Hypergraph Partitioning for Compiling Pseudo-Boolean Formulae

Romain Wallon

ROADEF'21, Session *Partitionnement des Graphes* – April 29, 2021

Laboratoire d'Informatique de l'X (LIX), École Polytechnique, X-Uber Chair

Symbolic AI and Boolean Reasoning

We consider Boolean (i.e., $\{0, 1\}$) variables to represent knowledge

Symbolic AI and Boolean Reasoning

We consider Boolean (i.e., $\{0, 1\}$) variables to represent knowledge

Most of the time, knowledge is encoded using *conjunctions of clauses*, a.k.a. CNF formulae

$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

Symbolic AI and Boolean Reasoning

We consider Boolean (i.e., $\{0, 1\}$) variables to represent knowledge

Most of the time, knowledge is encoded using *conjunctions of clauses*, a.k.a. CNF formulae

$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

The problem is often to check whether such a formula is satisfiable, i.e., has a solution

Dual Hypergraph of CNF Formulae

It is often convenient to use (hyper)graph representations of CNF Formulae to get some information about the structure of the formula

Dual Hypergraph of CNF Formulae

It is often convenient to use (hyper)graph representations of CNF Formulae to get some information about the structure of the formula

$$\gamma_1 \equiv a \vee \bar{b} \qquad \gamma_2 \equiv \bar{a} \vee c \qquad \gamma_3 \equiv b \vee \bar{d} \vee \bar{e} \qquad \gamma_4 \equiv \bar{b} \vee e \vee f$$

Dual Hypergraph of CNF Formulae

It is often convenient to use (hyper)graph representations of CNF Formulae to get some information about the structure of the formula

$$\gamma_1 \equiv a \vee \bar{b} \qquad \gamma_2 \equiv \bar{a} \vee c \qquad \gamma_3 \equiv b \vee \bar{d} \vee \bar{e} \qquad \gamma_4 \equiv \bar{b} \vee e \vee f$$

Its dual hypergraph has as hypervertices the clauses of the formula and as hyperedges the variables of this formula

Dual Hypergraph of CNF Formulae

It is often convenient to use (hyper)graph representations of CNF Formulae to get some information about the structure of the formula

$$\gamma_1 \equiv a \vee \bar{b} \qquad \gamma_2 \equiv \bar{a} \vee c \qquad \gamma_3 \equiv b \vee \bar{d} \vee \bar{e} \qquad \gamma_4 \equiv \bar{b} \vee e \vee f$$

Its dual hypergraph has as hypervertices the clauses of the formula and as hyperedges the variables of this formula

A hyperedge covers the clauses containing the corresponding variable

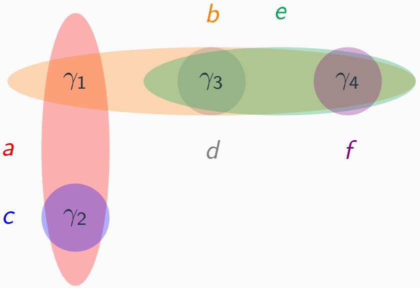
Dual Hypergraph of CNF Formulae

It is often convenient to use (hyper)graph representations of CNF Formulae to get some information about the structure of the formula

$$\gamma_1 \equiv a \vee \bar{b} \qquad \gamma_2 \equiv \bar{a} \vee c \qquad \gamma_3 \equiv b \vee \bar{d} \vee \bar{e} \qquad \gamma_4 \equiv \bar{b} \vee e \vee f$$

Its dual hypergraph has as hypervertices the clauses of the formula and as hyperedges the variables of this formula

A hyperedge covers the clauses containing the corresponding variable



Reasoning on CNF Formulae and Limitations

The main reason for using CNF formulae to represent knowledge is that we can use SAT solvers to reason with such formulae

Reasoning on CNF Formulae and Limitations

The main reason for using CNF formulae to represent knowledge is that we can use SAT solvers to reason with such formulae

Modern SAT solvers (Silva and Sakallah, 1996; Moskewicz et al., 2001; Eén and Sörensson, 2004) are very efficient in practice

Reasoning on CNF Formulae and Limitations

The main reason for using CNF formulae to represent knowledge is that we can use SAT solvers to reason with such formulae

