Answer Set Programming for Optimization and its Application

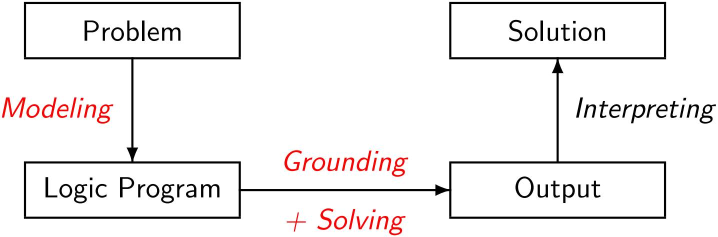
Gerhard Friedrich & Martin Gebser

University of Klagenfurt

Graz University of Technology

ASP'ish Knowledge Representation and Reasoning

- Answer Set Programming (ASP) offers expressive first-order modeling language and powerful reasoning technology
  - Ground instantiation by semi-naive database evaluation
  - Search/optimization by conflict-driven learning



Problem

Solution

Modeling

Interpreting

Logic Program

Grounding

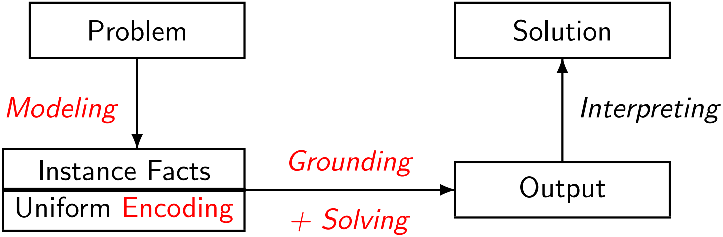
+ Solving

Output

- Uniform problem representations separate instance data from high-level problem encoding for elaboration-tolerant modeling
  - Logic Program = $\underbrace{\text{Facts}}_{\text{Instance}} + \underbrace{\text{Generate} + \text{Define} + \text{Test} + \text{Optimize}}_{\text{Encoding}}$

ASP’ish Knowledge Representation and Reasoning

- Answer Set Programming (ASP) offers expressive first-order modeling language and powerful reasoning technology
  - Ground instantiation by semi-naive database evaluation
  - Search/optimization by conflict-driven learning



- Uniform problem representations separate instance data from high-level problem encoding for elaboration-tolerant modeling
  - Logic Program $= \underbrace{\text{Facts}}_{\text{Instance}} + \underbrace{\text{Generate} + \text{Define} + \text{Test} + \text{Optimize}}_{\text{Encoding}}$

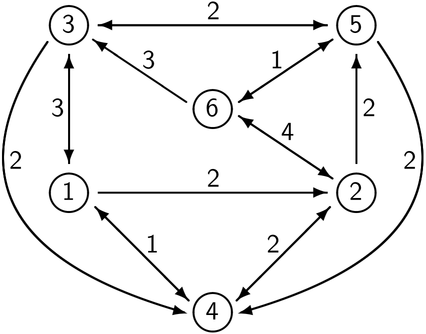
Outline

1. Traveling Salesperson Problem

2. Linux Package Configuration

3. Conclusion

Traveling Salesperson (TSP) Example

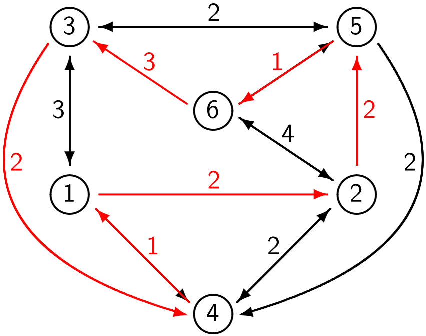


Total Cost: 11

Instance Representation

```
node(1). node(2). node(3).
node(4). node(5). node(6).


edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).
edge(2,4,2). edge(4,2,2).
edge(2,5,2).
edge(2,6,4). edge(6,2,4).
edge(3,4,2).
edge(3,5,2). edge(5,3,2).
edge(5,4,2).
edge(5,6,1). edge(6,5,1).
edge(6,3,3).
```

Traveling Salesperson (TSP) Example

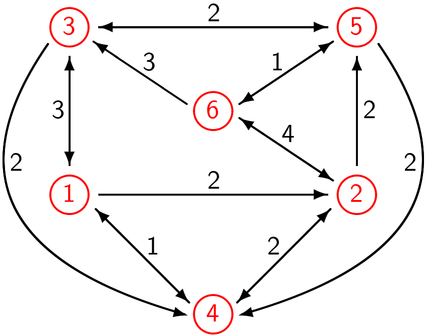


Total Cost: 11

Instance Representation

```
node(1). node(2). node(3).
node(4). node(5). node(6).

edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).
edge(2,4,2). edge(4,2,2).
edge(2,5,2).
edge(2,6,4). edge(6,2,4).
edge(3,4,2).
edge(3,5,2). edge(5,3,2).
edge(5,4,2).
edge(5,6,1). edge(6,5,1).
edge(6,3,3).
```

