Machine learning of strategies for efficiently solving QBF with abstraction refinement

Ricardo Joel Silva **Mikoláš Janota**

Instituto Superior Técnico, Universidade de Lisboa, Portugal

Rende, 20 November 2019

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
$\forall x \exists y.(x \leftrightarrow y)$

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:

$\forall x \exists y.(x \leftrightarrow y)$

$\forall x.(x \leftrightarrow 0) \vee (x \leftrightarrow 1)$

Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
$\forall x \exists y.(x \leftrightarrow y)$
$\forall x.(x \leftrightarrow 0) \vee (x \leftrightarrow 1)$
$((0 \leftrightarrow 0) \vee (0 \leftrightarrow 1)) \wedge ((1 \leftrightarrow 0) \vee (1 \leftrightarrow 1))$
1

QBF is the paradigmatic PSPACE-complete problem

- Model checking
- Circuit synthesis
- Non-monotonic reasoning
- Conformant planning
- ...

Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2 .\, (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$

Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2.\, (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.

Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2.\, (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.

Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2.\ (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.
- $\exists$ wins the game if the matrix becomes true.

Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.
- $\exists$ wins the game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.

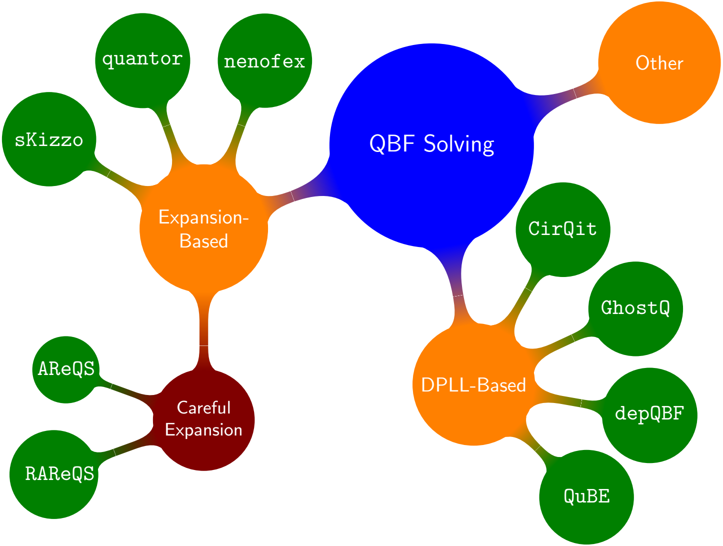
Games and strategies

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2 .\, (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.
- $\exists$ wins the game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.
- A QBF is true iff there exists a winning strategy for $\exists$.

- We consider prenex form: *Quantifier-prefix*. *Matrix*
  Example $\forall x_1 x_2 \exists y_1 y_2 . (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.
- $\exists$ wins the game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.
- A QBF is true iff there exists a winning strategy for $\exists$.
- **Example**

$$\forall u \exists e.\, u \leftrightarrow e$$

Solving QBF: DPLL versus expansion based solvers



Solving by Expansion

$$\exists x_1 \ldots x_m \forall y_1 \ldots y_m.\ \phi$$

Equivalent to:

Solving by Expansion

$$\exists x_1 \ldots x_m \forall y_1 \ldots y_m .\ \phi$$

Equivalent to:

$$(\exists x_1 \ldots x_m) \left.\begin{array}{cccccc} & & \boldsymbol{y_1}, & \ldots, & \boldsymbol{y_{n-1}}, & \boldsymbol{y_n} \\ \wedge & \phi[& 0, & \ldots, & 0, & 0 \quad] \\ \wedge & \phi[& 0, & \ldots, & 0, & 1 \quad] \\ \wedge & \phi[& 0, & \ldots, & 1, & 0 \quad] \\ \wedge & \phi[& 0, & \ldots, & 1, & 1 \quad] \\ \wedge & \ldots & & & & \\ \wedge & \phi[& 1, & \ldots, & 1, & 1 \quad] \end{array}\right\}$$

Solving by Expansion

$$\exists x_1 \ldots x_m \forall y_1 \ldots y_m.\ \phi$$

Equivalent to:

$$
(\exists x_1 \ldots x_m) \quad
\left.
\begin{array}{llllll}
 & & \boldsymbol{y_1}, & \ldots, & \boldsymbol{y_{n-1}}, & \boldsymbol{y_n} \\
\wedge & \phi[& 0, & \ldots, & 0, & 0 \ ] \\
\wedge & \phi[& 0, & \ldots, & 0, & 1 \ ] \\
\wedge & \phi[& 0, & \ldots, & 1, & 0 \ ] \\
\wedge & \phi[& 0, & \ldots, & 1, & 1 \ ] \\
\wedge & \ldots & & & & \\
\wedge & \phi[& 1, & \ldots, & 1, & 1 \ ]
\end{array}
\right\}
$$