Modern SAT solvers (Silva and Sakallah, 1996; Moskewicz et al., 2001; Eén and Sörensson, 2004) are very efficient in practice

However, they do not offer time guarantees when given an input to solve

Reasoning on CNF Formulae and Limitations

The main reason for using CNF formulae to represent knowledge is that we can use SAT solvers to reason with such formulae

Modern SAT solvers (Silva and Sakallah, 1996; Moskewicz et al., 2001; Eén and Sörensson, 2004) are very efficient in practice

However, they do not offer time guarantees when given an input to solve

For some applications, especially those involving interactions with users, this is not acceptable

Reasoning on CNF Formulae and Limitations

The main reason for using CNF formulae to represent knowledge is that we can use SAT solvers to reason with such formulae

Modern SAT solvers (Silva and Sakallah, 1996; Moskewicz et al., 2001; Eén and Sörensson, 2004) are very efficient in practice

However, they do not offer time guarantees when given an input to solve

For some applications, especially those involving interactions with users, this is not acceptable

In such cases, it may be interesting to rely on knowledge compilation

Knowledge Compilation

Given a formula written in a specific language (e.g., CNF), some operations may be too expensive in practice to be performed online

Knowledge Compilation

Given a formula written in a specific language (e.g., CNF), some operations may be too expensive in practice to be performed online

Compiling a formula is translating it (offline) into another language to obtain an equivalent formula on which performing the wanted (online) operations is easier

Targetting the d-DNNF Language

The language of d-DNNF is the language of deterministic and decomposable NNF

Targetting the d-DNNF Language

The language of d-DNNF is the language of deterministic and decomposable NNF

NNF is the language of Boolean circuits in Negation Normal Form, in which negations are only applied on variables

Targetting the d-DNNF Language

The language of d-DNNF is the language of deterministic and decomposable NNF

NNF is the language of Boolean circuits in Negation Normal Form, in which negations are only applied on variables

Deterministic means that, for any disjunction $\varphi \lor \psi$, there is no common model between φ and ψ (i.e., $\varphi \land \psi \models \bot$)

Targetting the d-DNNF Language

The language of d-DNNF is the language of deterministic and decomposable NNF

NNF is the language of Boolean circuits in Negation Normal Form, in which negations are only applied on variables

Deterministic means that, for any disjunction $\varphi \lor \psi$, there is no common model between φ and ψ (i.e., $\varphi \land \psi \models \bot$)

Decomposable means that, for each conjunction $\varphi \land \psi$, there is no common variable between φ and ψ (i.e., $\mathsf{var}(\varphi) \cap \mathsf{var}(\psi) = \emptyset$)

Targetting the d-DNNF Language

The language of d-DNNF is the language of deterministic and decomposable NNF

NNF is the language of Boolean circuits in Negation Normal Form, in which negations are only applied on variables

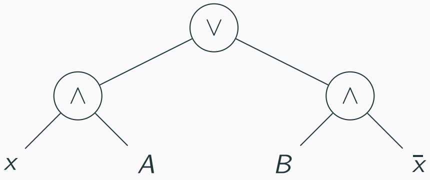
Deterministic means that, for any disjunction $\varphi \vee \psi$, there is no common model between φ and ψ (i.e., $\varphi \wedge \psi \models \bot$)

Decomposable means that, for each conjunction $\varphi \wedge \psi$, there is no common variable between φ and ψ (i.e., $\mathsf{var}(\varphi) \cap \mathsf{var}(\psi) = \emptyset$)

These two properties allow the efficient computation of different queries

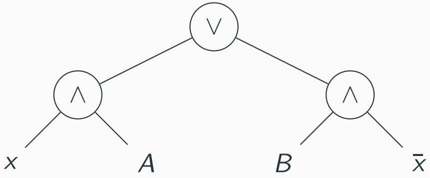
Ensuring Determinism

To ensure determinism, each disjunction node in the circuit will be a decision node

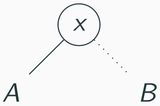


Ensuring Determinism

To ensure determinism, each disjunction node in the circuit will be a decision node