Traveling Salesperson (TSP) Example

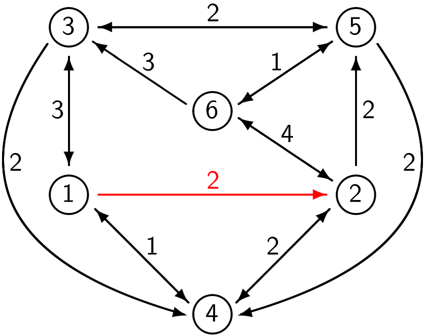


Total Cost: 11

Instance Representation

```
node(1). node(2). node(3).
node(4). node(5). node(6).

edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).
edge(2,4,2). edge(4,2,2).
edge(2,5,2).
edge(2,6,4). edge(6,2,4).
edge(3,4,2).
edge(3,5,2). edge(5,3,2).
edge(5,4,2).
edge(5,6,1). edge(6,5,1).
edge(6,3,3).
```

Traveling Salesperson (TSP) Example

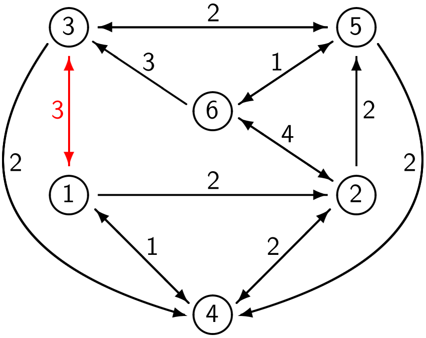


Total Cost: 11

Instance Representation

```
node(1). node(2). node(3).
node(4). node(5). node(6).

edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).
edge(2,4,2). edge(4,2,2).
edge(2,5,2).
edge(2,6,4). edge(6,2,4).
edge(3,4,2).
edge(3,5,2). edge(5,3,2).
edge(5,4,2).
edge(5,6,1). edge(6,5,1).
edge(6,3,3).
```

Traveling Salesperson (TSP) Example



Total Cost: 11

Instance Representation

```
node(1). node(2). node(3).
node(4). node(5). node(6).

edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).
edge(2,4,2). edge(4,2,2).
edge(2,5,2).
edge(2,6,4). edge(6,2,4).
edge(3,4,2).
edge(3,5,2). edge(5,3,2).
edge(5,4,2).
edge(5,6,1). edge(6,5,1).
edge(6,3,3).
```

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

TSP Solution Specification

1. Exactly one outgoing edge per node
2. Exactly one incoming edge per node
3. All nodes reached from (arbitrary) start node
4. Minimum sum of edge costs

Problem Encoding

```
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X,Y]
```

☞ More encoding “optimization” feasible

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(X).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,Y) : edge(1,Y,C)} = 1 :- node(1).
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,Y) : edge(1,Y,C)} = 1.
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.
{cycle(X,Y) : edge(X,Y,C)} = 1 :- node(Y).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.
{cycle(X,1) : edge(X,1,C)} = 1 :- node(1).

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.
{cycle(3,1); cycle(4,1)} = 1.

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(X) :- #min{Y : node(Y)} = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(X) :- #min{1 : ; 2 : ; 3 : ; 4 : ; 5 : ; 6 : } = X.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1) :- #min{1 : ; 2 : ; 3 : ; 4 : ; 5 : ; 6 : } = 1.
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(Y) :- cycle(1,Y), reached(1).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(Y) :- cycle(1,Y), reached(1).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- node(Y), not reached(Y).

:~ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- node(Y), not reached(Y).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- not reached(1).  ...   :- not reached(6).

:∼ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- not reached(1).  ...   :- not reached(6).

:~ cycle(X,Y), edge(X,Y,C). [C,X]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- not reached(1).  ...   :- not reached(6).

:∼ cycle(1,2). [2,1]  :∼ cycle(1,3). [3,1]
:∼ cycle(1,4). [1,1]
```

(Intelligent) Grounding

Ground Instantiation

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2,2).
edge(1,3,3). edge(3,1,3).
edge(1,4,1). edge(4,1,1).  ...

{cycle(1,2); cycle(1,3); cycle(1,4)} = 1.  ...
{cycle(3,1); cycle(4,1)} = 1.  ...

reached(1).
reached(2) :- cycle(1,2), reached(1).
reached(3) :- cycle(1,3), reached(1).
reached(4) :- cycle(1,4), reached(1).  ...
:- not reached(1).  ...   :- not reached(6).

:∼ cycle(1,2). [2,1]  :∼ cycle(1,3). [3,1]
:∼ cycle(1,4). [1,1]  ...
```

