A personal perspective on the past and the future of automated algorithm design

Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles (ULB)
Brussels, Belgium

stuetzle@ulb.ac.be
iridia.ulb.ac.be/~stuetzle







Context

The algorithmic solution of hard optimization problems is one of the OR/AI success stories!

- Exact (systematic search) algorithms
  - Branch&Bound (A^*), Branch&Cut, constraint programming, ..
  - powerful general-purpose software available
  - guarantees on optimality but often time/memory consuming
- Approximate algorithms
  - constructive algorithm, local search, metaheuristics, ...
  - typically special-purpose software
  - rarely provable guarantees but often fast and accurate

Much active research on hybrids between exact and approximate algorithms!

Design choices and parameters everywhere

Todays high-performance optimizers involve a large number of design choices and parameter settings

- exact solvers
  - design choices include alternative models, pre-processing, variable selection, value selection, branching rules ...
  - many design choices have associated numerical parameters
  - example: SCIP 3.0.1 solver (fastest non-commercial MIP solver) has more than 200 relevant parameters that influence the solver's search mechanism
- approximate algorithms
  - design choices include solution representation, operators, neighborhoods, pre-processing, strategies, ...
  - many design choices have associated numerical parameters
  - example: hybrid SLS algorithm with about 500+ parameters

Parameter types

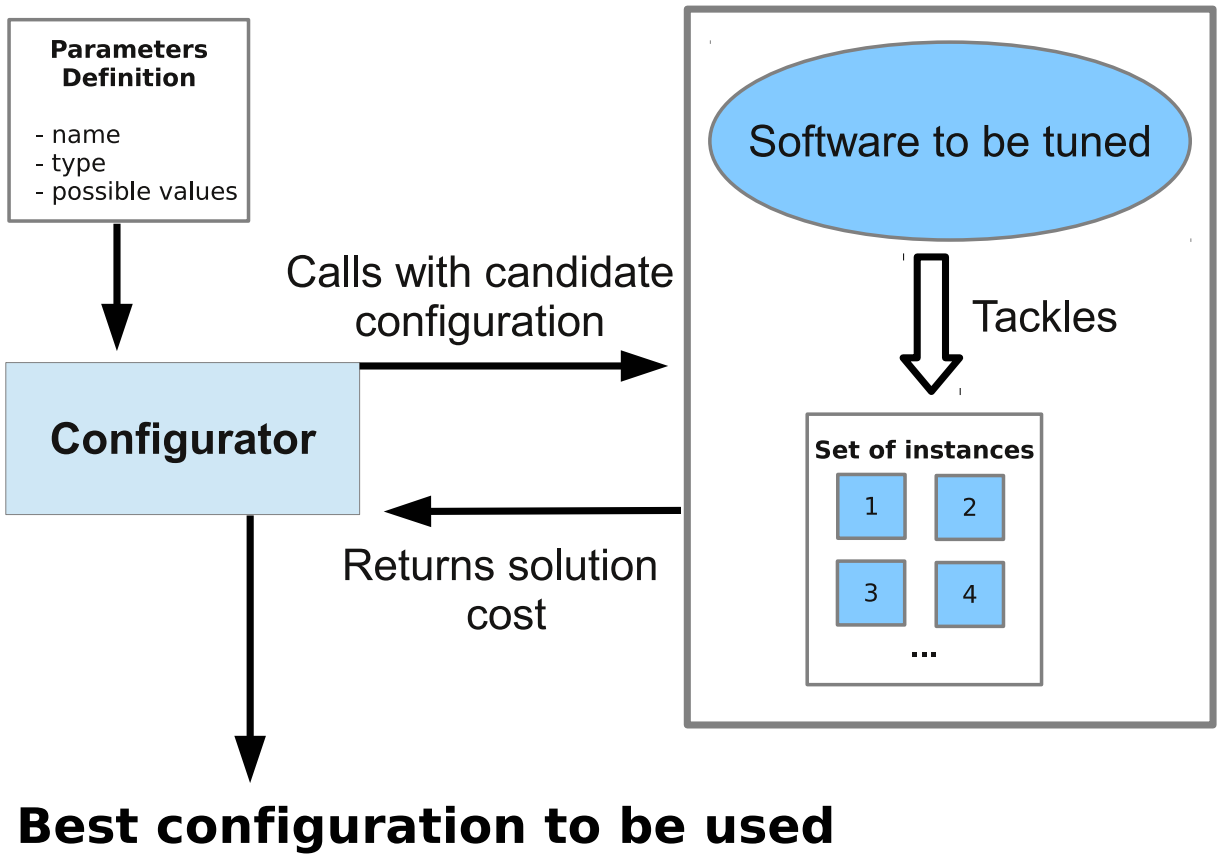
- *categorical* parameters **design**
  - choice of constructive procedure, choice of recombination operator, choice of branching strategy,...
- *ordinal* parameters **design**
  - neighborhoods, lower bounds, ...
- *numerical* parameters **tuning, calibration**
  - integer or real-valued parameters
  - weighting factors, population sizes, temperature, hidden constants, ...
  - numerical parameters may be *conditional* to specific values of categorical or ordinal parameters