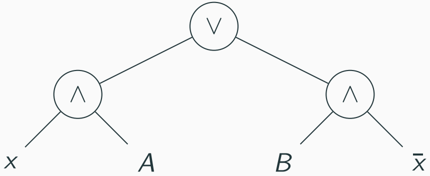


For more readablity, we will represent the decision node above as

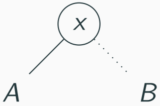


Ensuring Determinism

To ensure determinism, each disjunction node in the circuit will be a decision node



For more readablity, we will represent the decision node above as



The d-DNNFs we obtain in this case are called Decision-DNNF

Ensuring Decomposability

To ensure decomposability, a partition of the dual hypergraph of the CNF to compile is computed, to extract independent connected components

Ensuring Decomposability

To ensure decomposability, a partition of the dual hypergraph of the CNF to compile is computed, to extract independent connected components

This operation yields a cutset, which is a set of hyperedges (i.e., variables) that must be removed (i.e., assigned) to get disjoint components

Ensuring Decomposability

To ensure decomposability, a partition of the dual hypergraph of the CNF to compile is computed, to extract independent connected components

This operation yields a cutset, which is a set of hyperedges (i.e., variables) that must be removed (i.e., assigned) to get disjoint components

By construction, each connected component do not share variables

Ensuring Decomposability

To ensure decomposability, a partition of the dual hypergraph of the CNF to compile is computed, to extract independent connected components

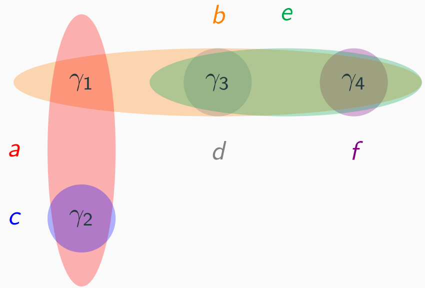
This operation yields a cutset, which is a set of hyperedges (i.e., variables) that must be removed (i.e., assigned) to get disjoint components

By construction, each connected component do not share variables

The connected components can then be compiled independently, before adding their conjunction to the build d-DNNF

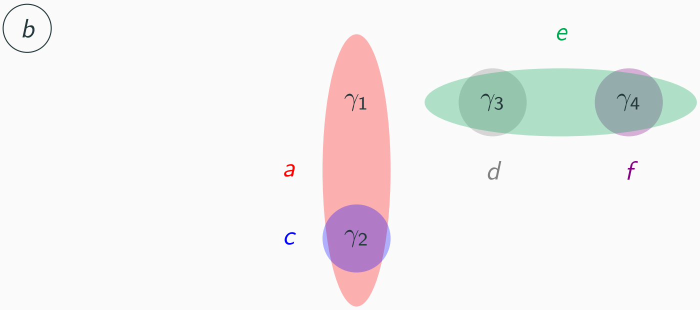
Compiling our CNF Formula

$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$



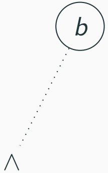
Compiling our CNF Formula

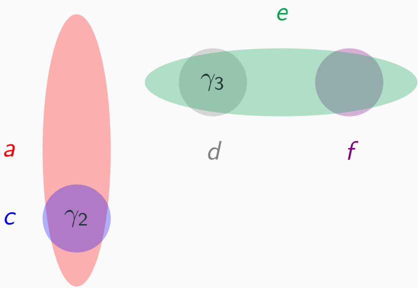
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$



Compiling our CNF Formula

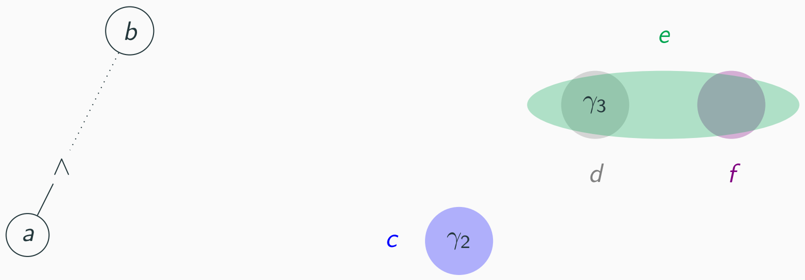
$$(a \lor \bar{b}) \land (\bar{a} \lor c) \land (b \lor \bar{d} \lor \bar{e}) \land (\bar{b} \lor e \lor f)$$





