Performance Tuning in Answer Set Programming

Matt Buddenhagen and Yuliya Lierler

University of Nebraska at Omaha

Motivation and Questions to Address

- 2011: inception of ASP-based application PARSER (Lierler and Schüller, 2012)
- 2012: performance of PARSER is manually tuned using "hints on modeling" by Gebser, Kaminski, Kaufmann, and Schaub (2011): $\times 4$ faster
  - ⇒ What is performance tuning in ASP?
  - ⇒ Are "hints on modeling" reasonable ground for establishing performance tuning methodology within ASP?
- 2012: Automatic configuration tools were used to find best performing CLASP configuration for PARSER: $\times 3$ faster
  - ⇒ What is the true role of these tools: can they "replace" manual processes?

Outline

- Features of ASP shaping "non-imperative" performance tuning
- "Hints on modeling" (Gebser et al., 2011)
- Reconstruct manual performance tuning process for PARSER
- Review experiments with automated configuration tools for performance tuning on PARSER
- Conclude by drawing on the experimental findings

Performance Tuning in ASP Software Engineering

In ASP, first:

- tools for processing problem *encodings* are called *solvers*,
- connection between the encoding and solver's execution is very subtle
  - ⇒ performance analysis methods of imperative programming are not applicable

Second:

- applications in ASP are often NP complete resulting in significant computational effort by solvers
- typically a variety of ways to encode the same problem
- solvers offer different heuristics, expose numerous parameters, and are sensitive to these
  - ? What is Performance Tuning for ASP?

“Hints on modeling” (Gebser et al., 2011)

1. Keep the grounding compact:
   (i) If possible, use aggregates; (ii) Try to avoid combinatorial blow-up; (iii) Project out unused variables; (iv) But don't remove too many inferences!
2. Add additional constraints to prune the search space:
   (i) Consider special cases; (ii) Break symmetries; (iii) Test whether the additional constraints really help
3. Try different approaches to model the problem
4. It (still) helps to know the systems:
   (i) GRINGO offers options to trace the grounding process; (ii) CLASP offers many options to configure the search

⋆ *n-queens* problem as illustration

⇒ *Performance Guidelines*

Case Study: Overview

1. Reconstruct a way from PARSER-0.1 to PARSER-0.2
   - comprised 20 encodings
   - Performance Guidelines (PG) items 1 and 2 are followed
     - ⋆ keep the grounding compact
     - ⋆ add additional constraints to prune the search space
   - no change in *model* of a problem (no PG item 3)
   - grounding size and solving time: primary performance measures
     - ⇒ significant performance change: ×4 faster

   ⇒ ontology of rewriting techniques

   ⇒ performance tuning methodology
2. An automated configuration tool SMAC is used along the way from PARSER-0.1 to PARSER-0.2 for tuning the parameters of CLASP: PG item 4.

   ⇒ additional performance change: ×3 faster

PARSER Tuning: Ontology of Rewriting Techniques

- **Concretion** ($\mathcal{C}$) replaces overly general rules by their effectively used, partial instantiations.
- **Projection** ($\mathcal{P}$) reduces the number of variables in a rule to produce a fewer number of ground instances.
- **Simplification** ($\mathcal{S}$) eliminates some rules of a program that are "entailed" by the rest of the program.
- **Equivalence** ($\mathcal{E}$) replaces some rules of the program by strongly equivalent rules.
- **Auxiliary Signature Reduction** ($\mathcal{A}$) reduces the program's signature by reformulating problem specifications by means of fewer predicates.
- **Output Signature Change** ($\mathcal{O}$) changes the output signature of a program to allow different sets of predicates to encode the solution.

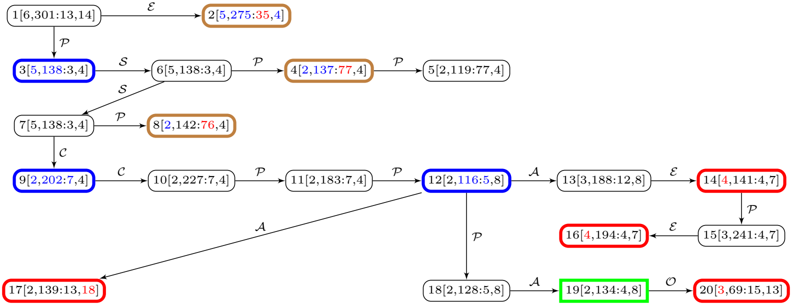
PARSER Tuning: Manual Process Methodology

- Smallest possible change per encoding revision was attempted
  - ⋆ e.g., in case of Auxiliary Signature Reduction only one predicate symbol at a time was eliminated
- Set of random 30 problem instances (Penn Treebank)
- Parameters used to evaluate the quality of each encoding:
  - ⋆ number of time or memory outs
  - ⋆ avg ground size
  - ⋆ avg solving time (CLASP-default)
  - ⋆ avg grounding time (GRINGO-default)

Nodes' form:

id[timeout,solving:grounding,ground-size]

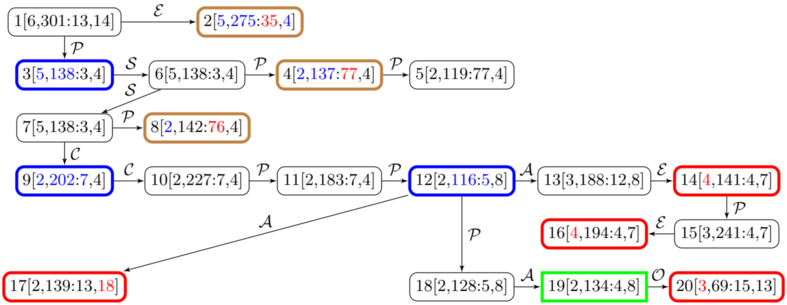
Encoding tree:



Better, when:

- number of timeouts is smaller, otherwise if
- avg ground size is smaller, otherwise if
- avg solving time is smaller, otherwise if
- avg grounding time is smaller.

