Cliquewidth and Knowledge Compilation

Igor Razgon [⋆1] and Justyna Petke[2]

[1] Department of Computer Science and Information Systems,
Birkbeck, University of London igor@dcs.bbk.ac.uk
[2] Department of Computer Science,
University College London J.Petke@cs.ucl.ac.uk

Abstract. In this paper we study the role of cliquewidth in succinct representation of Boolean functions. Our main statement is the following: Let Z be a Boolean circuit having cliquewidth k. Then there is another circuit Z^* computing the same function as Z having treewidth at most $18k+2$ and which has at most $4|Z|$ gates where $|Z|$ is the number of gates of Z. In this sense, cliquewidth is not more 'powerful' than treewidth for the purpose of representation of Boolean functions. We believe this is quite a surprising fact because it contrasts the situation with graphs where an upper bound on the treewidth implies an upper bound on the cliquewidth but not vice versa.
We demonstrate the usefulness of the new theorem for knowledge compilation. In particular, we show that a circuit Z of cliquewidth k can be compiled into a Decomposable Negation Normal Form (DNNF) of size $O(9^{18k}k^2|Z|)$ and the same runtime. To the best of our knowledge, this is the first result on efficient knowledge compilation parameterized by cliquewidth of a Boolean circuit.

1 Introduction

Statement of the results. Cliquewidth is a graph parameter, probably best known for its role in the design of fixed-parameter algorithms for graph-theoretic problems [2]. In this context the most interesting property of cliquewidth is that it is 'stronger' than treewidth in the following sense: if all graphs in some (infinite) class have treewidth bounded by some constant c, then the cliquewidth of the graphs of this class is also bounded by a constant $O(2^c)$. However, the opposite is not true. Consider, for example, the class of all complete graphs. The treewidth of this class is unbounded while the cliquewidth of any complete graph is 2. Thus, classes of bounded cliquewidth contain dense graphs, unlike the case of bounded treewidth.

In this paper we essentially show that, roughly speaking, cliquewidth of a Boolean function is not a stronger parameter than its treewidth. In particular, given a Boolean circuit Z, we define its cliquewidth as the cliquewidth of the DAG of this circuit and the treewidth as the treewidth of the undirected graph

⋆ I would like to thank Fedor Fomin for his help in shaping of my understanding of the structural graph parameters.

underlying this DAG. The main theorem of this paper states that for any circuit Z of cliquewidth k there is another circuit Z^* computing the same function whose treewidth is at most $18k + 2$ and the number of gates is at most 4 times the number of gates of Z. Moreover, if Z is accompanied with the respective clique decomposition then such a circuit Z^* (and the tree decomposition of width $18k + 2$) can be obtained in time $O(k^2 n)$. The definition of circuit treewidth is taken from [14] and the definition of circuit cliquewidth naturally follows from the treewidth definition. In fact, the relationship between circuit treewidth and cliquewidth is put in [14] as an open question.

We demonstrate that the main theorem is useful for knowledge compilation. In particular, we show that any circuit Z of cliquewidth k can be compiled into decomposable negation normal form (DNNF) [3] of size $O(9^{18k} k^2 |Z|)$ (where $|Z|$ is the number of gates) by an algorithm taking the same runtime. To the best of our knowledge, this is the first result on space-efficient knowledge compilation parameterized by cliquewidth. We believe this result is interesting because the parameterization by cliquewidth, compared to treewidth, allows to capture a wider class of inputs including those circuits whose underlying graphs are dense.

This bound is obtained as an immediate corollary of the main theorem and the $O(9^t t^2 |Z|)$ bound on the DNNF size for the given circuit Z, where t is the treewidth of Z. The intermediate step for the latter result is an $O(3^p(|C| + n))$ bound of the DNNF size of the given CNF where C and n are, respectively the number of clauses and variables of this CNF and p is the treewidth of its *incidence* graph. All these 3 bounds significantly extend the currently existing bound $O(2^r n)$ of [3] where r is the treewidth of the *primal* graph of the given CNF. For example, if the given CNF has large clauses (and hence a large treewidth of the primal graph) then the $O(2^r n)$ bound becomes practically infeasible while the $O(3^p(C + n))$ bound may be still feasible provided a small treewidth of the incidence graph and a number of clauses polynomially dependent on n.

Related Work The algorithmic power of cliquewidth stems from the meta-theorem of [2] stating that any problem definable in Monadic Second Order Logic (MSO_1) can be solved in linear time for a class of graphs of fixed cliquewidth k. The cliquewidth of the given graph is NP-hard to compute [8] and it is not known to be FPT. On the other hand, cliquewidth is FPT approximable by an FPT computable parameter called *rankwidth* [13, 11]. As said above, there are classes of graphs with unrestricted treewidth and bounded cliquewidth. However, it has been shown in [10] that the only reason for treewidth to be much larger than cliquewidth is the presence of a large complete bipartite graph (biclique) in the considered graph. In fact, we prove the main theorem of this paper by applying a transformation that eliminates all bicliques from the DAG of the given circuit.

DNNFs have been introduced as a knowledge compilation formalism in [3], where it has been shown that any CNF on n variables of treewidth t of the primal graph can be compiled into a DNNF of size $O(2^t tn)$ with the same runtime. A detailed analysis of special cases of DNNF has been provided in [6]. In particular, it has been shown that Free Binary Decision Diagrams (FBDD) and hence Ordered Binary Decision Diagrams (OBDD) can be seen as special cases of DNNF.

