Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Potassco

Heuristic programming: Overview

1. Motivation
2. Heuristically modified ASP
3. Experimental results

Outline

1 Motivation
2 Heuristically modified ASP
3 Experimental results

Potassco

Motivation

- Observation Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information
  - domain-specific knowledge can be added for improving propagation
  - domain-specific heuristics can be used for making better choices
- Idea Incorporation of domain-specific heuristics by extending
  - input language and/or solver options for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

Motivation

- Observation Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information
  - domain-specific knowledge can be added for improving propagation
  - domain-specific heuristics can be used for making better choices
- Idea Incorporation of domain-specific heuristics by extending
  - input language and/or solver options for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

Motivation

- Observation Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information
  - domain-specific knowledge can be added
    for improving propagation
  - domain-specific heuristics can be used
    for making better choices
- Idea Incorporation of domain-specific heuristics by extending
  - input language and/or solver options
    for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

Basic CDCL decision algorithm

loop

    propagate // compute deterministic consequences

    if no conflict **then**

        if all variables assigned **then return** variable assignment

        else decide // non-deterministically assign some literal

    else

        if top-level conflict **then return** unsatisfiable

        else

            analyze // analyze conflict and add a conflict constraint

            backjump // undo assignments until conflict constraint is unit

Basic CDCL decision algorithm

loop

    propagate // compute deterministic consequences

    if no conflict **then**

        if all variables assigned **then return** variable assignment
        else decide // non-deterministically assign some literal

    else

        if top-level conflict **then return** unsatisfiable
        else

            analyze // analyze conflict and add a conflict constraint
            backjump // undo assignments until conflict constraint is unit

Inside *decide*

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

$$A^{\mathbf{T}} = \{a \in \mathcal{A} \mid \mathbf{T}a \in A\} \quad \text{and} \quad A^{\mathbf{F}} = \{a \in \mathcal{A} \mid \mathbf{F}a \in A\}$$

- Heuristic functions

$$h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$

$$A^{\mathbf{T}} = \{a \in \mathcal{A} \mid \mathbf{T}a \in A\} \text{ and } A^{\mathbf{F}} = \{a \in \mathcal{A} \mid \mathbf{F}a \in A\}$$

- Heuristic functions

$$h : \mathcal{A} \to [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$
    $$A^{\mathbf{T}} = \{a \in \mathcal{A} \mid \mathbf{T}a \in A\} \text{ and } A^{\mathbf{F}} = \{a \in \mathcal{A} \mid \mathbf{F}a \in A\}$$
- Heuristic functions

$$h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$

    $A^{\mathbf{T}} = \{a \in \mathcal{A} \mid \mathbf{T}a \in A\}$ and $A^{\mathbf{F}} = \{a \in \mathcal{A} \mid \mathbf{F}a \in A\}$
- Heuristic functions

  $$h : \mathcal{A} \to [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$$
- Algorithmic scheme

  1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
  2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
  3. $C := argmax_{a \in U} h(a)$
  4. $a := \tau(C)$
  5. $A := A \cup \{a \mapsto s(a)\}$

Outline

1 Motivation

2 Heuristically modified ASP

3 Experimental results

Heuristic language

- Heuristic directive

  $$\#\texttt{heuristic}\ a\ :\ l_1,\ldots,l_n.\ [k@p,m]$$

  where
  - a is an atom, and $l_1,\ldots,l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier

- Heuristic modifiers

  `init` for initializing the heuristic value of a with k
  `factor` for amplifying the heuristic value of a by factor k
  `level` for ranking all atoms; the rank of a is k
  `sign` for attributing the sign of k as truth value to a

- Example

```
#heuristic occurs(A,T) : action(A), time(T). [T, factor]
```

Heuristic language

- Heuristic directive

  $\#\texttt{heuristic}\ a\ :\ l_1,\ldots,l_n.\ [k@p,m]$

  where
  - a is an atom, and $l_1,\ldots,l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier

- Heuristic modifiers

`init`	for initializing the heuristic value of a with k
`factor`	for amplifying the heuristic value of a by factor k
`level`	for ranking all atoms; the rank of a is k
`sign`	for attributing the sign of k as truth value to a

