Pocket calculators for hard combinatorial search and optimization problems

Torsten Schaub

University of Potsdam
1 Introduction

2 Modeling

3 Solving

4 Summary
Characteristics of good pocket calculators

0. handy
1. easy to use
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0 handy
1 easy to use
2 lots of operations
3 computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Introduction

Characteristics of good pocket calculators

0 handy
1 easy to use
2 lots of operations
3 computes effectively

Claim

*Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems*
Characteristics of good pocket calculators

0  handy ✓
1  easy to use
2  lots of operations
3  computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

ASP may run on your cell phone
Characteristics of good pocket calculators

0. handy ✓
1. easy to use
2. lots of operations
3. computes effectively

ASP may run on your cell phone
ASP has a high-level modeling language

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0. handy ✓
1. easy to use ✓
2. lots of operations
3. computes effectively

ASP may run on your cell phone
ASP has a high-level modeling language

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0 handy ✓
1 easy to use ✓
2 lots of operations
3 computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0. handy ✓
1. easy to use ✓
2. lots of operations ✓
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems.
Characteristics of good pocket calculators

0 handy ✓
1 easy to use ✓
2 lots of operations ✓
3 computes effectively

ASP may run on your cell phone
ASP has a high-level modeling language
ASP offers various reasoning modes
ASP solvers are highly effective

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0. handy ✓
1. easy to use ✓
2. lots of operations ✓
3. computes effectively ✓

ASP may run on your cell phone
ASP has a high-level modeling language
ASP offers various reasoning modes
ASP solvers are highly effective

Claim
Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems
Characteristics of good pocket calculators

0 handy ✓
1 easy to use ✓
2 lots of operations ✓
3 computes effectively ✓

ASP may run on your cell phone
ASP has a high-level modeling language
ASP offers various reasoning modes
ASP solvers are highly effective

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems ✓
ASPS solving

1. Problem
2. Logic Program
3. Grounder
4. Solver
5. Stable Models
6. Solution

Modeling

Interpreting

Solving
Introduction

ASP solving

Problem

Logic Program

Grounder

Solver

Stable Models

Solution

Modeling

Interpreting

Solving

Elaborating

Pocket calculators for combinatorial problems
Introduction

SAT solving

Problem:
Formula (CNF):
Solver:
Solution:
Classical Models:

Programming
Solving
Interpreting
Rooting ASP solving

**Introduction**

- **Problem**
- **Logic Program**
- **Grounder**
- **Solver**
- **Stable Models**
- **Solution**

**Modeling**

- **Solving**
- **Interpreting**
Rooting ASP solving

Introduction

Modeling

Problem

Logic Program

LP

KR

Grounder

DB

Solving

Solver

SAT

Stable Models

Solution

DB+KR+LP

Interpreting

Torsten Schaub (KRR@UP)

Pocket calculators for combinatorial problems
Outline

1. Introduction
2. Modeling
3. Solving
4. Summary
Language constructs

- **Variables**
  - $p(X) :- q(X)$

- **Conditional literals**
  - $p :- q(X) : r(X)$

- **Disjunction**
  - $p(X) ; q(X) :- r(X)$

- **Integrity constraints**
  - $:- q(X), p(X)$

- **Choice**
  - $2 \{ \ p(X,Y) : q(X) \ \} 7 :- r(Y)$

- **Aggregates**
  - $s(Y) :- r(Y), 2 \#\sum \{ X : p(X,Y), q(X) \} 7$

- **Optimization**
  - $:\sim q(X), p(X,C) [C]$
  - $\#\text{minimize} \{ C : q(X), p(X,C) \}$
Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

\[ p(X) :- q(X) \]
\[ p :- q(X) : r(X) \]
\[ p(X) ; q(X) :- r(X) \]
\[ :- q(X), p(X) \]
\[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]
\[ s(Y) :- r(Y), 2 \sum \{ X : p(X,Y), q(X) \} 7 \]
\[ :- q(X), p(X,C) [C] \]
\[ \#\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  \[ :\sim q(X), p(X,C) \ [C] \]
  \[ #\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  - p(X) :- q(X)

- Conditional literals
  - p :- q(X) : r(X)

- Disjunction
  - p(X) ; q(X) :- r(X)