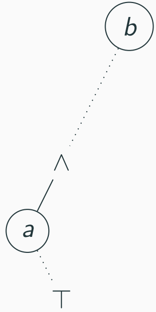
Compiling our CNF Formula

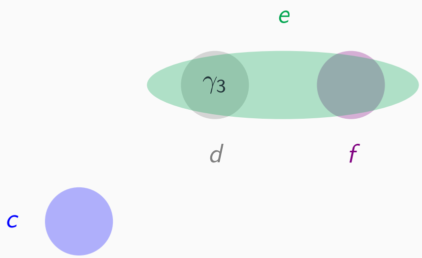
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$



Compiling our CNF Formula

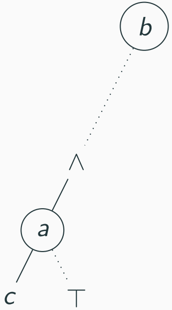
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

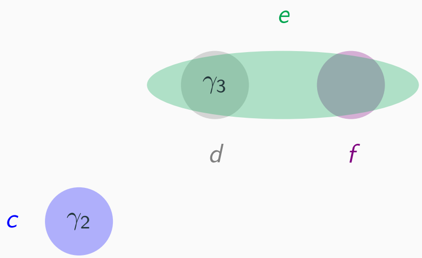




Compiling our CNF Formula

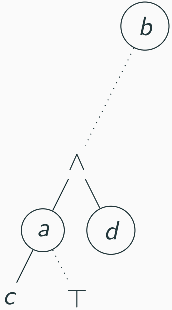
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

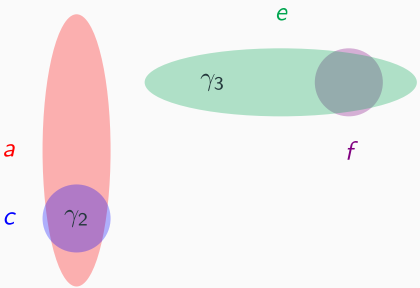




Compiling our CNF Formula

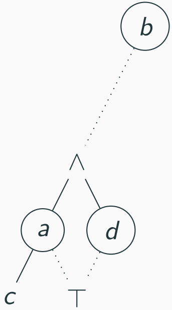
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

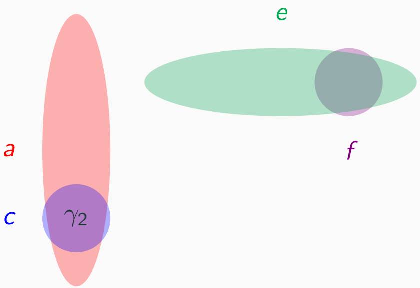




Compiling our CNF Formula

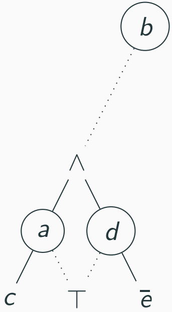
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

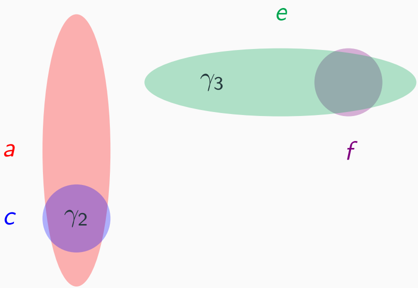




Compiling our CNF Formula

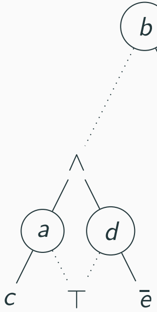
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

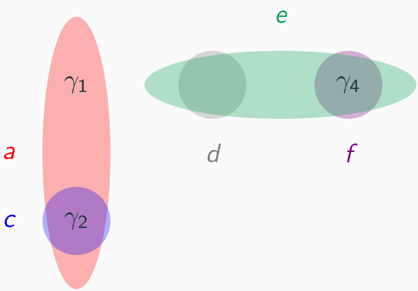




Compiling our CNF Formula

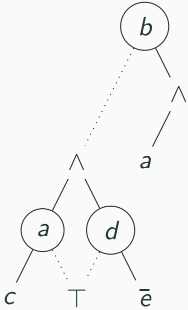
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

