“Are Preferences Giving You a Headache?” “Take asprin!”

Gerhard Brewka James Delgrande Javier Romero Torsten Schaub

University of Leipzig Simon Fraser University University of Potsdam

Potassco

Outline

Potassco

Outline

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of today's ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of today's ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of today's ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of today's ASP systems
- Example $\#\mathit{minimize}\{40 : \mathit{sauna}, 70 : \mathit{dive}\}$

Approach

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - without any modifications to the ASP solver
  - significantly reducing redundancies
  - via an implementation through ordinary ASP encodings

Approach

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings

Potassco

Approach

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings

Potassco

Approach

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings

Example

$$\#preference(costs, less(weight))\{40 : sauna, 70 : dive\}$$
$$\#preference(fun, superset)\{sauna, dive, hike, \neg bunji\}$$
$$\#preference(temps, aso)\{dive > sauna \parallel hot, sauna > dive \parallel \neg hot\}$$
$$\#preference(all, pareto)\{name(costs), name(fun), name(temps)\}$$

$$\#optimize(all)$$

Outline

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$

- A preference type is a (parametric) class of preference relations

Potassco

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$
- A preference type is a (parametric) class of preference relations

Potassco

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$
- A preference type is a (parametric) class of preference relations

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$
- A preference type is a (parametric) class of preference relations

Outline

Language

- weighted formula $w_1, \dots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s, t)\{e_1, \dots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element

- optimization directive $\#optimize(s)$
  where s is the name of a preference

- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Potassco

Language

- weighted formula $w_1, \ldots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s, t)\{e_1, \ldots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element
- optimization directive $\#optimize(s)$
  where s is the name of a preference
- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Potassco

Language

- weighted formula $w_1, \ldots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s, t)\{e_1, \ldots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element
- optimization directive $\#optimize(s)$
  where s is the name of a preference
- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Preference type

- A preference type t is a function mapping a set of preference elements E to a preference relation

$$t : E \mapsto \{(X, Y) \mid def_t(E, X, Y), X, Y \subseteq \mathcal{A}\}$$

where

  - $def_t(E, X, Y)$ defines the relation among sets X and Y wrt E
  - $dom(t)$ is the domain of t fixing admissible preference elements for t

- Example $less(cardinality)$

$$def_{less(cardinality)}(E, X, Y) = |\{\ell \in E \mid X \models \ell\}| < |\{\ell \in E \mid Y \models \ell\}|$$

$$dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$$

Potassco

Preference type

- A preference type t is a function mapping a set of preference elements E to a preference relation

$$t : E \mapsto \{(X,Y) \mid def_t(E,X,Y), X,Y \subseteq \mathcal{A}\}$$

where

  - $def_t(E,X,Y)$ defines the relation among sets X and Y wrt E
  - $dom(t)$ is the domain of t fixing admissible preference elements for t

- Example $less(cardinality)$
  - $def_{less(cardinality)}(E,X,Y) = |\{\ell \in E \mid X \models \ell\}| < |\{\ell \in E \mid Y \models \ell\}|$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$

Potassco

Preference type

- A preference type t is a function mapping a set of preference elements E to a preference relation

$$t : E \mapsto \{(X,Y) \mid def_t(E,X,Y), X, Y \subseteq \mathcal{A}\}$$

where

  - $def_t(E,X,Y)$ defines the relation among sets X and Y wrt E
  - $dom(t)$ is the domain of t fixing admissible preference elements for t

- Example $less(cardinality)$
  - $def_{less(cardinality)}(E,X,Y) = |\{\ell \in E \mid X \models \ell\}| < |\{\ell \in E \mid Y \models \ell\}|$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$

Potassco

Preference type

- A preference type t is a function mapping a set of preference elements E to a preference relation

$$t : E \mapsto \{(X, Y) \mid \mathit{def}_t(E, X, Y), X, Y \subseteq \mathcal{A}\}$$

where

  - $\mathit{def}_t(E, X, Y)$ defines the relation among sets X and Y wrt E
  - $\mathit{dom}(t)$ is the domain of t fixing admissible preference elements for t

- Example $\mathit{less}(\mathit{cardinality})$
  - $\mathit{def}_{\mathit{less}(\mathit{cardinality})}(E, X, Y) = |\{\ell \in E \mid X \models \ell\}| < |\{\ell \in E \mid Y \models \ell\}|$
  - $\mathit{dom}(\mathit{less}(\mathit{cardinality})) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$

More examples

- *more*(*weight*) is defined by
  - $def_{more(weight)}(E, X, Y) = \sum_{(w:\ell)\in E, X\models\ell} w > \sum_{(w:\ell)\in E, Y\models\ell} w$
  - $dom(more(weight)) = \mathcal{P}(\{w : a, w : \neg a \mid w \in \mathbb{Z}, a \in \mathcal{A}\})$; and
- *subset* is defined by
  - $def_{subset}(E, X, Y) = \{\ell \in E \mid X \models \ell\} \subset \{\ell \in E \mid Y \models \ell\}$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$.
- *pareto* is defined by
  - $def_{pareto}(E, X, Y) = \bigwedge_{name(s)\in E}(X \succeq_s Y) \wedge \bigvee_{name(s)\in E}(X \succ_s Y)$
  - $dom(pareto) = \mathcal{P}(\{n \mid n \in N\})$;
- *lexico* is defined by
  - $def_{lexico}(E, X, Y) = \bigvee_{w:name(s)\in E}((X \succ_s Y) \wedge \bigwedge_{v:name(s')\in E, v<w}(X =_{s'} Y))$
  - $dom(lexico) = \mathcal{P}(\{w : n \mid w \in \mathbb{Z}, n \in N\})$.