Parameter types

- *categorical* parameters **design**
  - choice of constructive procedure, choice of recombination operator, choice of branching strategy,...
- *ordinal* parameters **design**
  - neighborhoods, lower bounds, ...
- *numerical* parameters **tuning, calibration**
  - integer or real-valued parameters
  - weighting factors, population sizes, temperature, hidden constants, ...
  - numerical parameters may be *conditional* to specific values of categorical or ordinal parameters

Design and configuration of algorithms involves setting categorical, ordinal, **and** *numerical parameters*

Automated algorithm configuration



Typical performance measures

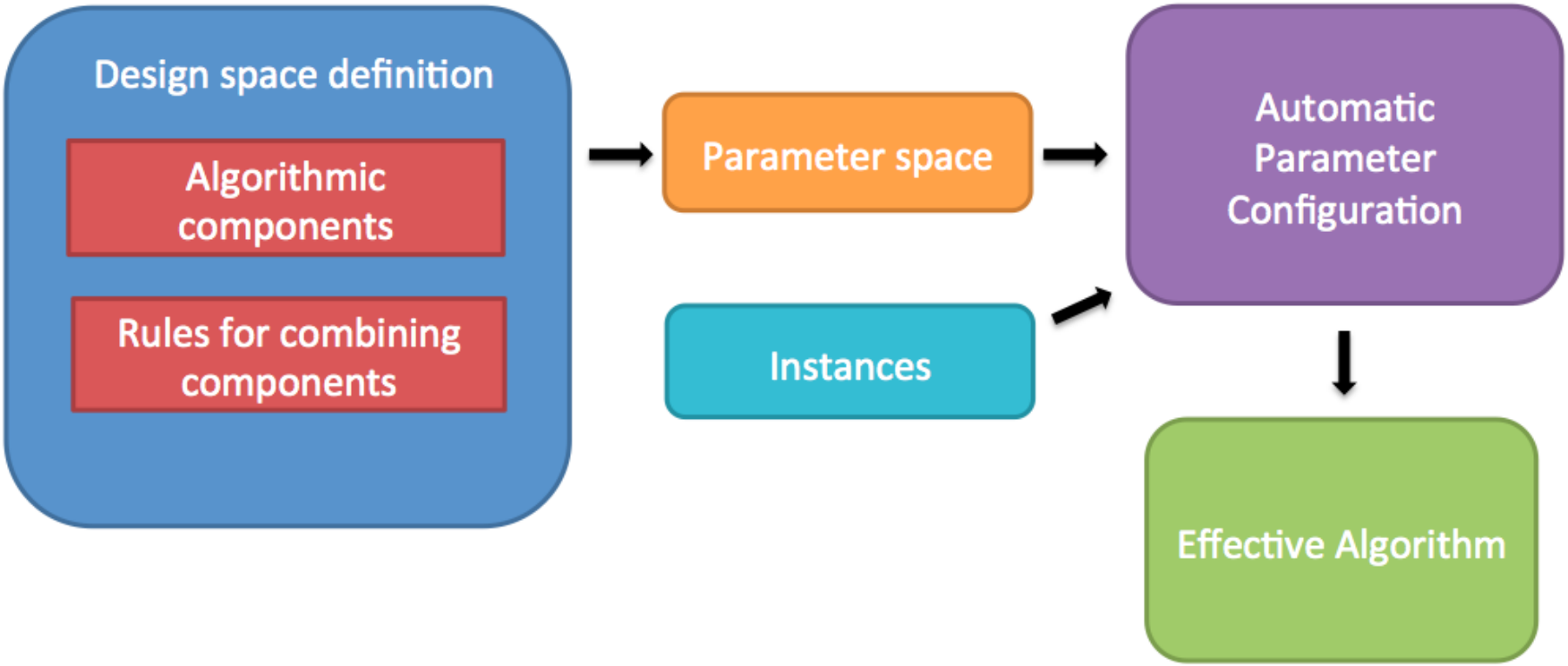
- maximize solution quality (within given computation time)
- minimize computation time (to reach optimal solution)

Overview of research in IRIDIA

An overview of research at IRIDIA

- Automatic configuration
  - F-race, irace, crace
- Parameter configuration
  - combinatorial and continuous single-objective optimization SLS algorithms
  - combinatorial and continuous multi-objective optimization SLS algorithms
  - anytime optimization (improve time-quality trade-offs)
  - exact MIP solvers (CPLEX, SCIP)
- Automatic algorithm design
  - from a flexible framework of algorithm components
  - from a grammar description
- Design of control software for swarm robotics