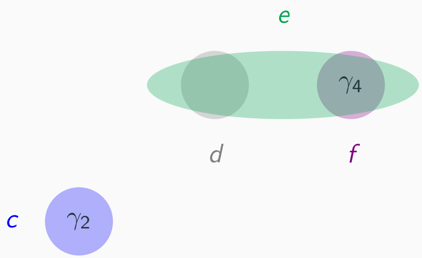




Compiling our CNF Formula

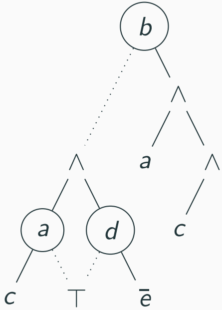
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

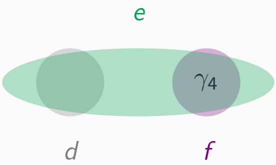




Compiling our CNF Formula

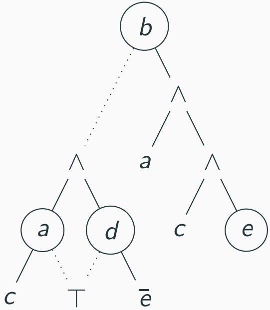
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

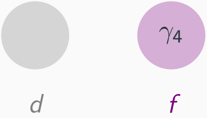




Compiling our CNF Formula

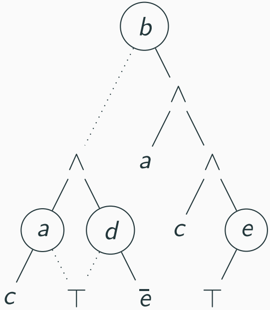
$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$





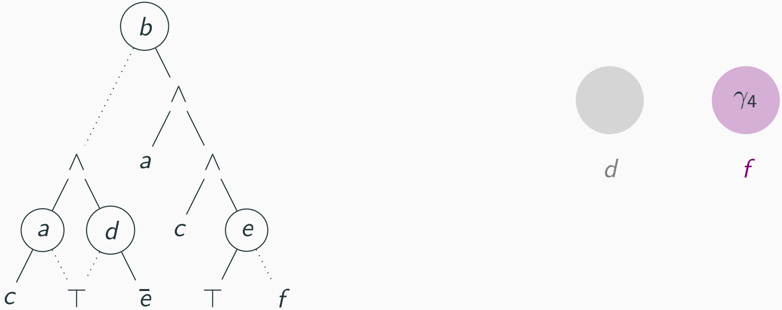
Compiling our CNF Formula

$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$



Compiling our CNF Formula

$$(a \vee \bar{b}) \wedge (\bar{a} \vee c) \wedge (b \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee e \vee f)$$

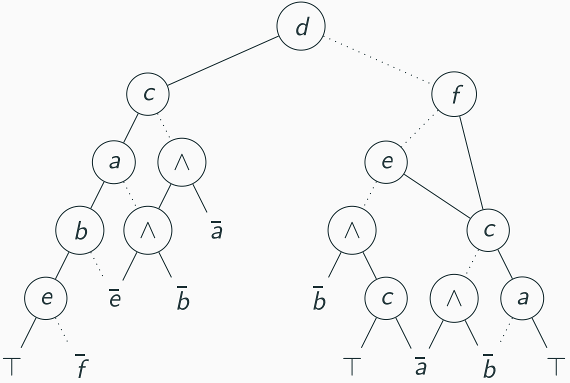


Impact of the Quality of the Partition

Finding a good partition is crucial for compiling the input into a small Decision-DNNF

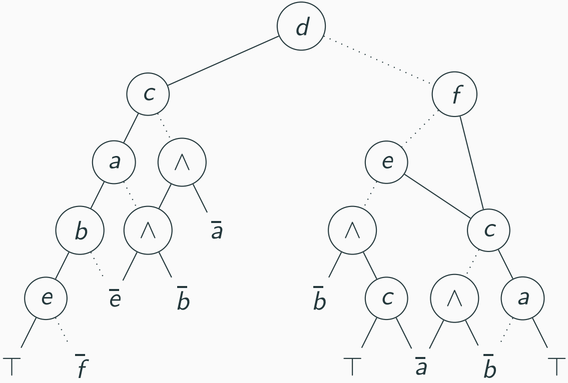
Impact of the Quality of the Partition

Finding a good partition is crucial for compiling the input into a small Decision-DNNF



