Computer-Aided Algorithm Design:

Automated Tuning, Configuration, Selection and Beyond

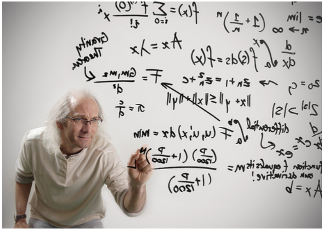
Holger H. Hoos

BETA Lab
Department of Computer Science
University of British Columbia
Canada

How to build better solvers
for hard problems?

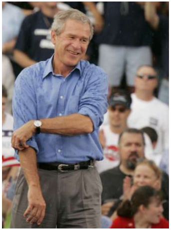
How to build better solvers for hard problems?

- construct a provably good solver



How to build better solvers for hard problems?

- construct a provably good solver
- roll up your sleeves and do the best you can



How to build better solvers for hard problems?

- construct a provably good solver
- roll up your sleeves and do the best you can

How to build better solvers for hard problems?

- construct a provably good solver
- roll up your sleeves and do the best you can
- use your little grey cells, then your little black chips

+

How to build better solvers for hard problems?

- construct a provably good solver
- roll up your sleeves and do the best you can
- use your little grey cells, then your little black chips

$\rightsquigarrow$ principled experimentation + generic techniques

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues

- enables effective exploration of larger design spaces

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms
- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues
- enables effective exploration of larger design spaces
- facilitates principled design of heuristic algorithms

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms
- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues
- enables effective exploration of larger design spaces
- facilitates principled design of heuristic algorithms
- profoundly changes how we build and use algorithms

High-performance heuristic algorithms are difficult to design

- many design choices (representation / search space; neighbourhoods; search strategy; variable/value selection heuristic; restart rules; pre-processing; data structures; ...)

High-performance heuristic algorithms are difficult to design

- many design choices (representation / search space; neighbourhoods; search strategy; variable/value selection heuristic; restart rules; pre-processing; data structures; ...)
- best performance often achieved by combination of various heuristics

High-performance heuristic algorithms are difficult to design

- many design choices (representation / search space; neighbourhoods; search strategy; variable/value selection heuristic; restart rules; pre-processing; data structures; ...)
- best performance often achieved by combination of various heuristics
  (Howe *et al.* 1999; Fox & Long 2001; Roberts *et al.* 2007; Richter & Westphal 2009; Valenzano *et al.* 2010; ...)

High-performance heuristic algorithms are difficult to design

- many design choices (representation / search space; neighbourhoods; search strategy; variable/value selection heuristic; restart rules; pre-processing; data structures; ...)
- best performance often achieved by combination of various heuristics
  (Howe *et al.* 1999; Fox & Long 2001; Roberts *et al.* 2007; Richter & Westphal 2009; Valenzano *et al.* 2010; . . .)
- various heuristic components interact in complex ways
  $\rightsquigarrow$ unexpected, emergent behaviour

High-performance heuristic algorithms are difficult to design

- many design choices (representation / search space; neighbourhoods; search strategy; variable/value selection heuristic; restart rules; pre-processing; data structures; ...)
- best performance often achieved by combination of various heuristics
  (Howe *et al.* 1999; Fox & Long 2001; Roberts *et al.* 2007; Richter & Westphal 2009; Valenzano *et al.* 2010; ...)
- various heuristic components interact in complex ways
  $\rightsquigarrow$ unexpected, emergent behaviour
- performance can be tricky to assess due to
  - differences in behaviour across problem instances
  - stochasticity

Therefore ...

- time-consuming design process, success often critically dependent on experience, intuition, luck

Therefore ...

- time-consuming design process, success often critically dependent on experience, intuition, luck
- resulting algorithms often complex, somewhat ad-hoc, not fully optimised

Real-world example:

- Application: Solving SAT-encoded software verification problems

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Given: High-performance DPLL-type SAT solver (SPEAR)
  - 26 parameters (7 categorical, 3 Boolean, 12 continuous, 4 integer-valued)
  - control variable/value ordering heuristics, clause learning, restarts, ...

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Given: High-performance DPLL-type SAT solver (SPEAR)
  - 26 parameters (7 categorical, 3 Boolean, 12 continuous, 4 integer-valued)
  - control variable/value ordering heuristics, clause learning, restarts, ...
- Goal: Minimize expected run-time on ‘typical’ SAT instances from software verification tool

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Given: High-performance DPLL-type SAT solver (SPEAR)
  - 26 parameters (7 categorical, 3 Boolean, 12 continuous, 4 integer-valued)
  - control variable/value ordering heuristics, clause learning, restarts, ...
- Goal: Minimize expected run-time on 'typical' SAT instances from software verification tool
- Problems:
  - default settings $\rightsquigarrow$ $\approx$ 300 seconds / run
  - good performance on some instances may not generalise

Outline

1. Introduction
2. From traditional to computer-aided algorithm design
3. Design spaces and design patterns
4. Meta-algorithmic search and optimisation procedures
5. Three success stories (SAT, timetabling, MIP)
6. The next step: Programming by Optimisation

From traditional to computer-aided algorithm design

Traditional algorithm design approach:

- iterative, manual process

From traditional to computer-aided algorithm design

Traditional algorithm design approach:

- iterative, manual process
- designer gradually introduces/modifies components or mechanisms

From traditional to computer-aided algorithm design

Traditional algorithm design approach:

- iterative, manual process
- designer gradually introduces/modifies components or mechanisms
- test performance on benchmark instances

From traditional to computer-aided algorithm design

Traditional algorithm design approach:

- iterative, manual process
- designer gradually introduces/modifies components or mechanisms
- test performance on benchmark instances
- design often starts from generic or broadly applicable problem solving method (*e.g.*, evolutionary algorithm)

Note:

- During the design process, many decisions are made.

Note:

- During the design process, many decisions are made.
- Some choices take the form of parameters,
  others are hard-coded.

Note:

- During the design process, many decisions are made.
- Some choices take the form of parameters,
  others are hard-coded.
- Design decisions interact in complex ways.

Problems:

- Design process is labour-intensive.

Problems:

- Design process is labour-intensive.
- Design decisions often made in *ad-hoc* fasion, based on limited experimentation and intuition.

Problems:

- Design process is labour-intensive.
- Design decisions often made in *ad-hoc* fasion, based on limited experimentation and intuition.
- Human designers typically over-generalise observations, explore few designs.

Problems:

- Design process is labour-intensive.
- Design decisions often made in *ad-hoc* fasion,
  based on limited experimentation and intuition.
- Human designers typically over-generalise observations,
  explore few designs.
- Implicit assumptions of independence, monotonicity
  are often incorrect.