(Pseudo-)Boolean Optimization

Model-guided Approach

```
$ clingo <instance> <encoding>

Answer: 1
cycle(1,4) cycle(4,2) cycle(2,6) cycle(6,5) cycle(5,3) cycle(3,1)
Optimization: 13

Answer: 2
cycle(1,4) cycle(4,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,1)
Optimization: 12

Answer: 3
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 12
```

(Pseudo-)Boolean Optimization

Model-guided Approach

```
$ clingo <instance> <encoding>

Answer: 1
cycle(1,4) cycle(4,2) cycle(2,6) cycle(6,5) cycle(5,3) cycle(3,1)
Optimization: 13

Answer: 2
cycle(1,4) cycle(4,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,1)
Optimization: 12

Answer: 3
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 12
```

(Pseudo-)Boolean Optimization

Model-guided Approach

```
$ clingo <instance> <encoding>

Answer: 1
cycle(1,4) cycle(4,2) cycle(2,6) cycle(6,5) cycle(5,3) cycle(3,1)
Optimization: 13

Answer: 2
cycle(1,4) cycle(4,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,1)
Optimization: 12

Answer: 3
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 12
```

(Pseudo-)Boolean Optimization

Core-guided Approach

```
$ clingo <instance> <encoding> --opt-strategy=usc
Progression : [3;inf]
Progression : [5;inf]
Progression : [6;inf]
Progression : [7;inf]
Progression : [8;inf]
Progression : [9;inf]
Progression : [10;inf]
Progression : [11;inf]

Answer: 1
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 10
```

(Pseudo-)Boolean Optimization

Core-guided Approach

```
$ clingo <instance> <encoding> --opt-strategy=usc
Progression : [3;inf]
Progression : [5;inf]
Progression : [6;inf]
Progression : [7;inf]
Progression : [8;inf]
Progression : [9;inf]
Progression : [10;inf]
Progression : [11;inf]

Answer: 1
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 10
```

(Pseudo-)Boolean Optimization

Core-guided Approach

```
$ clingo <instance> <encoding> --opt-strategy=usc
Progression : [3;inf]
Progression : [5;inf]
Progression : [6;inf]
Progression : [7;inf]
Progression : [8;inf]
Progression : [9;inf]
Progression : [10;inf]
Progression : [11;inf]

Answer: 1
cycle(1,2) cycle(2,5) cycle(5,6) cycle(6,3) cycle(3,4) cycle(4,1)
Optimization: 11

Time : 0.002s
Conflicts : 10
```

Free and Open Source Software Management

- Maintaining packages in modern Linux distributions is difficult
  - Complex dependencies
  - Large package repositories
  - Ever changing in view of software development
- Challenges for package configuration tools
  - Large problem size
  - Soft (and hard) constraints
  - Multiple optimization criteria

☞ Targeted in the EU research project *Mancoosi*

- Contributions of ASP
  - Uniform modeling by encoding plus instances
  - Solving techniques for (multi-criteria) optimization

☞ *Instead of the standard* `apt-get install libreoffice` *that failed to propose a decent upgrade, as detailed later, I typed* `apt-get --solver aspcud install libreoffice` *that returned this pretty good solution . . .*

Free and Open Source Software Management

- Maintaining packages in modern Linux distributions is difficult
  - Complex dependencies
  - Large package repositories
  - Ever changing in view of software development
- Challenges for package configuration tools
  - Large problem size
  - Soft (and hard) constraints
  - Multiple optimization criteria

☞ Targeted in the EU research project *Mancoosi*

- Contributions of ASP
  - Uniform modeling by encoding plus instances
  - Solving techniques for (multi-criteria) optimization

☞ *Instead of the standard* `apt-get install libreoffice` *that failed to propose a decent upgrade, as detailed later, I typed* `apt-get --solver aspcud install libreoffice` *that returned this pretty good solution . . .*

Free and Open Source Software Management

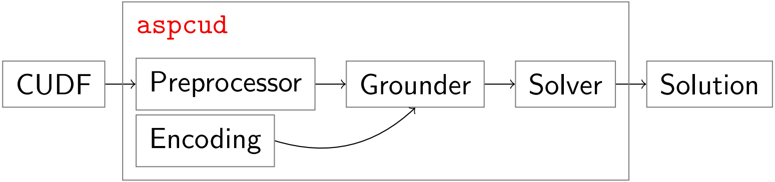
- ▶ Maintaining packages in modern Linux distributions is difficult
  - Complex dependencies
  - Large package repositories
  - Ever changing in view of software development
- ▶ Challenges for package configuration tools
  - Large problem size
  - Soft (and hard) constraints
  - Multiple optimization criteria

☞ Targeted in the EU research project *Mancoosi*

- ▶ Contributions of ASP
  - Uniform modeling by encoding plus instances
  - Solving techniques for (multi-criteria) optimization