Impact of the Quality of the Partition

Finding a good partition is crucial for compiling the input into a small Decision-DNNF



Ideally, we need small cutsets and balanced partitions

Outline of *D4* (Lagniez and Marquis, 2017)

1. Invoke a SAT Solver on the input
2. If the formula is `UNSAT`, then the compiled form is $\bot$
3. If all variables are assigned, then the compiled form is $\top$
4. For each connected component φ of the formula:
   a. Choose a variable v based on a cutset of φ computed with PaToH (Çatalyürek and Aykanat, 2011)
   b. Compile $\varphi|v$ as φ_v
   c. Compile $\varphi|\bar{v}$ as $\varphi_{\bar{v}}$
   d. The compiled form of φ is $\text{ite}(v, \varphi_v, \varphi_{\bar{v}})$
5. The compiled form is the conjunction of the compiled forms obtained above

Outline of *D4* (Lagniez and Marquis, 2017)

1. Invoke a SAT Solver on the input
2. If the formula is `UNSAT`, then the compiled form is $\bot$
3. If all variables are assigned, then the compiled form is $\top$
4. For each connected component φ of the formula:
    a. Choose a variable v based on a cutset of φ computed with PaToH (Çatalyürek and Aykanat, 2011)
    b. Compile $\varphi|v$ as φ_v
    c. Compile $\varphi|\bar{v}$ as $\varphi_{\bar{v}}$
    d. The compiled form of φ is $\text{ite}(v, \varphi_v, \varphi_{\bar{v}})$
5. The compiled form is the conjunction of the compiled forms obtained above

D4 is available at `https://github.com/crillab/d4`

SAT Solver Limitations

The algorithm we presented uses SAT solvers as oracles to benefit from their practical efficiency

SAT Solver Limitations

The algorithm we presented uses SAT solvers as oracles to benefit from their practical efficiency

However, some instances remain completely out of reach for modern SAT solvers, especially when counting capabilities are required

SAT Solver Limitations

The algorithm we presented uses SAT solvers as oracles to benefit from their practical efficiency

However, some instances remain completely out of reach for modern SAT solvers, especially when counting capabilities are required

For instance, SAT solvers cannot prove efficiently that *"n pigeons do not fit in $n-1$ holes"* (Haken, 1985)

SAT Solver Limitations

The algorithm we presented uses SAT solvers as oracles to benefit from their practical efficiency

However, some instances remain completely out of reach for modern SAT solvers, especially when counting capabilities are required

For instance, SAT solvers cannot prove efficiently that *"n pigeons do not fit in $n-1$ holes"* (Haken, 1985)

On such instances, pseudo-Boolean reasoning can offer better performance

Pseudo-Boolean (PB) Constraints

PB solvers are generalizations of SAT solvers that allow to consider

- normalized PB constraints $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$
- cardinality constraints $\sum_{i=1}^{n} \ell_i \geq \delta$
- clauses $\sum_{i=1}^{n} \ell_i \geq 1$

in which

- the coefficients α_i are non-negative integers
- ℓ_i are literals, i.e., a variable v or its negation $\bar{v} = 1 - v$
- the degree δ is a non-negative integer

Pseudo-Boolean (PB) Constraints

PB solvers are generalizations of SAT solvers that allow to consider

- normalized PB constraints $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$
- cardinality constraints $\sum_{i=1}^{n} \ell_i \geq \delta$
- clauses $\sum_{i=1}^{n} \ell_i \geq 1$

in which

- the coefficients α_i are non-negative integers
- ℓ_i are literals, i.e., a variable v or its negation $\bar{v} = 1 - v$
- the degree δ is a non-negative integer

PB constraints allow in general more succinct encodings than CNF, and are often more natural to use