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s,t)\,E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

$$X \succ_1 Y \text{ as } |\{\ell \in \{a,\neg b,c\} \mid X \models \ell\}| < |\{\ell \in \{a,\neg b,c\} \mid Y \models \ell\}|$$

where $\succ_1$ stands for $less(cardinality)(\{a,\neg b,c\})$

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s,t)\,E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

$$X \succ_1 Y \text{ as } |\{\ell \in \{a, \neg b, c\} \mid X \models \ell\}| < |\{\ell \in \{a, \neg b, c\} \mid Y \models \ell\}|$$

where $\succ_1$ stands for $less(cardinality)(\{a, \neg b, c\})$

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s, t)\, E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

  $X \succ_1 Y$ as $|\{\ell \in \{a, \neg b, c\} \mid X \models \ell\}| < |\{\ell \in \{a, \neg b, c\} \mid Y \models \ell\}|$

  where $\succ_1$ stands for $less(cardinality)(\{a, \neg b, c\})$

Outline

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let E_{t_s}, F_s, and A be "certain" logic programs

  We define $E_{t_s} \cup F_s \cup A$ as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad E_{t_s} \cup F_s \cup A \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note Dynamic versions of H_X and H_Y must be used for optimization

Potassco

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let E_{t_s}, F_s, and A be "certain" logic programs

  We define $E_{t_s} \cup F_s \cup A$ as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad E_{t_s} \cup F_s \cup A \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note Dynamic versions of H_X and H_Y must be used for optimization

Potassco

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let E_{t_s}, F_s, and A be "certain" logic programs

  We define $E_{t_s} \cup F_s \cup A$ as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad E_{t_s} \cup F_s \cup A \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note Dynamic versions of H_X and H_Y must be used for optimization

$$\#preference(3, subset)\{a, \neg b, c\}$$

$E_{subset} =$
```
better(P) :- preference(P,subset),
             holds'(X) : preference(P,_,_,for(X),_), holds(X);
             1 #sum { 1,X : not holds(X), holds'(X),
                            preference(P,_,_,for(X),_) }.
```

$F_3 =$
```
preference(3,subset).   preference(3,1,1,for(a),()).
                        preference(3,2,1,for(neg(b)),()).
                        preference(3,3,1,for(c),()).
```

$A =$
```
holds(neg(A))  :- not holds(A), preference(_,_,_,for(neg(A)),_).
holds'(neg(A)) :- not holds'(A),preference(_,_,_,for(neg(A)),_).
```

$H_{\{a,b\}} =$
```
holds(a).  holds(b).
```

$H'_{\{a\}} =$
```
holds'(a).
```

We get a stable model containing `better(3)` indicating that $\{a, b\} \succ_3 \{a\}$, or $\{a\} \subset \{a, \neg b\}$

Potassco

$\#preference(3, subset)\{a, \neg b, c\}$

$E_{subset} =$
```
better(P) :- preference(P,subset),
             holds'(X) : preference(P,_,_,for(X),_), holds(X);
             1 #sum { 1,X : not holds(X), holds'(X),
                            preference(P,_,_,for(X),_) }.
```

$F_3 =$
```
preference(3,subset).   preference(3,1,1,for(a),()).
                        preference(3,2,1,for(neg(b)),()).
                        preference(3,3,1,for(c),()).
```

$A =$
```
holds(neg(A))  :- not holds(A), preference(_,_,_,for(neg(A)),_).
holds'(neg(A)) :- not holds'(A),preference(_,_,_,for(neg(A)),_).
```

$H_{\{a,b\}} =$
```
holds(a).  holds(b).
```

$H'_{\{a\}} =$
```
holds'(a).
```

We get a stable model containing `better(3)` indicating that $\{a, b\} \succ_3 \{a\}$, or $\{a\} \subset \{a, \neg b\}$

Potassco

Basic algorithm $solveOpt(P, s)$

Input : A program P over $\mathcal{A}$ and preference statement s
Output : A $\succ_s$-preferred stable model of P, if P is satisfiable, and $\bot$ otherwise

$Y \leftarrow solve(P)$
if $Y = \bot$ **then return** $\bot$
repeat
  $X \leftarrow Y$
  $Y \leftarrow solve(P \cup E_{t_s} \cup F_s \cup R_{\mathcal{A}} \cup H'_X) \cap \mathcal{A}$
until $Y = \bot$
return X

where $R_X = \{holds(a) \leftarrow a \mid a \in X\}$

Potassco

Embedded Python Implementation

```
#script (python)

from gringo import *
holds = []

def getHolds():
    global holds
    return holds

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1: prg.ground([("doholds",[step-1]),("preference",[0,step-1])]
        ret = prg.solve(onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1
#end.

#program base.                     #program doholds(m).             #program preference(m1,m2).
#show _holds(X,0) : _holds(X,0).   _holds(X,m) :- X = @getHolds().  _volatile(m1,m2).

#end.
```

Embedded Python Implementation

```
#script (python)

from gringo import *
holds = []

def getHolds():
    global holds
    return holds

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1: prg.ground([("doholds",[step-1]),("preference",[0,step-1])]
        ret = prg.solve(onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1
#end.
```

```
#program base.
#show _holds(X,0) : _holds(X,0).
```

```
#program doholds(m).
_holds(X,m) :- X = @getHolds().
```

```
#program preference(m1,m2).
_volatile(m1,m2).
```

```
#end.
```

Potassco

Outline

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations

- `http://potassco.sourceforge.net/labs.html#asprin`

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations
- `http://potassco.sourceforge.net/labs.html#asprin`

Potassco

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations
- http://potassco.sourceforge.net/labs.html#asprin

Potassco

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations

- http://potassco.sourceforge.net/labs.html#asprin

Potassco