In fact, there is a separation between DNNF and FBDD [4]. This additional expression power of DNNF has its disadvantages: a number of queries that can be answered in polynomial time (polytime) for FBDD and OBDD are NP-complete for DNNF [6]. This trade-off led to investigation of subclasses of DNNF that, on the one hand, retain the succinctness of DNNF for CNFs of small treewidth and, on the other hand, have an increased set of queries that can be answered in polytime. Probably the most notable result obtained in this direction are Sentential Decision Diagrams (SDD) [5] that, on one hand, can answer in polytime the equivalence query (possibility to answer this query in polytime for OBDDs is probably the main reason why this formalism is very popular in the area of verification) and, on the other hand, retain the same upper bound dependence on treewidth as DNNF.

In fact the size of OBDD can also be efficiently parameterized by the treewidth of the initial representation of the considered function. Indeed, there is an OBDD of size $O(n2^p)$ where p is the pathwidth of the primal graph of the given CNF and of size $(n^{O(t)})$ where t is the treewidth of the graph, see e.g. [9]. It is shown in [14] that similar pattern retains if we consider the pathwidth and treewidth of a circuit but in the former case p is replaced by an exponential function of p and in the latter case, t is replaced by a double exponential function of t.

Important remark. Due to space constraints, proofs of some papers are either omitted or replaced by sketches. A complete version of this work is available at `http://arxiv.org/abs/1303.4081`

2 Preliminaries

A *labeled* graph $G = (V, E, \mathbf{S})$ is defined by the usual set $V(G)$ of vertices and a set $E(G)$ of edges and also by $\mathbf{S}(G)$, a partition of $V(G)$. Each element of $\mathbf{S}(G)$ is called a *label*. A *simplified clique decomposition* (SCD) is a pair $(T, \mathbf{G})$ where T is a rooted tree and $\mathbf{G}$ is a family of labeled graphs. Each node t of T is associated with a graph $G(t)$, which is defined as follows. If t is a leaf node, then $G(t) = (\{v\}, \emptyset, \{\{v\}\})$. Assume that t has two children t_1 and t_2 and let $G_1 = G(t_1)$ and $G_2 = G(t_2)$. Then $V(G_1) \cap V(G_2) = \emptyset$ and $G(t) = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2), \mathbf{S}(G_1) \cup \mathbf{S}(G_2))$. Finally, assume that t has only one child t_1 and let $G_1 = G(t_1)$. Graph $G(t)$ can be obtained from G_1 by one of the following three operations:

- **Adding a new vertex**. There is $v \notin V(G_1)$ such that $G(t) = (V(G_1) \cup \{v\}, E(G_1), \mathbf{S}(G_1) \cup \{\{v\}\})$.
- **Union of labels**. There are $S_1, S_2 \in \mathbf{S}(G_1)$ such that $G(t) = (V(G_1), E(G_1), (\mathbf{S}(G_1) \setminus \{S_1, S_2\}) \cup \{S\})$. We say that S_1 and S_2 are the *children* of S.
- **New adjacency**. There are $S_1, S_2 \in \mathbf{S}(G_1)$ such that $G(t) = (V(G_1), E(G_1) \cup \{\{u, v\} | u \in S_1, v \in S_2\}, \mathbf{S}(G_1))$. We say that S_1 and S_2 are *adjacent*.

The width of a node t of T is $|\mathbf{S}(G(t))|$. The width of $(T, \mathbf{G})$ is the largest width of a node t of T. Let r be the root of T. Then we say that $(T, \mathbf{G})$ is an SCD of $G(r)$ and of $(V(G(r)), E(G(r)))$ (the unlabeled version of $G(r)$). The *simplified*

cliquewidth (SCW) of a graph G is the smallest width among all SCDs of G. The definition of SCD is closely related to the standard notion of clique decomposition. In fact SCW of a graph G is at most twice larger than the cliquewidth of G. The details are provided in the complete version.

Clique decomposition and SCD are easily extended to the directed case. In fact the notion of cliquewidth has been initially proposed for the directed case, as noted in [7]. The only change is that the new adjacency operation adds to $G(t)$ all possible directed arcs from label S_1 to label S_2 instead of undirected edges. In this case we say that there is an arc from S_1 to S_2.

We denote $\bigcup_{t \in V(T)} \mathbf{S}(G(t))$ by $\mathbf{S} = \mathbf{S}(T, \mathbf{G})$ and call it *the set of labels* of $(T, \mathbf{G})$. The set $\mathbf{S}$ is very important for our reasoning. In fact, we use SCD because we believe it allows a more intuitive definition of set $\mathbf{S}$ than the stadard clique deocmposition.

A tree decomposition of a graph G is a pair $(T, \mathbf{B})$ where T is a tree and the elements of $\mathbf{B}$ are subsets of vertices called *bags*. There is a mapping between the nodes of T and elements of $\mathbf{B}$. Let us say a vertex v of G is *contained* in a node t of T if v belongs to the bag $B(t)$ of t. Two properties of a tree decomposition are *connectedness* (all the nodes containing the given vertex v form a subtree of T), *adjacency* (each edge $\{u, v\}$ is a subset of some bag), and *union* (the union of all bags is $V(G)$). In this paper we consider the treewidth of a directed graph as the treewidth of the underlying undirected graph.

Boolean circuits considered in this paper are over the basis $\{\vee, \wedge, \neg\}$ with the unbounded fan-in. In such a circuit there are input gates (having only output wires) corresponding to variables and constants $true$ and $false$. The output of each gate of a circuit Z computes a function on the set of input variables. We denote by $functions(Z)$ the set of all functions computed by the gates of Z. The number of gates of Z is denoted by $|Z|$.