- Now only one type of quantifier: $\exists x_1 \ldots x_m$

Solving by Expansion

$$\exists x_1 \ldots x_m \forall y_1 \ldots y_m .\ \phi$$

Equivalent to:

$$
(\exists x_1 \ldots x_m) \quad
\left.
\begin{array}{llcccccr}
 & & \boldsymbol{y_1}, & \ldots, & \boldsymbol{y_{n-1}}, & \boldsymbol{y_n} & \\
\wedge & \phi[& 0, & \ldots, & 0, & 0 &] \\
\wedge & \phi[& 0, & \ldots, & 0, & 1 &] \\
\wedge & \phi[& 0, & \ldots, & 1, & 0 &] \\
\wedge & \phi[& 0, & \ldots, & 1, & 1 &] \\
\wedge & \ldots & & & & & \\
\wedge & \phi[& 1, & \ldots, & 1, & 1 &]
\end{array}
\right\}
$$

- Now only one type of quantifier: $\exists x_1 \ldots x_m$
- We can call a SAT solver!

Solving by Expansion

$$\exists x_1 \ldots x_m \forall y_1 \ldots y_m.\ \phi$$

Equivalent to:

$$(\exists x_1 \ldots x_m) \quad \left.\begin{array}{cllllll} & & \boldsymbol{y_1}, & \ldots, & \boldsymbol{y_{n-1}}, & \boldsymbol{y_n} & \\ \wedge & \phi[& 0, & \ldots, & 0, & 0 &] \\ \wedge & \phi[& 0, & \ldots, & 0, & 1 &] \\ \wedge & \phi[& 0, & \ldots, & 1, & 0 &] \\ \wedge & \phi[& 0, & \ldots, & 1, & 1 &] \\ \wedge & \ldots & & & & & \\ \wedge & \phi[& 1, & \ldots, & 1, & 1 &] \end{array}\right\} 2^n$$

- Now only one type of quantifier: $\exists x_1 \ldots x_m$
- We can call a SAT solver!

Solving by Expansion (contd.)

- Expanding everything = exponential blow-up

Solving by Expansion (contd.)

- Expanding everything = exponential blow-up
- Do we need to expand everything?

Solving by Expansion (contd.)

- Expanding everything = exponential blow-up
- Do we need to expand everything?
- **Example:**

$$\exists x_1 x_2 \forall y_1 y_2.\ (x_1 \land x_2 \land y_1 \land y_2)$$

- Expanding everything = exponential blow-up
- Do we need to expand everything?
- **Example:**

$$\exists x_1 x_2 \forall y_1 y_2 .\ (x_1 \wedge x_2 \wedge y_1 \wedge y_2)$$

- . . . sufficient to expand $y_1 = y_2 = 0$

Solving by Expansion (contd.)

- Expanding everything = exponential blow-up
- Do we need to expand everything?
- **Example:**

$$\exists x_1 x_2 \forall y_1 y_2 .\ (x_1 \wedge x_2 \wedge y_1 \wedge y_2)$$

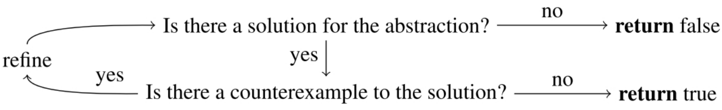
- ... sufficient to expand $y_1 = y_2 = 0$
- How to come up with the right expansions?

CEGAR paradigm : careful expansion

- Expand the formula gradually, to avoid exponential blow-up.
- . . . means gradually strengthening abstraction of the formula.

CEGAR paradigm : careful expansion

- Expand the formula gradually, to avoid exponential blow-up.
- ... means gradually strengthening abstraction of the formula.

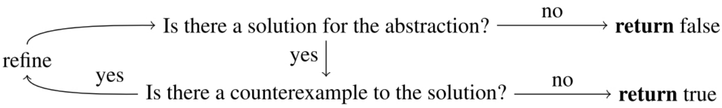


CEGAR loop

refine = expand more

CEGAR paradigm : careful expansion

- Expand the formula gradually, to avoid exponential blow-up.
- ... means gradually strengthening abstraction of the formula.