- Example

```
#heuristic occurs(A,T) : action(A), time(T). [T, factor]
```

Heuristic language

- Heuristic directive

  `#heuristic` a : $l_1, \dots, l_n$. $[k@p, m]$

  where
  - a is an atom, and $l_1, \dots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier

- Heuristic modifiers

`init`	for initializing the heuristic value of a with k
`factor`	for amplifying the heuristic value of a by factor k
`level`	for ranking all atoms; the rank of a is k
`sign`	for attributing the sign of k as truth value to a
`true/false`	combine `level` and `sign`

- Example

```
#heuristic occurs(A,T) : action(A), time(T). [T, factor]
```

Heuristic language

- Heuristic directive

  `#heuristic` a : $l_1, \dots, l_n$. $[k@p, m]$

  where

  - a is an atom, and $l_1, \dots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier

- Heuristic modifiers

`init`	for initializing the heuristic value of a with k
`factor`	for amplifying the heuristic value of a by factor k
`level`	for ranking all atoms; the rank of a is k
`sign`	for attributing the sign of k as truth value to a

- Example

```
#heuristic occurs(A,T) : action(A), time(T). [T, factor]
```

Heuristic language

- Heuristic directive

  $\#\texttt{heuristic}\ a\ :\ l_1,\ldots,l_n.\ [k@p,m]$

  where
  - a is an atom, and $l_1,\ldots,l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier

- Heuristic modifiers

`init`	for initializing the heuristic value of a with k
`factor`	for amplifying the heuristic value of a by factor k
`level`	for ranking all atoms; the rank of a is k
`sign`	for attributing the sign of k as truth value to a

- Example

```
#heuristic occurs(mv,5) : action(mv), time(5). [5, factor]
```

Simple STRIPS planning

```
time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Simple STRIPS planning

```
time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic occurs(A,T) : action(A), time(T). [2, factor]
```

Simple STRIPS planning

```
time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic occurs(A,T) : action(A), time(T). [1, level]
```

Simple STRIPS planning

```
time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic occurs(A,T) : action(A), time(T). [T, factor]
```

Simple STRIPS planning

```
time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic holds(F,T-1) :     holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T)  [t-T+1, false]
                          fluent(F), time(T).
```

Heuristic options

- Alternative for specifying structure-oriented heuristics in *clasp*

```
--dom-mod=<arg> : Default modification for
                  domain heuristic
   <arg>: <mod>[,<pick>]
     <mod>   : Modifier
               {1=level|2=pos|3=true|4=neg|
                5=false|6=init|7=factor}
     <pick> : Apply <mod> to
               {0=all|1=scc|2=hcc|4=disj|
                8=min|16=show} atoms
```

Engage heuristic modifications (in both settings!)

```
--heuristic=Domain
```

Heuristic options

- Alternative for specifying structure-oriented heuristics in *clasp*

```
--dom-mod=<arg> : Default modification for
                  domain heuristic
   <arg>: <mod>[,<pick>]
     <mod>   : Modifier
               {1=level|2=pos|3=true|4=neg|
                5=false|6=init|7=factor}
     <pick> : Apply <mod> to
               {0=all|1=scc|2=hcc|4=disj|
                8=min|16=show} atoms
```

- Engage heuristic modifications (in both settings!)

```
--heuristic=Domain
```

Heuristic options

- Alternative for specifying structure-oriented heuristics in *clasp*

```
--dom-mod=<arg> : Default modification for
                  domain heuristic
   <arg>: <mod>[,<pick>]
     <mod>  : Modifier
              {1=level|2=pos|3=true|4=neg|
               5=false|6=init|7=factor}
     <pick> : Apply <mod> to
              {0=all|1=scc|2=hcc|4=disj|
               8=min|16=show} atoms
```

- Engage heuristic modifications (in both settings!)