A clique or tree decomposition of a circuit Z is the respective decomposition of the DAG of Z. In our discussion, we often associate the vertices of the DAG with the respective gates. *De Morgan circuits* are a subclass of circuits where the inputs of all the NOT gates are variables (i.e. the outputs of NOT gates serve as negative literals). For a gate g of Z, denote by $Var(g)$ the set of variables having a path to g in the DAG of Z. A circuit Z has the *decomposability* property if for any two in-neighbours g_1 and g_2 of an AND gate g, $Var(g_1) \cap Var(g_2) = \emptyset$. DNNF is a decomposable De Morgan circuit. When we consider a general circuit Z, we assume that it does not have constant input gates, since these gates can be propagated by removal of some gates of Z, which in turn does not increase the cliquewidth nor the treewidth of the circuit. However, for convenience of reasoning, we may use constant input gates when we describe construction of a DNNF. If the given circuit Z is a CNF then its variables-clauses relation can be represented by the *incidence graph*, a bipartite graph with parts corresponding to variables and clauses and a variable-clause edge representing occurrence of a variable in a clause.

3 From small cliquewidth to small treewidth

The central result of this section is the following theorem:

Theorem 1. *Let F be a circuit of cliquewidth k over n variables Then there is a circuit F^* of treewidth at most $18k+2$ and $|F^*| \leq 4|F|$ such that $functions(F) \subseteq functions(F^*)$. Moreover, given F and a clique decomposition of F of width k there is an $O(k^2 n)$ algorithm constructing F^* and a tree decomposition of F^* of width at most $18k + 2$ having at most $2|F|$ bags.*

The rest of this section is the proof of Theorem 1. The main idea of the proof is to replace 'parts' of the given circuit forming large bicliques by circuits computing equivalent functions where such bicliques do not occur. As an example consider a CNF of 3 clauses $C_1 = (a_1 \vee a_2 \vee a_3 \vee b_1)$, $C_2 = (a_1 \vee a_2 \vee a_3 \vee b_2)$ and $C_3 = (a_1 \vee a_2 \vee a_3 \vee b_3)$. The circuit of this graph contains a biclique of order 3 created by C_1, C_2, C_3 on one side and a_1, a_2, a_3 on the other one. This biclique can be eliminated by the introduction of an additional OR gate C_4 having input a_1, a_2, a_3 and output c_4 so that the clauses C_1, C_2, C_3 are transformed into $(b_1 \vee c_4), (b_2 \vee c_4), (b_3 \vee c_4)$, respectively. It is not hard to see that the new circuit computes the same function as the original one. This is the main idea behind the construction of circuit F^*. The formal description of the construction is given below.

For the purpose of construction of F^* we consider a type respecting SCD $(T, \mathbf{G})$ of F where each non-singleton label is one of the following:

- A *unary* label containing input gates and negation gates.
- An AND label containing AND gates.
- An OR label containing OR gates.

The following lemma essentially follows from splitting each label of the given clique decomposition into three type respecting labels.

Lemma 1. *Let k be the cliquewidth of F and let k^* be the smallest width of an* SCD *of F that respects types. Then $k^* \leq 6k$.*

Given a type respecting SCD $(T, \mathbf{G})$, let us construct the circuit F^*. In the first stage, we associate each label $S \in \mathbf{S}$ with a set of gates as follows:

- If S is non-singleton then it is associated with an AND gate denoted by $oand(S)$ and an OR gate denoted by $oor(S)$.
- If S is non-singleton and does not contain input gates then it is associated with an additional gate called $in(S)$ whose type is determined as follows: If S is an AND or OR label then $in(S)$ is an AND or OR gate, respectively. If S is a unary label then $in(S)$ is a circuit (perceived as a single atomic gate) consisting of two NOT gates, the output of one of them is the input of the other. So, the input of the former and the output of the latter are, respectively, the input and output of $in(S)$.

– Each singleton label $\{g\}$ is associated with the gate g of F. We call the gates associated with singleton labels *original gates* because they are the gates of F^* appearing in F. For the sake of uniformity, for each original gate g associated with a singleton label S, we put $g = oand(S) = oor(S) = in(S)$.

The wires of F^* are described below. When we say that there is a wire from gate g_1 to gate g_2, we mean that the wire is *from the output* of g_1 *to the input* of g_2.

– **Child-parent wires.** Let S_1 and S_2 be labels of $(T, \mathbf{G})$ such that S_1 is a child of S_2. Then there is a wire from $oand(S_1)$ to $oand(S_2)$ and a wire from $oor(S_1)$ to $oor(S_2)$.
– **Parent-child wires.** Let S_1 and S_2 be as above and assume that S_2 does not contain input gates. Then there is a wire from $in(S_2)$ to $in(S_1)$. That is, the direction of child-parent wires is opposite to the direction of parent-child wires. [3]
– **Adjacency wires.** Assume that in $(T, \mathbf{G})$ there is an arc from S_1 to S_2 (established by the new adjacency node). Then the following cases apply:
  • If S_2 is an AND label then put a wire from $oand(S_1)$ to $in(S_2)$.
  • If S_2 is an OR label then put a wire from $oor(S_1)$ to $in(S_2)$.
  • If S_2 is a unary label consisting of negation gates only then put a wire from an arbitrary one of $oand(S_1)$ or $oor(S_1)$ to $in(S_2)$.

Finally, we remove $in(S)$ gates that have no inputs. This removal may be iterative as removal of one gate may leave without input another one.

It is not hard to see by construction that F and F^* have the same input gates. This gives us possibility to state the following theorem with proof in Section 3.1.

Theorem 2. *F^* is a well formed circuit (that is, F^* satisfies the definition of a Boolean circuit). The output of each original gate g of F^* computes exactly the same function (in terms of input gates) as in F.*

In Section 3.2, we prove that the treewidth of F^* is not much larger than the width of $(T, \mathbf{G})$.

