,w,i}) \wedge (\bar{c}_{u,v,i} \vee \bar{c}_{u,w,i} \vee c_{v,w,i}) \wedge (\bar{c}_{u,w,i} \vee \bar{c}_{v,w,i} \vee c_{u,v,i}) \ \wedge$$
$$(\bar{g}_{u,v,i} \vee \bar{g}_{v,w,i} \vee g_{u,w,i}) \wedge (\bar{g}_{u,v,i} \vee \bar{g}_{u,w,i} \vee g_{v,w,i}) \wedge (\bar{g}_{u,w,i} \vee \bar{g}_{v,w,i} \vee g_{u,v,i}) \quad \text{for } u, v, w \in V,\ u < v < w,\ 0 \leq i \leq t.$$

For the *edge property*, we need to include the clauses for any two vertices $u, v \in V$ with $u < v$, $uv \in E$ and $1 \leq i \leq t$:

$$(c_{u,v,i-1} \vee \bar{g}_{u,v,i}).$$

For the *neighborhood property* the following clauses are needed for any three vertices $u, v, w \in V$ with $uv \in E$ and $uw \notin E$ and $1 \leq i \leq t$:

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

For the *path property*, we need to include the following clauses for any four vertices u, v, w, x, such that $uv, uw, vx \in E$, and $wx \notin E$, $u < v$ and $1 \leq i \leq t$:

$$(c_{u,v,i-1} \vee \bar{g}_{\min(u,x),\max(u,x),i} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

The following statement is a direct consequence of the above definitions.

Lemma 9. *$F_{\text{der}}(G, t)$ is satisfiable if and only if G has a derivation of length t.*

Until this part, we have set of clauses to ensure that we can obtain a derivation with length t. Now we need to introduce new variables and clauses to present the width of a derivation of at most k. Heule and Szeider have two approaches for encoding the width of derivation. Their first approach is *direct encoding*. The idea of *direct coding* is to introduce a variable for each vertex to indicate its group number for each template. But this approach can produce clauses such that unit propagation of SAT solvers can not result in a conflict. Therefore, they proposed *representative encoding* which uses two types of variables, the *representative variable* $r_{v,i}$ and the *order variable* $o^{>}_{v,a,i}$. The representative variable $r_{v,i}$ indicates that if the vertex v represents a group in template T_i. For each group, just one vertex can present it which will the smaller vertex from the group since the vertices are given in fixed linear order. We can express this property by the following clauses:

$$(r_{v,i} \vee \bigvee_{u \in V, u<v} g_{u,v,i}) \wedge \bigwedge_{u \in V, u<v} (\bar{r}_{v,i} \vee \bar{g}_{u,v,i}) \qquad \text{for } v \in V,\ 0 \leq i \leq t$$

The order variable $o^{>}_{v,a,i}$ expresses that the label number of the vertex v is greater than a in template T_i, with $v \in V$, $D = \{1, \ldots, k\}$, $a \in D \setminus \{k\}$, $0 \leq i \leq t$. For example, assigning $o^{>}_{v,2,2} = 1$ means that the label of v in T_2 is greater than 2. Heule and Szeider coded the number of the representative variables at most k in a component with the order variables.

If two vertices u and v are representative in the same component of a template T_i and $u < v$, then $o^{>}_{u,a,i} = 0$ and $o^{>}_{v,a,i} = 1$ must be hold. Since $u < v$ and u, v are representatives, u can not have the highest numbered label and v can not have the lowest numbered label. Furthermore, if $o^{>}_{u,a,i} = 1$ and $u < v$ then $o^{>}_{v,a,i} = 1$. The related clauses are:

$(\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee \bar{o}^{>}_{u,k-1,i}) \wedge (\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee o^{>}_{v,1,i}) \wedge$
$\bigwedge_{1 \leq a < k-1} (\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee \bar{o}^{>}_{u,a,i} \vee o^{>}_{v,a+1,i})$
for $u, v \in V$, $u < v$, $0 \leq i \leq t$.

The representative encoding requires $n(n+k-1)(n-k+2)$ variables and worst case $\mathcal{O}(n^5 - n^4 k)$ clauses.

4.2 Directed Graphs

Directed graphs have quite similar encoding to undirected graphs. The main differences appear in the encoding of properties. In particular, the encoding of the derivation conditions D1-D4 and the representative encoding of directed graphs stay exactly same for undirected graphs. Therefore, in this section we just consider the clauses for digraph conditions: *arc property, neighborhood property* and *path property.*

We have the same assumptions as for undirected graphs. The vertices are given in some arbitrary but fixed linear order. For any two distinct vertices $u, v \in V(G)$, we introduce a propositional variable $c_{u,v,i}$ which indicates whether u and v are in the same *component* of template T_i. Also in the same sense, we introduce a propositional variable $g_{u,v,i}$ for any two distinct vertices $u, v \in V(G)$ to indicate whether the vertices are in the same *group* of template T_i.