Problems:

- Design process is labour-intensive.
- Design decisions often made in *ad-hoc* fasion, based on limited experimentation and intuition.
- Human designers typically over-generalise observations, explore few designs.
- Implicit assumptions of independence, monotonicity are often incorrect.
- Number of components and mechanisms tends to grow in each stage of design process.

Problems:

- Design process is labour-intensive.
- Design decisions often made in *ad-hoc* fasion, based on limited experimentation and intuition.
- Human designers typically over-generalise observations, explore few designs.
- Implicit assumptions of independence, monotonicity are often incorrect.
- Number of components and mechanisms tends to grow in each stage of design process.

$\rightsquigarrow$ complicated designs, unfulfilled performance potential

Solution: Computer-aided Algorithm Design

- Goal: construct high-performance algorithms automatically

Solution: Computer-aided Algorithm Design

- Goal: construct high-performance algorithms automatically
- Key idea: use fully formalised procedures to effectively explore large space of candidate designs

Solution: Computer-aided Algorithm Design

- Goal: construct high-performance algorithms automatically
- Key idea: use fully formalised procedures to effectively explore large space of candidate designs

$\rightsquigarrow$ genetic programming, hyper-heuristics, reactive search

Solution: Computer-aided Algorithm Design

- Goal: construct high-performance algorithms automatically
- Key idea: use fully formalised procedures to effectively explore large space of candidate designs

$\rightsquigarrow$ genetic programming, hyper-heuristics, reactive search; learning and intelligent optimisation, SLS engineering; meta-learning; program synthesis

Human designer:

- specifies (possibly large) space of candidate algorithm design

Human designer:

- specifies (possibly large) space of candidate algorithm design
- supplies set of problem instances for performance evaluation

Human designer:

- specifies (possibly large) space of candidate algorithm design
- supplies set of problem instances for performance evaluation
- specifies performance metric

Human designer:

- specifies (possibly large) space of candidate algorithm design
- supplies set of problem instances for performance evaluation
- specifies performance metric

Meta-algorithmic system:

- explores design space in principled manner

Human designer:

- specifies (possibly large) space of candidate algorithm design
- supplies set of problem instances for performance evaluation
- specifies performance metric

Meta-algorithmic system:

- explores design space in principled manner
- evaluates candidate design

Human designer:

- specifies (possibly large) space of candidate algorithm design
- supplies set of problem instances for performance evaluation
- specifies performance metric

Meta-algorithmic system:

- explores design space in principled manner
- evaluates candidate design
- finds high-performance designs

Advantages:

- lets human designer focus on higher-level issues

Advantages:

- lets human designer focus on higher-level issues
- enables better exploration of larger design spaces

Advantages:

- lets human designer focus on higher-level issues
- enables better exploration of larger design spaces
- exploits complementary strengths of different approaches for solving a given problem

Advantages:

- lets human designer focus on higher-level issues
- enables better exploration of larger design spaces
- exploits complementary strengths of different approaches for solving a given problem
- uses principled, fully formalised methods for algorithm design

Advantages:

- lets human designer focus on higher-level issues
- enables better exploration of larger design spaces
- exploits complementary strengths of different approaches for solving a given problem
- uses principled, fully formalised methods for algorithm design
- can be used to customise algorithms for use in specific applications with minimal human effort

Example: SAT-based software verification

Hutter, Babic, HH, Hu (2007)

- **Goal:** Solve suite of SAT-encoded software verification instances as fast as possible

Example: SAT-based software verification

Hutter, Babic, HH, Hu (2007)

- **Goal:** Solve suite of SAT-encoded software verification instances as fast as possible
- new DPLL-style SAT solver SPEAR (by Domagoj Babic)

  = highly parameterised heuristic algorithm

  (26 parameters, $\approx 8.3 \times 10^{17}$ configurations)

Example: SAT-based software verification

Hutter, Babic, HH, Hu (2007)

- **Goal:** Solve suite of SAT-encoded software verification instances as fast as possible
- new DPLL-style SAT solver SPEAR (by Domagoj Babic)

  = highly parameterised heuristic algorithm

  (26 parameters, $\approx 8.3 \times 10^{17}$ configurations)
- manual configuration by algorithm designer

Example: SAT-based software verification

Hutter, Babic, HH, Hu (2007)

- **Goal:** Solve suite of SAT-encoded software verification instances as fast as possible
- new DPLL-style SAT solver SPEAR (by Domagoj Babic)

  = highly parameterised heuristic algorithm

  (26 parameters, $\approx 8.3 \times 10^{17}$ configurations)
- manual configuration by algorithm designer
- automated configuration using ParamILS, a generic algorithm configuration procedure

  Hutter, HH, Stützle (2007)

SPEAR: Empirical results on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec

SPEAR: Empirical results on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec

SPEAR: Empirical results on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec
SPEAR generic. opt. config.	302/302	35.9 CPU sec

Spear: Empirical results on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
Spear original	298/302	787.1 CPU sec
Spear generic. opt. config.	302/302	35.9 CPU sec
Spear specific. opt. config.	302/302	1.5 CPU sec

SPEAR: Empirical results on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec
SPEAR generic. opt. config.	302/302	35.9 CPU sec
SPEAR specific. opt. config.	302/302	1.5 CPU sec

- $\approx$ 500-fold speedup through use automated algorithm configuration procedure (ParamILS)
- new state of the art
  (winner of 2007 SMT Competition, QF_BV category)

Design spaces and design patterns

Special cases of computer-aided algorithm design:

- parameter optimisation (for given set of instances)

  Birattari *et al.* (2002); Adenso-Diaz & Laguna (2006),
  Hutter *et al.* (2007–9), Ansótegui *et al.* (2009); Bartz-Beielstein (2006)

Design spaces and design patterns

Special cases of computer-aided algorithm design:

- parameter optimisation (for given set of instances)

  Birattari *et al.* (2002); Adenso-Diaz & Laguna (2006),
  Hutter *et al.* (2007–9), Ansótegui *et al.* (2009); Bartz-Beielstein (2006)

- algorithm configuration from components
  (for given set of instances)

  Fukunaga (2002), Chiarandini *et al.* (2008), KhudaBukhsh *et al.* (2009)

Design spaces and design patterns

Special cases of computer-aided algorithm design:

- parameter optimisation (for given set of instances)

  Birattari *et al.* (2002); Adenso-Diaz & Laguna (2006),
  Hutter *et al.* (2007–9), Ansótegui *et al.* (2009); Bartz-Beielstein (2006)