Theorem 3. *There is a tree decomposition of F^* with at most $2|F|$ bags having width at most $3k + 2$, where k is the width of $(T, \mathbf{G})$.*

Proof of Theorem 1. Due to Theorem 2, $functions(F) \subseteq functions(F^*)$. If we take $(T, \mathbf{G})$ to be of the smallest possible type respecting width then the treewidth of F^* is at most $18k + 2$ by combination of Theorem 3 and Lemma 1.

To compute the number of gates of F^*, let n be the number of gates of F, which is also the number of singleton labels of $(T, \mathbf{G})$. Since each non-singleton label has two children (i.e. in the respective tree of labels each non-leaf node is binary), the number of non-singleton labels is at most $n - 1$. By construction, F^* has one gate per singleton label plus at most 3 gates per non-singleton label, which adds up to at most $4n$. The technical details of the runtime derivation are omitted due to space constraints.

[3] We would like to thank the anonymous referee, for helping us to identify a typo in this definition that occurred in the first version of the manuscript.

3.1 Proof of Theorem 2

We start with establishing simple combinatorial properties of F^* (Lemmas 2,3, 4,5). A *path* in a circuit is a sequence of gates so that the output of every gate (except the last one) is connected by a wire to the input of its successor. Let us call a path a *connecting* path if it contains exactly one adjacency wire.

Lemma 2. – *Any path P of F^* starting at an original gate and not containing adjacency wires contains child-parent wires only.*
– *Any path P of F^* ending at an original gate and not containing adjacency wires contains parent-child wires only.*

Proof. The only possible wire to leave the original gate is a child-parent wire. Any path starting from an original gate and containing child-parent wires only ends up in an *oand* or *oor* gate. This means that the next wire (if not an adjacency one) can be only another child-parent wire. Thus the correctness of the lemma for all the paths of length i implies its correctness for all such paths of length $i + 1$, confirming the first statement.

For the second statement, we start from an original gate and go back *against* the direction of wires. The reasoning similar to the previous paragraph applies with the *in* gates of non-singleton labels replacing the *oor* and *oand* ones. ■

Lemma 3. *Let g_1 and g_2 be gates of F such that g_2 is an* AND *or an* OR *gate. Then there is a wire from g_1 to g_2 in F if and only if F^* has a connecting path from g_1 to g_2 such that all the gates of this path except possibly g_1 are of the same type as g_2.*

Proof. We prove only the case where g_2 is an AND gate, the other case is symmetric. Let P be a connecting path of F^* from g_1 to g_2 of the specified kind. Let g'_1 and g'_2 be, respectively, the tail and the head gates of the adjacency wire. Then either $g_1 = g'_1$ or the suffix of P ending at g'_1 consists of child-parent wires only according to Lemma 2. It follows that g'_1 corresponds to a label containing g_1. Analogously, we conclude that either $g_2 = g'_2$ or the suffix of P starting at g'_2 contains only parent-child labels and hence the label corresponding to g'_2 contains g_2. Existence of the adjacency wire from the label of g'_1 to the label of g'_2 means that the SCD introduces all wires from the gates in the label of g'_1 to the gates in the label of g'_2. In particular, there is a wire from g_1 to g_2 in F.

Conversely, assume that there is a wire from g_1 to g_2 in F. Then there are labels S_1 and S_2 containing g_1 and g_2, respectively, such that $(T, \mathbf{G})$ introduces an adjacency arc from S_1 to S_2. By construction of F^* there is a gate g'_1 corresponding to S_1 and a gate g'_2 corresponding to S_2 such that F^* has an adjacency wire from g'_1 to g'_2. Moreover, by the definition of a type respecting SCD, S_2 is an AND label, hence $g'_2 = in(S_2)$ is an AND gate. Furthermore, by construction of F^* either $g_2 = g'_2$ or there is a path from g'_2 to g_2 consisting of parent-child arcs only and AND gates only. Indeed, if S_2 is not a singleton then there is a wire from $in(S_2)$ to $in(S_3)$ containing g_2 since S_2 is partitioned by its children. Iterative application of this argument produces a path from g'_2 to g_2. Since g_2

is an AND gate, all gates in this path are AND gates by construction. Thus the suffix exists. What about the prefix? By construction, $g'_1 = oand(S_1)$. Since S_1 contains g_1, either $g'_1 = g_1$ or there is a path from g_1 to g'_1 involving child-parent wires and AND gates only: just start at g_1 and go every time to the *oand*-gate of the parent until S_1 has been reached. Thus we have established existence of the desired prefix.

It remains to be shown that the prefix and suffix do not intersect. However, this is impossible due to the disjointness of S_1 and S_2. ■

Lemma 4. *Let g_1 and g_2 be the gates of F such that g_2 is a* NOT *gate. Then F has a wire from g_1 to g_2 if and only if there is a connecting path P in F^* from g_1 to g_2 with the adjacency wire (g'_1, g'_2) such that $g_1 = g'_1$ and all the intermediate vertices in the suffix of P starting from g'_1 are in-gates of unary labels containing negation gates only.*

Proof. Let P be a connecting path of F^* of the specified form. Then either $g'_2 = g_2$ or g'_2 corresponds to a label containing g_2. In both cases this means that F has a wire from g_1 to g_2.