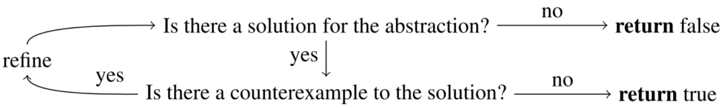
CEGAR loop

refine = expand more

AReQS: CEGAR-based solver for 2-QBF [J. and Silva SAT'11]

CEGAR paradigm : careful expansion

- Expand the formula gradually, to avoid exponential blow-up.
- . . . means gradually strengthening abstraction of the formula.



CEGAR loop

refine = expand more

AReQS: CEGAR-based solver for 2-QBF **[J. and Silva SAT'11]**
RAReQS: generalises AReQS through recursion **[J. et al. SAT'12]**

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move Counter-move

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move	Counter-move
$000\ldots0001$	

Issue with Expansion

Example

$$\exists x_1 \dots x_{100} \forall y_1 \dots y_{100}. \bigvee_{i=1\dots 100} (x_i \neq y_i)$$

Move	Counter-move
000 . . . 0001	000 . . . 0001

Issue with Expansion

Example

$$\exists x_1 \dots x_{100} \forall y_1 \dots y_{100} . \bigvee_{i=1 \dots 100} (x_i \neq y_i)$$

Move	Counter-move
000 . . . 0001	000 . . . 0001
000 . . . 0010	

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move	Counter-move
$000\ldots0001$	$000\ldots0001$
$000\ldots0010$	$000\ldots0010$

Issue with Expansion

Example

$$\exists x_1 \dots x_{100} \forall y_1 \dots y_{100} . \bigvee_{i=1\dots100} (x_i \neq y_i)$$

Move	Counter-move
$000 \dots 0001$	$000 \dots 0001$
$000 \dots 0010$	$000 \dots 0010$
$000 \dots 0011$	

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move	Counter-move
000 . . . 0001	000 . . . 0001
000 . . . 0010	000 . . . 0010
000 . . . 0011	000 . . . 0011

Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move	Counter-move
000 . . . 0001	000 . . . 0001
000 . . . 0010	000 . . . 0010
000 . . . 0011	000 . . . 0011
⋮	

Issue with Expansion

Example

$$\exists x_1 \dots x_{100} \forall y_1 \dots y_{100} . \bigvee_{i=1\dots100} (x_i \neq y_i)$$

Move	Counter-move
000 . . . 0001	000 . . . 0001
000 . . . 0010	000 . . . 0010
000 . . . 0011	000 . . . 0011

⋮

Expansion necessarily exponential

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$

Expansion by Strategies

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$
- BUT: $\forall$ has a short **winning strategy**

Expansion by Strategies

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$
- BUT: $\forall$ has a short **winning strategy**
- $\forall$-player wins by playing $y_i \triangleq x_i$.

Expansion by Strategies

Example

$$\exists x_1 \dots x_{100} \forall y_1 \dots y_{100}. \bigvee_{i=1\dots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$
- BUT: $\forall$ has a short **winning strategy**
- $\forall$-player wins by playing $y_i \triangleq x_i$.
- Plug it in gives UNSAT:

$$\exists x_1 \dots x_{100}. \bigvee_{i=1\dots100} (x_i \neq x_i)$$

Expansion by Strategies

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$
- BUT: $\forall$ has a short **winning strategy**
- $\forall$-player wins by playing $y_i \triangleq x_i$.
- Plug it in gives UNSAT:

$$\exists x_1 \ldots x_{100}. \bigvee_{i=1\ldots100} (x_i \neq x_i)$$

- $y_i \triangleq x_i$ is **winning strategy for** $\forall$

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} . \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

- has exponential expansion by constants $0/1$
- BUT: $\forall$ has a short **winning strategy**
- $\forall$-player wins by playing $y_i \triangleq x_i$.
- Plug it in gives UNSAT:

$$\exists x_1 \ldots x_{100} . \bigvee_{i=1\ldots100} (x_i \neq x_i)$$

- $y_i \triangleq x_i$ is **winning strategy for** $\forall$
- QUESTION: So, how do we obtain good strategies?

- **QUESTION:** So, how do we obtain good strategies?