In order to enforce the *arc property* of a directed graph $D = (V, A)$, we add the following clauses for any $u, v \in V$, $(u, v) \in A$ and $1 \leq i \leq t$:

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(u,v),\max(u,v),i}).$$

For the *neighborhood property*, the following clauses are needed for any three vertices $u, v, w \in V$ $1 \leq i \leq t$:

i) $(u, v) \in A$ and $(u, w) \notin A$

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

ii) $(v, u) \in A$ and $(w, u) \notin A$

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

For the *path property*, we need the following clauses for any four vertices $u, v, w, x \in V$, such that $(u, v), (u, w), (x, v) \in A$, and $(x, w) \notin A$ and $1 \leq i \leq t$:

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(u,x),\max(u,x),i} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

The representative encoding of digraphs requires $n(n+k-1)(n-k+2)$ variables. The number of clauses depends on the digraph and is $\mathcal{O}(n^5 - n^4k)$ which is same for undirected graphs.

CHAPTER 5

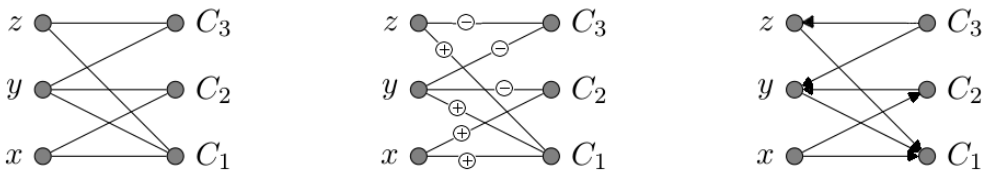
Model Counting with Clique-width

The model counting problems have been studied in recent research areas of informatics such as AI, probabilistic reasoning, contingency planning, and hard combinatorial problems. Simply finding a solution to such problems can be challenging, to count the number of solutions is much harder [GSS09]. Fischer et al. proposed a model counting algorithm that counts the number of models of formulas of bounded clique-width in polynomial time [FMR08]. The algorithm calculates the number of models of a formula using a k-expression tree of the directed incidence graph of the formula. In Chapter 4.2 we obtained an approach for determining the clique-width of a directed graph and in this chapter we consider the approach of Fischer et al. and in Section 5.3 we present the result of an application (program) based on this approach.

5.1 Directed Incidence Graph

The *incidence graph* of a CNF formula F is a bipartite graph such that each clause and variable is presented as a vertex and there is an edge between a variable vertex and a clause vertex if the variable occurs in the clause. The *signed incidence graph* of a clause set F is an incidence graph of F with edge labels $(+)$ and $(-)$ to indicate whether variables occur positively or negatively in a clause. The *directed incidence graph* of a clause set F is a digraph obtained from a signed incidence graph of F with oriented edges such that if the sign of an edge is positive then the orientation of the arc is from the vertex of the variable to the vertex of the clause and if it is negative the other way around.

Example 3. Consider propositional formula $F = \{\{x, y, z\}, \{x, \bar{y}\}, \{\bar{y}, \bar{z}\}\}$ with $C_1 = \{x, y, z\}$, $C_2 = \{x, \bar{y}\}$ and $C_3 = \{\bar{y}, \bar{z}\}$. The incidence graph of F is shown in Figure 5.1a and the directed incidence graph of F is shown in Figure 5.1c. ⊣



(a) $I(F)$ (b) $SI(F)$ (c) $DI(F)$

Figure 5.1: Incidence graph (a), signed incident graph (b) and directed incidence graph (c) of formula $F = \{C_1, C_2, C_3\}$ with $C_1 = \{x, y, z\}$, $C_2 = \{x, \bar{y}\}$ and $C_3 = \{\bar{y}, \bar{z}\}$

5.2 Model Counting of Formulas of Bounded Clique Width

This section is based on the paper [FMR08] from Fischer, Makowsky and Ravve. Fischer et al. proposed an algorithm that calculates the number of models of a formula on the nodes of the k-expression tree of its directed incidence graph. The intuitive idea is that we have a table for each node of the k-expression tree. We initiate the tables of the leaf nodes and calculate the table of the inner nodes by starting with the leaves of the k-expression tree and going up from child nodes to parent nodes. We repeat this process until we reach the root node. At the end, we have the number of models of the formula in the table of the root node.