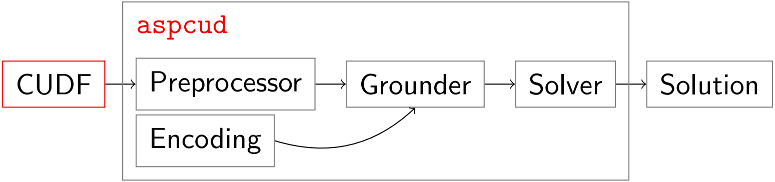
☞ *Instead of the standard* `apt-get install libreoffice` *that failed to propose a decent upgrade, as detailed later, I typed* `apt-get --solver aspcud install libreoffice` *that returned this pretty good solution . . .*

Linux Package Configurator aspcud



Preprocessor	Converts CUDF input to ASP instance
Encoding	First-order problem specification
Grounder	Instantiates first-order variables
Solver	Searches for (optimal) answer sets

Linux Package Configurator aspcud : Input



Preprocessor Converts CUDF input to ASP instance

Encoding First-order problem specification

Grounder Instantiates first-order variables

Solver Searches for (optimal) answer sets

Common Upgradability Description Format (CUDF)

- Language to represent package interdependencies
  - Conflicts
  - Dependencies
  - Recommendations
- and user goals
  - Installation
  - Removal
  - Upgrade
- subject to optimization
  - Package deletions
  - Package additions
  - Package recommendations
  - Version changes
  - Version up-to-dateness
  - Version coherence
  - Installation size

CUDF Input

```
package:     firefox
version:     3
conflicts:   firefox
depends:     xserver > 2

recommends:  thunderbird

request:
install:     firefox
remove:      firefox < 3

upgrade:     firefox > 2
```

Common Upgradability Description Format (CUDF)

- Language to represent package interdependencies
  - Conflicts
  - Dependencies
  - Recommendations
- and user goals
  - Installation
  - Removal
  - Upgrade
- subject to optimization
  - Package deletions
  - Package additions
  - Package recommendations
  - Version changes
  - Version up-to-dateness
  - Version coherence
  - Installation size

CUDF Input

```
package:     firefox
version:     3
conflicts:   firefox
depends:     xserver > 2

recommends:  thunderbird

request:
install:     firefox
remove:      firefox < 3

upgrade:     firefox > 2
```

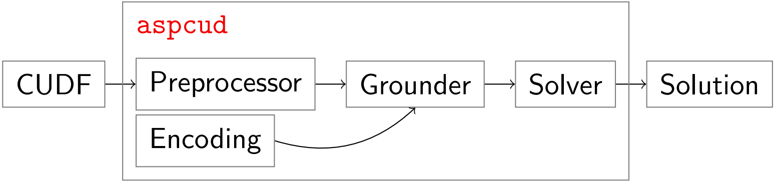
Common Upgradability Description Format (CUDF)

- Language to represent package interdependencies
  - Conflicts
  - Dependencies
  - Recommendations
- and user goals
  - Installation
  - Removal
  - Upgrade
- subject to optimization
  - Package deletions
  - Package additions
  - Package recommendations
  - Version changes
  - Version up-to-dateness
  - Version coherence
  - Installation size

CUDF Input

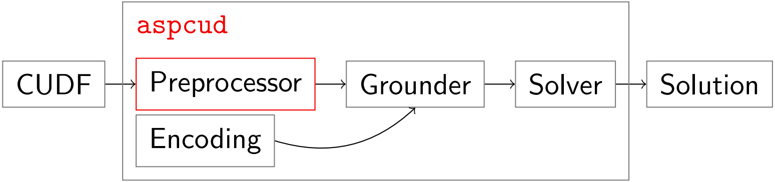
```
package:     firefox
version:     3
conflicts:   firefox
depends:     xserver > 2

recommends:  thunderbird

request:
install:     firefox
remove:      firefox < 3

upgrade:     firefox > 2
```

Linux Package Configurator aspcud



Preprocessor	Converts CUDF input to ASP instance
Encoding	First-order problem specification
Grounder	Instantiates first-order variables
Solver	Searches for (optimal) answer sets

Linux Package Configurator aspcud : Preprocessor



Preprocessor Converts CUDF input to ASP instance

Encoding First-order problem specification

Grounder Instantiates first-order variables

Solver Searches for (optimal) answer sets

Setting the Focus

Scenario

- Modern Linux distributions are large (50K packages or more)

☞ Problem representation and search space are of significant size

Observations

- Some packages can't be installed (`remove` or `upgrade` goals)
- An empty installation is conflict-free and thus valid

☞ Packages to install should serve (hard) `install` or `upgrade` goals, or satisfy (soft) constraints

Approach

1. Identify packages whose installation may be of direct use
2. Saturate such packages wrt. dependencies and soft constraints
3. Restrict the ASP instance to closure of "interesting" packages
4. (Greedily) partition these packages into mutual conflict cliques

Setting the Focus

Scenario

- Modern Linux distributions are large (50K packages or more)