QFUN : Machine Learning for Strategies

- **QUESTION:** So, how do we obtain good strategies?
- Use machine learning
  - **repeatedly during** the execution of the solver
  - **on** previous moves and counter-moves

QFUN : Machine Learning for Strategies

- **QUESTION:** So, how do we obtain good strategies?
- Use machine learning
  - **repeatedly during** the execution of the solver
  - **on** previous moves and counter-moves
- **periodically** refine abstraction with the learned strategy

- **QUESTION:** So, how do we obtain good strategies?
- Use machine learning
  - **repeatedly during** the execution of the solver
  - **on** previous moves and counter-moves
- **periodically** refine abstraction with the learned strategy

Example:

Move	Counter-move
000...0001	000...0001
100...1000	000...1000
010...0011	010...0011
⋮	

- **QUESTION:** So, how do we obtain good strategies?
- Use machine learning
  - **repeatedly during** the execution of the solver
  - **on** previous moves and counter-moves
- **periodically** refine abstraction with the learned strategy

Example:

Move	Counter-move
000 . . . 0001	000 . . . 0001
100 . . . 1000	000 . . . 1000
010 . . . 0011	010 . . . 0011
⋮	

$y_i \triangleq x_i$ is **learnt from** $\ll 2^n$ **expansions**

[J. AAAI'18]

Requirements on ML

- Learning occurs during the solver's execution, therefore we have a tight time constraint

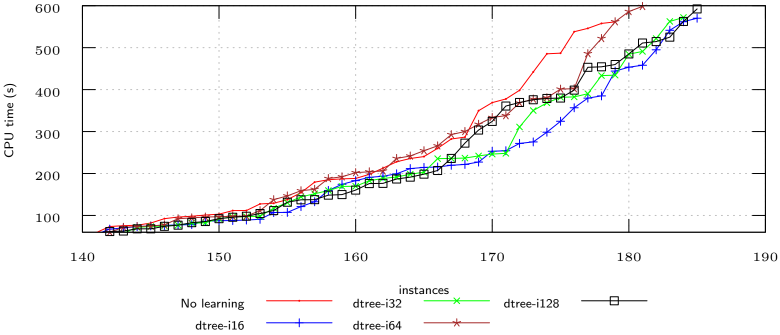
Requirements on ML

- Learning occurs during the solver's execution, therefore we have a tight time constraint
- We need to learn boolean formulae that can be passed onto the solver

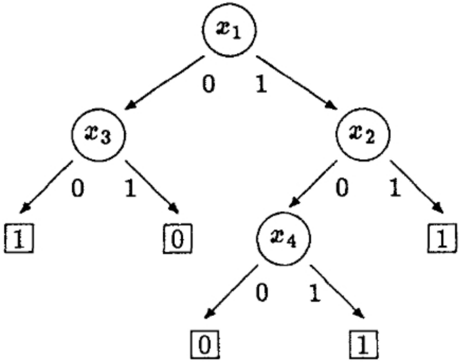
Requirements on ML

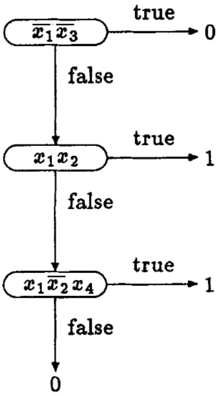
- Learning occurs during the solver's execution, therefore we have a tight time constraint
- We need to learn boolean formulae that can be passed onto the solver
- Our samples are small (especially for ML standards), but the number of variables can be quite large.

Originally: Decision Trees and ID3



Alternative to Decision Trees: Decision Lists





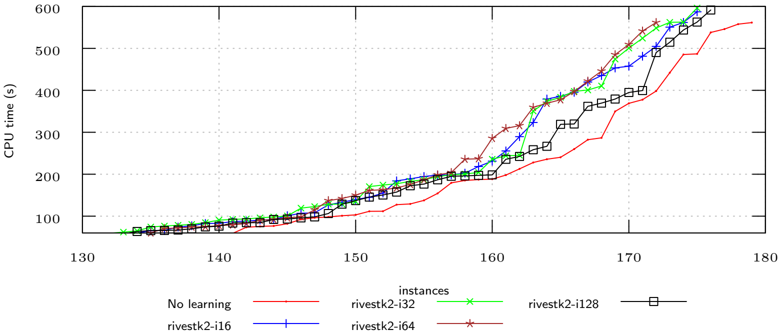
Learning Algorithms

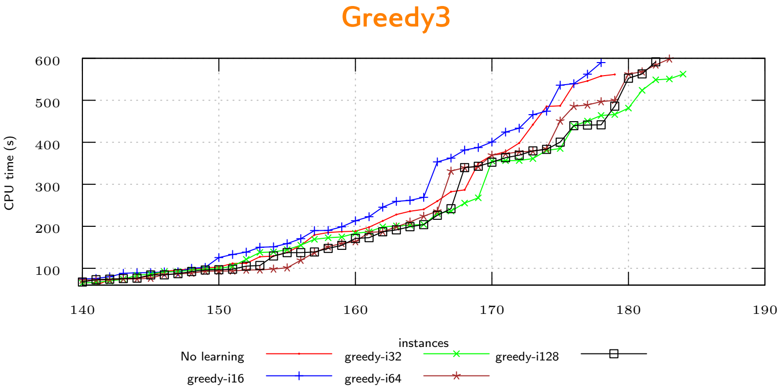
- k-decision list ... each rule at most k literals
- k-decision list are PAC-learnable **[Rivest '87]**