Theorem 1 ([FMR08])**.** *Given a CNF formula F and signed parse tree $der_{SI}(F)$ for clique-width of up to k, it is possible to calculate $\#F$, with a number of algebraic operations that is linear in the size of the parse tree $der_{SI}(F)$, and exponential in k.*

Before starting the formulation and describing the steps of the algorithm, we need to clarify that if there are no clauses in the formula $F = \{\}$ then the number of models of F is 2^n, where n is the number of variables in the scope and if there are no variables in the scope and no clauses in F, then number of models of F is $2^0 = 1$.

The table of the nodes of a k-expression tree keeps the number of models of its *transformation*. A *transformation* $F^{(A,B,C)}$ of a formula F for given subsets A, B, C of $\{1, \ldots, k\}$ is described with the following operations:

1. Remove every clause labeled with $i \in A$ from F.
2. Let X_i be the set that consists of all variables labeled with $i \in \{1, \ldots, k\}$. For each label $i \in B$ add a clause that consists of the disjunction of the all variables $x \in X_i$.
3. For each label $i \in C$ add a clause that consists of the disjunction of the all negation of variables $x \in X_i$.

Example 4. Let $F = \{C_1, C_2\}$, $C_1 = \{x, y\}$ and $C_2\{y, \bar{z}\}$, where label $l_{C_1} = 1, l_{C_2} = 2, l_x = 3, l_y = 3, l_z = 2$. Then $F^{\{1\},\{2\},\{3\}} = \{C_2, \{z\}, \{\bar{x}, \bar{y}\}\}$ ⊣

Fischer et al. assume that all unions are made by disjoint subsets of the label set. This can be achieved by constructing a k-expression tree with duplicate the number of labels. The idea is that for each union node with binary child nodes, we can shift the label numbers of one child by k, and after the union operation we can use the relabeling operation to set back the labels from $\{k+1, \ldots, 2k\}$ to $\{1, \ldots, k\}$. Also, if there is a union node consisting of more than two child nodes in the k-expression tree, it can be reconstructed with a union of only two child nodes and vice versa. Since the algorithm is required in exponential time in k, the duplication of k reduces the performance of its practical application.

For each node of the k-expression tree of a formula F, we have a table that consists of the number of models $\#F^{(A,B,C)}$ for each possible triple subsets A, B, C (they do not need to be disjoint) of $\{1, \ldots, k\}$. The calculation of tables of each node starts from leaf nodes to inner nodes by the following:

1. If the node v is a disjoint union of u and w:

$$\#F_v^{A,B,C} = \#F_u^{A,B,C}.\#F_w^{A,B,C}$$

2. If the node v is an arc creation $\alpha_{i,j}(F_w)$ where w is the child node of v.

   a) If i is the label of clauses and j is the label of variables:

      i. if $i \in A$ then

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A,B,C)}),$$

      ii. if $j \in C$

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A\cup\{i\},B,C)}),$$

      iii. otherwise

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A,B,C)}) + \#(F_w^{(A\cup\{i\},B,C\cup\{j\})}) - \#(F_w^{(A,B,C\cup\{j\})}).$$

   b) If j is the label of clauses and i is the label of variables:

      i. if $j \in A$ then

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A,B,C)}),$$

      ii. if $i \in B$

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A\cup\{j\},B,C)}),$$

      iii. otherwise

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A,B,C)}) + \#(F_w^{(A\cup\{j\},B\cup\{i\},C)}) - \#(F_w^{(A,B\cup\{i\},C)}).$$

3. If the node v is a relabeling operation $\rho_{i,j}(F_w)$ where w is a child node of v:

a) $i \in B \cup C$

$$\#(F_v^{A,B,C}) = 0$$

b) if $j \notin B \cup C$ and our setting is: if $j \in A$ then $A' = A \cup \{i\}$, otherwise $A' = A \setminus \{i\}$,

$$\#(F_v^{(A,B,C))} = \#(F_w^{(A',B,C))}$$

c) if $j \in B$ and $j \notin C$, and with the setting: if $j \in A$ then $A' = A \cup \{i\}$ otherwise $A' = A \setminus \{i\}$, $B_1 = B \cup \{i\} \setminus \{j\}$, $B_2 = B$ and $B_3 = B \cup i$

$$\#(F_v^{(A,B,C)}) = \#(F_w^{(A',B_1,C)}) + \#(F_w^{(A',B_2,C)}) - \#(F_w^{(A',B_3,C)})$$

d) if $j \notin B$ and $j \in C$, and with the setting: if $j \in A$ then $A' = A \cup \{i\}$, otherwise $A' = A \setminus \{i\}$, $C_1 = C \cup \{i\} \setminus \{j\}$, $C_2 = C$ and $C_3 = C \cup \{i\}$