Succinctness of PB Constraints

To illustrate the succinctness of PB constraints compared to CNF, consider the cardinality constraint

$$a + b + c + d + e \geq 3$$

Succinctness of PB Constraints

To illustrate the succinctness of PB constraints compared to CNF, consider the cardinality constraint

$$a + b + c + d + e \geq 3$$

Its CNF encoding is given by

$$\begin{aligned} &(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee d) \wedge (a \vee c \vee e) \\ \wedge &(a \vee d \vee e) \wedge (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee d \vee e) \wedge (c \vee d \vee e) \end{aligned}$$

Succinctness of PB Constraints

To illustrate the succinctness of PB constraints compared to CNF, consider the cardinality constraint

$$a + b + c + d + e \geq 3$$

Its CNF encoding is given by

$$\begin{aligned}
&(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee d) \wedge (a \vee c \vee e) \\
\wedge &(a \vee d \vee e) \wedge (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee d \vee e) \wedge (c \vee d \vee e)
\end{aligned}$$

In general, PB representations may be exponentially smaller than CNF representations

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

They allow to consider instances that are too big when represented in CNF to be compiled

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

They allow to consider instances that are too big when represented in CNF to be compiled

PB solvers may find exponentially shorter proofs while inheriting many of the efficient techniques developped in SAT solvers

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

They allow to consider instances that are too big when represented in CNF to be compiled

PB solvers may find exponentially shorter proofs while inheriting many of the efficient techniques developped in SAT solvers

Using such solvers as oracles may allow to speed up the compilation time

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

They allow to consider instances that are too big when represented in CNF to be compiled

PB solvers may find exponentially shorter proofs while inheriting many of the efficient techniques developped in SAT solvers

Using such solvers as oracles may allow to speed up the compilation time

Supporting PB constraints only requires to extend the existing algorithm, without forcing to redesign a completely new approach

Compiling PB Formulae

A first advantage of the native support of PB constraints for knowledge compilation is that PB representations may be more succinct and more natural than CNF representations

They allow to consider instances that are too big when represented in CNF to be compiled

PB solvers may find exponentially shorter proofs while inheriting many of the efficient techniques developped in SAT solvers

Using such solvers as oracles may allow to speed up the compilation time

Supporting PB constraints only requires to extend the existing algorithm, without forcing to redesign a completely new approach

To support PB compilation, one basically needs to replace by a PB solver the SAT solver used in the compilation procedure

Dual Hypergraph of a PB Formula

The dual hypergraph of a PB formula is defined as for CNF formulae

Dual Hypergraph of a PB Formula

The dual hypergraph of a PB formula is defined as for CNF formulae

$\chi_1 \equiv a + \bar{b} \geq 1 \quad \chi_2 \equiv \bar{a} + c \geq 1 \quad \chi_3 \equiv b + \bar{d} + \bar{e} \geq 2 \quad \chi_4 \equiv 2\bar{b} + e + f \geq 3$

Dual Hypergraph of a PB Formula

The dual hypergraph of a PB formula is defined as for CNF formulae

$\chi_1 \equiv a + \bar{b} \geq 1 \quad \chi_2 \equiv \bar{a} + c \geq 1 \quad \chi_3 \equiv b + \bar{d} + \bar{e} \geq 2 \quad \chi_4 \equiv 2\bar{b} + e + f \geq 3$

Its dual hypergraph has as hypervertices the constraints of the formula and as hyperedges the variables of this formula

A hyperedge covers the constraints containing the corresponding variable

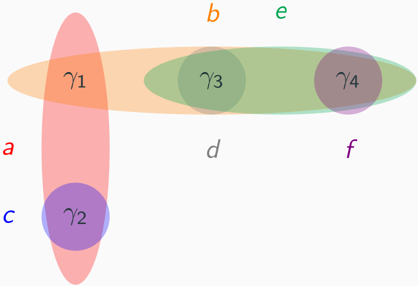
Dual Hypergraph of a PB Formula

The dual hypergraph of a PB formula is defined as for CNF formulae

$$\chi_1 \equiv a + \bar{b} \geq 1 \quad \chi_2 \equiv \bar{a} + c \geq 1 \quad \chi_3 \equiv b + \bar{d} + \bar{e} \geq 2 \quad \chi_4 \equiv 2\bar{b} + e + f \geq 3$$

Its dual hypergraph has as hypervertices the constraints of the formula and as hyperedges the variables of this formula