TÉCNICO LISBOA

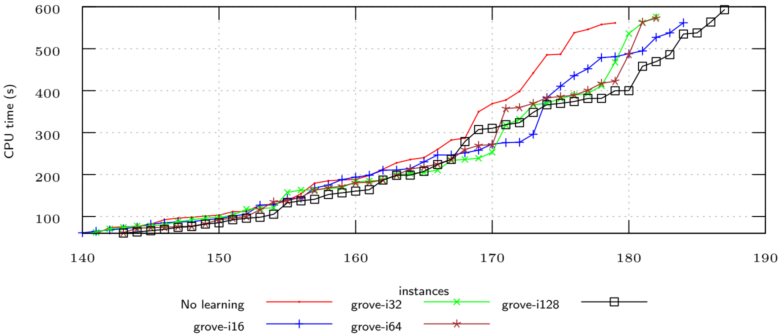
- k-decision list ... each rule at most k literals
- k-decision list are PAC-learnable **[Rivest '87]**

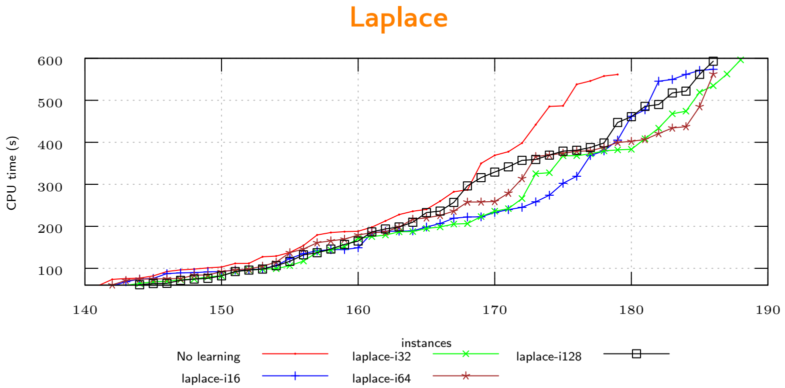
Decision Lists and Rivest ($k = 2$)

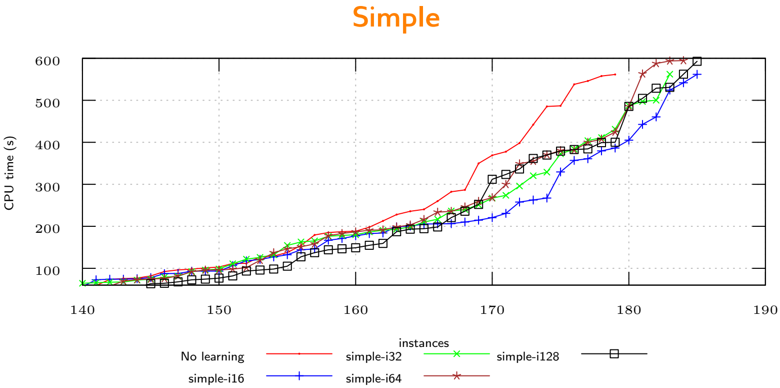


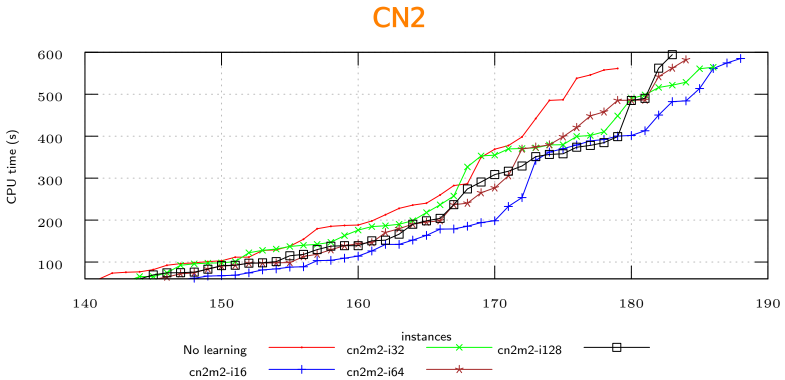

Greedy3
600
500
400
300
200
100
CPU time (s)
140
150
160
170
180
190
instances
No learning
greedy-i32
greedy-i128
greedy-i16
greedy-i64