$$\#(F_v^{(A,B,C)}) = \#(F_v^{(A',B,C_1)}) + \#(F_v^{(A',B,C_2)}) - \#(F_v^{(A',B,C_3)})$$

e) if $j \in B$ and $j \in C$

if $j \in A$ then $A' = A \cup \{i\}$, otherwise $A' = A \setminus \{i\}$,
$B_1 = B \cup \{i\} \setminus \{j\}$, $B_2 = B$,$B_3 = B \cup i$,$C_1 = C \cup \{i\} \setminus \{j\}$, $C_2 = C$ and $C_3 = C \cup \{i\}$

$$\begin{aligned}\#(F_v^{(A,B,C)}) = {} & \#(F_v^{(A',B_1,C_1)}) + \#(F_v^{(A',B_1,C_2)}) + \#(F_v^{(A',B_2,C_1)}) + \\ & \#(F_v^{(A',B_2,C_2)}) - \#(F_v^{(A',B_3,C_1)}) - \#(F_v^{(A',B_3,C_2)}) - \#(F_v^{(A',B_1,C_3)}) - \\ & \#(F_v^{(A',B_2,C_3)}) + \#(F_v^{(A',B_3,C_3)})\end{aligned}$$

The intuitive idea behind the calculation is that we already keep the possible transformations in the tables, and calculate the current table values from the previous table by the set inclusion/exclusion principle.

5.3 An Application on Model Counting

We can solve a model counting problem of a formula in polynomial time if we are given a k-expression of the directed incidence graph of the formula where k is a constant. We can achieve this by having a table for each node of the k-expression tree of the directed incidence graph of a formula and calculating tables of parent nodes from the table of their child nodes, once we initiate the tables of leaf nodes. Therefore, we wrote a program that transforms a k-derivation to its k-expression tree based on the Lemma 6. We wrote the program [1] in programming language *Python*. It takes the output of the decoder program and generates the related k-expression tree. In order to obtain the k-expression tree from a graph, we run the sequence of programs in the order: encoder, SAT solver,

[1] The code of the program is available on `https://bitbucket.org/ayParlak/thesis-codes`.

Table 5.1: A table of transformations for $\{0, \ldots, 3\}$.

A	B	C	$\#F^{(A,B,C)}$
$\emptyset$	$\emptyset$	$\emptyset$	n_1
$\emptyset$	$\emptyset$	$\{1\}$	n_2
$\emptyset$	$\emptyset$	$\{2\}$	n_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
{1,2,3}	{1,2,3}	{1,2,3}	n_{512}

decoder and k-expression tree generator. We automated the running of the sequence with a shell script. We used a 4-core QEMU Virtual CPU(2665.908 Mhz, 4096 KB) 16 GB RAM machine running Ubuntu 10.04 for all our testing and the SAT solver *Glucose* version 2.2 [AS09] to check the satisfiability of the CNF formulas, as it was one of the best solvers for our instances [HS15].

Example 5. Consider the propositional formula $F = \{\{x, y, z\}, \{x, \bar{y}\}, \{\bar{y}, \bar{z}\}\}$ with $C_1 = \{x, y, z\}$, $C_2 = \{x, \bar{y}\}$ and $C_3 = \{\bar{y}, \bar{z}\}$. The directed incidence graph of F is shown in Figure 5.2. The k-derivation of the directed incidence graph is $z^2y^1C_3^3x^1C_2^4C_1 : z^3x^4y^5C_3^3C_2^5C_1$. ⊣

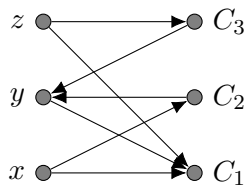


Figure 5.2: Directed incidence graph of formula $F = \{C_1, C_2, C_3\}$ with $C_1 = \{x, y, z\}$, $C_2 = \{x, \bar{y}\}$ and $C_3 = \{\bar{y}, \bar{z}\}$

.

The algorithm from Fischer et al. proposes keeping a table (see Table 5.1) that includes all possible *transformations* of the formula for each node of the k-expression tree. The size of the table is exponential with k such that 2^{3k} and the number of operations is linear in size of the k-expression tree. We can solve a $\#P$ problem in polynomial time but since it is exponential in k, the performance of the program strictly depends on k (for $k \geq 7$, we can run out of space on our current testing machine, see Table 5.2). As we pointed out in Section 5.2, the algorithm requires a disjoint subset of labels used by the child nodes of union nodes. This can force the algorithm to duplicate the number of labels and increase the performance manifestly.