☞ Problem representation and search space are of significant size

Observations

- Some packages can't be installed (`remove` or `upgrade` goals)
- An empty installation is conflict-free and thus valid

☞ Packages to install should serve (hard) `install` or `upgrade` goals, or satisfy (soft) constraints

Approach

1. Identify packages whose installation may be of direct use
2. Saturate such packages wrt. dependencies and soft constraints
3. Restrict the ASP instance to closure of “interesting” packages
4. (Greedily) partition these packages into mutual conflict cliques

Setting the Focus

Scenario

- Modern Linux distributions are large (50K packages or more)

☞ Problem representation and search space are of significant size

Observations

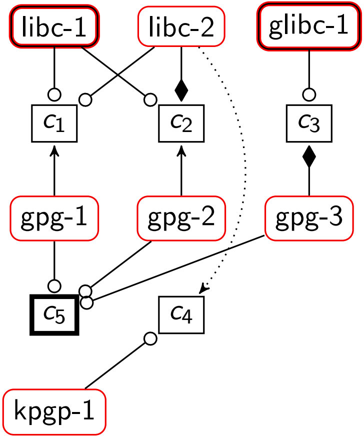
- Some packages can't be installed (`remove` or `upgrade` goals)
- An empty installation is conflict-free and thus valid

☞ Packages to install should serve (hard) `install` or `upgrade` goals, or satisfy (soft) constraints

Approach

1. Identify packages whose installation may be of direct use
2. Saturate such packages wrt. dependencies and soft constraints
3. Restrict the ASP instance to closure of "interesting" packages
4. (Greedily) partition these packages into mutual conflict cliques

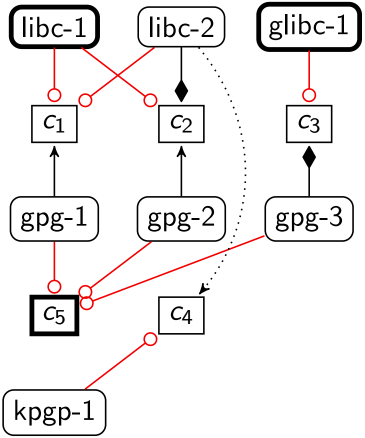
Instance Representation



Installable Packages

```
package(libc,1).
package(libc,2).

package(glibc,1).

package(gpg,1).
package(gpg,2).
package(gpg,3).

package(kpgp,1).
```

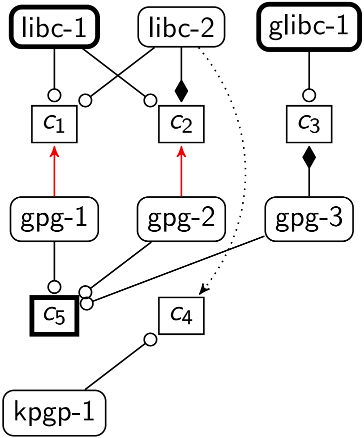
Instance Representation



Package Conditions

```
satisfies(libc,1,c1).
satisfies(libc,2,c1)

satisfies(libc,1,c2).

satisfies(glibc,1,c3).

satisfies(kpgp,1,c4).

satisfies(gpg,1,c5).
satisfies(gpg,2,c5).
satisfies(gpg,3,c5).
```