- algorithm configuration from components
  (for given set of instances)

  Fukunaga (2002), Chiarandini *et al.* (2008), KhudaBukhsh *et al.* (2009)

- restart strategies

  Luby *et al.* (1993); Gagliolo & Schmidhuber (2007);
  Streeter *et al.* (2007)

Special cases of computer-aided algorithm design (2):

- instance-based algorithm configurators

  Hutter *et al.* (2006); Malitsky & Sellmann (2009)

Special cases of computer-aided algorithm design (2):

- instance-based algorithm configurators
  Hutter *et al.* (2006); Malitsky & Sellmann (2009)
- on-line algorithm control / reactive search
  Carchrae & Beck (2005); Battiti *et al.* (2008)

Special cases of computer-aided algorithm design (2):

- instance-based algorithm configurators

  Hutter *et al.* (2006); Malitsky & Sellmann (2009)

- on-line algorithm control / reactive search

  Carchrae & Beck (2005); Battiti *et al.* (2008)

- instance-based algorithm selection

  Rice (1976); Leyton-Brown *et al.* (2003); Guerri & Milano (2004); Xu *et al.* (2008)

Special cases of computer-aided algorithm design (2):

- instance-based algorithm configurators

  Hutter *et al.* (2006); Malitsky & Sellmann (2009)

- on-line algorithm control / reactive search

  Carchrae & Beck (2005); Battiti *et al.* (2008)

- instance-based algorithm selection

  Rice (1976); Leyton-Brown *et al.* (2003); Guerri & Milano (2004); Xu *et al.* (2008)

- algorithm portfolios (static and dynamic)

  Huberman *et al.* (1997), Gomes & Selman (2001); Gagliolo & Schmidhuber (2007)

Special cases of computer-aided algorithm design (2):

- instance-based algorithm configurators

  Hutter *et al.* (2006); Malitsky & Sellmann (2009)

- on-line algorithm control / reactive search

  Carchrae & Beck (2005); Battiti *et al.* (2008)

- instance-based algorithm selection

  Rice (1976); Leyton-Brown *et al.* (2003); Guerri & Milano (2004); Xu *et al.* (2008)

- algorithm portfolios (static and dynamic)

  Huberman *et al.* (1997), Gomes & Selman (2001); Gagliolo & Schmidhuber (2007)

$\rightsquigarrow$ meta-algorithmic design patterns, induce design spaces

Meta-algorithmic search and optimisation procedures

How to search design spaces?

- use powerful heuristic search and optimisation procedures, combined with significant amounts of computing power

Meta-algorithmic search and optimisation procedures

How to search design spaces?

- use powerful heuristic search and optimisation procedures, combined with significant amounts of computing power
- use machine learning methods (classification, regression), combined with significant amount of training data

Some examples:

- parameter tuning:
  - numerical optimisation techniques
    e.g., CMA-ES (Hansen & Ostermeier 2001)

Some examples:

- parameter tuning:
  - numerical optimisation techniques
    e.g., CMA-ES (Hansen & Ostermeier 2001)
  - model-based optimisation methods
    e.g., SPO (Bartz-Beielstein 2006),
    SPO$^+$, TB-SPO (Hutter *et al.* 2009–10)

Some examples:

- parameter tuning:
  - numerical optimisation techniques
    e.g., CMA-ES (Hansen & Ostermeier 2001)
  - model-based optimisation methods
    e.g., SPO (Bartz-Beielstein 2006),
    SPO^{+}, TB-SPO (Hutter *et al.* 2009–10)
- algorithm configuration:
  - genetic programming
    e.g., CLASS (Fukunaga 2002)

Some examples:

- parameter tuning:
  - numerical optimisation techniques
    e.g., CMA-ES (Hansen & Ostermeier 2001)
  - model-based optimisation methods
    e.g., SPO (Bartz-Beielstein 2006),
    SPO$^+$, TB-SPO (Hutter *et al.* 2009–10)
- algorithm configuration:
  - genetic programming
    e.g., CLASS (Fukunaga 2002)
  - racing procedures
    e.g., F-Race (Birattari *et al.* 2002)

Some examples:

- parameter tuning:
  - numerical optimisation techniques
    e.g., CMA-ES (Hansen & Ostermeier 2001)
  - model-based optimisation methods
    e.g., SPO (Bartz-Beielstein 2006),
    SPO^+, TB-SPO (Hutter *et al.* 2009–10)
- algorithm configuration:
  - genetic programming
    e.g., CLASS (Fukunaga 2002)
  - racing procedures
    e.g., F-Race (Birattari *et al.* 2002)
  - advanced stochastic local search procedures
    e.g., ParamILS (Hutter *et al.* 2007)

More examples:

- instance-based algorithm selection
  - classification approaches (*e.g.*, Guerri & Milano 2004)

More examples:

- instance-based algorithm selection
  - classification approaches (*e.g.*, Guerri & Milano 2004)
  - regression approaches (*e.g.*, Leyton-Brown *et al.* 2003, Xu *et al.* 2008)

More examples:

- instance-based algorithm selection
  - classification approaches (*e.g.*, Guerri & Milano 2004)
  - regression approaches (*e.g.*, Leyton-Brown *et al.* 2003, Xu *et al.* 2008)

- dynamic algorithm portfolios (time allocators)
  - bandit solvers (*e.g.*, Gagliolo & Schmidhuber 2007)

More examples:

- instance-based algorithm selection
  - classification approaches (*e.g.*, Guerri & Milano 2004)
  - regression approaches (*e.g.*, Leyton-Brown *et al.* 2003, Xu *et al.* 2008)
- dynamic algorithm portfolios (time allocators)
  - bandit solvers (*e.g.*, Gagliolo & Schmidhuber 2007)
  - evolutionary algorithms (*e.g.*, Harik & Lobo 1999)

Many open questions:

- Which procedure for which type of design space?

Many open questions:

- Which procedure for which type of design space?
- How to deal with hybrid design patterns?

Many open questions:

- Which procedure for which type of design space?
- How to deal with hybrid design patterns?
- How to best deal with censored, sparse data?

How good are current methods for computer-aided algorithm design?

Three success stories

How good are current methods for computer-aided algorithm design?

"The proof is in the pudding":

- Propositional Satisfiability
- Course Timetabling
- Mixed Integer Programming

Three success stories

How good are current methods for computer-aided algorithm design?