General approach



Software for configuration of algorithms

F-race, iterated F-race or irace, crace

- F-race
  - first proposal in 2002
  - use F-race to eliminate worse configurations
- iterated F-race or irace (2007 till today)
  - iteratively generate F-races or t-races
  - irace is the software to be used since 2011
  - . . . has several parameters . . .
- crace (2019 till today)
  - continuous racing instead of iterative racing
  - remove some disadvantages of irace due to the kind of parallelization

What is important of irace, crace, and others

- easy to use
- details for specific tasks (parameters)
- performant also with many (hundreds) of parameters
- they are *one tool for the different algorithms*

Multi-objective evolutionary algorithms (MOEA)

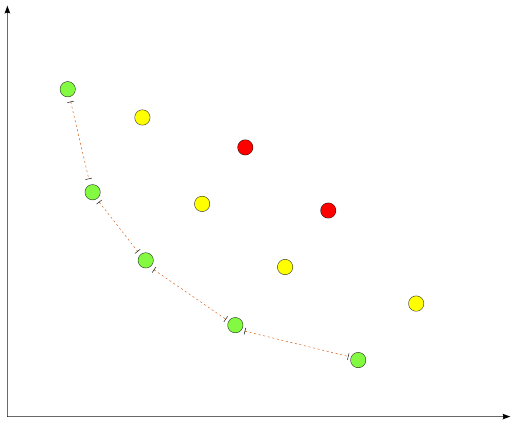




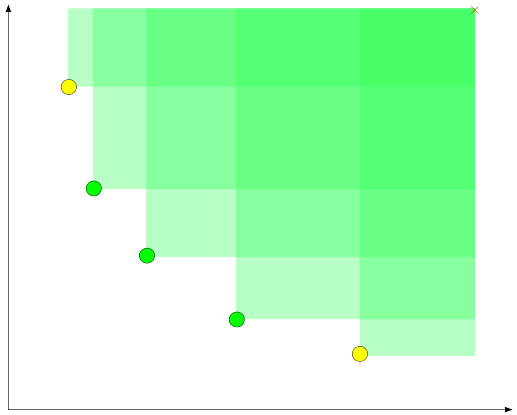
Multi-objective Evolutionary Algorithms

- +30 years of research
- most researched multi-objective metaheuristic/SLS algorithm
- mostly applied to continuous (real-valued) problems
- real-world applications in many domains
- numerous high-quality libraries/frameworks:
  Shark, jMetal, PaGMO, ...

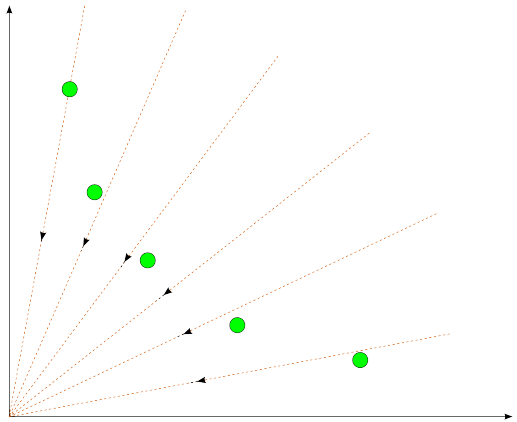
Multi-objective evolutionary algorithms



Pareto based
(NSGA-II, SPEA2)

