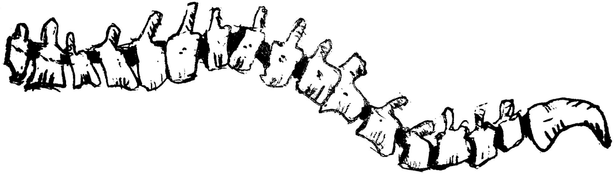
Experimental Analysis of Backbone Computation Algorithms

Mikoláš Janota[1] Inês Lynce[2] Joao Marques-Silva[3]

[1] INESC-ID, Lisbon, Portugal
[2] INESC-ID/IST, Lisbon, Portugal
[3] CASL/CSI, University College Dublin, Ireland



Backbones

- Backbones of propositional theories are literals that are true in every model.

Backbones

- Backbones of propositional theories are literals that are true in every model.

$\dots$	$\mathbf{x_j}$	$\dots$	$\mathbf{x_k}$	$\dots$	$\mathbf{x_n}$
$\dots$	x_j	$\dots$	$\neg x_k$	$\dots$	$\neg x_n$

Backbones

- Backbones of propositional theories are literals that are true in every model.

$\ldots$	$\mathbf{x_j}$	$\ldots$	$\mathbf{x_k}$	$\ldots$	$\mathbf{x_n}$
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	$\neg x_n$
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	x_n

Backbones

- Backbones of propositional theories are literals that are true in every model.

$\ldots$	$\mathbf{x_j}$	$\ldots$	$\mathbf{x_k}$	$\ldots$	$\mathbf{x_n}$
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	$\neg x_n$
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	x_n
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	$\neg x_n$
$\ldots$	x_j	$\ldots$	$\neg x_k$	$\ldots$	x_n

Backbones

- Backbones of propositional theories are literals that are true in every model.

...	x_j	...	x_k	...	x_n
...	x_j	...	$\neg x_k$	...	$\neg x_n$
...	x_j	...	$\neg x_k$	...	x_n
...	x_j	...	$\neg x_k$	...	$\neg x_n$
...	x_j	...	$\neg x_k$	...	x_n

Backbones

- Backbones of propositional theories are literals that are true in every model.

$\ldots$	$\mathbf{x_j}$	$\ldots$	$\mathbf{x_k}$	$\ldots$	$\mathbf{x_n}$
$\ldots$	$\mathbf{x_j}$	$\ldots$	$\neg\mathbf{x_k}$	$\ldots$	$\neg x_n$
$\ldots$	$\mathbf{x_j}$	$\ldots$	$\neg\mathbf{x_k}$	$\ldots$	x_n
$\ldots$	$\mathbf{x_j}$	$\ldots$	$\neg\mathbf{x_k}$	$\ldots$	$\neg x_n$
$\ldots$	$\mathbf{x_j}$	$\ldots$	$\neg\mathbf{x_k}$	$\ldots$	x_n

$\phi \models x_j$ $\qquad$ $\phi \models \neg x_k$

Motivation

- backbones tell us more about the formula, e.g.
  - upper bound for number of models

$$2^{n-k}, \text{ where } n \text{ \#variables and } k \text{ \#backbones}$$

Motivation

- backbones tell us more about the formula, e.g.
  - upper bound for number of models

$$2^{n-k}, \text{ where } n \ \#\text{variables and } k \ \#\text{backbones}$$

  - product configuration

$$\left.\begin{array}{l}\text{gas-engine} \vee \text{electric-engine} \\ \text{electric-engine} \Rightarrow \text{automatic} \\ \neg\text{automatic} \vee \neg\text{manual} \\ \\ \end{array}\right\}$$

Motivation

- backbones tell us more about the formula, e.g.
  - upper bound for number of models

    2^{n-k}, where n #variables and k #backbones

  - product configuration

$$\left.\begin{array}{l}\text{gas-engine} \vee \text{electric-engine}\\ \text{electric-engine} \Rightarrow \text{automatic}\\ \neg\text{automatic} \vee \neg\text{manual}\\ \text{electric-engine}\end{array}\right\}$$

Motivation

- backbones tell us more about the formula, e.g.
  - upper bound for number of models

$$2^{n-k}, \text{ where } n \text{ \#variables and } k \text{ \#backbones}$$

  - product configuration

$$\left.\begin{array}{l} \text{gas-engine} \lor \text{electric-engine} \\ \text{electric-engine} \Rightarrow \text{automatic} \\ \neg\text{automatic} \lor \neg\text{manual} \\ \text{electric-engine} \end{array}\right\} \boxed{\text{automatic}}, \boxed{\neg\text{manual}}$$

Motivation

- Can we compute backbones for large instances?
- How many backbone literals do real-world instances have?