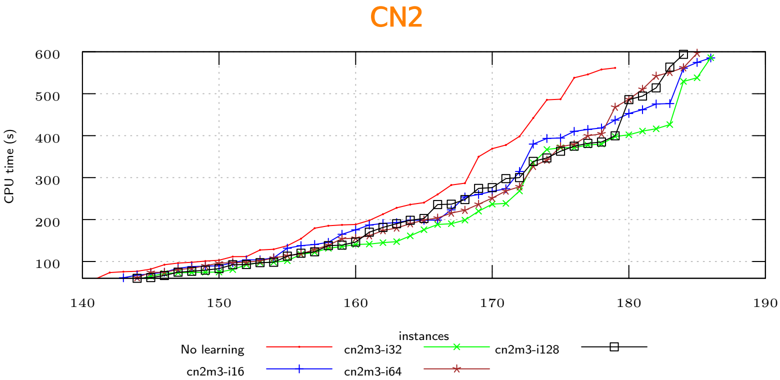
Grove


CPU time (s)
600
500
400
300
200
100
140
150
160
170
180
190
instances
No learning
grove-i32
grove-i128
grove-i16
grove-i64

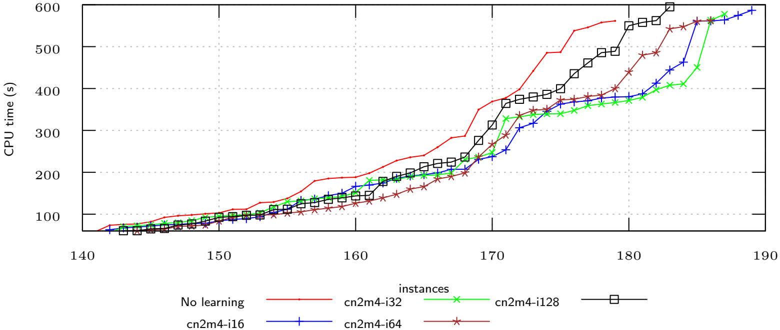

Laplace
CPU time (s)
600
500
400
300
200
100
140
150
160
170
180
190
instances
No learning
laplace-i16
laplace-i32
laplace-i64
laplace-i128


Simple
600
500
400
300
200
100
CPU time (s)
140
150
160
170
180
190
instances
No learning
simple-i32
simple-i128
simple-i16
simple-i64

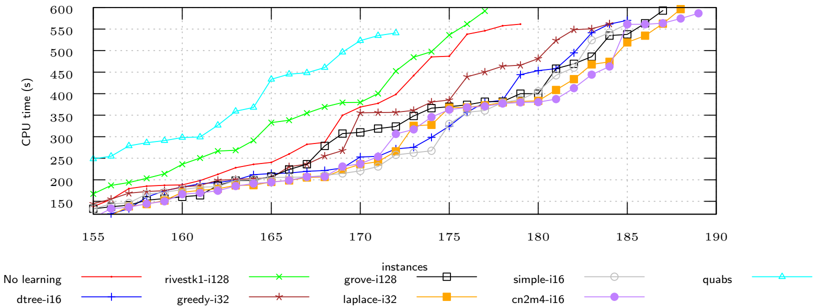

CN2
600
500
400
300
200
100
CPU time (s)
140
150
160
170
180
190
instances
No learning
cn2m2-i32
cn2m2-i128
cn2m2-i16
cn2m2-i64

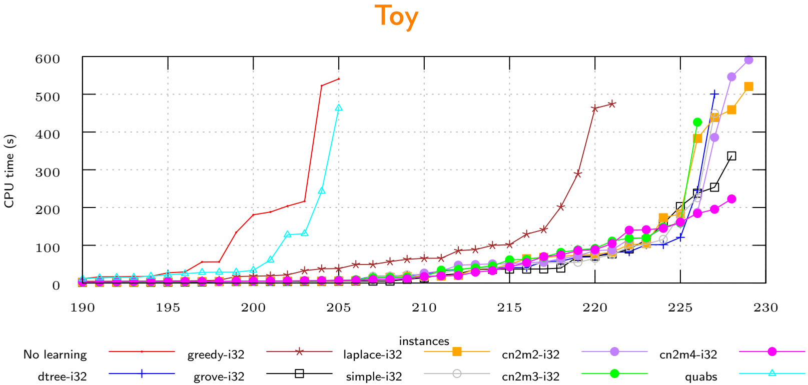

CN2
CPU time (s)
600
500
400
300
200
100
140
150
160
170
180
190
instances
No learning
cn2m3-i32
cn2m3-i128
cn2m3-i16
cn2m3-i64