Conversely, assume that F has a wire from g_1 to g_2. Then there are labels S_1 and S_2 containing g_1 and g_2 such that $(T, \mathbf{G})$ sets an adjacency wire from S_1 to S_2. Observe that S_1 cannot contain more than one element because in this case g_2, a NOT gate, will have two inputs. Furthermore, either S_2 contains g_2 only or S_2 is a unary label containing negation gates only (because the input gates do not have input wires). In the latter case, the desired suffix from the head of the adjacency arc to g_2 follows by construction. ■

Lemma 5. *Any path of F^* between two original gates that does not involve other original gates is a connecting path.*

Proof. First of all, let us show that any path of F^* between original gates involves at least one adjacency wire. Indeed, by Lemma 2, any path leaving an original gate and not having adjacency wires has only child-parent wires. Such wires lead only to bigger and bigger labels and cannot end up at a singleton one. It follows that at least one adjacency wire is needed.

Let us show that additional adjacency wires cannot occur without original gates as intermediate vertices. Indeed, the head of the first adjacency wire is an *in* gate of some label S. Unless S is a singleton, the only wires leaving $in(S)$ are parent-child wires to the *in* gates of the children of S. Applying this argumentation iteratively, we observe that no other wires except parent-child wires are possible until the path meets the *in* gate of a singleton label. However, this is an original gate that cannot be an intermediate node in our path. It follows that any path between two original gates without other original gates cannot involve 2 adjacency wires. Combining with the previous paragraph, it follows that any such path involves exactly one adjacency wire, i.e. it is a connecting path. ■

Using the lemmas above, it can be shown that any cycle in F^* involves at least one original gate and that this implies that F contains a cycle as well, a

contradiction showing that F^* is acyclic. The technical details of this derivation are omitted due to space constraints. By construction, each wire connects output to input and there are no gates (except the input gates of course) having no input. It follows that F^* is a well formed circuit.

For each gate g of F^* denote by $f(g, F^*)$ the function computed by a subcircuit of F^* rooted by g. We establish properties of these functions from which Theorem 2 will follow by induction. In the following we sometimes refer to $f(g, F^*)$ as the function of g.

Lemma 6. *For each* NOT *gate g of F^*, $f(g, F^*)$ is the negation of $f(g', F^*)$, where g' is the input of g in F.*

Proof. According to Lemma 4, F^* has a path from g' to g where all vertices except the first one are NOT gates. Since all of them but the last one are doubled, there is an odd number of such NOT gates. Each NOT gate has a single input, hence the function of each gate of the path (except the first one) is the negation of the function of its predecessor. Hence these functions are, alternatively, the negation of the function of g' and the function of g'. Since the number of NOT gates in the path is odd, the function of g is the negation of the function of g', as required. ■

In order to establish a similar statement regarding AND and OR gates we need two auxiliary lemmas.

Lemma 7. *For each label S, $f(oand(S), F^*)$ is the conjunction of $f(g, F^*)$ of all original gates g contained in S. Similarly, $f(oor(S), F^*)$ is the disjunction of the functions of such gates.*

Proof. We prove the lemma only for the *oand* gates as for the *oor* gates the proof is symmetric. The proof easily goes by induction. For an original gate this is just a conjunction of a single element, namely itself, and this is clear by construction. For a larger label S, it follows by construction that $f(oand(S), F^*) = f(oand(S_1), F^*) \wedge f(oand(S_2, F^*))$, where S_1 and S_2 are the children of S. For S_1 and S_2 the rule holds by the induction assumption. Hence, $f(oand(S), F^*)$ is the conjunction of all the functions of all the original gates in the union of S_1 and S_2, that is, the conjunction of the functions of all the original gates contained in S, as required. ■

Let us call a path of F^* *semi-connecting* if it starts with an adjacency wire and the rest of the wires are parent-child ones.

Lemma 8. *Let S be an* AND *label. Then $f(in(S), F^*)$ is the conjunction of the functions of all gates from which there is a semi-connecting path to $in(S)$. For the* OR *label the statement is analogous with the conjunction replaced by disjunction.*

Proof. We provide the proof only for the AND label, for the OR label the proof is analogous with the corresponding replacements of AND by OR and conjunctions by disjunctions.

The proof is by induction on the decreasing size of labels. For the largest AND label S, all the input wires are the adjacency wires. Clearly the considered

function is the conjunction of the functions of the gates at the tails of these adjacency wires. It remains to see if there are no more gates to arrive at $in(S)$ by semi-connected paths. But any such gate, after passing through the adjacency wire must meet an ancestor of S and, by the maximality assumption, S has no ancestors.

The same reasoning as above is valid for any label S without ancestors. If S has ancestors, then $f(in(S), F^*)$ is the conjunction of the functions of the gates at the tails of the adjacency wires incident to $in(S)$ and the function of the in gate of the parent of S . By the induction assumption, this function is in fact a conjunction of the gates at the tails of the adjacency wires incident to $in(S)$ plus those connected to $in(S)$ by semi-connected paths through the parent. Since any semi-connected path either directly hits $in(S)$ at the head of an adjacency wire or approaches it through the parent, the statement is proven. ■

Lemma 9. *The function of any original* AND *gate g of F^* is the conjunction of the functions of the singleton gates whose outputs are the inputs of g in F. The same happens for the* OR *gate and the disjunction.*

Proof. As before, we prove the statement for the AND gate, for the OR gate it is analogous with the respective substitutions. By construction and Lemma 8, $f(g, F^*)$ is the conjunction of functions of all *oand* gates (since there are no other ones) connected to g by semi-connected paths. Let us call the labels of these *oand* gates the *critical labels*. Combining this with Lemma 7, we see that $f(g, F^*)$ is in fact a conjunction of the functions of all original gates contained in the critical labels. It remains to show that these gates are exactly the in-neighbours of g in F. Let us take a particular in-neighbour g'. By Lemma 3, there is a connecting path from g' to g and by Lemma 7, the tail of the adjacency wire of this path is the *oand* gate of a critical label, so g' is in the required set. Conversely, assume that g' is a gate in the required set. Specify a critical label S g' belongs to. Clearly, there is a child-parent path from g' to $oand(S)$ which, together with a semi-connected path from $oand(S)$ to g, makes a connecting path. The latter means that in F there is a wire from g' to g according to Lemma 3, as required. ■