- Integrity constraints
  - :- q(X), p(X)

- Choice
  - 2 { p(X,Y) : q(X) } 7 :- r(Y)

- Aggregates
  - s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7

- Optimization
  - :~ q(X), p(X,C) [C]
  - #minimize { C : q(X), p(X,C) }

Torsten Schaub (KRR@UP) Pocket calculators for combinatorial problems 9 / 21
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ \ p(X,Y) : q(X) \ \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \ \#sum \{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  \[ :- q(X), p(X,C) \ [C] \]
  \[ \#minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- **Variables**
  - \( p(X) :- q(X) \)
- **Conditional literals**
  - \( p :- q(X) : r(X) \)
- **Disjunction**
  - \( p(X) ; q(X) :- r(X) \)
- **Integrity constraints**
  - \( :- q(X), p(X) \)
- **Choice**
  - \( 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \)
- **Aggregates**
  - \( s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \)
- **Optimization**
  - \( :\sim q(X), p(X,C) [C] \)
  - \( \#\text{minimize} \{ C : q(X), p(X,C) \} \)
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization

  - Weak constraints
    \[ :\sim q(X), p(X,C) [C] \]

    \#minimize \{ C : q(X), p(X,C) \}
Language constructs

- Variables
  \[ p(X) : \neg q(X) \]

- Conditional literals
  \[ p : \neg q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) : \neg r(X) \]

- Integrity constraints
  \[ :\neg q(X) , p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :\neg r(Y) \]

- Aggregates
  \[ s(Y) : \neg r(Y) , 2 \#sum \{ X : p(X,Y) , q(X) \} 7 \]

- Optimization
  - Weak constraints
  \[ :\sim q(X) , p(X,C) [C] \]
  - Statements
  \[ \#minimize \{ C : q(X) , p(X,C) \} \]
Language constructs

- **Variables**
  \[ p(X) :- q(X) \]

- **Conditional literals**
  \[ p :- q(X) : r(X) \]

- **Disjunction**
  \[ p(X) ; q(X) :- r(X) \]

- **Integrity constraints**
  \[ :- q(X), p(X) \]

- **Choice**
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- **Aggregates**
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- **Optimization**
  - **Weak constraints**
    \[ :) q(X), p(X,C) [C] \]
  - **Statements**
    \[ #minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- **Variables**
  
  ```prolog
  p(X) :- q(X)
  ```

- **Conditional literals**
  
  ```prolog
  p :- q(X) : r(X)
  ```

- **Disjunction**
  
  ```prolog
  p(X) ; q(X) :- r(X)
  ```

- **Integrity constraints**
  
  ```prolog
  :- q(X), p(X)
  ```

- **Choice**
  
  ```prolog
  2 { p(X,Y) : q(X) } 7 :- r(Y)
  ```

- **Aggregates**
  
  ```prolog
  s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
  ```

- **Optimization**
  
  - **Weak constraints**
    
    ```prolog
    :~ q(X), p(X,C) [C]
    ```

  - **Statements**
    
    ```prolog
    #minimize { C : q(X), p(X,C) }
    ```
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Multi-objective optimization
  - Weak constraints
    \[ \sim q(X), p(X,C) \ [C@42] \]
  - Statements
    \[ \#minimize \{ C@42 : q(X), p(X,C) \} \]
Basic methodology

Methodology

**Generate and Test**  (or: **Guess and Check**)

**Generator** Generate potential stable model candidates
   (typically through non-deterministic constructs)

**Tester** Eliminate invalid candidates
   (typically through integrity constraints)

Peanutshell

Logic program  =  Data + Generator + Tester ( + Optimizer)
**Basic methodology**

**Methodology**

**Generate and Test** (or: Guess and Check)

**Generator**
- Generate potential stable model candidates
  - (typically through non-deterministic constructs)

**Tester**
- Eliminate invalid candidates
  - (typically through integrity constraints)

**Peanutshell**

Logic program = Data + Generator + Tester (+ Optimizer)
Satisfiability testing

\[(a \leftrightarrow b) \land c\]
Satisfiability testing

\[(a \leftrightarrow b) \land c\]

\{ a ; b ; c \}.

:- not a, b.
:- a, not b.
:- not c.
Maximum satisfiability testing

“(a ↔ b) ∧ c”

{ a ; b ; c }.