A hyperedge covers the constraints containing the corresponding variable



Outline of *PBD4*

1. Invoke a PB Solver on the input
2. If the formula is `UNSAT`, then the compiled form is $\bot$
3. If all variables are assigned, then the compiled form is $\top$
4. For each connected component φ of the formula:
   a. Choose a variable v based on a cutset of φ computed with KaHyPar (Schlag, 2020)
   b. Compile $\varphi|v$ as φ_v
   c. Compile $\varphi|\bar{v}$ as $\varphi_{\bar{v}}$
   d. The compiled form of φ is $\text{ite}(v, \varphi_v, \varphi_{\bar{v}})$
5. The compiled form is the conjunction of the compiled forms obtained above

Outline of *PBD4*

1. Invoke a PB Solver on the input
2. If the formula is `UNSAT`, then the compiled form is $\bot$
3. If all variables are assigned, then the compiled form is $\top$
4. For each connected component φ of the formula:
   a. Choose a variable v based on a cutset of φ computed with KaHyPar (Schlag, 2020)
   b. Compile $\varphi|v$ as φ_v
   c. Compile $\varphi|\bar{v}$ as $\varphi_{\bar{v}}$
   d. The compiled form of φ is $\text{ite}(v, \varphi_v, \varphi_{\bar{v}})$
5. The compiled form is the conjunction of the compiled forms obtained above

PBD4 is available at `https://github.com/crillab/pbd4`

Conclusion

- Knowledge compilation ensures runtime guarantees for online operations

Conclusion

- Knowledge compilation ensures runtime guarantees for online operations
- Hypergraph partitioning provides a heuristic to decide in which order to assign variables when building the compiled form

Conclusion

- Knowledge compilation ensures runtime guarantees for online operations
- Hypergraph partitioning provides a heuristic to decide in which order to assign variables when building the compiled form
- Modern and efficient SAT solvers are used as oracles to determine whether it is worth compiling subformulae

Conclusion

- Knowledge compilation ensures runtime guarantees for online operations
- Hypergraph partitioning provides a heuristic to decide in which order to assign variables when building the compiled form
- Modern and efficient SAT solvers are used as oracles to determine whether it is worth compiling subformulae
- For compiling certain problems, using PB solvers instead may be more efficient

Perspectives

- Take advantage of native PB compilation for considering new applications of knowledge compilation (e.g., for explaining (binarized) neural networks)

Perspectives

- Take advantage of native PB compilation for considering new applications of knowledge compilation (e.g., for explaining (binarized) neural networks)
- Use speculation techniques to speed up compilation:

Perspectives

- Take advantage of native PB compilation for considering new applications of knowledge compilation (e.g., for explaining (binarized) neural networks)
- Use speculation techniques to speed up compilation:
  - by predicting satisfiability before invoking the SAT/PB solver as an oracle

Perspectives

- Take advantage of native PB compilation for considering new applications of knowledge compilation (e.g., for explaining (binarized) neural networks)
- Use speculation techniques to speed up compilation:
    - by predicting satisfiability before invoking the SAT/PB solver as an oracle
    - by predicting cutsets before computing a partition of the hypergraph

Hypergraph Partitioning for Compiling Pseudo-Boolean Formulae

Romain Wallon

ROADEF'21, Session *Partitionnement des Graphes* – April 29, 2021

Laboratoire d'Informatique de l'X (LIX), École Polytechnique, X-Uber Chair

References i

Çatalyürek, Ü. V. and Aykanat, C. (2011). Patoh (partitioning tool for hypergraphs). In Padua, D. A., editor, *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer.

Eén, N. and Sörensson, N. (2004). An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518.

Haken, A. (1985). The intractability of resolution. *Theoretical Computer Science*, 39:297 – 308. Third Conference on Foundations of Software Technology and Theoretical Computer Science.

Lagniez, J.-M. and Marquis, P. (2017). An improved decision-dnnf compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 667–673.

References ii

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA. ACM.

Schlag, S. (2020). *High-Quality Hypergraph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, Germany.

Silva, J. a. P. M. and Sakallah, K. A. (1996). GRASP – New Search Algorithm for Satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA. IEEE Computer Society.