The performance of the program for small size problems is not significant for comparing it with other exact model counting programs. It can have comparable performance with big size problems, or instance with small clique-width. However, it is a still challenging topic to find the clique-width for the big size of graphs (see Figure 6.2).

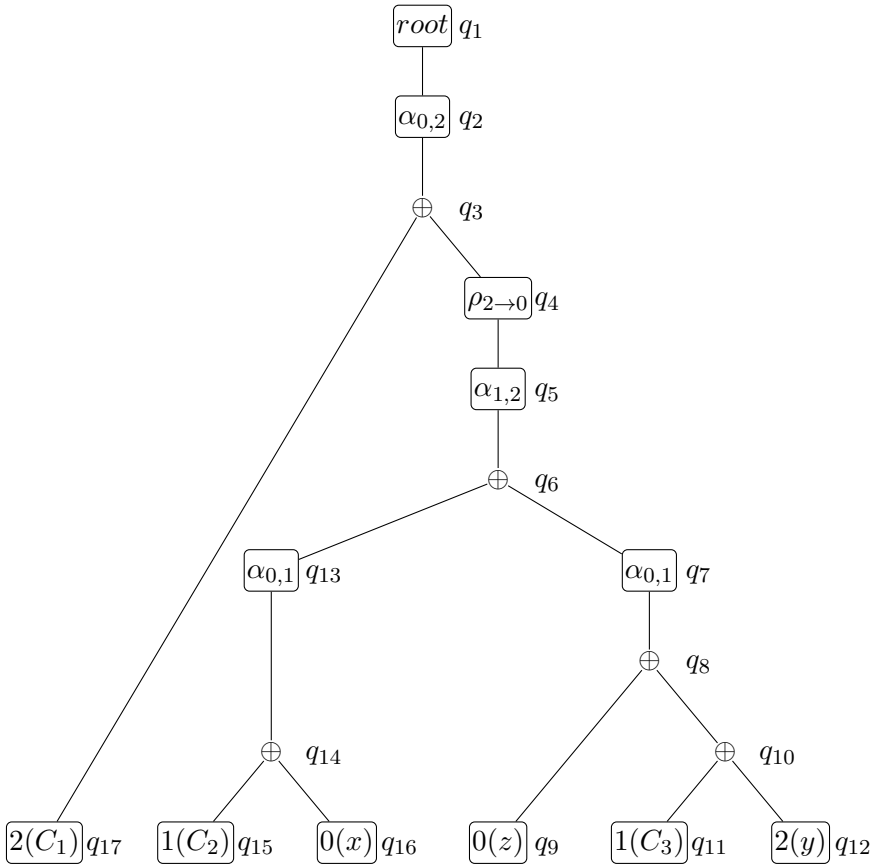


Figure 5.3: 3-expression tree of the directed incidence graph of Figure 5.2, Example 5.

Table 5.2: Size of transformations table for different values of k.

k	Size of transformations table (2^{3k})
1	8
2	64
3	512
4	4 096
5	32 768
6	262 144
7	2 097 152
8	16 777 216

CHAPTER 6

Experimental Results

In this chapter, we review the experimental results of the SAT approach to clique-width from the paper by Heule and Szeider [HS15], as well as the experimental results of our SAT approach to clique-width of digraphs. We used a 4-core QEMU Virtual CPU(2665.908 Mhz,4096 KB) 16 Gb RAM machine running Ubuntu 10.04 for all our testing and the SAT solver *Glucose* version 2.2 [AS09] to check the satisfiability of the CNF formulas, as it was one of the best solvers for our instances [HS15].

6.1 Clique-Width of Undirected Graphs

In this section we review the results from Heule and Szeider. They calculate the clique-width of a graph by determining in which value of $k \in \{1 \dots |V|\}$, it holds that $F(G, k, |V| - k + 1)$ is satisfiable and $F(G, k-1, |V| - k + 2)$ is unsatisfiable.

They provided two programs that are written in the programming language C.[1] The first program is the decoder which decodes a given graph in DIMACS format and a parameter k into CNF in DIMACS format. The CNF formula is satisfiable if and only if the graph has a clique-width of less or equal to k. The second program is the decoder which converts the CNF formula in DIMACS format into a k-derivation. They used *string notation* to present a k-derivation as a string.

For a graph $G = (E, V), V = \{a_1, \dots, a_n\}$, a *component string*

$$a_1^{c(a_1,a_2)} a_2^{c(a_2,a_3)} a_3^{c(a_3,a_4)} \dots a_{n-1}^{c(a_{n-1},a_n)} a_n$$

of the components of a derivation $\mathcal{D} = (T_0, \dots, T_t)$ is an order of vertices of graph with the smallest template number $c(a, a')$ such that a and a' appear in the same component.