CN2

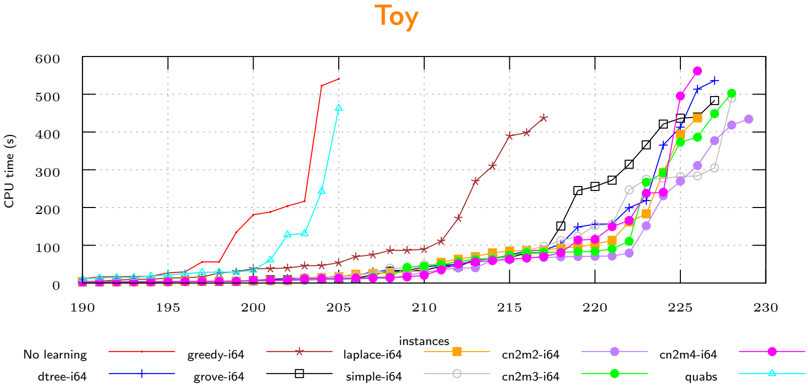

600
500
400
300
200
100
CPU time (s)
140
150
160
170
180
190
instances
No learning
cn2m4-i32
cn2m4-i128
cn2m4-i16
cn2m4-i64

Overview

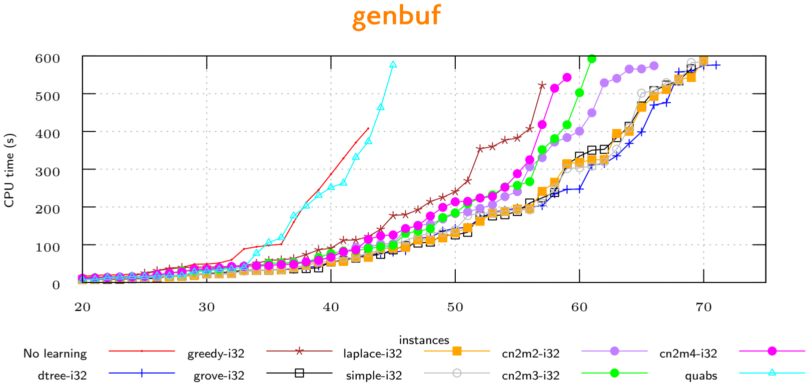

600
550
500
450
400
350
300
250
200
150
CPU time (s)
155
160
165
170
175
180
185
190
instances
No learning
rivestk1-i128
grove-i128
simple-i16
quabs
dtree-i16
greedy-i32
laplace-i32
cn2m4-i16



270 instances, consisting of QBF encodings for a number of basic building blocks of circuits

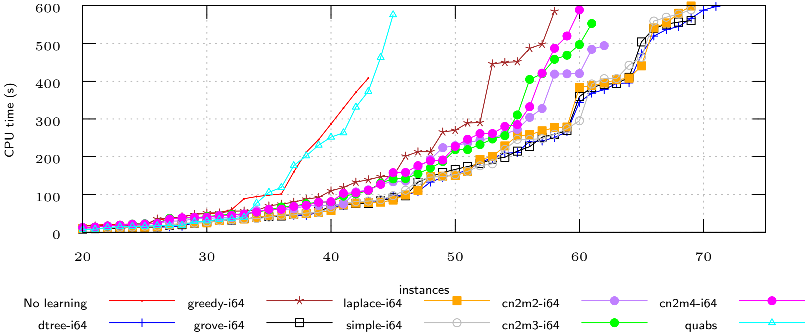


270 instances, consisting of QBF encodings for a number of basic building blocks of circuits