Proof of Theorem 2. Let us order the gates of F topologically and do induction on the topological order. The first gate is an input gate and the function of the input is just the corresponding variable both in F and in F^*. Otherwise, the gate is AND or OR or NOT gate. In the former two cases, according to Lemma 9 the function of g in F^* is the conjunction (or disjunction, in case of OR) of the functions of its inputs in F, the same relation as in F. The theorem holds regarding the inputs by the induction assumption, hence the function of g in F^* is the same as in F. Regarding the NOT gate, the argumentation is analogous, employing Lemma 6. ■

3.2 Proof of Theorem 3

Let us define the undirected graph $H = H(T, \mathbf{G})$ called the *representation graph* of $(T, \mathbf{G})$ as follows. The vertices of this graph are the labels of $(T, \mathbf{G})$ and two

vertices S_1 and S_2 are adjacent if and only if either S_1 is a child of S_2 (or vice versa of course) or S_1 and S_2 are adjacent in $(T, \mathbf{G})$ (meaning that the new adjacency operation is applied on S_1 and S_2). We call the first type of edges *child-parent* edges and the second type *adjacency* edges.

Lemma 10. *Let t be the treewidth of H. Then the treewidth of F^* is at most $3t + 2$.*

Proof (Sketch). Observe that if we contract the gates in F^* of each label into a single vertex, eliminate directions and remove multiple occurrences of edges, we obtain a graph isomorphic to H. The desired tree decompositom is obtained from the tree decomposition of H by replacing the occurrence of each vertex of H in a bag by the gates corresponding to this vertex. Thus, there is a tree decomposition of F^* with at most $3(t+1)$ elements in each bag, that is the treewidth of F^* is at most $3t + 2$. ■

Lemma 11. *The treewidth of H is at most k, where k is the width of $(T, \mathbf{G})$.*

Proof. For each node t of T, let $S(t)$ be the set of labels of $G(t)$ and let $B(t)$ be the set of vertices of H corresponding to $S(t)$. Denote the set of all $B(t)$ by $\mathbf{B}$. We are going to show that $(T, \mathbf{B})$ is a tree decomposition of graph H' obtained from H by removal of all child-parent edges.

First of all, observe that for each $v \in V(H)$, the subgraph T_v of T consisting of all nodes containing v is a subtree of T. Indeed, let us consider T as a rooted tree with the root t being the same as in $(T, \mathbf{G})$. Let t_1 and t_2 be two nodes containing v. Then one of them is an ancestor of the other. Indeed, otherwise t_1 and t_2 are nodes of two disjoint subtrees T_1 and T_2 whose roots t'_1 and t'_2 are children of some node t^*. By the definition of SCD, $G(t'_1)$ is disjoint with $G(t'_2)$ and it is not hard to conclude from the definition that $V(G(t_1)) \subseteq V(G(t'_1))$ and $V(G(t_2)) \subseteq V(G(t'_2))$ are disjoint. Since any label is a subset of the set of vertices of the graph it belongs to, $S(t_1)$ and $S(t_2)$ cannot have a common label and hence $B(t_1)$ and $B(t_2)$ cannot have a joint node. Furthermore, it is not hard to observe, if t_1 is ancestor of t_2 and $S \in S(t_1) \cap S(t_2)$ then S belongs to $S(t')$ of all nodes t' in the path between t_1 and t_2. It follows that the node of H corresponding to S belongs to $B(t')$ of all these nodes t'. Thus we have shown that if t_1 and t_2 contain v they cannot belong to different connected components of T_v, confirming the connectedness of T_v.

Next, we observe that if v_1 and v_2 are incident to an adjacency edge then there is a node t containing both v_1 and v_2. Indeed, let S_1 and S_2 be the labels corresponding to v_1 and v_2, respectively. Let t be the node where the adjacency operation regarding S_1 and S_2 is applied. Then both S_1 and S_2 belong to $S(t)$ and, consequently, t contains both v_1 and v_2. Finally, by construction, each vertex of H is contained in some node.

To obtain the desired tree decomposition of H, we are going to modify $(T, \mathbf{B})$ to acquire two properties: that the number of nodes of the resulting tree is at most $2|F|$ and that each parent-child pair u, v is contained in some node t. For the former just iteratively remove all nodes whose operations are new adjacency.

If the node t being removed is not the root then make the parent of t to be the parent of the only child of t (since t has only one child the tree remains binary). The latter property can be established by adding at most one vertex to each bag of the resulting structure $(T', \mathbf{B}')$. Indeed, for each non-singleton label S, let $t(S)$ be the node where this label is created by the union operation. Then both children of S belong to the only child of $t(S)$. Let $(T', \mathbf{B}^*)$ be obtained from $(T', \mathbf{B}')$ as follows. For each non-singleton label S, add the vertex corresponding to S to the bag of the child of $t(S)$. Since at most one new label is created per node of T', at most one vertex is added to each bag. It is not hard to see both of the modifications preserve properties stated in the previous paragraphs and achieve the desired properties regarding the child-parent edges. Since each bag of $(T, \mathbf{B}')$ contains at most $k+1$ elements, we conclude that the treewidth of H is at most k. Since the number of bags is at most as the number of labels, we conclude that the number of bags is at most $2|F|$ ■

Proof of Theorem 3. Immediately follows from the combination of Lemma 10 and Lemma 11. ■

4 Application to knowledge compilation

In this section we demonstrate an application of Theorem 1 to knowledge compilation by showing the existence of an algorithm compiling the given circuit Z into DNNF. Both the time complexity of the algorithm and the space complexity of the resulting DNNF are fixed-parameter linear parameterized by the cliquewidth of Z. More precisely, the statement is the following:

Theorem 4. *Given a single-output circuit Z of cliquewidth k, there is a* DNNF *of Z having size $O(9^{18k}k^2|Z|)$. Moreover, given a clique decomposition of Z of width k, there is a $O(9^{18k}k^2|Z|)$ algorithm constructing such a* DNNF.

