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, $A : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$

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

- Heuristic functions

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

- 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, $A : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$
    $$A^{\mathbf{T}} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{T} \in A\} \text{ and } A^{\mathbf{F}} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{F} \in A\}$$
- 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, $A : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$

    $A^{\mathbf{T}} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{T} \in A\}$ and $A^{\mathbf{F}} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{F} \in A\}$
- 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. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Inside *decide*

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

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

- 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. $C := argmax_{a \in U} h(a)$
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 `factor`

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

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

- `sign`

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

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

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`

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

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

- `sign`

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

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

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`

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

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

- `sign`

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

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

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

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

Abductive problems with optimization

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

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

Abductive problems with optimization

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

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`