SATenstein: Automatically Building Local Search SAT Solvers From Components

Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, Kevin Leyton-Brown
Department of Computer Science
University of British Columbia
Canada
SATenstein?

• Frankenstein
  – Create “perfect” human being from scavenged body parts

• SATenstein
  – Create high-performance SAT solvers using components scavenged from existing solvers
Algorithm Design Approach

• Traditional approach
  – Hard-code various design choices
  – Iteratively conduct small experiments to improve the design

• Our approach
  – Make all design options explicit, encoding them as parameters
    • Results in a generalized, highly parameterized algorithm
    • Instantiation produces many different solvers
  – Given a distribution, set the parameters using an automatic algorithm configuration procedure
SATenstein

• A highly parameterized, generalized SLS solver built on top of UBCSAT solver framework
  [Tompkins & Hoos, 2004]
  – 3 categories of SLS algorithms
    • WalkSAT, G²WSAT, dynamic local search algorithms
  – 25 known algorithms
  – 41 parameters
  – > 2 × 10¹¹ possible instantiations

• For each distribution, configured using ParamILS [Hutter et al., 2007-2009]
Related Work

• SLS SAT solvers
  – GSAT [Selman et al., 1992]
  – WalkSAT [Selman et al., 1994]
  – SAPS [Hutter et al., 2002]
  – gNovelty\textsuperscript{+} [Pham and Gretton, 2007]

• UBCSAT [Tompkins & Hoos, 2004]
  – SLS solver development framework

• Genetic programming [Fukunaga, 2002; 2004]
  – Evolve variable selection mechanism for SLS solver

• SATzilla [Xu et al., 2008]
  – Instance-based algorithm selection from portfolio of SAT solvers
SATenstein vs SATzilla

**SATzilla** [Xu et al., 2008]

- Relatively small number of known solvers
- Selects a given algorithm on a per-instance basis
- Creates empirical hardness model from given run-time data

**SATenstein**

- Can instantiate billions of solvers, most never studied before
- Selects a given configuration on a per-distribution basis
- Does not use runtime prediction

- The approaches are complementary
- SATenstein solvers can be used in SATzilla
  - Satzilla2009_R in SAT competition 2009
    - Gold in random SAT+UNSAT
    - 4th in random SAT
Performance objective

Penalized Average Runtime (PAR)

• Want: Minimize mean runtime
  • What about capped runs?

PAR = \text{avg}(\text{completed runs} + \text{penalty} \times \text{cutoff time})

• here: penalty = 10
Experimental setup

• 6 well-known distributions of SAT instances
  – Application/Industrial: FAC, CBMC-SE
  – Crafted: QCP, SW-GCP,
  – Random: HGEN, R3SAT

• 11 challenger algorithms (medal-winning SLS solvers in the 2003 - 2008 SAT competitions)
Automatic configurator: ParamILS 2.2 [Hutter et al., 2007-2009]

- Iterated local search (ILS) based automated parameter tuning tool

- Previously used to tune:
  - SPEAR, a highly parametric DPLL solver [Hutter et al., 2007a]
  - SLS algorithm for timetabling [Chiarandini et al., 2008]
  - CPLEX for mixed integer programming [Hutter et al., 2009]
Results

- Factor of **70 - 1000** performance improvement over best challenger on QCP, HGEN, CBMC-SE
- Factor of **1.4 - 2** performance improvement over best challenger on SW-GCP, R3SAT and FAC
- Improved the state-of-the-art across all the solvers on SW-GCP, QCP, HGEN and R3SAT
- On CBMC-SE and FAC, reduced the gap between complete solvers and SLS solvers
PAR comparison on QCP
PAR comparison on CBMC(SE)
SATenstein-LS vs Top 3 challengers on HGEN
SATenstein-LS vs Top 3 challengers on CBMC-SE
SATenstein-LS vs Oracle on CBMC-SE

- Oracle selects the challenger with minimum median runtime on a per-instance basis
Conclusion

• SATenstein: A new approach for building high-performance algorithms.
  – A framework that flexibly combines components from high-performance algorithms
  – A powerful algorithm configuration tool
• New state-of-the-art SAT solvers in 4 distributions
• Substantial improvement on 3 distributions (QCP, HGEN, CBMC-SE)
• Reduced gap between DPLL solvers and SLS solvers on CBMC-SE
Future Work

• Use of preprocessing
• Mixed strategies
• Better understanding of the configurations found
• More problem distributions / other problems