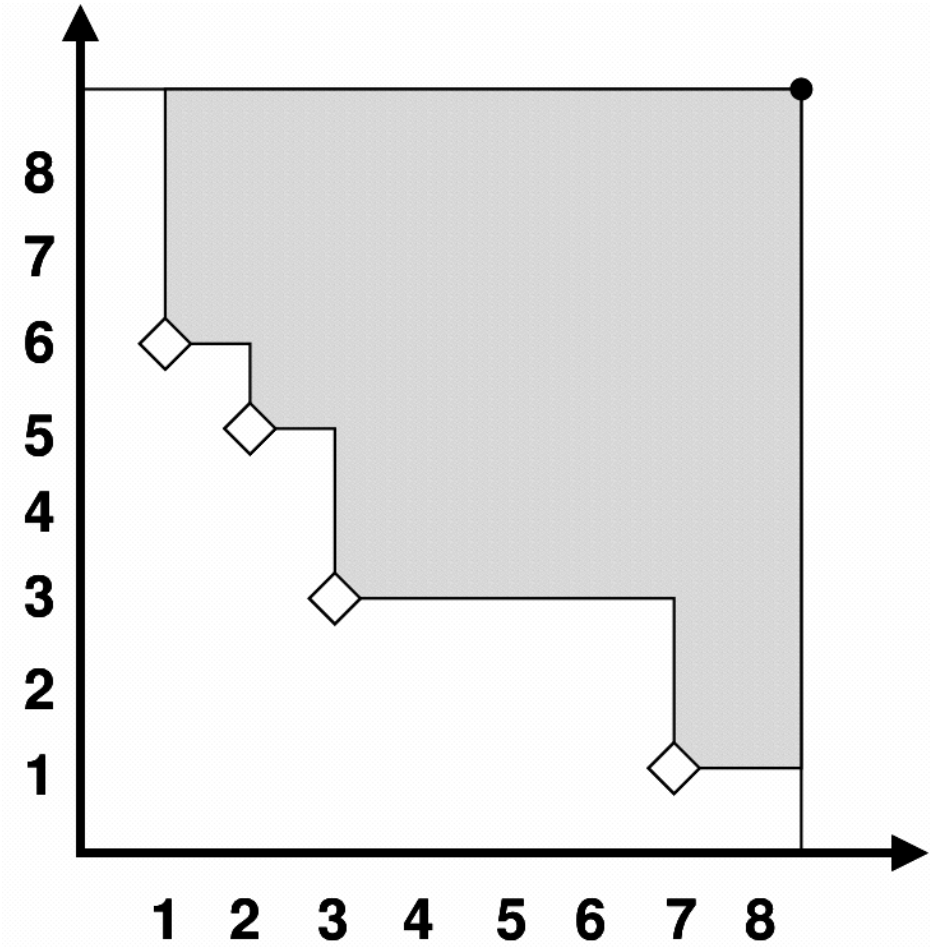


Indicator based
(IBEA, SMS-EMOA)



Weight based
(MOGLS, MOEA/D)

Hypervolume framework



irace + *hypervolume* = *automatic configuration of multi-objective solvers!*

AutoMOEA: Automatic Design (2015 – 2020)

Automatic configuration (**irace**)
+ Flexible algorithmic framework (AutoMOEA)
= Automatic design of state-of-the-art MOEAs

	BuildMatingPool			Replacement		
	SetPart	Quality	Diversity	SetPart	Quality	Diversity
MOGA	rank	—	niche-sharing	—	—	—
NSGA-II	depth	—	crowding dist.	depth	—	crowding dist.
SPEA2	strength	—	kNN	strength	—	kNN
IBEA	—	binary indicator	—	—	binary ind.	—
HypE	—	I_H^h	—	depth	I_H^h	—
SMS-EMOA	—	—	—	depth-rank	I_H^1	—
DTLZ 2-obj	—	—	crowding	depth-rank	I_ϵ	sharing
DTLZ 3-obj	depth-rank	I_ϵ	kNN	rank	I_H^1	sharing
DTLZ 5-obj	rank	I_H^1	crowding	depth	I_H^1	—
WFG 2-obj	rank	—	crowding	depth-rank	I_H^1	—
WFG 3-obj	count	I_H^1	crowding	strength	I_H^1	sharing
WFG 5-obj	count	I_H^h	crowding	—	I_H^1	—

Experimental results

DTLZ			WFG		
2-obj $\Delta R = 126$	3-obj $\Delta R = 127$	5-obj $\Delta R = 107$	2-obj $\Delta R = 169$	3-obj $\Delta R = 130$	5-obj $\Delta R = 97$
Auto_{D2} (1339)	**Auto_{D3} (1500)**	**Auto_{D5} (1002)**	**Auto_{W2} (1692)**	**Auto_{W3} (1375)**	**Auto_{W5} (1170)**
SPEA2_{D2} (1562)	IBEA_{D3} (1719)	SMS_{D5} (1550)	SPEA2_{W2} (2097)	SMS_{W3} (1796)	SMS_{W5} (1567)
IBEA_{D2} (1940)	SMS_{D3} (1918)	IBEA_{D5} (1867)	$\text{NSGA-II}_{\text{W2}}$ (2542)	IBEA_{W3} (1843)	IBEA_{W5} (1746)
$\text{NSGA-II}_{\text{D2}}$ (2143)	HypE_{D3} (2019)	SPEA2_{D5} (2345)	SMS_{W2} (2621)	SPEA2_{W3} (2600)	SPEA2_{W5} (2747)
HypE_{D2} (2338)	SPEA2_{D3} (2164)	$\text{NSGA-II}_{\text{D5}}$ (2346)	IBEA_{W2} (2777)	$\text{NSGA-II}_{\text{W3}}$ (3315)	$\text{NSGA-II}_{\text{W5}}$ (3029)
SMS_{D2} (2406)	$\text{NSGA-II}_{\text{D3}}$ (2528)	HypE_{D5} (2674)	HypE_{W2} (2851)	HypE_{W3} (3431)	MOGA_{W5} (4268)
MOGA_{D2} (2970)	MOGA_{D3} (2851)	MOGA_{D5} (2915)	MOGA_{W2} (4320)	MOGA_{W3} (4540)	HypE_{W5} (4373)

Additional remarks

- additional results with old variant
  - time-constrained scenarios
  - applications to multi-objective flow-shop scheduling
- with new variant
  - compares up to 10 objectives on continuous benchmarks
  - dominates the state-of-the-art algorithms
- extensions
  - more comprehensive benchmarks sets
  - design space analysis (e.g. ablation)
  - extensions of template (local search, etc.)

