Cliquewidth and Knowledge Compilation

Igor Razgon[1] & Justyna Petke[2]

[1]Birkbeck, University of London, UK

[2]University College London, UK

Boolean functions

$$f(x) : B^n \to B$$
$$B : \{0, 1\}$$
n : a positive integer
$$x = (x_1, x_2, \cdots, x_n) : x_i \in B$$

Boolean functions

Clausal entailment query:

Given a partial truth assignment, can it be extended to a complete satisfying assignment?

Boolean functions

Clausal entailment query:

Given a partial truth assignment, can it be extended to a complete satisfying assignment?

Good representation of Boolean functions:

The clausal entailment query can be answered in poly-time.

Some applications require good representations of Boolean functions.

Boolean function representations - normal forms

- Conjunctive Normal Form (CNF)
- Disjunctive Normal Form (DNF)

DNF representation:

$$\bigvee_{Y \in T} \Big(\bigwedge_{i | y_i = 1} x_i \bigwedge_{j | y_j = 0} \neg x_j \Big)$$

where T is a set of solutions to a Boolean function f

DNF is a good representation while CNF is not.

Knowledge compilation

- Off-line phase:
  - propositional theory is compiled into some target language
  - the target language must be a good representation!
  - can be slow

Knowledge compilation

- On-line phase:
  - the compiled target is used to efficiently answer a number of queries
  - fast (partly due to being good)

Knowledge compilation representation

NNF : Negation Normal Form

- conjunctions and disjunctions are the only connectives used (e.g. CNF, DNF)

DNNF : Decomposable Negation Normal Form

- conjunctions and disjunctions are the only connectives used
- atoms are not shared across conjunctions

Knowledge compilation representation

Properties:

- DNNF is a highly tractable representation
- every DNF is also a DNNF
- $\exists$ exponential DNF & linear DNNF for the same Boolean function

Automated DNNF construction & graph parameters

- efficient DNNF compilation achieved when the input clausal form is parameterised by the *treewidth* of the primal graph of the input CNF

Automated DNNF construction & graph parameters

- efficient DNNF compilation achieved when the input clausal form is parameterised by the *treewidth* of the primal graph of the input CNF
- treewidth is always high for dense graphs

Automated DNNF construction & graph parameters

- efficient DNNF compilation achieved when the input clausal form is parameterised by the *treewidth* of the primal graph of the input CNF
- treewidth is always high for dense graphs
- better parameter: *cliquewidth*

Knowledge compilation result

Given a 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.

Main result

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.

Consequence: cliquewidth is not more ‘powerful’ than treewidth for Boolean function representation

Obtaining the Know. Comp. Res. from the Main Result

- upgrade from DNNF parameterized by treewidth of the primal graph of the input CNF to the treewidth of its incidence graph

Primal vs. incidence graph

C = a ∨ b ∨ c

Obtaining the Know. Comp. Res. from the Main Result

- upgrade from DNNF parameterized by treewidth of the primal graph of the input CNF to the treewidth of its incidence graph
- extension from input CNF to input circuits (by Tseitin transformation plus projection removing additional variables)
- replacing the treewidth of the input circuit by the cliquewidth of the input circuit using the main result

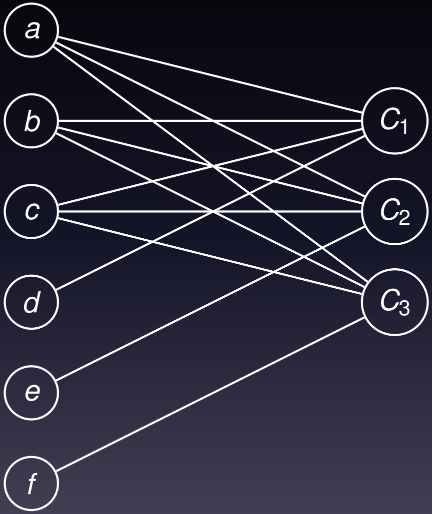
Small Cliquewidth and Large Treewidth

- a necessary condition: existence of large complete bipartite subgraphs
- examples: complete graph, complete bipartite graph

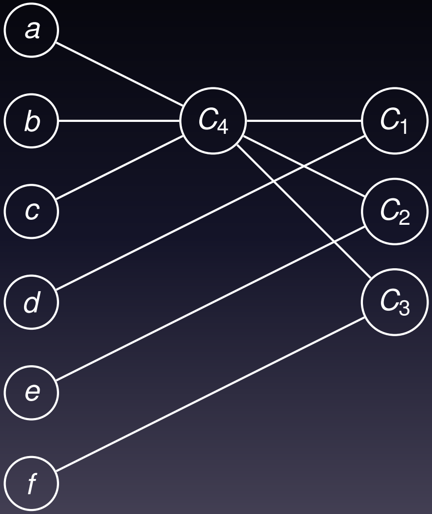
Elimination of large bicliques in Boolean circuits

- necessary and sufficient condition:
  a set X of many gates of the same type ($\vee$ or $\wedge$) share a large set of Y common inputs
- elimination: introduce a new gate g of the same type with inputs Y; connect the output of g to all of X instead Y
- example: $(a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee e) \wedge (a \vee b \vee c \vee f)$
- new gate: $C = (a \vee b \vee c)$
- modified circuit: $(C \vee d) \wedge (C \vee e) \wedge (C \vee f)$

Elimination of large bicliques in Boolean circuits



Elimination of large bicliques in Boolean circuits



Conclusions

- showed an efficient knowledge compilation parameterised by cliquewidth of a Boolean circuit
- showed that cliquewidth is not more ‘powerful’ than treewidth for Boolean function representation