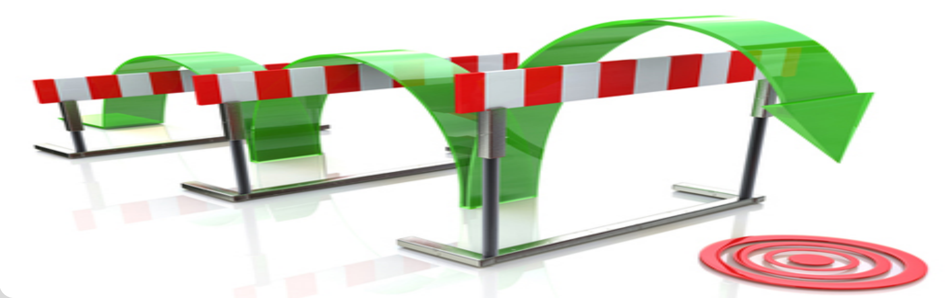
Accelerating SAT Based Planning with Incremental SAT Solving

ICAPS 2017, Pittsburgh, Pennsylvania, USA

Stephan Gocht and Tomáš Balyo | June 22, 2017

INSTITUTE OF THEORETICAL INFORMATICS, ALGORITHMICS II



Planning

classical planning is considered:

- discrete and finite state space
- find sequence of actions from initial state to goal state
- actions are deterministic
- no action cost

Definitions

CNF Formula

- A Boolean variable has two values: True and False
- A literal is Boolean variables or its negation
- A clause is a disjunction (or) of literals
- A CNF formula is a conjunction (and) of clauses

$$F = (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_3} \vee x_1) \wedge (x_1) \wedge (\overline{x_2} \vee \overline{x_4})$$

Satisfiability

- A CNF formula is satisfiable if it has a satisfying assignment.
- The problem of satisfiability (SAT) is to determine whether a given CNF formula is satisfiable

Planning as SAT:

Encode to SAT that a plan of length *i* exists (Kautz and Selman 1992)

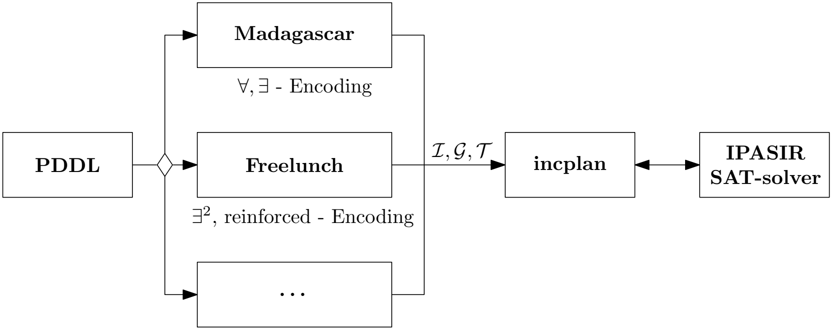
$$F_i := \mathcal{I}(t_0) \wedge \left(\bigwedge_{k=0}^{i-1} \mathcal{T}(t_k, t_{k+1}) \right) \wedge \mathcal{G}(t_i)$$

- $\mathcal{I}$ initial clauses
- $\mathcal{G}$ goal clauses
- $\mathcal{T}$ transition clauses

$\Rightarrow$ generated using arbitrary modern aproaches to SAT based planning:

- exists-step ($\exists$): Rintanen, Heljanko, and Niemelä 2006
- reinforced (reinf.): Balyo, Barták, and Trunda 2015
- ...

Toolchain


Madagascar
$\forall, \exists$ - Encoding
PDDL
Freelunch
$\exists^2$, reinforced - Encoding
...
$\mathcal{I}, \mathcal{G}, \mathcal{T}$
incplan
IPASIR
SAT-solver

Incremental SAT Solving

Interface: Eén and Sörensson 2003

- $add(C)$, for a clause C
- $solve(assumptions = A)$, for a set of literals A

Advantages

- no input overhead for adding the same clauses over and over again
- reuse from previous solve:
  - internally learned clauses
  - importance of clauses and variables
  - good assignments (phase saving)
  - ...