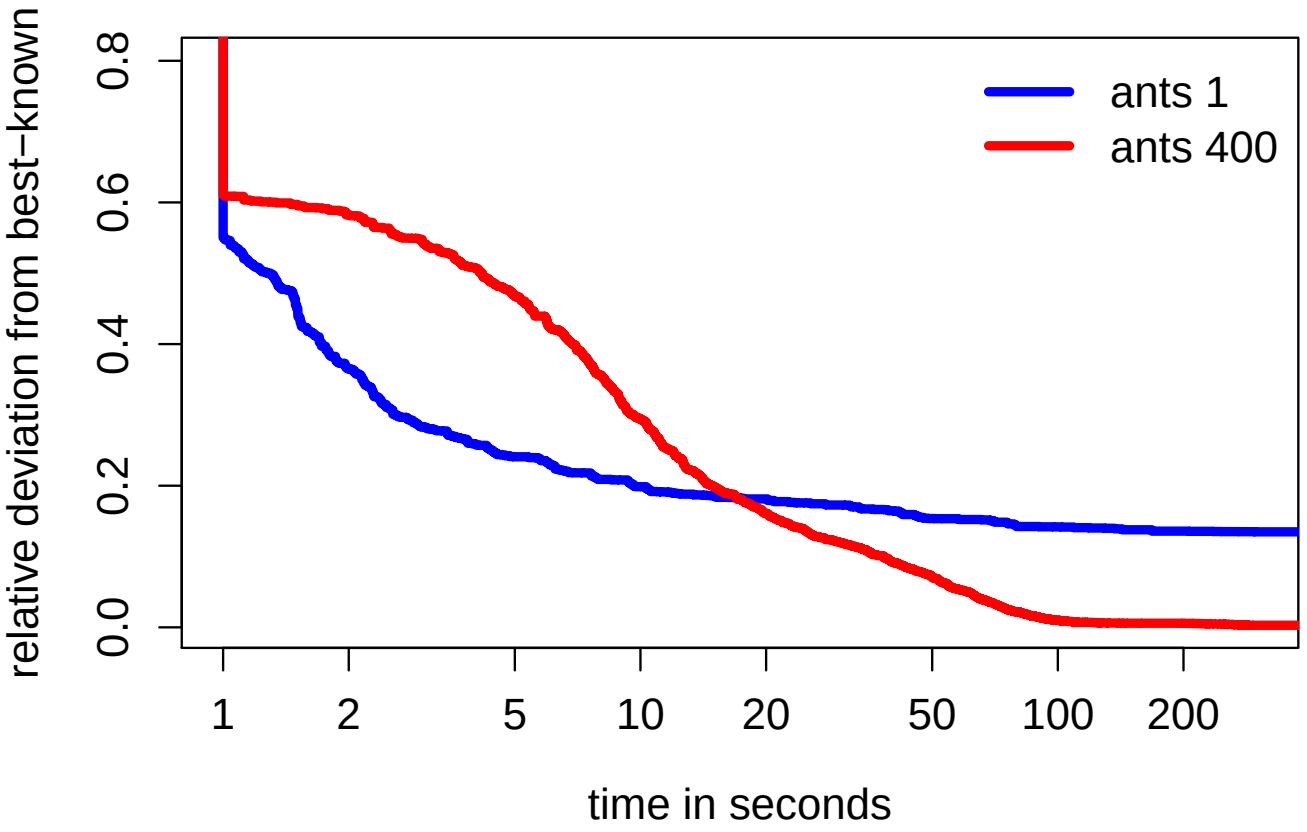
Time has come to automatically configure MOEAs (and other algorithms)