:- not a, b.
:- a, not b.
:- not c.

{ a ; b ; c }.

:- not a, b.
:- a, not b. [10@2]
:- not c. [100@1]
n-Queens
Basic encoding

\{ \text{queen}(1..n,1..n) \}.

:- \{ \text{queen}(I,J) \} \neq n.
:- \text{queen}(I,J), \text{queen}(I,JJ), J \neq JJ.
:- \text{queen}(I,J), \text{queen}(II,J), I \neq II.
:- \text{queen}(I,J), \text{queen}(II,JJ), (I,J) \neq (II,JJ), I-J = II-JJ.
:- \text{queen}(I,J), \text{queen}(II,JJ), (I,J) \neq (II,JJ), I+J = II+JJ.
n-Queens
Advanced encoding

\{ \text{queen}(I,1..n) \} = 1 :- I = 1..n.
\{ \text{queen}(1..n,J) \} = 1 :- J = 1..n.

:- \{ \text{queen}(D-J,J) \} > 1, D = 2..2*n.
:- \{ \text{queen}(D+J,J) \} > 1, D = 1-n..n-1.
Traveling salesperson
Basic encoding

1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 \ :- \ \text{node}(X).
1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 \ :- \ \text{node}(Y).

\text{reached}(X) :- X = \min \{ Y : \text{node}(Y) \}.
\text{reached}(Y) :- \text{cycle}(X,Y), \text{reached}(X).

:- \text{node}(Y), \text{not} \ \text{reached}(Y).

\#\text{minimize} \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.

\text{node}(X) :- \text{edge}(X,\_).
\text{node}(X) :- \text{edge}(\_,X).

\text{edge}(1,2). \ \text{edge}(1,3). \ \text{edge}(1,4).
\text{edge}(2,4). \ \text{edge}(2,5). \ \text{edge}(2,6). \ [\ldots]
Traveling salesperson
Basic encoding

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

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

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

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.

node(X) :- edge(X,\_).
node(X) :- edge(\_,X).

dge(1,2). edge(1,3). edge(1,4).
dge(2,4). edge(2,5). edge(2,6). [...]

Torsten Schaub (KRR@UP)  Pocket calculators for combinatorial problems
Traveling salesperson
Basic encoding

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

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

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

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.

node(X) :- edge(X,_).
node(X) :- edge(_,X).

dge(1,2).  edge(1,3).  edge(1,4).
dge(2,4).  edge(2,5).  edge(2,6).  [...]
controls(X,Y) :-
    #sum+ { S: owns(X,Y,S);
      S,Z: controls(X,Z), owns(Z,Y,S) } > 50,
    company(X), company(Y), X != Y.

company(c_1). owns(c_1,c_2,60).
    owns(c_1,c_3,20).
company(c_2). owns(c_2,c_3,35).
company(c_3). owns(c_3,c_4,51).
company(c_4).
Reasoning modes

- ASP solvers offer
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization
  - and combinations of them

- For instance, `clasp` allows for
  - ASP solving (`smodels` format)
  - MaxSAT and SAT solving (extended `dimacs` format)
  - PB solving (`opb` and `wbo` format)
Reasoning modes

- **ASP solvers offer**
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization
  - and combinations of them

- **For instance, clasp allows for**
  - ASP solving (*smodes* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
Solving

Reasoning modes

- ASP solvers offer
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization
  - and combinations of them

- For instance, *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
    - Program Builder
- **Logic Program**
- **Solver 1...n**
  - Conflict Resolution
    - Decision Heuristic
      - Assignment Atoms/Bodies
        - Recorded Nogoods
  - Propagation
    - Unit Propagation
    - Post Propagation
- **ParallelContext**
  - Threads: $S_1, S_2, \ldots, S_n$
  - Counter: $T, W, \ldots, S$
  - Queue: $P_1, P_2, \ldots, P_n$
- **Enumerator**
  - Nogood Distributor
  - Shared Nogoods
1 Introduction

2 Modeling

3 Solving

4 Summary
Well then . . . ?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems

0 handy
1 easy to use
2 lots of operations
3 computes effectively

ASP may run on your cell phone ✗
ASP has a high-level modeling language ✓
ASP offers various reasoning modes ✓
ASP solvers are highly effective ✓
Well then . . . ?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems