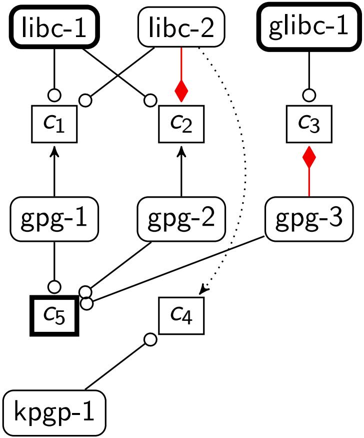
Instance Representation



Package Dependencies

```
depends(gpg,1,c1).

depends(gpg,2,c2).
```

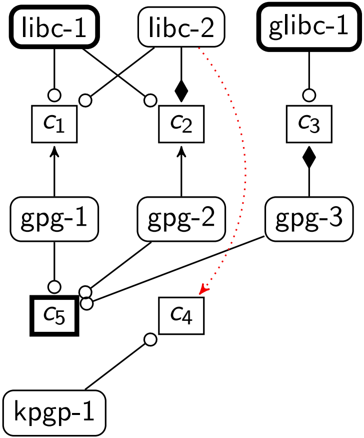
Instance Representation



Package Conflicts

```
conflicts(libc,2,c2).

conflicts(gpg,3,c3).
```

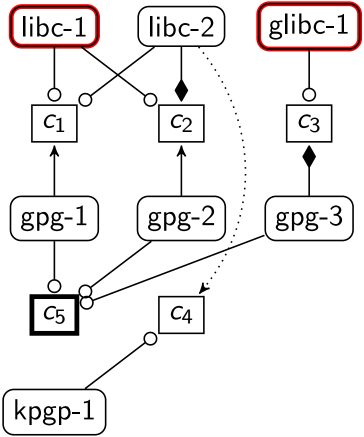
Instance Representation



Package Recommendations

```
recommends(libc,2,c4).
```

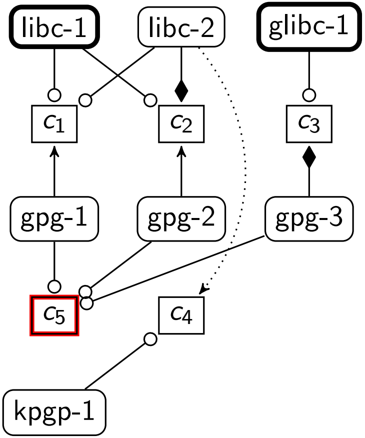
Instance Representation



Installed Packages

```
installed(libc,1).

installed(glibc,1).
```

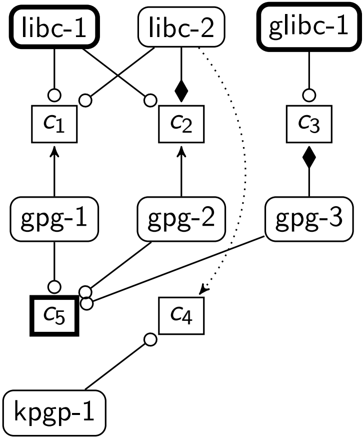
Instance Representation



User Goals

```
requested(c5).
```

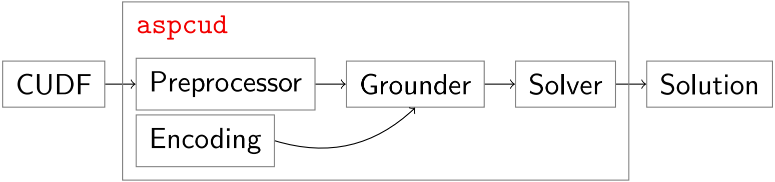
Instance Representation



Optimization Criteria

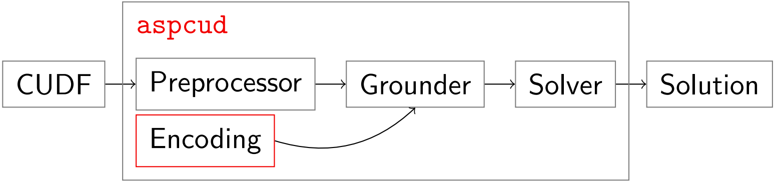
```
utility(delete,1).

utility(change,2).
```

Linux Package Configurator aspcud



Preprocessor	Converts CUDF input to ASP instance
Encoding	First-order problem specification
Grounder	Instantiates first-order variables
Solver	Searches for (optimal) answer sets

Linux Package Configurator aspcud : Encoding



Preprocessor	Converts CUDF input to ASP instance
Encoding	First-order problem specification
Grounder	Instantiates first-order variables
Solver	Searches for (optimal) answer sets

Hard Constraints

1. Can install any installable package
2. Excluded, included, and satisfied conditions (packages) follow
3. Respective conditions and user goals must be fulfilled

Problem Encoding

```
{install(P,V)} :- package(P,V).

exclude(C) :- install(P,V), conflicts(P,V,C).
include(C) :- install(P,V), depends(P,V,C).

satisfy(C) :- install(P,V), satisfies(P,V,C).

:- exclude(C),     satisfy(C).
:- include(C), not satisfy(C).
:- request(C), not satisfy(C).
```

Hard Constraints

1. Can install any installable package
2. Excluded, included, and satisfied conditions (packages) follow
3. Respective conditions and user goals must be fulfilled

Problem Encoding

```
{install(P,V)} :- package(P,V).

exclude(C) :- install(P,V), conflicts(P,V,C).
include(C) :- install(P,V), depends(P,V,C).

satisfy(C) :- install(P,V), satisfies(P,V,C).

:- exclude(C),     satisfy(C).
:- include(C), not satisfy(C).
:- request(C), not satisfy(C).
```

Hard Constraints

1. Can install any installable package
2. Excluded, included, and satisfied conditions (packages) follow
3. Respective conditions and user goals must be fulfilled

Problem Encoding

```
{install(P,V)} :- package(P,V).

exclude(C) :- install(P,V), conflicts(P,V,C).
include(C) :- install(P,V), depends(P,V,C).

satisfy(C) :- install(P,V), satisfies(P,V,C).

:- exclude(C),     satisfy(C).
:- include(C), not satisfy(C).
:- request(C), not satisfy(C).
```

Hard Constraints

1. Can install any installable package
2. Excluded, included, and satisfied conditions (packages) follow
3. Respective conditions and user goals must be fulfilled