Improve automatically the anytime behavior of algorithms



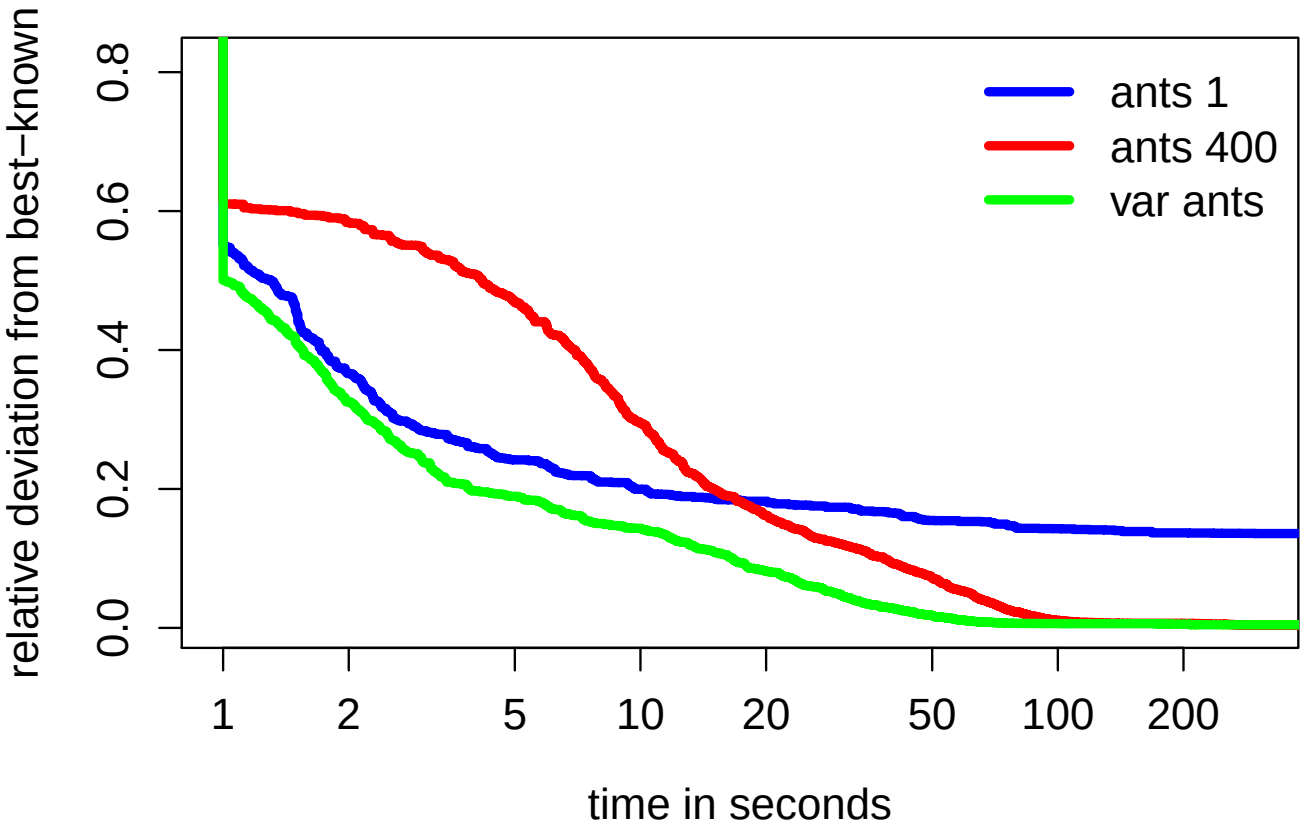

“Anytime” Algorithms [Zilberstein, 1996]

“Anytime” algorithms aim to produce as high quality results as possible, independent of the computation time allowed.



“Anytime” Algorithms [Zilberstein, 1996]

“Anytime” algorithms aim to produce as high quality results as possible, independent of the computation time allowed.

