

Domain-specific Heuristics in Answer Set Programming

Martin Gebser, Benjamin Kaufmann, Ramon Otero, Javier Romero,
Torsten Schaub, and Philipp Wanko

University of Potsdam and University of Corunna

Outline

Outline

Motivation

- **Answer Set Programming (ASP)**
 - General purpose approach to declarative problem solving
 - Combination of a rich yet simple modeling language with highly performant solving capacities
- 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
- **Proposal** A declarative framework for incorporating domain-specific heuristics into ASP by extending its
 - input language for expressing domain-specific heuristics
 - solving capacities for integrating domain-specific heuristics

Motivation

- **Answer Set Programming (ASP)**
 - General purpose approach to **declarative problem solving**
 - Combination of a rich yet simple **modeling language** with highly performant **solving capacities**
- 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
- **Proposal** A declarative framework for incorporating domain-specific heuristics into ASP by extending its
 - input language for expressing domain-specific heuristics
 - solving capacities for integrating domain-specific heuristics

Motivation

- **Answer Set Programming (ASP)**
 - General purpose approach to declarative problem solving
 - Combination of a rich yet simple modeling language with highly performant solving capacities
- 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
- **Proposal** A declarative framework for incorporating domain-specific heuristics into ASP by extending its
 - input language for expressing domain-specific heuristics
 - solving capacities for integrating domain-specific heuristics

Motivation

- **Answer Set Programming (ASP)**
 - General purpose approach to declarative problem solving
 - Combination of a rich yet simple modeling language with highly performant solving capacities
- 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
- Proposal A declarative framework for **incorporating domain-specific heuristics into ASP** by extending its
 - input language 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, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

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

■ 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 $C := \operatorname{argmax}_{a \in U} h(a)$
- 4 $a := \tau(C)$
- 5 $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

■ Basic concepts

■ Atoms, $\mathcal{A}$

■ Assignments, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

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

■ 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 $C := \operatorname{argmax}_{a \in U} h(a)$
- 4 $a := \tau(C)$
- 5 $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

■ Basic concepts

■ Atoms, $\mathcal{A}$

■ Assignments, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

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

■ 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 $C := \operatorname{argmax}_{a \in U} h(a)$
- 4 $a := \tau(C)$
- 5 $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

■ Basic concepts

■ Atoms, $\mathcal{A}$

■ Assignments, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

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

■ 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 $C := \operatorname{argmax}_{a \in U} h(a)$
- 4 $a := \tau(C)$
- 5 $A := A \cup \{a \mapsto s(a)\}$

Outline

Heuristic language elements

■ Heuristic predicate `_heuristic`

■ Heuristic modifiers (atom, a , and integer, v)

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

■ Heuristic atoms

```
_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a , and integer, v)
 - `init` for initializing the heuristic value of a with v
 - `factor` for amplifying the heuristic value of a by factor v
 - `level` for ranking all atoms; the rank of a is v
 - `sign` for attributing the sign of v as truth value to a
- Heuristic atoms
 - `_heuristic(occurs(A,T),factor,T) :- action(A), time(T).`

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a , and integer, v)
 - `init` for initializing the heuristic value of a with v
 - `factor` for amplifying the heuristic value of a by factor v
 - `level` for ranking all atoms; the rank of a is v
 - `sign` for attributing the sign of v as truth value to a
- Heuristic atoms
 - `_heuristic(occurs(A,T),factor,T) :- action(A), time(T).`

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a , and integer, v)
 - `init` for initializing the heuristic value of a with v
 - `factor` for amplifying the heuristic value of a by factor v
 - `level` for ranking all atoms; the rank of a is v
 - `sign` for attributing the sign of v as truth value to a
- Heuristic atoms


```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (`atom`, a , and integer, v)
 - `init` for initializing the heuristic value of a with v
 - `factor` for amplifying the heuristic value of a by factor v
 - `level` for ranking all atoms; the rank of a is v
 - `sign` for attributing the sign of v as truth value to a
- Heuristic atoms


```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a , and integer, v)
 - `init` for initializing the heuristic value of a with v
 - `factor` for amplifying the heuristic value of a by factor v
 - `level` for ranking all atoms; the rank of a is v
 - `sign` for attributing the sign of v as truth value to a
- Heuristic atoms
 - `_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).`

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)
 - `init` for initializing the heuristic value of *a* with *v*
 - `factor` for amplifying the heuristic value of *a* by factor *v*
 - `level` for ranking all atoms; the rank of *a* is *v*
 - `sign` for attributing the sign of *v* as truth value to *a*
- Heuristic atoms


```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Simple STRIPS planner

```
time(1..t).

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

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

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

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

Simple STRIPS planner

```
time(1..t).