Problem Encoding

```
{install(P,V)} :- package(P,V).

exclude(C) :- install(P,V), conflicts(P,V,C).
include(C) :- install(P,V), depends(P,V,C).

satisfy(C) :- install(P,V), satisfies(P,V,C).

:- exclude(C),     satisfy(C).
:- include(C), not satisfy(C).
:- request(C), not satisfy(C).
```

Soft Constraints

1. Package additions and deletions
2. Version changes

Problem Encoding (ctd)

```
install(P)    :- install(P,V).
installed(P) :- installed(P,V).

violate(newpkg,L,P) :-
    utility(newpkg,L), install(P), not installed(P).
violate(delete,L,P) :-
    utility(delete,L), installed(P), not install(P).
violate(change,L,P) :-
    utility(change,L), installed(P,V), not install(P,V).
violate(change,L,P) :-
    utility(change,L), install(P,V), not installed(P,V).

:~ violate(U,L,P). [1@L,U,P]
```

Soft Constraints

1. Package additions and deletions
2. Version changes

Problem Encoding (ctd)

```
install(P)   :- install(P,V).
installed(P) :- installed(P,V).

violate(newpkg,L,P) :-
    utility(newpkg,L), install(P), not installed(P).
violate(delete,L,P) :-
    utility(delete,L), installed(P), not install(P).
violate(change,L,P) :-
    utility(change,L), installed(P,V), not install(P,V).
violate(change,L,P) :-
    utility(change,L), install(P,V), not installed(P,V).

:~ violate(U,L,P). [1@L,U,P]
```

Soft Constraints

1. Package additions and deletions
2. Version changes

Problem Encoding (ctd)

```
install(P)   :- install(P,V).
installed(P) :- installed(P,V).

violate(newpkg,L,P) :-
    utility(newpkg,L), install(P), not installed(P).
violate(delete,L,P) :-
    utility(delete,L), installed(P), not install(P).
violate(change,L,P) :-
    utility(change,L), installed(P,V), not install(P,V).
violate(change,L,P) :-
    utility(change,L), install(P,V), not installed(P,V).

:∼ violate(U,L,P). [1@L,U,P]
```

Soft Constraints

1. Package additions and deletions
2. Version changes

Problem Encoding (ctd)

```
install(P)   :- install(P,V).
installed(P) :- installed(P,V).

violate(newpkg,L,P) :-
    utility(newpkg,L), install(P), not installed(P).
violate(delete,L,P) :-
    utility(delete,L), installed(P), not install(P).
violate(change,L,P) :-
    utility(change,L), installed(P,V), not install(P,V).
violate(change,L,P) :-
    utility(change,L), install(P,V), not installed(P,V).

:~ violate(U,L,P). [1@L,U,P]
```

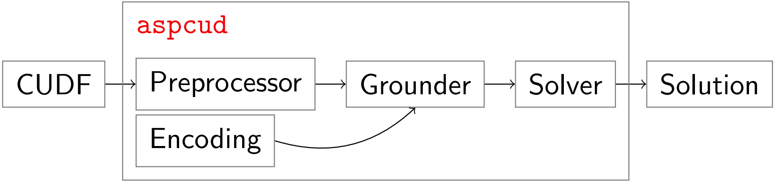
Soft Constraints

1. Package additions and deletions
2. Version changes

Problem Encoding (ctd)

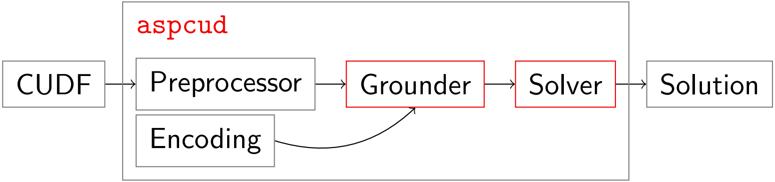
```
install(P)   :- install(P,V).
installed(P) :- installed(P,V).

violate(newpkg,L,P) :-
    utility(newpkg,L), install(P), not installed(P).
violate(delete,L,P) :-
    utility(delete,L), installed(P), not install(P).
violate(change,L,P) :-
    utility(change,L), installed(P,V), not install(P,V).
violate(change,L,P) :-
    utility(change,L), install(P,V), not installed(P,V).
...
:~ violate(U,L,P). [1@L,U,P]
```

Linux Package Configurator aspcud



Preprocessor	Converts CUDF input to ASP instance
Encoding	First-order problem specification
Grounder	Instantiates first-order variables
Solver	Searches for (optimal) answer sets