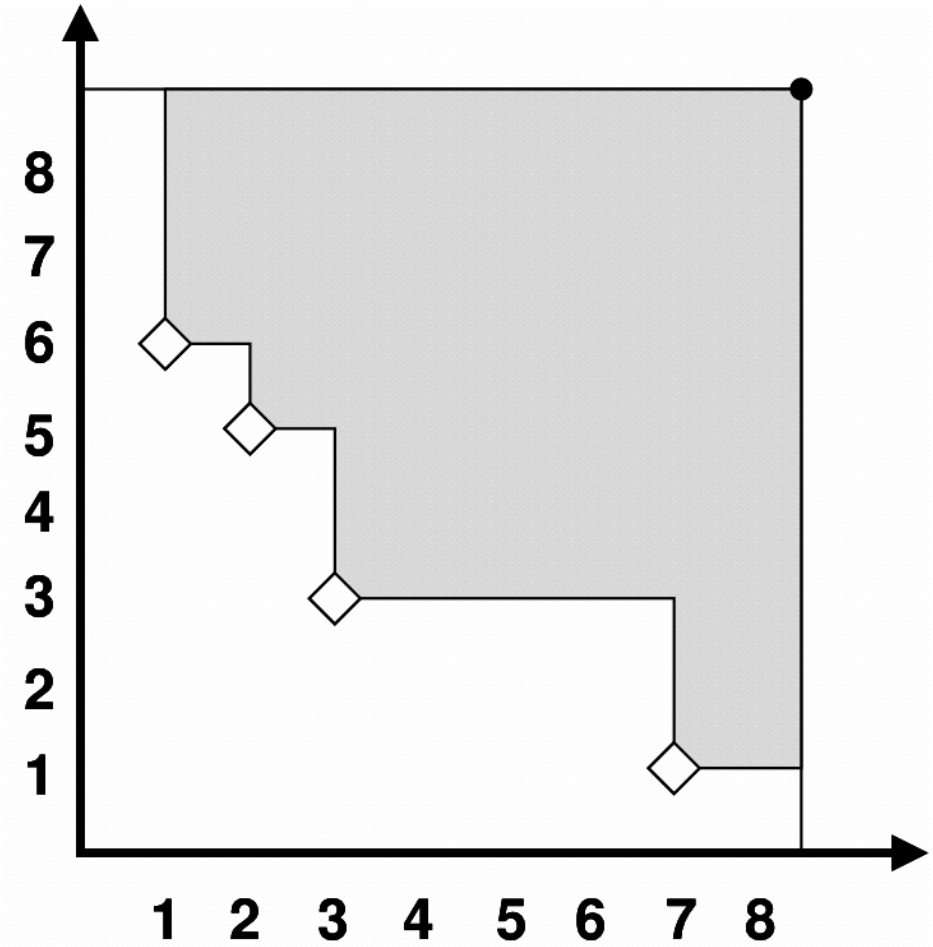


Aapproach for automatic configuration (2014)

- multi-objective optimization
  - \+ Objectively defined comparison
  - \+ Performance assessment techniques (hypervolume)

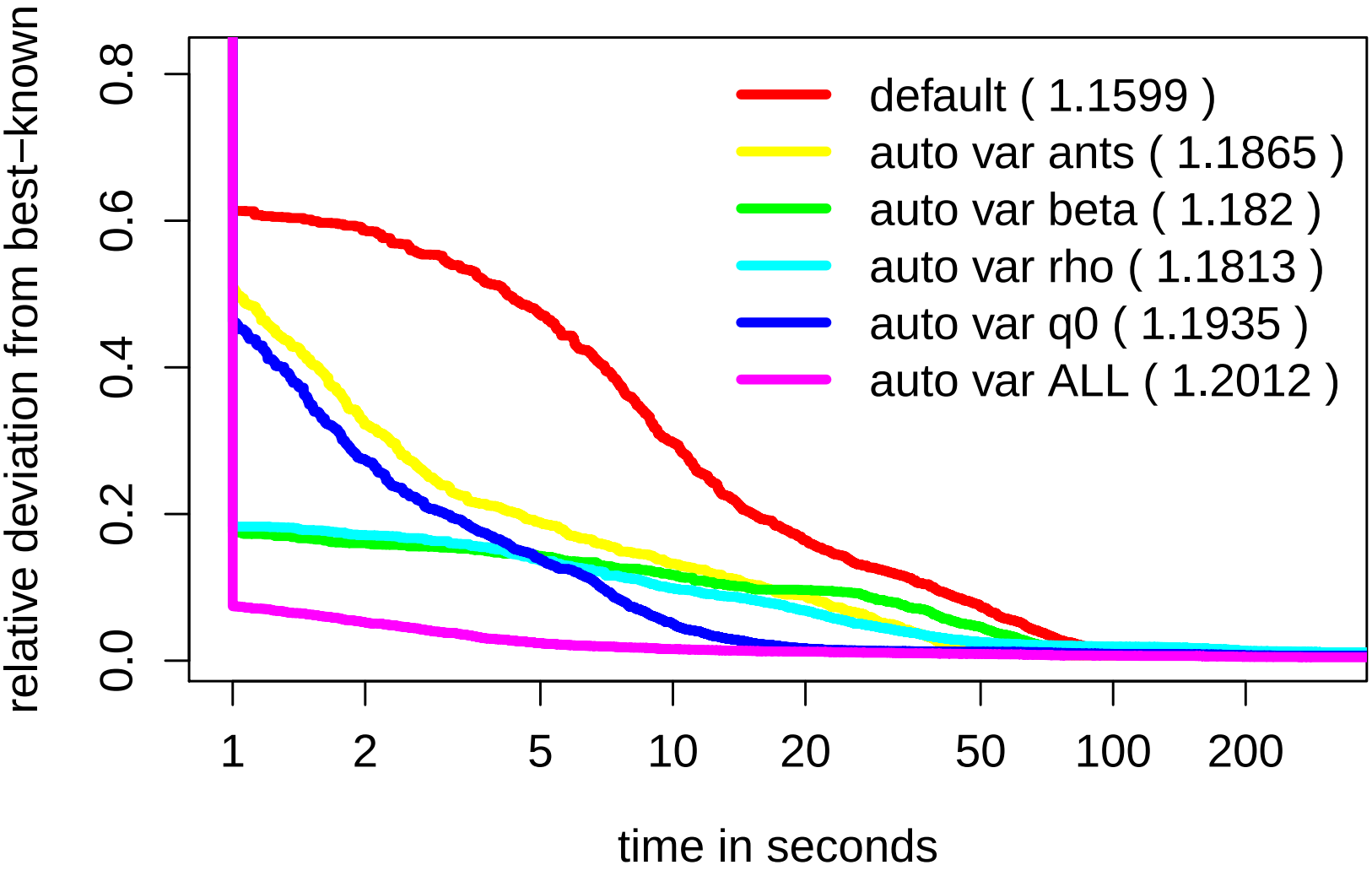
- Automatic configuration
  - \+ Most effort done by the computer
  - \+ Best configurations selected by the computer: *Unbiased*