"The proof is in the pudding":

- Propositional Satisfiability
- Course Timetabling
- Mixed Integer Programming

Further successes:

- protein structure prediction (Thachuk *et al.* 2007)
- SAT (KhudaBukhsh *et al.* 2009; Xu *et al.* – to appear; Tompkins & HH – to appear)
- TSP (Styles & HH – in preparation)

SATzilla: Portfolio-based algorithm selection for SAT

Xu, Hutter, HH, Leyton-Brown (2008)

Key idea: Instance-based Algorithm Selection (Rice 1976)

- *Given:* set S of algorithms for a problem, problem instance π
- *Select* from S the algorithm expected to solve π *most efficiently*, based on (cheaply computable) *features* of π.

SATzilla: Portfolio-based algorithm selection for SAT

Xu, Hutter, HH, Leyton-Brown (2008)

Key idea: Instance-based Algorithm Selection (Rice 1976)

- *Given:* set S of algorithms for a problem, problem instance π
- *Select* from S the algorithm expected to solve π *most efficiently*, based on (cheaply computable) *features* of π.

SATzilla in a nutshell:

- CNF formula $\rightsquigarrow$ 84 polytime-computable instance features

SATzilla: Portfolio-based algorithm selection for SAT

Xu, Hutter, HH, Leyton-Brown (2008)

Key idea: Instance-based Algorithm Selection (Rice 1976)

- *Given:* set S of algorithms for a problem, problem instance π
- *Select* from S the algorithm expected to solve π *most efficiently*, based on (cheaply computable) *features* of π.

SATzilla in a nutshell:

- CNF formula $\rightsquigarrow$ 84 polytime-computable instance features
- features $\rightsquigarrow$ performance prediction for set of SAT solvers

SATzilla: Portfolio-based algorithm selection for SAT

Xu, Hutter, HH, Leyton-Brown (2008)

Key idea: Instance-based Algorithm Selection (Rice 1976)

- *Given:* set S of algorithms for a problem, problem instance π
- *Select* from S the algorithm expected to solve π *most efficiently*, based on (cheaply computable) *features* of π.

SATzilla in a nutshell:

- CNF formula $\rightsquigarrow$ 84 polytime-computable instance features
- features $\rightsquigarrow$ performance prediction for set of SAT solvers
- run solver with best predicted performance

Under the hood:

- Use state-of-the-art complete (DPLL) and incomplete (local search) SAT solvers.

Under the hood:

- Use state-of-the-art complete (DPLL) and incomplete (local search) SAT solvers.
- Use ridge regression on selected features to predict solver run-times from instance features.

Under the hood:

- Use state-of-the-art complete (DPLL) and incomplete (local search) SAT solvers.
- Use ridge regression on selected features to predict solver run-times from instance features.
- Use method by Schmee & Hahn (1979) to deal with censored run-time data.

Some bells and whistles:

- Use pre-solvers to solve ‘easy’ instances quickly.

Some bells and whistles:

- Use pre-solvers to solve ‘easy’ instances quickly.
- Build run-time predictors for various types of instances, use classifier to select best predictor based on instance features.

Some bells and whistles:

- Use pre-solvers to solve ‘easy’ instances quickly.
- Build run-time predictors for various types of instances, use classifier to select best predictor based on instance features.
- Predict time required for feature computation; if that time is too long, use back-up solver.

Some bells and whistles:

- Use pre-solvers to solve 'easy' instances quickly.
- Build run-time predictors for various types of instances, use classifier to select best predictor based on instance features.
- Predict time required for feature computation; if that time is too long, use back-up solver.

$\rightsquigarrow$ prizes in 5 of the 9 main categories of the 2009 SAT Solver Competition (3 gold, 2 silver medals)

Post-Enrolment Course Timetabling

Chiarandini, Fawcett, HH (2008); Fawcett, HH, Chiarandini (in preparation)

Post-Enrolment Course Timetabling:

- students enroll in courses
- courses are assigned to rooms and time slots, subject to *hard constraints*
- preferences are represented by *soft constraints*

Post-Enrolment Course Timetabling

Chiarandini, Fawcett, HH (2008); Fawcett, HH, Chiarandini (in preparation)

Post-Enrolment Course Timetabling:

- students enroll in courses
- courses are assigned to rooms and time slots, subject to *hard constraints*
- preferences are represented by *soft constraints*

Our solver:

- modular multiphase stochastic local search algorithm
- hard constraint solver: finds feasible course schedules
- soft constraint solver: optimise schedule (maintaining feasibility)

Our first solver:

- developed over ca. 1 month
- starting point: Chiarandini *et al.* (2003)
- *soft constraint solver* unchanged
- automatically configured *hard constraint solver*

Our first solver:

- developed over ca. 1 month
- starting point: Chiarandini *et al.* (2003)
- *soft constraint solver* unchanged
- automatically configured *hard constraint solver*

Design space for hard constraint solver:

- parameterised combination of constructive search, tabu search, diversification strategy
- 7 parameters, 50 400 configurations

Our first solver:

- developed over ca. 1 month
- starting point: Chiarandini *et al.* (2003)
- *soft constraint solver* unchanged
- automatically configured *hard constraint solver*

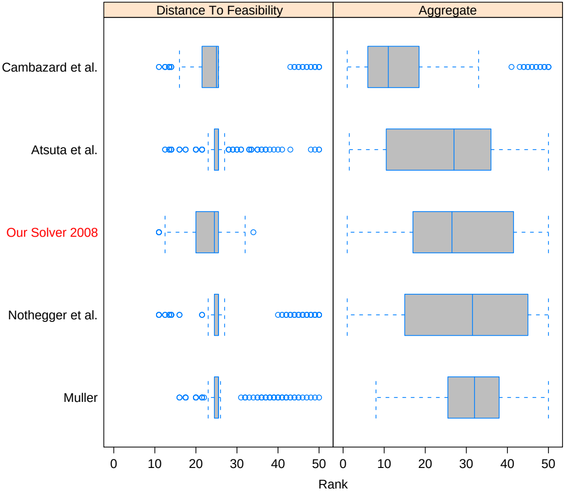
Design space for hard constraint solver:

- parameterised combination of constructive search, tabu search, diversification strategy
- 7 parameters, 50 400 configurations

Automated configuration process:

- configurator: FocusedILS 2.3 (Hutter *et al.* 2009)
- performance objective: solution quality after 300 CPU sec

2nd International Timetabling Competition (ITC), Track 2



Our latest solver:

- developed over ca. 6 months
- starting point: our previous solver
- automatically configured *hard & soft constraint* solvers

Our latest solver:

- developed over ca. 6 months
- starting point: our previous solver
- automatically configured *hard & soft constraint* solvers

Design space for soft constraint solver:

- highly parameterised simulated annealing algorithm
- 11 parameters, 2.7×10^9 configurations

Our latest solver:

- developed over ca. 6 months
- starting point: our previous solver
- automatically configured *hard & soft constraint* solvers

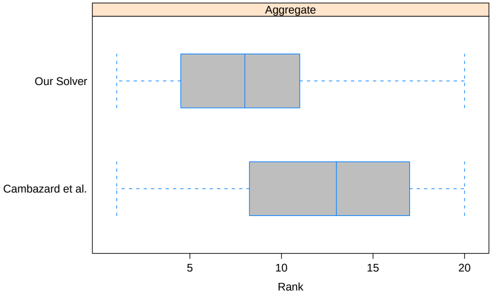
Design space for soft constraint solver:

- highly parameterised simulated annealing algorithm
- 11 parameters, 2.7×10^9 configurations

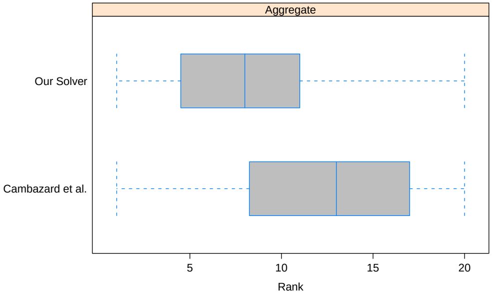
Automated configuration process:

- configurator: FocusedILS 2.4 (new version, multiple stages)
- multiple performance objectives
  (final stage: solution quality after 600 CPU sec)

2-way race against ITC Track 2 winner



2-way race against ITC Track 2 winner



- our solver wins beats ITC winner on 20 out of 24 competition instances
- application to university-wide exam scheduling at UBC
  ($\approx$ 1650 exams, 28 000 students)

Mixed Integer Programming (MIP)

Hutter, HH, Leyton-Brown, Stützle (2009); Hutter, HH, Leyton-Brown (2010)

- MIP is widely used for modelling optimisation problems
- MIP solvers play an important role for solving broad range of real-world problems

CPLEX:

- prominent and widely used commercial MIP solver
- exact solver, based on sophisticated branch & cut algorithm and numerous heuristics
- 159 parameters, 81 directly control search process

"A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models."

[CPLEX 12.1 user manual, p. 478]

"A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models."

[CPLEX 12.1 user manual, p. 478]

Automatically Configuring CPLEX:

- starting point: factory default settings
- 63 parameters (some with 'AUTO' settings)
- 1.38×10^{37} configurations

"A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models."

[CPLEX 12.1 user manual, p. 478]

Automatically Configuring CPLEX:

- starting point: factory default settings
- 63 parameters (some with 'AUTO' settings)
- 1.38×10^{37} configurations
- configurator: FocusedILS 2.3 (Hutter *et al.* 2009)
- performance objective: minimal mean run-time
- configuration time: 10×2 CPU days

"A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models."

[CPLEX 12.1 user manual, p. 478]

Automatically Configuring CPLEX:

- starting point: factory default settings
- 63 parameters (some with 'AUTO' settings)
- 1.38×10^{37} configurations
- configurator: FocusedILS 2.3 (Hutter *et al.* 2009)
- performance objective: minimal mean run-time
- configuration time: 10×2 CPU days

CPLEX on various MIPS benchmarks

Benchmark	Default performance [CPU sec]	Optimised performance [CPU sec]	Speedup factor
BCOL/CONIC.SCH	5.37	2.35 (2.4 ± 0.29)	2.2
BCOL/CLS	712	23.4 (327 ± 860)	30.4
BCOL/MIK	64.8	1.19 (301 ± 948)	54.4
CATS/REGIONS200	72	10.5 (11.4 ± 0.9)	6.8
RNA-QP	969	525 (827 ± 306)	1.8

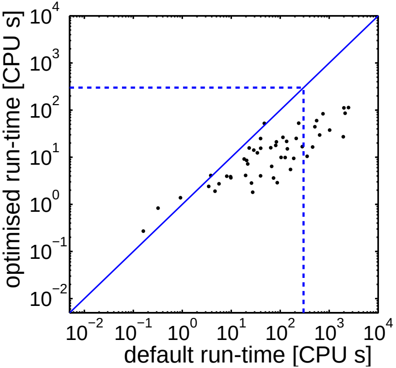
(Timed-out runs are counted as $10 \times$ cutoff time.)

CPLEX on various MIPS benchmarks

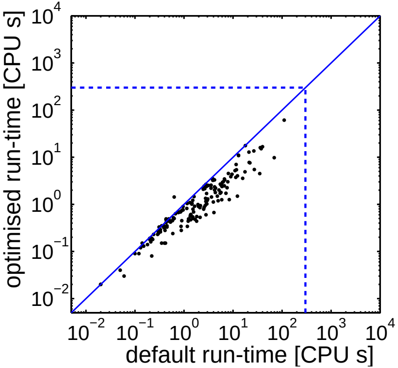
Benchmark	Default performance [CPU sec]	Optimised performance [CPU sec]	Speedup factor
BCOL/CONIC.SCH	5.37	2.35 (2.4 ± 0.29)	2.2
BCOL/CLS	712	23.4 (327 ± 860)	30.4
BCOL/MIK	64.8	1.19 (301 ± 948)	54.4
CATS/REGIONS200	72	10.5 (11.4 ± 0.9)	6.8
RNA-QP	969	525 (827 ± 306)	1.8

(Timed-out runs are counted as 10 $\times$ cutoff time.)