```
--heuristic=Domain
```

Inclusion-minimal stable models

- Consider a logic program containing a mimimize statement of form
  - `#minimize`$\{a_1, \dots, a_n\}$
- Computing one inclusion-minimal stable model can be done either via
  - `#heuristic` a_i `[1,false].` for $i = 1, \dots, n$, or
  - `--dom-mod=5,16`
- Computing all inclusion-minimal stable model can be done
  - by adding `--enum-mod=domRec` to the two options

Potassco

Inclusion-minimal stable models

- Consider a logic program containing a mimimize statement of form
  - `#minimize`$\{a_1, \ldots, a_n\}$
- Computing one inclusion-minimal stable model can be done either via
  - `#heuristic` a_i `[1,false].` for $i = 1, \ldots, n$, or
  - `--dom-mod=5,16`
- Computing all inclusion-minimal stable model can be done
  - by adding `--enum-mod=domRec` to the two options

Inclusion-minimal stable models

- Consider a logic program containing a mimimize statement of form
  - `#minimize`$\{a_1, \dots, a_n\}$
- Computing one inclusion-minimal stable model can be done either via
  - `#heuristic` a_i `[1,false].` for $i = 1, \dots, n$, or
  - `--dom-mod=5,16`
- Computing all inclusion-minimal stable model can be done
  - by adding `--enum-mod=domRec` to the two options

Heuristic modifications to functions h and s

- $\nu_{a,m}(A)$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and `factor`

$$d_0(a) = \quad \nu_{a,\texttt{init}}(A_0) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu_{a,\texttt{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \mathbf{T} & \text{if } \nu_{a,\texttt{sign}}(A_i) > 0 \\ \mathbf{F} & \text{if } \nu_{a,\texttt{sign}}(A_i) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\quad \ell_{A_i}(\mathcal{A}') = argmax_{a \in \mathcal{A}'} \nu_{a,\texttt{level}}(A_i) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu_{a,m}(A)$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and

$$d_0(a) = \quad \nu_{a,\mathtt{init}}(A_0) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu_{a,\mathtt{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\mathtt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \mathbf{T} & \text{if } \nu_{a,\mathtt{sign}}(A_i) > 0 \\ \mathbf{F} & \text{if } \nu_{a,\mathtt{sign}}(A_i) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\quad \ell_{A_i}(\mathcal{A}') = argmax_{a \in \mathcal{A}'} \nu_{a,\mathtt{level}}(A_i) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu_{a,m}(A)$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and `factor`

$$d_0(a) = \nu_{a,\mathtt{init}}(A_0) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu_{a,\mathtt{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\mathtt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \mathbf{T} & \text{if } \nu_{a,\mathtt{sign}}(A_i) > 0 \\ \mathbf{F} & \text{if } \nu_{a,\mathtt{sign}}(A_i) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\quad \ell_{A_i}(\mathcal{A}') = argmax_{a \in \mathcal{A}'} \nu_{a,\mathtt{level}}(A_i) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Inside *decide*, heuristically modified

0 $h(a) := d(a)$ for each $a \in \mathcal{A}$

1 $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$

2 $U := \ell_A(\mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}}))$

3 $C := argmax_{a \in U} d(a)$

4 $a := \tau(C)$

5 $A := A \cup \{a \mapsto t(a)\}$

Inside *decide*, heuristically modified

0. $h(a) := d(a)$ for each $a \in \mathcal{A}$
1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \ell_A(\mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}}))$
3. $C := argmax_{a \in U} d(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto t(a)\}$

Potassco

Inside *decide*, heuristically modified

0 $h(a) := d(a)$ for each $a \in \mathcal{A}$

1 $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$

2 $U := \ell_A(\mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}}))$

3 $C := argmax_{a \in U} d(a)$

4 $a := \tau(C)$

5 $A := A \cup \{a \mapsto t(a)\}$

Outline

1 Motivation

2 Heuristically modified ASP

3 Experimental results

Abductive problems with optimization

Setting		*Diagnosis*	*Expansion*	*Repair (H)*	*Repair (S)*
base configuration		111.1s (115)	161.5s (100)	101.3s (113)	33.3s (27)
`sign,-1`		324.5s (407)	7.6s (3)	8.4s (5)	3.1s (0)
`sign,-1`	`factor,2`	310.1s (387)	7.4s (2)	3.5s (0)	3.2s (1)
`sign,-1`	`factor,8`	305.9s (376)	7.7s (2)	3.1s (0)	2.9s (0)
`sign,-1`	`level,1`	76.1s (83)	6.6s (2)	0.8s (0)	2.2s (1)
	`level,1`	77.3s (86)	12.9s (5)	3.4s (0)	2.1s (0)

(abducibles subject to optimization via `#minimize` statements)

Abductive problems with optimization

Setting		*Diagnosis*	*Expansion*	*Repair (H)*	*Repair (S)*
base configuration		111.1s (115)	161.5s (100)	101.3s (113)	33.3s (27)
`sign,-1`		324.5s (407)	7.6s (3)	8.4s (5)	3.1s (0)
`sign,-1`	`factor,2`	310.1s (387)	7.4s (2)	3.5s (0)	3.2s (1)
`sign,-1`	`factor,8`	305.9s (376)	7.7s (2)	3.1s (0)	2.9s (0)
`sign,-1`	`level,1`	76.1s (83)	6.6s (2)	0.8s (0)	2.2s (1)
	`level,1`	77.3s (86)	12.9s (5)	3.4s (0)	2.1s (0)

(abducibles subject to optimization via `#minimize` statements)

Planning benchmarks