Theorem 4 is an immediate corollary of Theorem 1 and the following one:

Theorem 5. *Given a circuit Z of treewidth p, there is a* DNNF *of Z having size $O(9^p p^2|Z|)$. Moreover, such a* DNNF *can be constructed by an algorithm of the same runtime that gets as input the circuit Z and a tree decomposition of Z of width p having $O(Z)$ bags.*

The rest of this section is a proof of Theorem 5. Our first step is Tseitin transformation from circuit Z into a CNF F'. For this purpose we assume that Z does not have paths of 2 or more NOT gates. Depending on whether this path is of odd or even length, it can be replaced by a single NOT gate or by a wire, without treewidth increase. In this case the variables $y_1, \ldots, y_m$ of F' are the variables of Z and the outputs of AND and OR gates of Z. Under this assumption, it is not hard to see that the inputs of each gate are literals of $y_1, \ldots, y_m$. Then the output x of Z is either y_i or $\neg y_i$ for some i. Let us call x the output literal.

The CNF F' is a conjunction of the singleton clause containing the output literal and the CNFs associated with each AND and OR gate. Let C be an AND gate with inputs $t_1, \ldots, t_r$ and output z. Then the resulting CNF is $(t_1 \vee \neg z) \wedge$

$\ldots \wedge (t_r \vee \neg z) \wedge (\neg t_1 \vee \ldots \vee \neg t_r \vee z)$. If C is an OR gate then the resulting CNF is $(\neg t_1 \vee z) \wedge \ldots \wedge (\neg t_r \vee z) \wedge (t_1 \vee \ldots \vee t_r \vee \neg z)$. We call the last clause of the CNF of C the *carrying* clause w.r.t. C and the rest are *auxiliary* ones w.r.t. C and the corresponding input.

To formulate the property of Tseitin transformation that we need for our transformation, let us extend the notation. We consider sets of literals that do not contain a variable and its negation. For a set S of literals, $Var(S)$ is the set of variables of S. Let $V' \subseteq Var(S)$. The *projection* $Pr(S, V')$ of S to V' is the subset S' of S such that $Var(S') = V'$. Let $\mathbf{S}$ be a family of sets of literals over a set V of variables. Then the projection $Proj(\mathbf{S}, V')$ of $\mathbf{S}$ to $V' \subseteq V$ is $\{Proj(S, V') | S \in \mathbf{S}\}$. Denote by $Var(Z)$ and $Var(F')$ the sets of variables of Z and F', respectively. Let us say that a set S of literals with $Var(S) = Var(Z)$ is a *satisfying assignment* of Z if Z is true on the truth assignment on $Var(Z)$ that assigns all the literals of S to true. For a CNF, the definition is analogous. The well known property of Tseitin transformation is the following:

Lemma 12. *Let* $\mathbf{S_1}$ *and* $\mathbf{S_2}$ *be the sets of satisfying assignments of* F' *and* Z*, respectively. Then* $Proj(\mathbf{S_1}, Var(Z)) = \mathbf{S_2}$.

Lemma 12 is useful because of the following nice property of DNNF.

Lemma 13. *(Theorem 9 of [3]). Let* Z *be a DNNF let* $V' \subseteq Var(Z)$ *and let* Z' *be the* DNNF *obtained from* Z *by replacing the variables of* $Var(Z) \setminus V'$ *with the true constant. Let* $\mathbf{S}$ *and* $\mathbf{S}'$ *be sets of satisfying assignments of* Z *and* Z'*, respectively. Then* $\mathbf{S}' = Proj(\mathbf{S}, V')$.

Thus it follows from Lemmas 12 and 13 that having compiled F' into a DNNF D', a DNNF D of Z can be obtained by replacing the variables of $Var(F') \setminus Var(Z)$ with the *true* constant. Clearly, this does not incur any additional gates. In order to obtain a DNNF of F', we observe that the treewidth of the incidence graph of F' is not much larger than the treewidth of Z.

Lemma 14. *Let* $(T, \mathbf{B})$ *be a tree decompositoion of* Z *of width* p*. There is a* $O(p^2|T|)$ *time algorithm (*$|T|$ *is the number of nodes of* T*) transforming* $(T, \mathbf{B})$ *into a tree decomposition* $(T^*, \mathbf{B}^*)$ *of the incidence graph* G' *of* F' *having width at most* $2p + 1$ *and with* $|T^*| = O(p^2|T|)$.

Proof (Sketch). Let F'' be the CNF obtained from F' by removal of all the clauses but the carrying ones and let G'' be the respective incidence graph. Transform $(T, \mathbf{B})$ into $(T, \mathbf{B}'')$ as follows:

- Replace each occurrence of an AND or OR gate X with the respective carrying clause and the variable corresponding to the output of X.
- Replace each occurrence of a NOT gate with the variable corresponding to the input of the gate (it may either be an input variable of Z or the output variable of some AND or OR gate).

It can be observed by a straightforward inspection that $(T, \mathbf{B}'')$ is indeed a tree decomposition of G'' of width $2p + 1$.