[1] The sources of the encoding are available on `https://bitbucket.org/mjhheule/cwd-encode/`.

In the same way a *group string*

$$a_1^{g(a_1,a_2)} a_2^{g(a_2,a_3)} a_3^{g(a_3,a_4)} \dots a_{n-1}^{g(a_{n-1},a_n)} a_n$$

of the groups of a derivation $\mathcal{D} = (T_0, \dots, T_t)$ is an order of vertices of graph with the smallest template number $g(a, a')$ such that a and a' appear in the same group. It is easy to see that such an order always exists and it may not be unique. Therefore, we can present a k-derivation in a string notation (see Example 6).

Example 6. Consider the Petersen graph (Figure 6.1) $G = (E, V)$, $V = \{a, b, c, d, e, f, g, h, i, j\}$, $E = \{ac, ad, af, bd, be, bg, ce, ch, di, ej, fg, fj, gh, hi, ij\}$. Its 5-derivation $(T_0, \dots, T_4)$ is

$$\begin{array}{lcllcl}
\mathrm{cmp}(T_0) & = & \{a, b, c, d, e, f, g, h, i, j\}, & \mathrm{grp}(T_0) & = & \{a, b, c, d, e, f, g, h, i, j\}, \\
\mathrm{cmp}(T_1) & = & \{ace, bfg, dhi, i\}, & \mathrm{grp}(T_1) & = & \{a, b, c, d, e, f, g, h, i, j\}, \\
\mathrm{cmp}(T_2) & = & \{abcefg, dhi, j\}, & \mathrm{grp}(T_2) & = & \{a, b, cg, d, e, f, h, i, j\}, \\
\mathrm{cmp}(T_3) & = & \{abcdefghi, j\}, & \mathrm{grp}(T_3) & = & \{ab, cg, d, efi, h, j\}, \\
\mathrm{cmp}(T_4) & = & \{abcdefghij\}, & \mathrm{grp}(T_4) & = & \{ab, cg, dh, efi, j\};
\end{array}$$

By the string notation we can present the 5-derivation of the Petersen graph with

$$a^1 c^1 e^2 b^1 f^1 g^3 d^1 h^1 i^4 j : a^3 b^5 c^2 g^5 d^4 h^5 e^3 f^3 i^5 j.$$

⊣

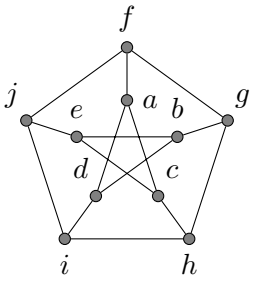


Figure 6.1: The Petersen graph

The performance of the program for upper bounds is better than for lower bounds (Table 6.1). There are two reason behind this performance difference. First, since the derivation length is smaller, by Proposition 1, the upper bounds formulas are smaller. Second, for lower bounds, all the search space needs to be explored, while for upper bounds the solution can be found on a branch without exploring other branches. Further more, the running time of the CNF formula creation is significantly less than the running time of the SAT solver. Therefore, the performance of the program is determined by the performance of the SAT solver (Table 6.1).

k	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Time_1(s)	0.22	0.20	0.19	0.19	0.16	0.15	0.15	0.16	0.13	0.15	0.13	0.11	0.12	0.22
Time_2(s)	0.76	3.17	8.88	12.99	26.40	43.20	121.85	0.22	0.30	0.17	0.13	0.11	0.09	0.07

Table 6.1: A random graph G with 20 vertices and 101 edges for different values of k. Time_1 presents the running time of creating clauses of the CNF formula and Time_2 presents the running time of the SAT solver. Up to $k = 9$ the formulas are unsatisfiable, after that they are satisfiable, therefore $\text{cw}(G) = 10$.