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

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

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

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

_heuristic(occurs(A,T),factor,2) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

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

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

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

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

_heuristic(occurs(A,T),level,1) :- action(A), time(T).
```


Simple STRIPS planner

```
time(1..t).

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

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

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

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

_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

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

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

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

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

_heuristic(A,level,V) :- _heuristic(A,true, V).
_heuristic(A,sign, 1) :- _heuristic(A,true, V).
```

Simple STRIPS planner

```
time(1..t).

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

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

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

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

_heuristic(A,level,V) :- _heuristic(A,false,V).
_heuristic(A,sign,-1) :- _heuristic(A,false,V).
```

Simple STRIPS planner

```

time(1..t).

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

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

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

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

_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
    fluent(F), time(T), not holds(F,T).

```

Heuristic modifications to functions h and s

■ $\nu(V_{a,m}(A))$ — “value for modifier m on atom a wrt assignment A ”

■ `init` and

$$d_0(a) = \nu(V_{a,\text{init}}(A_0)) + h_0(a)$$

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

■ `sign`

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

■ `level` $\ell_{A_i}(\mathcal{A}') = \operatorname{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\text{level}}(A_i)) \quad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

■ $\nu(V_{a,m}(A))$ — “value for modifier m on atom a wrt assignment A ”

■ `init` and

$$d_0(a) = \nu(V_{a,\text{init}}(A_0)) + h_0(a)$$

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

■ `sign`

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

■ `level` $\ell_{A_i}(\mathcal{A}') = \operatorname{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\text{level}}(A_i)) \quad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

■ $\nu(V_{a,m}(A))$ — “value for modifier m on atom a wrt assignment A ”

■ `init` and `factor`

$$d_0(a) = \nu(V_{a,\text{init}}(A_0)) + h_0(a)$$

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

■ `sign`

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

■ `level` $\ell_{A_i}(\mathcal{A}') = \operatorname{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\text{level}}(A_i)) \quad \mathcal{A}' \subseteq \mathcal{A}$

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 $U := \ell_A(\mathcal{A} \setminus (A^T \cup A^F))$
- 3 $C := \operatorname{argmax}_{a \in U} d(a)$
- 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 $U := \ell_A(\mathcal{A} \setminus (A^T \cup A^F))$
- 3 $C := \operatorname{argmax}_{a \in U} d(a)$
- 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 $U := \ell_{\mathcal{A}}(\mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}}))$
- 3 $C := \operatorname{argmax}_{a \in U} d(a)$
- 4 $a := \tau(C)$
- 5 $A := A \cup \{a \mapsto t(a)\}$

Outline

Abductive problems with optimization

Setting	<i>Diagnosis</i>	<i>Expansion</i>	<i>Repair (H)</i>	<i>Repair (S)</i>
<i>base configuration</i>	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	<i>Diagnosis</i>	<i>Expansion</i>	<i>Repair (H)</i>	<i>Repair (S)</i>
<i>base configuration</i>	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 Competition Benchmarks

```

_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
    fluent(F), time(T), not holds(F,T).

```

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

Planning Competition Benchmarks

```

_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
    fluent(F), time(T), not holds(F,T).

```

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

Planning Competition Benchmarks

```

_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
    fluent(F), time(T), not holds(F,T).

```

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

Outline

Summary

- A declarative framework for incorporating domain-specific heuristics into ASP
 - seamless integration into ASP's input language
 - general and flexible tool for expressing domain-specific heuristics
 - new possibilities of applying, experimenting, and studying domain-specific heuristics in a uniform setting
- <http://potassco.sourceforge.net/labs.html#hclasp>