Armory

- We use a satisfiability (SAT) solver as a blackbox

Armory

- We use a satisfiability (SAT) solver as a blackbox

$$\mathtt{SAT}(x \vee y) = (\mathsf{true}, \{x, \neg y\})$$

Armory

- We use a satisfiability (SAT) solver as a blackbox

$$\mathtt{SAT}(x \vee y) = (\text{true}, \{x, \neg y\})$$

$$\mathtt{SAT}(x \wedge \neg x) = (\text{false}, -)$$

Model Enumeration

Input : CNF formula φ
Output: Backbone of φ, ν_R

$\nu_R \leftarrow \{\neg x, x \mid x \in X\}$ `// initial backbone estimate`

repeat
- $(\text{outc}, \nu) \leftarrow \texttt{SAT}(\varphi)$ `// SAT solver call`
- **if** outc $=$ false **then**
  - **return** ν_R `// terminate if unsatisfiable`
- $\nu_R \leftarrow \nu_R \cap \nu$ `// update backbone estimate`
- $\omega_B \leftarrow \texttt{BlockClause}(\nu)$ `// block model`
- $\varphi \leftarrow \varphi \cup \omega_B$

until $\nu_R = \emptyset$
return $\emptyset$

Iterative SAT Testing

- Can we decide whether l is a backbone using a SAT solver?

Iterative SAT Testing

- Can we decide whether l is a backbone using a SAT solver?

$$\boxed{\phi \models l}$$

Iterative SAT Testing

- Can we decide whether l is a backbone using a SAT solver?

$$\phi \models l \quad \text{iff} \quad \text{UNSAT}(\phi \land \neg l)$$

Iterative SAT Testing

- Can we decide whether l is a backbone using a SAT solver?

$$\boxed{\phi \models l} \quad \text{iff} \quad \boxed{\text{UNSAT}(\phi \land \neg l)}$$

$$\boxed{\phi \models x}$$

Iterative SAT Testing

- Can we decide whether l is a backbone using a SAT solver?

$$\phi \models l \quad \text{iff} \quad \text{UNSAT}(\phi \wedge \neg l)$$

$$\phi \models x \quad \text{iff} \quad \text{UNSAT}(\phi \wedge \neg x)$$

Iterative SAT Testing

Input : CNF formula φ, with variables X
Output: Backbone of φ, ν_R

$\nu_R \leftarrow \emptyset$
foreach $l \in \{\neg x, x \mid x \in X\}$ **do**
  $(\text{outc}, \nu) \leftarrow \texttt{SAT}(\varphi \cup \{\neg l\})$
  if outc $=$ false **then**
    $\nu_R \leftarrow \nu_R \cup \{l\}$ // l is backbone
    $\varphi \leftarrow \varphi \cup \{l\}$
return ν_R

Iterative SAT Testing

Input : CNF formula φ, with variables X
Output: Backbone of φ, ν_R

$\nu_R \leftarrow \emptyset$
foreach $l \in \{\neg x, x \,|\, x \in X\}$ **do**
    $(\text{outc}, \nu) \leftarrow \texttt{SAT}(\varphi \cup \{\neg l\})$
    if outc $=$ false **then**
        $\nu_R \leftarrow \nu_R \cup \{l\}$ `// l is backbone`
        $\varphi \leftarrow \varphi \cup \{l\}$
return ν_R

- SAT is called twice per variable

Observation

- if ν is a model of ϕ and $l \in \nu$ *then* $\neg l$ is not a backbone

Observation

- if ν is a model of ϕ and $l \in \nu$ *then* $\neg l$ is not a backbone

$\dots$	$\mathbf{x_i}$	$\dots$
$\dots$	$\neg x_i$	$\dots$
$\vdots$	$\vdots$	$\vdots$

Observation

- if ν is a model of ϕ and $l \in \nu$ *then* $\neg l$ is not a backbone

$\dots$	$\mathbf{x_i}$	$\dots$
$\dots$	$\neg x_i$	$\dots$
$\vdots$	$\vdots$	$\vdots$

$\phi \not\models x_i$

Observation

- if ν is a model of ϕ and $\boxed{l \in \nu}$ *then* $\boxed{\neg l}$ is not a backbone

$$\begin{array}{|c|c|c|}
\dots & \mathbf{x_i} & \dots \\
\hline
\dots & \boxed{\neg x_i} & \dots \\
\vdots & \vdots & \vdots
\end{array} \quad \boxed{\phi \nvDash x_i}$$

- **OR:** if $l \notin \nu$, for some model ν, then l is not a backbone

Improving Iterative Testing

Input : CNF formula φ, with variables X
Output: Backbone of φ, ν_R

```
Λ ← {x, ¬x | x ∈ X}                    // candidates for backbone
ν_R ← ∅                                // initial backbone estimate
foreach l ∈ Λ do
    (outc, ν) ← SAT(φ ∪ {¬l})
    if outc = false then
        ν_R ← ν_R ∪ {l}                // Backbone identified
        φ ← φ ∪ {l}
    else
        Λ ← Λ ∩ ν
return ν_R
```

Characteristics

- model enumeration computes backbone from the upper bound (at the beginning everything can be a backbone)

Characteristics

- model enumeration computes backbone from the upper bound (at the beginning everything can be a backbone)
- iterative testing goes from the lower bound (at the beginning nothing is a backbone)

Characteristics

- model enumeration computes backbone from the upper bound (at the beginning everything can be a backbone)
- iterative testing goes from the lower bound (at the beginning nothing is a backbone)
- can we have a smarter upper bound algorithm?