Linux Package Configurator aspcud : Reasoning



Preprocessor Converts CUDF input to ASP instance

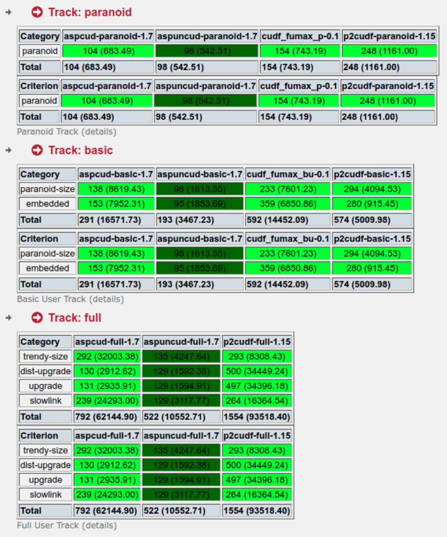
Encoding First-order problem specification

Grounder Instantiates first-order variables

Solver Searches for (optimal) answer sets

Mancoosi International Solver Competition (MISC)

- Encoding using conflict cliques and core-guided optimization



Track: paranoid

Category	aspcud-paranoid-1.7	aspuncud-paranoid-1.7	cudf_fumax_p-0.1	p2cudf-paranoid-1.15
paranoid	104 (683.49)	98 (542.51)	154 (743.19)	248 (1161.00)
Total	**104 (683.49)**	**98 (542.51)**	**154 (743.19)**	**248 (1161.00)**

Criterion	aspcud-paranoid-1.7	aspuncud-paranoid-1.7	cudf_fumax_p-0.1	p2cudf-paranoid-1.15
paranoid	104 (683.49)	98 (542.51)	154 (743.19)	248 (1161.00)
Total	**104 (683.49)**	**98 (542.51)**	**154 (743.19)**	**248 (1161.00)**

Paranoid Track (details)

Track: basic

Category	aspcud-basic-1.7	aspuncud-basic-1.7	cudf_fumax_bu-0.1	p2cudf-basic-1.15
paranoid-size	138 (8619.43)	98 (1613.55)	233 (7601.23)	294 (4094.53)
embedded	153 (7952.31)	95 (1853.69)	359 (6850.86)	280 (915.45)
Total	**291 (16571.73)**	**193 (3467.23)**	**592 (14452.09)**	**574 (5009.98)**

Criterion	aspcud-basic-1.7	aspuncud-basic-1.7	cudf_fumax_bu-0.1	p2cudf-basic-1.15
paranoid-size	138 (8619.43)	98 (1613.55)	233 (7601.23)	294 (4094.53)
embedded	153 (7952.31)	95 (1853.69)	359 (6850.86)	280 (915.45)
Total	**291 (16571.73)**	**193 (3467.23)**	**592 (14452.09)**	**574 (5009.98)**

Basic User Track (details)

Track: full

Category	aspcud-full-1.7	aspuncud-full-1.7	p2cudf-full-1.15
trendy-size	292 (32003.38)	135 (4247.64)	293 (8308.43)
dist-upgrade	130 (2912.62)	129 (1592.38)	500 (34449.24)
upgrade	131 (2935.91)	129 (1594.91)	497 (34396.18)
slowlink	239 (24293.00)	129 (3117.77)	264 (16364.54)
Total	**792 (62144.90)**	**522 (10552.71)**	**1554 (93518.40)**

Criterion	aspcud-full-1.7	aspuncud-full-1.7	p2cudf-full-1.15
trendy-size	292 (32003.38)	135 (4247.64)	293 (8308.43)
dist-upgrade	130 (2912.62)	129 (1592.38)	500 (34449.24)
upgrade	131 (2935.91)	129 (1594.91)	497 (34396.18)
slowlink	239 (24293.00)	129 (3117.77)	264 (16364.54)
Total	**792 (62144.90)**	**522 (10552.71)**	**1554 (93518.40)**

Full User Track (details)

Further Remarks

- Virtually all application problems require optimization
  - objective functions
  - lexicographic (multi-)criteria
- Complex criteria like ⊆-minimality or Pareto efficiency by
  - meta-programming (disjunctive ASP)
  - `asprin` framework
- Multi-shot solving, domain heuristics and theory reasoning
  - `clingo`
  - `clingo[DL]`
  - `clingo[LP]`
  - `clingcon`
  - `DLV2`
  - `dlvhex`
  - `EZCSP`
  - `EZSMT`
  - `IDP`
  - `wasp`
  - ASP tools (by Aalto SCI)

Further Remarks

- Virtually all application problems require optimization
  - objective functions
  - lexicographic (multi-)criteria
- Complex criteria like $\subseteq$-minimality or Pareto efficiency by
  - meta-programming (disjunctive ASP)
  - `asprin` framework
- Multi-shot solving, domain heuristics and theory reasoning
  - `clingo`
  - `clingo[DL]`
  - `clingo[LP]`
  - `clingcon`
  - `DLV2`
  - `dlvhex`
  - `EZCSP`
  - `EZSMT`
  - `IDP`
  - `wasp`
  - ASP tools (by Aalto SCI)

Thanks!

- to Roland Kaminski and Torsten Schaub for part of the slides
- to **you** for your attention and . . .

Questions?