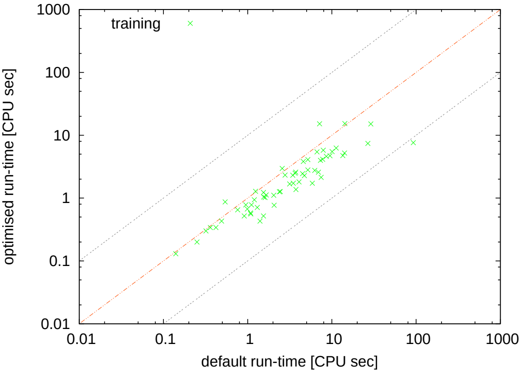
CPLEX on BCOL/CLS



CPLEX on BCOL/CONIC.SCH

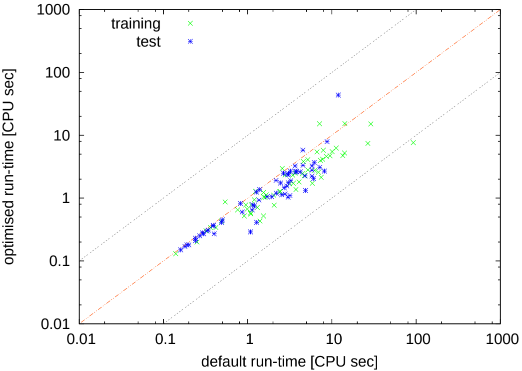


Latest results: Gurobi on BCOL/MIK



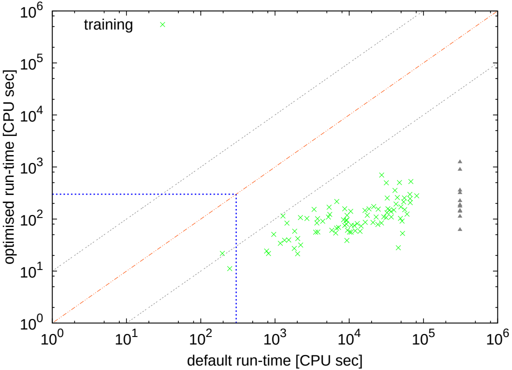
Configuration time: 10×2 CPU days

Latest results: Gurobi on BCOL/MIK



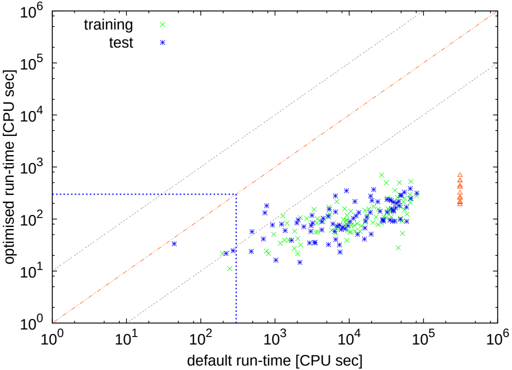
Configuration time: 10×2 CPU days

Latest results: lpsolve on CA-WDP



Configuration time: 10×2 CPU days

Latest results: lpsolve on CA-WDP



Configuration time: 10×2 CPU days

How to use computer-aided algorithm design?

How to use computer-aided algorithm design?

application context

How to use computer-aided algorithm design?

application context

$+$

design space

How to use computer-aided algorithm design?

application context

$+$

design space

$+$

optimisation procedure

How to use computer-aided algorithm design?

application context

$+$

design space

$+$

optimisation procedure

$+$

compute power

How to use computer-aided algorithm design?

application context

$+$

design space

$+$

optimisation procedure

$+$

compute power

$=$

success

The next step: Programming by Optimisation

How to *easily* use computer-aided algorithm design?

The next step: Programming by Optimisation

How to *easily* use computer-aided algorithm design?

Need effective support for ...

- specification of rich design spaces

The next step: Programming by Optimisation

How to *easily* use computer-aided algorithm design?

Need effective support for ...

- specification of rich design spaces
- automated design (and analysis) process

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*
- support wide range of design patterns, procedures

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*
- support wide range of design patterns, procedures
- support effective utilisation of parallel computation

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*
- support wide range of design patterns, procedures
- support effective utilisation of parallel computation
- support multiple platforms
  (Linux, MacOS; *later:* Windows, Chrome OS?)

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*
- support wide range of design patterns, procedures
- support effective utilisation of parallel computation
- support multiple platforms
  (Linux, MacOS; *later:* Windows, Chrome OS?)
- web-based UI, component-based architecture

HAL: High-performance Algorithm Lab

Nell, Fawcett, HH, Leyton-Brown (under review)

- support *algorithm design* and *empirical analysis*
- support wide range of design patterns, procedures
- support effective utilisation of parallel computation
- support multiple platforms
  (Linux, MacOS; *later:* Windows, Chrome OS?)
- web-based UI, component-based architecture
- open source, easy to use & expand

HAL 1.0

HAL 1.0	home	1q/3r	?

New Tasks

Evaluate algorithm performance
Analyse performance of an algorithm on an instance set.

Compare algorithm performance
Compare the performance of two algorithms on an instance set.

Configure algorithm
Optimize parameter settings to maximize algorithm performance on an instance set.

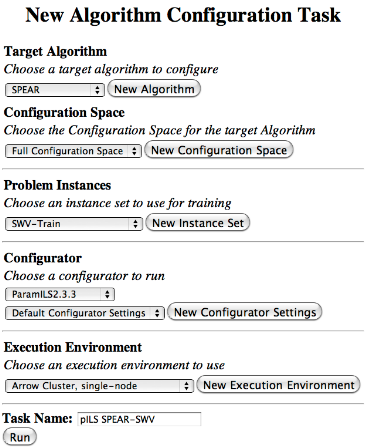
Active Tasks

Status	ID	Name	Start Time	CPU Time	
queued	5	Compare GGA/PILS SPEAR	N/A	0 s	
99%	3	ParamILS SPEAR SWV	2010-04-02 17:25:05.0	258122.70 s	KILL
97%	4	GGA SPEAR SWV	2010-04-02 17:31:31.0	253904.81 s	KILL
8%	6	Compare GGA/PILS SATenstein	2010-04-05 08:47:26.0	1322.16 s	KILL

Completed Tasks

Status	ID	Name	Start Time	CPU Time
done	1	ParamILS SATenstein QCP	2010-04-02 15:07:35.0	188920.02 s
done	2	GGA SATenstein QCP	2010-04-02 15:08:41.0	181342.54 s

HAL 1.0


New Algorithm Configuration Task
Target Algorithm
Choose a target algorithm to configure
SPEAR
New Algorithm
Configuration Space
Choose the Configuration Space for the target Algorithm
Full Configuration Space
New Configuration Space
Problem Instances
Choose an instance set to use for training
SWV-Train
New Instance Set
Configurator
Choose a configurator to run
ParamILS2.3.3
Default Configurator Settings
New Configurator Settings
Execution Environment
Choose an execution environment to use
Arrow Cluster, single-node
New Execution Environment
Task Name:
pILS SPEAR-SWV
Run

HAL 1.0

HAL 1.0 home 1q/3r ?

New Tasks

Evaluate algorithm performance

Analyse performance of an algorithm on an instance set.

Compare algorithm performance

Compare the performance of two algorithms on an instance set.

Configure algorithm

Optimize parameter settings to maximize algorithm performance on an instance set.