Next, we observe that for each AND or OR gate X of Z and for each variable u of F' corresponding to an input of X and for variable y of F' corresponding to the output of X, there is a node t of $(T, \mathbf{B}'')$ containing both y and u. Indeed, let C be the carrying clause corresponding to X. By construction, whenever t contains C, t also contains y. By the adjacency property, there is at least one t containing C and u. Since this last t contains also y, this is a desired clause. Pick one node with the specified property and denote it by $t(y, u)$. Add to T a new node t' with $t(y, u)$ being its only neighbour. The bag of t' will contain y, u, and $C(y, u)$ the auxiliary clause of X corresponding to the input u. Do so for all the auxiliary clauses. Finally, properly add a node whose bag contains the variable y of the output literal and the singleton clause containing this literal (the neighbour of this new node should be an existing node containing y). Let $(T^*, \mathbf{B}^*)$ be the resulting structure. It is not hard to observe by construction that $(T^*, \mathbf{B}^*)$ satisfies the statement of the lemma. ■

It remains to show that a space-efficient DNNF can be created parameterized by the treewidth of the incidence graph.

Theorem 6. *Let F be a* CNF *and let $(T', \mathbf{B}')$ be a tree decomposition of the incidence graph of F. Then F has a* DNNF *of size $O(3^t|T'|)$ where t is the width of $(T', \mathbf{B}')$. Moreover, given F and $(T', \mathbf{B}')$ such a* DNNF *can be constructed by an algorithm having the same runtime.*

We omit the proof of Theorem 6 due to space constraints. It is similar to the proof of Theorem 16 of [3], essentially based on dynamic programming. The difference is that in addition to branching on assignments of variables of the given bag, the algorithm also needs to branch on the clauses of that bag that are not satisfied by the currently considered assignment of variables. Three choices need to be considered for each clause: to not satisfy the clause at all (this choice is needed for 'coordination' with the 'parent bag'), to satisfy the clause by the variables of the left child and to satisfy the clause by the variables of the right child. These 3 choices increase the base of the exponent from 2 to 3.

Remark. It is not hard to see that any tree decomposition can be transformed (without the increase of width) into another one whose number of nodes is at most as big as the number of vertices. Having this in mind, Theorem 6 canbe reformulated with $O(3^t(CL + n))$ istead $O(3^t|T'|)$, where CL and n are, respectively the number of clauses and the number of variables of F. With this reformulation, Theorem 6 becomes of an independent interest because it extends the result of Darwiche [3] from the primal to the incidence graph of the given CNF without increasing much the base of the exponent.

Proof of Theorem 5. The construction of a DNNF for Z consists of 4 stages: transform Z into F' by the Tseitin transformation; transform the tree decomposition of Z into a tree decomposition of the incidence graph of F'; obtain a DNNF of F' as specified by Theorem 6 and obtain a DNNF of Z as specified in Lemma 13. The correctness of this procedure follows from the above discussion. The time and space complexities easily follow from the combination of the complexities of intermediate stages. ■

5 Discussion

In this paper we presented a theorem that shows that a circuit of cliquewidth k can be transformed into, roughly speaking, an equivalent circuit of treewidth $18k + 2$ with at most 4 times more gates. A consequence of this statement is that any space-efficient knowledge compilation parameterized by the *treewidth* of the input circuit can be transformed into a space efficient knowledge compilation parameterized by the *cliquewidth* of the input circuit. We elaborated this consequence on the example of DNNF. As a result we obtained a theoretically efficient but formidably looking space complexity of $(9^{18k}k^2n)$. Therefore, the first natural question is how to reduce the base of the exponent.

The next question for further investigation is to check if the proposed upper bound can be applied to SDD [5] which is more practical than DNNF in the sense that it allows a larger set of queries to be efficiently handled. To answer this question positively, it will be sufficient to extend Theorem 6 to the case of SDD, the 'upper' levels of the reasoning will be applied analogously to the case of DNNF.

It is important to note that rankwidth is a better parameter for capturing dense graphs than cliquewidth in the sense that rankwidth of a graph does not exceed its treewidth plus one [12] as well as cliquewidth [13], while cliquewidth can be exponentially larger than treewidth (and hence rankwidth) [1]. Also, computing of rankwidth, unlike cliquewidth, is known to be FPT [11]. Therefore, it is interesting to investigate the relationship between the rankwidth and the treewidth of a Boolean function. For this purpose rankwidth has to be extended to directed graphs [15]. It is worth saying that if the question is answered negatively, i.e. that the treewidth of a circuit can be exponentially larger than its rankwidth, it would be an interesting circuit complexity result.

Finally, recall that all the upper bounds on the DNNF size obtained in this paper are polynomial in the *size* of the circuit which can be much larger than the number of variables. On the other hand, the upper bound on the DNNF size parameterized by the treewidth of the primal graph of the given CNF is polynomial in the number of variables [3]. Can we do the same in the circuit case?

References

1. Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.
2. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
3. Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.
4. Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.

5. Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826, 2011.
6. Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.
7. Wolfgang Dvorák, Stefan Szeider, and Stefan Woltran. Reasoning in argumentation frameworks of bounded clique-width. In *COMMA*, pages 219–230, 2010.
8. 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.
9. Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in verification: Local vs. global. In *LPAR*, pages 489–503, 2005.
10. Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $k_{n,\ n}$. In *WG*, pages 196–205, 2000.
11. Petr Hlinený and Sang il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.
12. Sang il Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.
13. Sang il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.
14. Abhay Kumar Jha and Dan Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, pages 249–261, 2012.
15. Mamadou Moustapha Kanté and Michaël Rao. $\mathbb{F}$-rank-width of (edge-colored) graphs. In *CAI*, pages 158–173, 2011.