Characteristics

- model enumeration computes backbone from the upper bound (at the beginning everything can be a backbone)
- iterative testing goes from the lower bound (at the beginning nothing is a backbone)
- can we have a smarter upper bound algorithm?

idea

- look only for those models that show that something that still can be a backbone, is not a backbone

Upper Bound Algorithm

Input : CNF formula φ, with variables X
Output: Backbone of φ, ν_R

```
(outc, ν_R) ← SAT(φ)                          // initial backbone estimate
if outc = false then return ∅                  // unsatisfiable case
while ν_R ≠ ∅ do
    (outc, ν) ← SAT(φ ∧ ⋁_{l∈ν_R} ¬l)
    if outc = false then
        return ν_R                  // estimate contains only backbones
    else
        ν_R ← ν_R ∩ ν
return ν_R
```

Characteristics

- the estimate will eventually contain only backbones, which will need to be proven in the last call

Characteristics

- the estimate will eventually contain only backbones, which will need to be proven in the last call
- the SAT calls are getting gradually harder

Characteristics

- the estimate will eventually contain only backbones, which will need to be proven in the last call
- the SAT calls are getting gradually harder
- can we join the two approaches?

Characteristics

- the estimate will eventually contain only backbones, which will need to be proven in the last call
- the SAT calls are getting gradually harder
- can we join the two approaches?

idea

- split the estimate into chunks of size K
- test only one chunk at a time

Upper Bound Chunking Algorithm

Input : CNF formula φ, with variables X. $K \in \mathbb{N}^+$
Output: Backbone of φ, ν_R

```
(outc, Λ) ← SAT(φ)                              // initial backbone estimate
if outc = false then return ∅                      // unsatisfiable case
ν_R ← ∅                                          // initial backbone estimate
while Λ ≠ ∅ do
    k ← min(|ν_R|, K)
    Γ ← pick k literals from Λ
    (outc, ν) ← SAT(φ ∧ ⋁_{l∈Γ} ¬l)
    if outc = false then
        ν_R ← ν_R ∪ Γ                        // chunk contains only backbones
        φ ← φ ∧ ⋀_{l∈Γ} l
    else
        Λ ← Λ ∩ ν                       // something in the chunk not backbone
return ν_R
```

Characteristics

- K backbones can be shown in one call thus reducing the number of calls

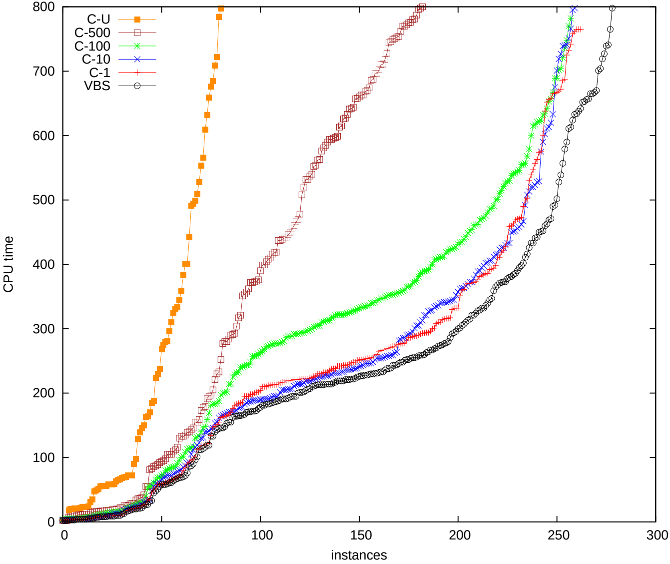
Characteristics

- K backbones can be shown in one call thus reducing the number of calls
- $K = 1$ is the iterative algorithm

Characteristics

- K backbones can be shown in one call thus reducing the number of calls
- $K = 1$ is the iterative algorithm
- $K = |X|$ is the upper-bound algorithm

Results



C-U
C-500
C-100
C-10
C-1
VBS
CPU time
instances
0
100
200
300
400
500
600
700
800
0
50
100
150
200
250
300

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox
- iterative algorithm (one call per variable)

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox
- iterative algorithm (one call per variable)
- upper bound (backbone proven in the last call)

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox
- iterative algorithm (one call per variable)
- upper bound (backbone proven in the last call)
- generalized by chunking algorithm
  (K literals can be shown as a backbone in one call)

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox
- iterative algorithm (one call per variable)
- upper bound (backbone proven in the last call)
- generalized by chunking algorithm
  (K literals can be shown as a backbone in one call)
- chunking overall does not outperform the iterative algorithm but helps in some cases

Summary and Future Work

- analysis of algorithms for computing backbones that use a SAT solver as a blackbox
- iterative algorithm (one call per variable)
- upper bound (backbone proven in the last call)
- generalized by chunking algorithm
  (K literals can be shown as a backbone in one call)
- chunking overall does not outperform the iterative algorithm but helps in some cases
- adaptive algorithms for chunks