PARSER Encoding Tree II



- ⇒ Projection $\mathcal{P}$ is most occurring rewriting technique
- ⇒ Output Signature Change $\mathcal{O}$ occurs only once, Simplification $\mathcal{S}$ and Concretion $\mathcal{C}$ occur twice each
- ⇒ $\mathcal{P}$ and $\mathcal{C}$ cause greatest changes for good
- ⇒ $\mathcal{S}$ seems fruitless
- ⇒ Equivalence $\mathcal{E}$ is ambivalent

PARSER Tuning: Automatic Algorithm Configuration

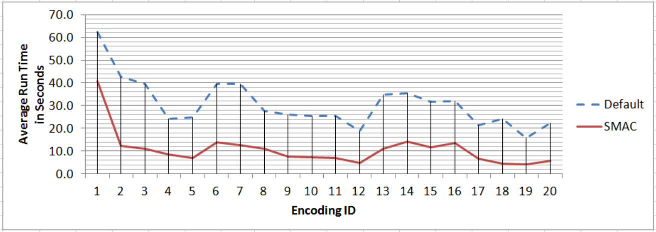
- In manual tuning only single CLASP-default was assumed
  - ? but what about CLASP sensitivity to parameters?

To answer this question, we consider

- Automatic configuration system SMAC by Hutter, Hoos, and Leyton-Brown (2011)
  - ⋆ apply SMAC to CLASP on each PARSER encoding
  - ⋆ a held-out set of 60 problem instances and a training set of 300 instances from PennTreebank of controlled hardness
  - ⋆ *cutoffTime* is 300 seconds
  - ⋆ *wallclock-limit* is 480000 seconds (5.56 days)
  - ⋆ *run-objective* is RUNTIME.
- 22 weeks to complete on a cluster with 24 virtual cores

PARSER Tuning: Results I

Encoding ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Default	62.4	42.6	39.7	24.2	24.9	39.6	39.7	27.7	26.1	25.5	25.4	19.0	35.0	35.5	31.6	32.0	21.3	24.4	15.8	22.3
SMAC	40.9	12.2	11.1	8.6	6.9	13.9	12.6	11.2	7.7	7.2	7.1	4.8	10.9	14.3	11.6	13.4	6.7	4.5	4.1	5.6



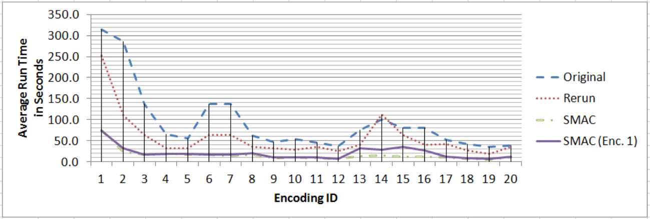
- ⋆ Curve of a tuned version is similar to that of default.
- ⋆ The winner identified by “default“ process stays the winner identified by “SMAC” process.

PARSER Tuning: Results II

- 30 random instances from manual tuning
- Original: 2012 results on CLASP-default
- Rerun: 2015 “SMAC platform” rerun of CLASP-default
- SMAC: SMAC-turned CLASP on individual encodings
- SMAC (Enc-1): SMAC-turned CLASP on encoding 1

Summary of Runtimes (Average for instances that never time out.)

Encoding ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Original	313.4	286.6	137.5	65.3	55.0	137.4	137.5	62.6	46.8	53.7	45.9	37.4	75.3	101.3	80.7	81.1	52.3	43.0	35.1	38.3
Rerun	251.8	111.2	64.6	32.7	32.2	63.3	63.4	36.4	32.9	28.8	35.8	26.0	41.5	112.1	64.3	41.4	42.9	28.0	18.6	35.6
SMAC	76.0	23.4	16.7	18.4	17.7	17.9	13.2	17.4	9.4	10.5	9.8	6.9	13.5	15.5	12.8	12.6	11.4	6.6	6.4	10.5
SMAC (Enc. 1)	74.4	31.6	17.7	19.8	18.9	17.7	17.7	20.4	10.3	10.1	10.1	8.0	32.4	29.0	35.0	28.1	12.1	8.5	7.6	12.0



PARSER Tuning: Results I and II Conclusions

- ⋆ Change from random instances to controlled hardness instances in tuning did not seem to change the winner
- ⋆ SMAC (Enc-1) seems to be rather close to SMAC in performance, yet it is substantially better than default

⇒ It makes sense to use automatic configuration tool early on in tuning to save time in the future

Conclusions and Future

- Performance Guidelines by Gebser et al. (2011) pave the way to a sensible performance tuning methodology in ASP
- Manual performance tuning of PARSER augmented by methodological evaluation process of incremental code changes is an illustration of this claim
- We believe that some if not all rewriting techniques discussed here can be automated
- Automation of this process is the future direction
- *Automatic* configuration tools are powerful tools for performance tuning yet their role is orthogonal to the role of *manual* rewriting techniques
- Another question for the future is whether we can combine automatic configuration tools with automatic rewriting processes (once available)

Thank you!

- Thanks
- Questions