```
#heuristic holds(F,T-1) :     holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T), fluent(F),time(T).
                                          [t-T+1, false]
```

Problem	*base configuration*		#heuristic		*base config.* (SAT)		#heu. (SAT)	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Planning benchmarks

```
#heuristic holds(F,T-1) :     holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T), fluent(F),time(T).
                                          [t-T+1, false]
```

Problem	*base configuration*		#heuristic		*base config.* (SAT)		#heu. (SAT)	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Planning benchmarks

```
#heuristic holds(F,T-1) :     holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T), fluent(F),time(T).
                                          [t-T+1, false]
```

Problem	*base configuration*		`#heuristic`		*base config.* (SAT)		`#heu.` (SAT)	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

[1] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[2] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[3] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[4] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[5] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.

In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[6] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.
In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[7] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.

IOS Press, 2009.

[9] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[10] G. Brewka, I. Niemelä, and M. Truszczyński.
Answer set optimization.
In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[11] K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[12] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

[13] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[14] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[15] M. Davis and H. Putnam.
A computing procedure for quantification theory.
Journal of the ACM, 7:201–215, 1960.

[16] E. Di Rosa, E. Giunchiglia, and M. Maratea.
Solving satisfiability problems with preferences.
Constraints, 15(4):485–515, 2010.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.

Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.
In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.
Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.

In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

[26] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub.
Abstract Gringo.
Theory and Practice of Logic Programming, 15(4-5):449–463, 2015.
Available at `http://arxiv.org/abs/1507.06576`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.
University of Potsdam, second edition edition, 2015.

[28] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[29] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[30] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.

On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [49], pages 250–264.

[31] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [3], pages 260–265.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [3], pages 136–148.

[34] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [74], pages 386–392.

[35] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[36] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[37] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*

(CPAIOR'09), volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[38] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [49], pages 235–249.

[39] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[40] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[41] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[42] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[43] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[44] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.

In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[45] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[46] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.
Journal of Automated Reasoning, 36(4):345–377, 2006.

[47] K. Gödel.
Zum intuitionistischen Aussagenkalkül.
In *Anzeiger der Akademie der Wissenschaften in Wien*, page 65–66. 1932.

[48] A. Heyting.

Die formalen Regeln der intuitionistischen Logik.
In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. 1930.
Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[49] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[50] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [74], pages 2318–2323.

[51] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[52] J. Lee.

A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[53] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[54] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[55] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[56] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?

ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[57] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

[58] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[59] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[60] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

[61] J. Marques-Silva and K. Sakallah.

GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[62] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.
In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[63] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[64] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

[65] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[66] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[67] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[68] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.

In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[69] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[70] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.
Theory and Practice of Logic Programming, 6(5):559–608, 2006.

[72] T. Syrjänen.
Lparse 1.0 user's manual, 2001.

[73] A. Van Gelder, K. Ross, and J. Schlipf.

The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[74] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[75] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.
In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.