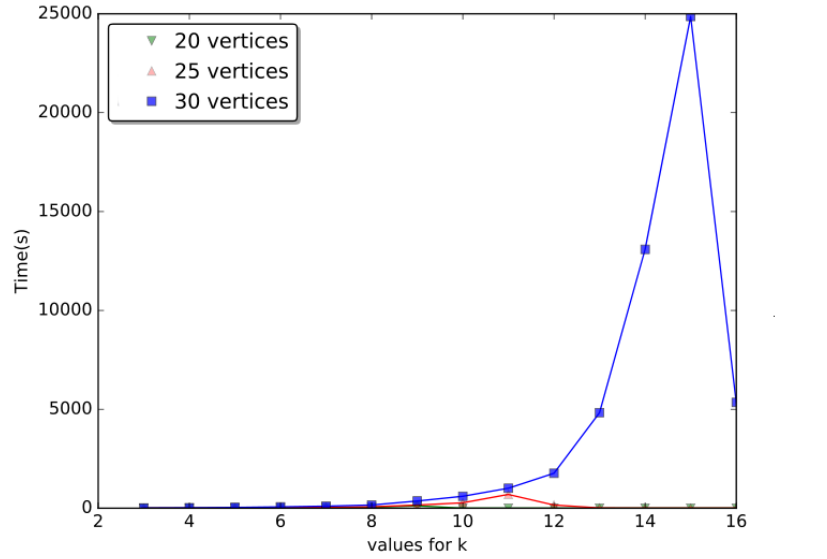


Figure 6.2: Time performance of the program for different candidate values of k for random graphs.

The program for a graph G and $\text{cwd}(G) = k$ has the greatest running time on candidate value $k - 1$ for $\text{cwd}(G)$. When we consider the greatest running time (24864 s) for the graph with 30 vertices in Figure 6.2, we can observe that the performance of the algorithm is restricted for the larger graphs.

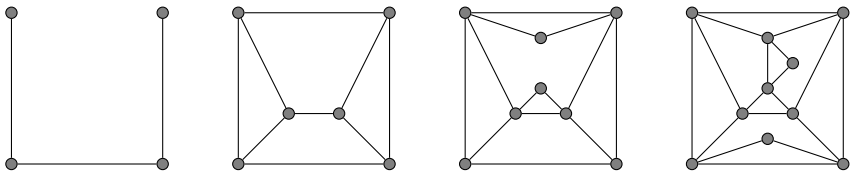


Figure 6.3: Smallest graphs with clique-width 3, 4, 5, and 6 (from left to right) [HS15].

k	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Time_1(s)	0.17	0.17	0.17	0.16	0.12	0.11	0.11	0.12	0.12	0.12	0.14	0.10	0.10	0.09
Time_2(s)	0.40	1.13	4.56	7.84	11.48	26.16	24.98	74.37	1.34	0.38	0.11	0.10	0.10	0.09

Table 6.2: A directed graph G' (obtained by randomly orienting the edge of graph G from the Table 6.1) with 20 vertices and 101 arcs for different values of k. Time_1 present the running time of creating clauses of CNF formula and Time_2 presents the running time of SAT solver. Up to $k = 10$ the formulas are unsatisfiable, after they are satisfiable, therefore $\text{dcw}(G) = 11$.

6.2 Clique-Width of Directed Graphs

We extended the approach of Heule and Szeider for directed graphs. We modified the encoder and decoder they provided, for directed graphs[2]. We also used Nauty [McK81] to generate non-isomorphic connected digraphs with up to seven vertices.

(a) Petersen graph (b) A directed version of Petersen graph

Figure 6.4: Petersen graph with 5-derivation: $a^1c^1e^2b^1f^1g^3d^1h^1i^4j{:}a^3b^5c^2g^5d^4h^5e^3f^3i^5j$ and orientation of it with 6-derivation: $f^5j^1h^1e^1c^2i^1a^1d^3g^4b{:}f^6j^5g^6i^3c^4h^6a^6e^2d^6b$.

In addition, we reviewed the directed version of some famous graphs. One of them is the Petersen graph (10 vertices, 15 edges graph named after the Danish mathematician Julius Petersen) which has clique-width of 5. It has 324 orientations and 2 of them have clique-width of 6 while others have 5, see Figure 6.4.

As we discussed in Section 3.2, an orientation of an undirected graph has clique-width greater than or equal to the clique-width of its underlying graph [CO00]. The performance of the algorithm for directed graphs and for undirected graphs do not have significant differences (Table 6.1, Table 6.2).

[2]The sources of the encoding are available on `https://bitbucket.org/ayParlak/thesis-codes`.

6.3 The Clique-Width Numbers

Heule and Szeider defined a term called *clique-width number*. The *clique-width number* for a number k is the smallest number n_k such that there exists an n_k-vertex graph with clique-width k. They published also the experimental results up to ten vertices (see Table 6.4). They used Nauty [McK81] to generate non-isomorphic connected graphs. Additionally they eliminated the non-prime [HP10] graphs. As a result of this practical work they determined the first seven clique-width numbers: $1, 2, 4, 6, 8, 10, 11$. Figure 6.3 shows the four smallest graphs that correspond to the clique-width numbers 3, 4, 5 and 6. In our experimental work, we determined the first five clique-width numbers for directed graphs: $1, 2, 3, 4, 6$. Figure 6.5 shows the three smallest digraphs that correspond to the clique-width numbers 3, 4 and 5.