Active Tasks

Status	ID	Name	Start Time	CPU Time	
queued	5	Compare GGA/PILS SPEAR	N/A	0 s	
99%	3	ParamILS SPEAR SWV	2010-04-02 17:25:05.0	258122.70 s	KILL
97%	4	GGA SPEAR SWV	2010-04-02 17:31:31.0	253904.81 s	KILL
8%	6	Compare GGA/PILS SATenstein	2010-04-05 08:47:26.0	1322.16 s	KILL

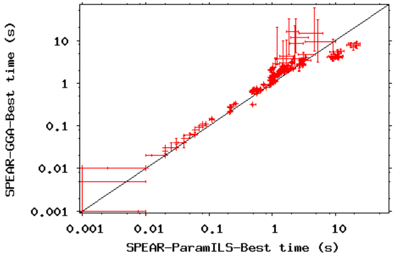
Completed Tasks

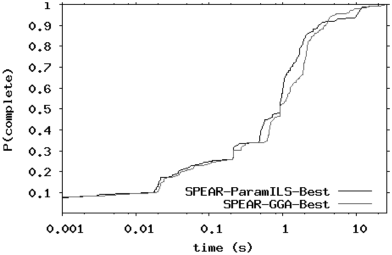
Status	ID	Name	Start Time	CPU Time
done	1	ParamILS SATenstein QCP	2010-04-02 15:07:35.0	188920.02 s
done	2	GGA SATenstein QCP	2010-04-02 15:08:41.0	181342.54 s

HAL 1.0

Wilcoxon signed-rank test: p=2.63E-15
Wilcoxon winner: SPEAR-ParamILS-Best
Spearman correlation test: p=2.22E-208

	q10	q25	q50	q75	q90	mean	sd
SPEAR-ParamILS-Best	0.017	0.104	0.922	1.670	3.171	1.712	3.216
SPEAR-GGA-Best	0.019	0.135	0.926	2.073	3.888	1.747	3.052





Programming by Optimisation (PbO)

HH (work in progress)

Programming by Optimisation (PbO)

HH (work in progress)

Key idea:

- avoid premature, uninformed, possibly detrimental design choices

- encourage developers to parameterise, provide functionally equivalent alternatives

Programming by Optimisation (PbO)

HH (work in progress)

Key idea:

- avoid premature, uninformed, possibly detrimental design choices

- encourage developers to parameterise, provide functionally equivalent alternatives

- automatically make choices to obtain algorithm / software / system that performs well in a given application context

Programming by Optimisation (PbO)

HH (work in progress)

Key idea:

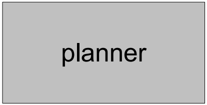
- avoid premature, uninformed, possibly detrimental design choices
- encourage developers to parameterise, provide functionally equivalent alternatives
  $\rightsquigarrow$ generic programming language extension
- automatically make choices to obtain algorithm / software / system that performs well in a given application context

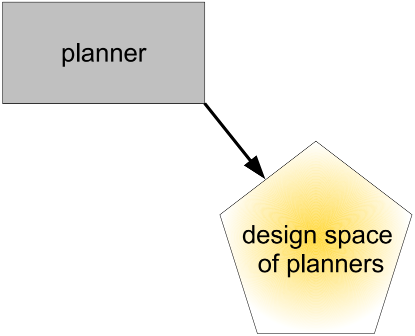
Programming by Optimisation (PbO)

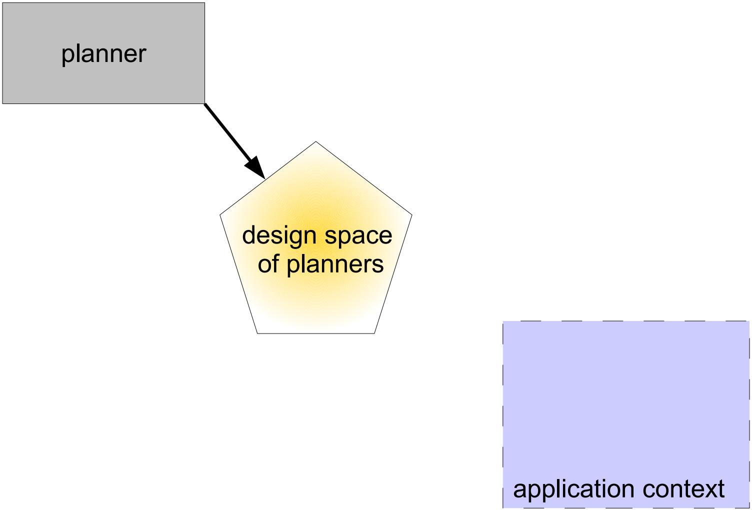
HH (work in progress)

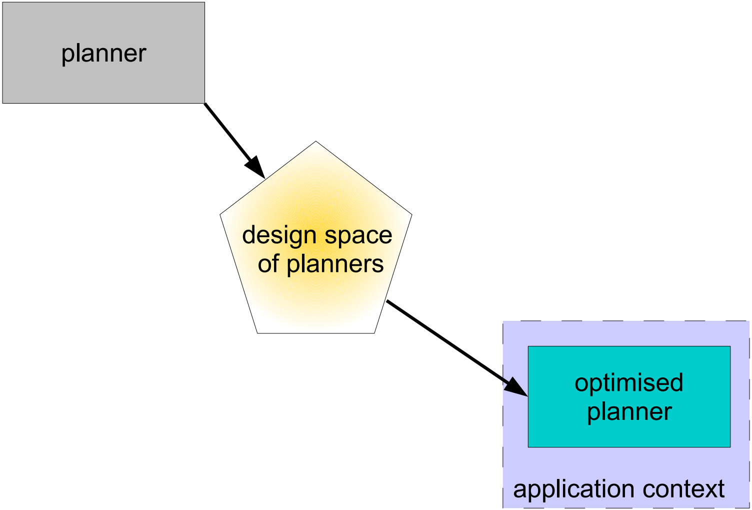
Key idea:

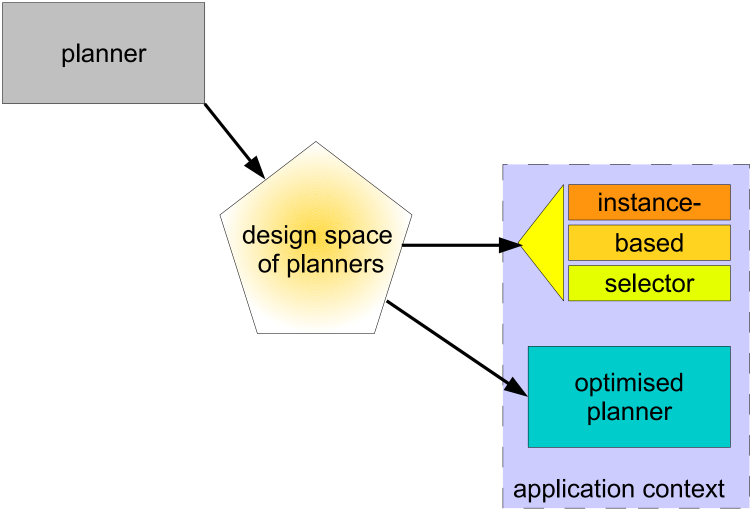
- avoid premature, uninformed, possibly detrimental design choices
- encourage developers to parameterise, provide functionally equivalent alternatives
  $\rightsquigarrow$ generic programming language extension
- automatically make choices to obtain algorithm / software / system that performs well in a given application context
  $\rightsquigarrow$ HAL + compute power

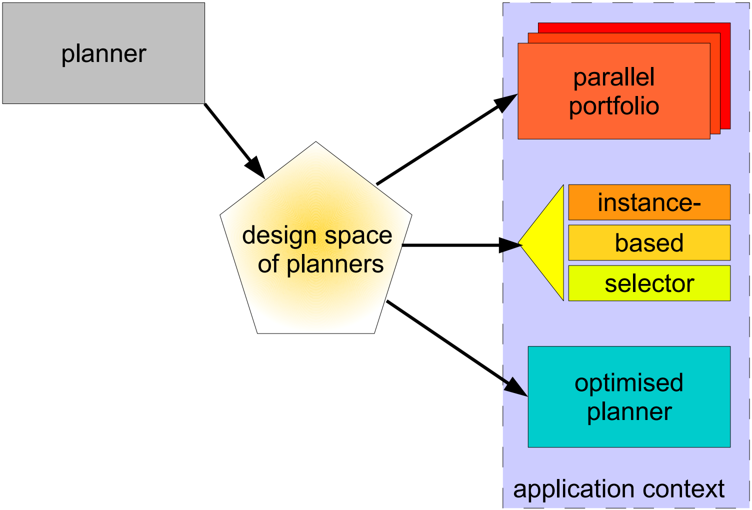

planner

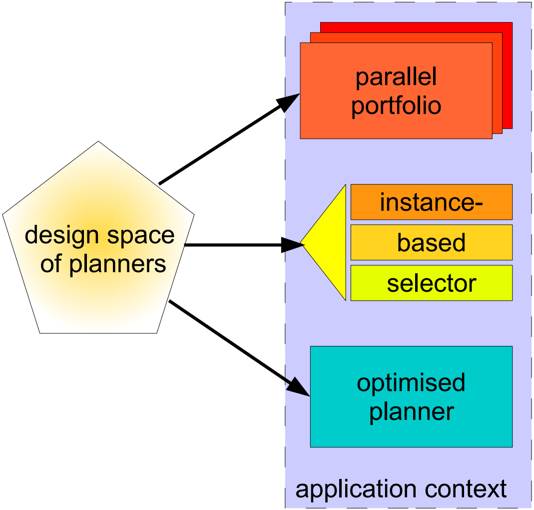

planner
design space
of planners


planner
design space
of planners
application context


planner
design space of planners
optimised planner
application context


planner
design space
of planners
instance-
based
selector
optimised
planner
application context


planner
design space of planners
parallel portfolio
instance-
based
selector
optimised planner
application context


parallel
portfolio
design space
of planners
instance-
based
selector
optimised
planner
application context

Computationally too expensive?

Recent example: Hydra SAT solver (Xu *et al.* – to appear)

- automated construction of the solver
  (using ParamILS, SATzilla):
  $\approx$ 70 CPU days

Computationally too expensive?

Recent example: Hydra SAT solver (Xu *et al.* – to appear)

- automated construction of the solver
  (using ParamILS, SATzilla):
  $\approx$ 70 CPU days
- wall-clock time on 10 CPU cluster:
  $\approx$ 7 CPU days

Computationally too expensive?

Recent example: Hydra SAT solver (Xu *et al.* – to appear)

- automated construction of the solver
  (using ParamILS, SATzilla):
  $\approx$ 70 CPU days
- wall-clock time on 10 CPU cluster:
  $\approx$ 7 CPU days
- cost on Amazon Elastic Compute Cloud (EC2):
  20.16 USD

Computationally too expensive?

Recent example: Hydra SAT solver (Xu *et al.* – to appear)

- automated construction of the solver
  (using ParamILS, SATzilla):
  $\approx$ 70 CPU days

- wall-clock time on 10 CPU cluster:
  $\approx$ 7 CPU days

- cost on Amazon Elastic Compute Cloud (EC2):
  20.16 USD

- 20.16 USD pays for ...
  - 0:38 hours of average software engineer

Computationally too expensive?

Recent example: Hydra SAT solver (Xu *et al.* – to appear)

- automated construction of the solver
  (using ParamILS, SATzilla):
  $\approx$ 70 CPU days
- wall-clock time on 10 CPU cluster:
  $\approx$ 7 CPU days
- cost on Amazon Elastic Compute Cloud (EC2):
  20.16 USD
- 20.16 USD pays for ...
  - 0:38 hours of average software engineer
  - 2:45 hours at minimum wage

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms

- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues

- enables effective exploration of larger design spaces

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms
- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues
- enables effective exploration of larger design spaces
- facilitates principled design of heuristic algorithms

Computer-aided Algorithm Design ...

- leverages computational power to construct better algorithms
- liberates human designers from boring, menial tasks and lets them focus on higher-level design issues
- enables effective exploration of larger design spaces
- facilitates principled design of heuristic algorithms
- profoundly changes how we build and use algorithms

Acknowledgements

Collaborators:

- Domagoj Babic
- Alex Devkar
- Chris Fawcett
- Frank Hutter
- Chris Nell
- Eugene Nudelman
- Alena Shmygelska
- Chris Thachuk
- James Styles
- Lin Xu

- Thomas Bartz-Beielstein
  (FH Köln)
- Marco Chiarandini
  (University of Southern Denmark)
- Alan Hu
- Kevin Leyton-Brown
- Kevin Murphy
- Yoav Shoham
  (Stanford University)
- Thomas Stützle
  (Université Libre de Bruxelles)

Research funding:

- NSERC, MITACS, CFI
- IBM, Actenum Corp.

Computing resources:

- Arrow, BETA, ICICS clusters
- WestGrid