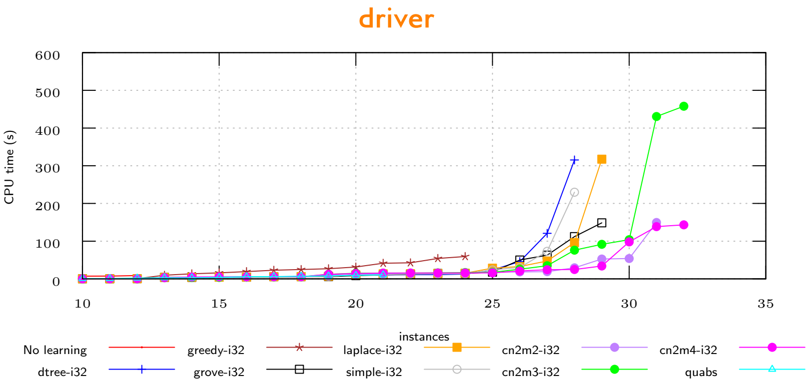


126 instances of encodings for generalized buffer specification

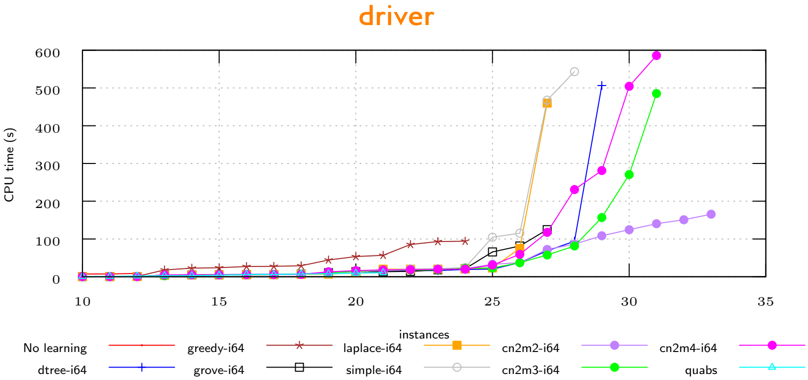
genbuf



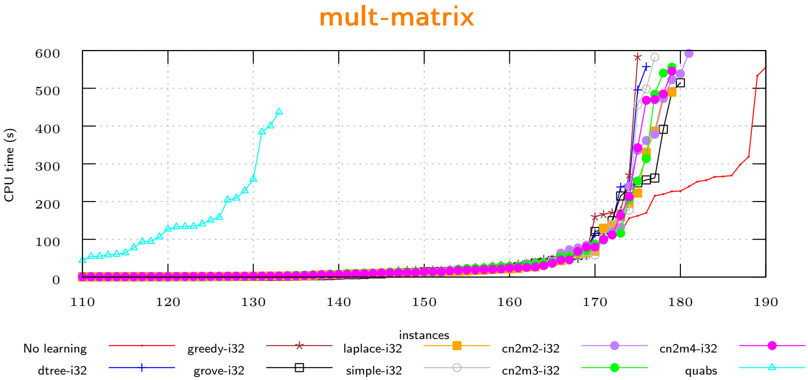
126 instances of encodings for generalized buffer specification



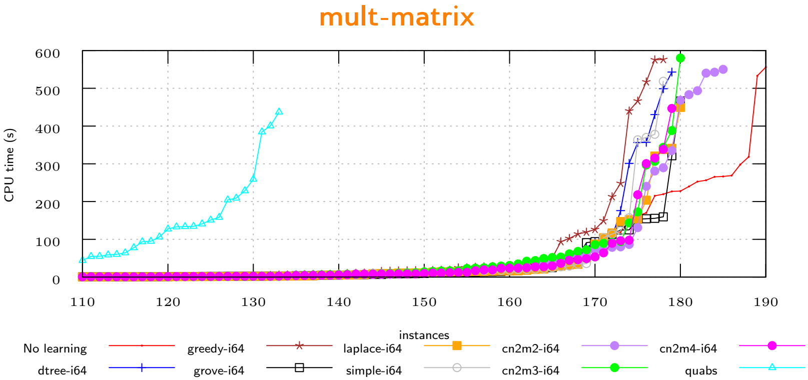
48 instances encoding specifications of a driver for a hard disk controller



48 instances encoding specifications of a driver for a hard disk controller



522 QBFs encoding the specifications for circuits that perform a single matrix multiplication, or repeated multiplication with a subset of controllable inputs



522 QBFs encoding the specifications for circuits that perform a single matrix multiplication, or repeated multiplication with a subset of controllable inputs

Conclusions and Future Work

Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver's performance.

Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver's performance.
- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.

Conclusions and Future Work

Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver's performance.
- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.
- Using beam search to select literals is feasible. More complex learning algorithms might be suitable candidates to improve QFUN.

Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver's performance.
- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.
- Using beam search to select literals is feasible. More complex learning algorithms might be suitable candidates to improve QFUN.
- For some families of QBF, QFUN with learning is particularly useful, namely the families **toy**, **genbuf**, **driver** and **cycle-sched**.

Future Work

- Learn strategies for multiple variables at once.

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm

Conclusions and Future Work

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning
- Improve the analysis of families of QBFs.

TÉCNICO LISBOA

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning
- Improve the analysis of families of QBFs.
- Parallelism / solver portfolio