0 handy
1 easy to use
2 lots of operations
3 computes effectively

ASP may run on your cell phone ✗
ASP has a high-level modeling language ✓
ASP offers various reasoning modes ✓
ASP solvers are highly effective ✓
Well then . . .?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems

0 handy
ASP may run on your cell phone

1 easy to use
ASP has a high-level modeling language

2 lots of operations
ASP offers various reasoning modes

3 computes effectively
ASP solvers are highly effective
ASPyclopedia

**Systems**
- asperix  http://www.info.univ-angers.fr/pub/claire/asperix
- assat   http://assat.cs.ust.hk
- clasp, gringo, clingo, etc.  http://potassco.sourceforge.net
- cmodels  http://www.cs.utexas.edu/users/tag/cmodels
- dlv   http://www.dlvsystem.com
- smodels, lparse, gnt  http://www.tcs.hut.fi/Software
- wasp   https://www.mat.unical.it/ricca/wasp
- sup  http://www.cs.utexas.edu/users/tag/sup

**User’s guides**
- DLV Systems
- Potassco
  http://sourceforge.net/projects/potassco/files/guide

**Literature**  [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.
ASPyclopedia

- **Systems** — *best suited for beginners*
  
  - clingo  
  
  - dlv  
    - http://www.dlvsystem.com

- **User’s guides**
  
  - DLV Systems  
  
  - Potassco  

- **Literature**  
  
  - [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.
ASPyclopedia

- **Systems**
  - clasp, gringo, clingo, etc.  [http://potassco.sourceforge.net](http://potassco.sourceforge.net)
  - dlv  [http://www.dlvsystem.com](http://www.dlvsystem.com)
  - smodels, lparse, gnt  [http://www.tcs.hut.fi/Software](http://www.tcs.hut.fi/Software)
  - wasp  [https://www.mat.unical.it/ricca/wasp](https://www.mat.unical.it/ricca/wasp)
  - sup  [http://www.cs.utexas.edu/users/tag/sup](http://www.cs.utexas.edu/users/tag/sup)

- **User's guides**

- **Literature**  [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.
Summary

ASPyclopedia

- **Systems**
  - clasp, gringo, clingo, etc.  [http://potassco.sourceforge.net](http://potassco.sourceforge.net)
  - dlv  [http://www.dlvsystem.com](http://www.dlvsystem.com)
  - smodels, lparse, gnt  [http://www.tcs.hut.fi/Software](http://www.tcs.hut.fi/Software)
  - wasp  [https://www.mat.unical.it/ricca/wasp](https://www.mat.unical.it/ricca/wasp)
  - sup  [http://www.cs.utexas.edu/users/tag/sup](http://www.cs.utexas.edu/users/tag/sup)

- **User’s guides**
  - DLV Systems
  - Potassco

- **Literature**  [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.
ASPyclopedia

■ Systems
  ■ asperix  http://www.info.univ-angers.fr/pub/claure/asperix
  ■ assat   http://assat.cs.ust.hk
  ■ clasp, gringo, clingo, etc.  http://potassco.sourceforge.net
  ■ cmodels http://www.cs.utexas.edu/users/tag/cmodels
  ■ dlv     http://www.dlvsystem.com
  ■ lp2*    http://research.ics.aalto.fi/software/asp
  ■ smodels, lparse, gnt  http://www.tcs.hut.fi/Software
  ■ wasp    https://www.mat.unical.it/ricca/wasp
  ■ sup     http://www.cs.utexas.edu/users/tag/sup

■ User’s guides
  ■ DLV Systems
  ■ Potassco
    http://sourceforge.net/projects/potassco/files/guide

■ Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.


Abstract Gringo. 

*Answer Set Solving in Practice*. 
Synthesis Lectures on Artificial Intelligence and Machine Learning. 

*Answer sets*. 


Summary


Logic programming and knowledge representation — the A-Prolog perspective.

The stable model semantics for logic programming.

Logic programs with classical negation.

Answer set programming and plan generation.

*Introduction to answer set programming.*

*Nonmonotonic logic: context-dependent reasoning.*

*Stable models and an alternative logic programming paradigm.*

*Logic programs with stable model semantics as a constraint programming paradigm.*