Incremental Encoding

Nonincremental:

$$\boxed{t_0 = \mathcal{I}}\ \}\ \mathcal{G}(t_0)$$

Single Ended Incremental:

$$\boxed{t_0 = \mathcal{I}}\ \}\ \mathcal{G}(t_0)$$

Incremental Encoding

Nonincremental:

Single Ended Incremental:

$$\boxed{t_0 = \mathcal{I}}$$

Incremental Encoding

Nonincremental:

$$\mathcal{T}(t_0, t_1) \begin{cases} \boxed{\quad t_1 \quad} \; \} \; \mathcal{G}(t_1) \\ \boxed{t_0 = \mathcal{I}} \end{cases}$$

Single Ended Incremental:

$$\mathcal{T}(t_0, t_1) \begin{cases} \boxed{\quad t_1 \quad} \; \} \; \mathcal{G}(t_1) \\ \boxed{t_0 = \mathcal{I}} \end{cases}$$

Incremental Encoding

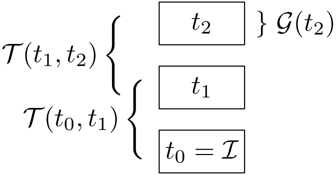
Nonincremental:

Single Ended Incremental:

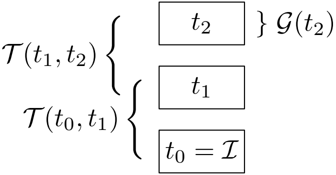
$$\mathcal{T}(t_0, t_1) \begin{cases} \boxed{t_1} \\ \boxed{t_0 = \mathcal{I}} \end{cases}$$

Incremental Encoding

Nonincremental:



Single Ended Incremental:



Incremental Encoding

Nonincremental:

Single Ended Incremental:

$$\boxed{t_2}$$

$$\mathcal{T}(t_1, t_2)$$

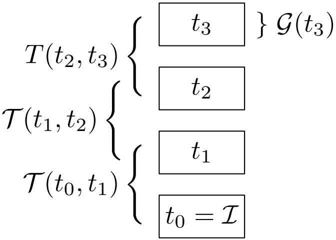
$$\boxed{t_1}$$

$$\mathcal{T}(t_0, t_1)$$

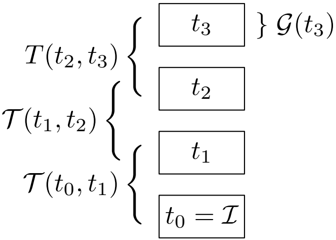
$$\boxed{t_0 = \mathcal{I}}$$

Incremental Encoding

Nonincremental:



Single Ended Incremental:



forward search

or

backward search?

forward search

or

backward search?

both!

Double Ended Encoding

$$\overbrace{\boxed{t_0 = \mathcal{I}} \qquad\qquad \boxed{t_n = \mathcal{G}}}^{L(t_0, t_n)}$$

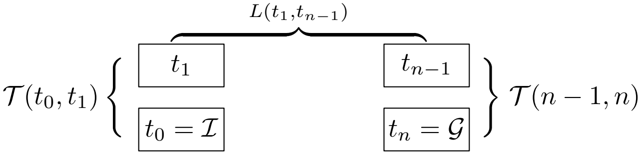
$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Double Ended Encoding

$\boxed{t_0 = \mathcal{I}}$ $\boxed{t_n = \mathcal{G}}$

$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Double Ended Encoding



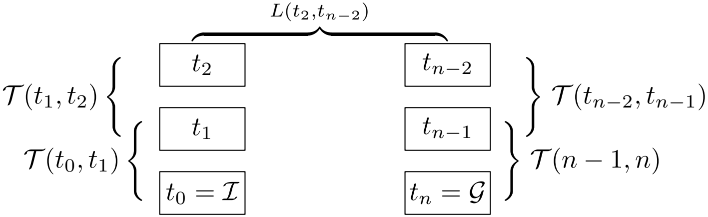
$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Double Ended Encoding

$$\mathcal{T}(t_0, t_1)\left\{\begin{array}{c}\boxed{t_1}\\[1em] \boxed{t_0 = \mathcal{I}}\end{array}\right. \qquad\qquad \left.\begin{array}{c}\boxed{t_{n-1}}\\[1em] \boxed{t_n = \mathcal{G}}\end{array}\right\}\mathcal{T}(n-1, n)$$

