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 Coruna

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

Conflict Driven Answer Set Solving

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 *add a conflict constraint* **and** *backjump*

Inside decide

- Basic concepts
  - Atoms $\mathcal{A}$, and partial assignments $\mathbf{A} : \mathcal{A} \to \{\mathbf{T}, \mathbf{F}\}$
    $\mathbf{A}^\mathbf{T} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{T} \in \mathbf{A}\}$ and $\mathbf{A}^\mathbf{F} = \{a \in \mathcal{A} \mid a \mapsto \mathbf{F} \in \mathbf{A}\}$
- Heuristic functions

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

- Algorithmic scheme

$h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
$U := \mathcal{A} \setminus (A^\mathbf{T} \cup A^\mathbf{F})$
$C := \mathit{argmax}_{a \in U} h(a)$
$a := \tau(C)$
$A := A \cup \{a \mapsto s(a)\}$

Heuristics in ASP

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(a,init,5)` Add 5 to the initial heuristic value of a
  `_heuristic(b,factor,2)` Multiply the heuristic value of b by 2
  `_heuristic(c,level,1)` The rank of c is 1 (the default rank is 0)
  `_heuristic(d,sign,-1)` If deciding on d, assign it to false

Heuristic modifications to functions h and s

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

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

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

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

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

Inside decide, heuristically modified

$h(a) := d(a)$ for each $a \in \mathcal{A}$
$h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
$U := \ell_A(\mathcal{A} \setminus (A^\mathbf{T} \cup A^\mathbf{F}))$
$C := \mathit{argmax}_{a \in U} d(a)$
$a := \tau(C)$
$A := A \cup \{a \mapsto t(a)\}$

Heuristics in ASP for PDDL planning

Simple STRIPS planner

```
time(1..last).

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 nholds(F,T), time(T).
holds(F,T)  :- occurs(A,T), add(A,F).
nholds(F,T) :- occurs(A,T), del(A,F).

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

Heuristic rules

Add 5 to the initial heuristic value of actions
```
_heuristic(occurs(A,T),init,5) :- action(A), time(T).
```

Multiply the heuristic value of actions at T by T
```
_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Decide first on actions
```
_heuristic(occurs(A,T),level,1) :- action(A), time(T).
```

Do a forward search on actions
```
_heuristic(occurs(A,T),level,last-T+1) :- action(A), time(T).
```

If deciding on actions, assign them to false
```
_heuristic(occurs(A,T),sign,-1) :- action(A), time(T).
```

New modifiers definition

Modifier `true` gives a rank and a positive sign, `false` gives a rank and a negative sign
```
_heuristic(X,level,Y) :- _heuristic(X,true,Y).
_heuristic(X,sign,1)  :- _heuristic(X,true,Y).
_heuristic(X,level,Y) :- _heuristic(X,false,Y).
_heuristic(X,sign,-1) :- _heuristic(X,false,Y).
```

Do a forward search on true actions
```
_heuristic(occurs(A,T),true,last-T+1) :- action(A), time(T).
```

Experiments in Abduction with Optimization and PDDL Planning

Abductive problems wih optimization

- Heuristic rules modify the abducibles, e.g., for the last row in Diagnosis we use:

```
_heuristic(abnormal(C),level,1) :- component(C).
```

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)

Showing average times (timeouts)

Planning Competition Benchmarks

- Heuristic rules make the fluents persist backwards:

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

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

Showing avg. times (plans/timeouts), and avg. times (timeouts) for satisfiable instances

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`