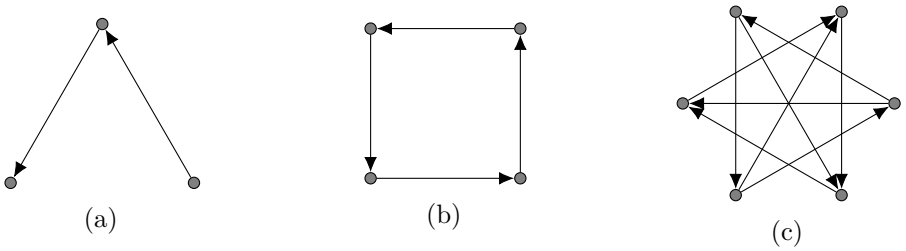


Figure 6.5: Smallest digraphs with clique-width 3, 4, and 5 (from left to right).

The clique-width number is different for undirected and directed graphs, since they have different operation in their construction. For the same value of k, a clique-width number for undirected graphs is greater or equal than the number for directed graphs. Figure 6.3 shows n_k for different values of k for undirected and directed graphs. n_1 and n_2 are trivial cases. Because we can not insert an edge or an arc with one label and we can build just a complete graph with two labels (the smallest complete graph with an edge or an arc is K_2). The differences start from three labels (see Figure 6.5a and 6.3).

Table 6.3: Clique-width numbers for different values of k for undirected and directed graphs.

k	n_k (undirected)	n_k (directed)
1	1	1
2	2	2
3	4	3
4	6	4
5	8	6
6	10	?
7	11	?

The reason behind this differences is, while we can introduce the edges between the

vertices of two different labels but we can not introduce the opposite oriented arcs between the vertices of two different labels (see Figure 6.6).

(a) Undirected graph G

(b) Digraphs G'

Figure 6.6: Digraph G' is an orientation of graph G and the vertices in the dashed area have the same label. We can introduce the edges of graph G with two labels but we can not introduce the arcs of the digraph G' with two labels.

Table 6.4: Number of connected, prime and directed graphs grouped by clique-width, modulo isomorphism. Space out (SO) is 16 GB.

				clique-width (prime graphs)						clique-width (directed graphs)			
$\|V\|$	connected graphs	connected prime graphs	weakly connected digraphs	2	3	4	5	6	2	3	4	5
3	2	0	5	0	0	0	0	0	3	2	0	0
4	6	1	34	0	1	0	0	0	9	24	1	0
5	21	4	535	0	4	0	0	0	30	461	44	0
6	112	26	20,848	0	25	1	0	0	103	11,457	9,287	1
7	853	260	2,120,098	0	210	50	0	0	333	318,441	1,712,294	89,030
8	11,117	4,670	SO	0	1,873	2,790	7	0	SO	SO	SO	SO
9	261,080	145,870	SO	0	16,348	125,364	4,4158	0	SO	SO	SO	SO
10	11,716,571	8,110,354	SO	0	142,745	5,520,350	2,447,190	68	SO	SO	SO	SO

CHAPTER 7

Conclusion

We revisited and extended the SAT approach to clique-width by Heule and Szeider from graphs to digraphs.

From the theoretical side, we generalized the key idea by Heule and Szeider, of certifying clique-width k by k-derivations instead of by k-expressions, from the graph to the digraph setting. We also presented a SAT encoding of the directed version of the notion of k-derivation.

We then exploited this theoretical work to implement an algorithm that is capable of finding the clique-width of small directed graphs (around 20 vertices), previously unknown; see Table 6.3. We also implemented a model counting algorithm by Fischer, Makowsky, and Ravve, and tested it on instances of small clique-width; to this aim we exploited the above implementation together with a program that computes a k-expression from a k-derivation of a digraph of clique-width k.

List of Figures

List of Tables

Bibliography

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[BJG97] Jørgen Bang-Jensen and Gregory Gutin. Alternating cycles and paths in edge-coloured multigraphs: a survey. *Discrete Math.*, 165/166:39–60, 1997.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[CMR01] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.*, 108(1-2):23–52, 2001.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

[FMR08] E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.

[FRRS09] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[GSS09] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, pages 633–654. IOS Press, 2009.

[HP10] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[HS15] Marijn J. H. Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Transactions on Computational Logic*, 16(24), 2015.

[McK81] Brendan D. McKay. Practical graph isomorphism. In *Proceedings of the Tenth Manitoba Conference on Numerical Mathematics and Computing, Vol. I (Winnipeg, Man., 1980)*, volume 30, pages 45–87, 1981.

[Oum08] Sang-Il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, December 2008.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Wah11] Magnus Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2), 2011.