$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Double Ended Encoding



$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

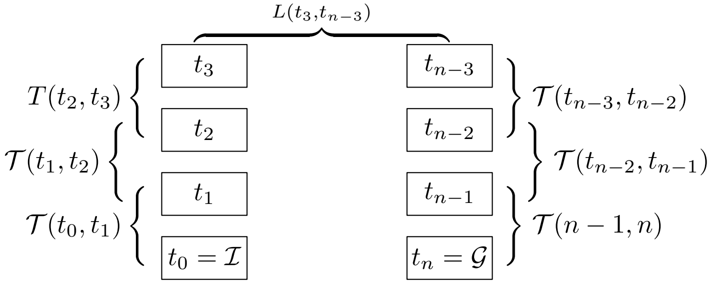
Double Ended Encoding

$$\mathcal{T}(t_1, t_2)\left\{\begin{array}{l}\boxed{t_2}\\ \boxed{t_1}\end{array}\right. \qquad \mathcal{T}(t_0, t_1)\left\{\begin{array}{l}\boxed{t_1}\\ \boxed{t_0 = \mathcal{I}}\end{array}\right.$$

$$\left.\begin{array}{l}\boxed{t_{n-2}}\\ \boxed{t_{n-1}}\end{array}\right\}\mathcal{T}(t_{n-2}, t_{n-1}) \qquad \left.\begin{array}{l}\boxed{t_{n-1}}\\ \boxed{t_n = \mathcal{G}}\end{array}\right\}\mathcal{T}(n-1, n)$$

$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Double Ended Encoding

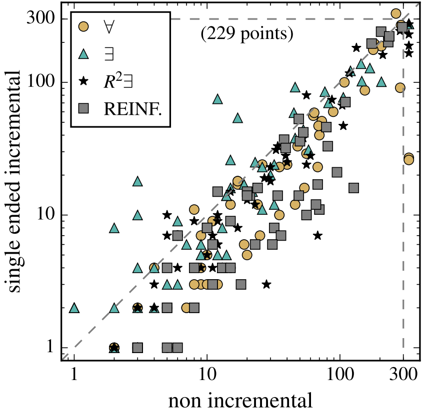


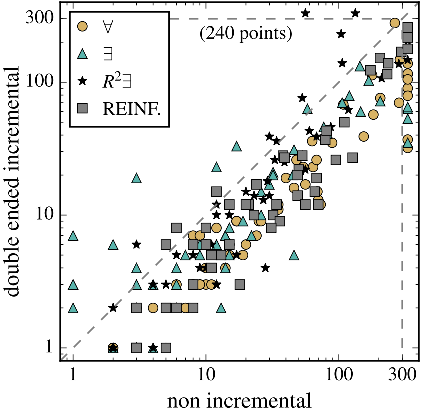
$L(t_i, t_k)$:
clauses such that all variables in t_i have the same value as in t_k and vice versa

Evaluation

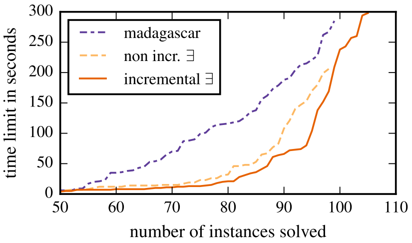
- Recent SAT solver from 2016 SAT Competition:
  - *COMiniSatPS 2Sun nopre*, Oh 2016
- Benchmarks:
  - Agile Track of the 2014 International Planning Competition (IPC) Vallati et al. 2015
- Limits:
  - timelimit: 300s
  - 1 CPU core @ 2.10GHz
  - 8 GB RAM

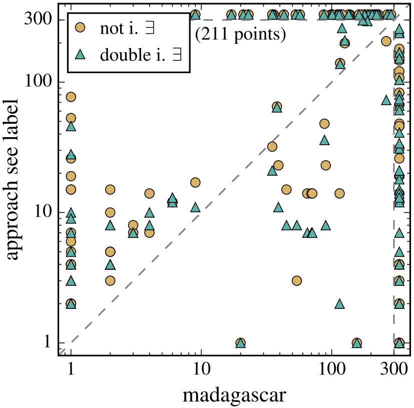
Nonincremental vs Incremental





Incremental vs Madagascar





Conclusion

Results:

- incremental SAT solving is very beneficial
- its important to keep track of SAT solver development

Future Work:

- use advanced scheduling techniques
- use planning specific SAT solver heuristics
- insights on reasons for speedup

Activation Literals

No Removal of Clauses

- removing clauses can invalidate learned clauses
- use activation literals instead

Activation Literals: Eén and Sörensson 2003

- $add(\overline{a} \vee C)$, where a is a fresh variable
- $solve(A \cup \{a\})$, to solve with clause C
- $solve(A)$, to solve without clause C