Hypervolume framework



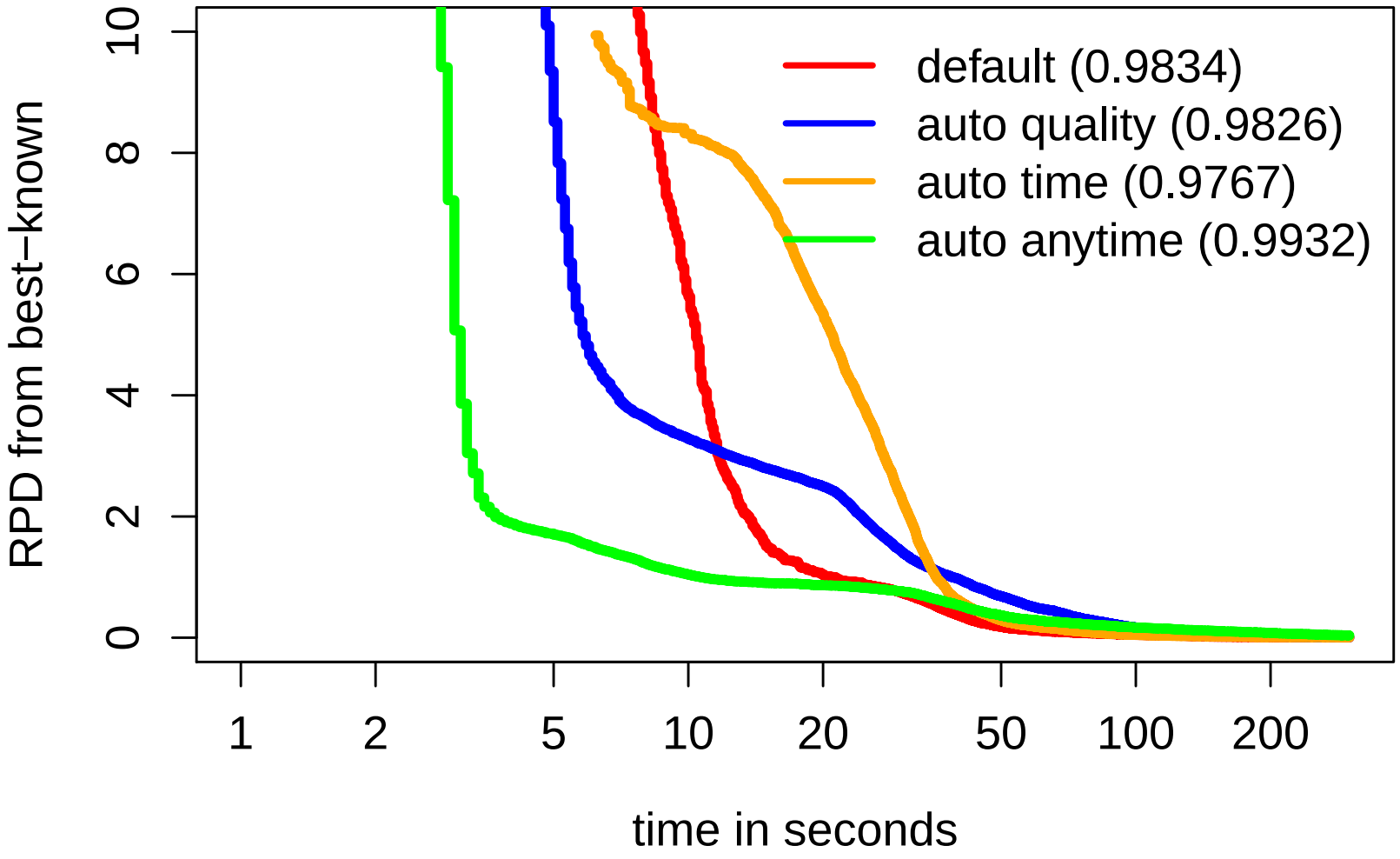
irace + *hypervolume* = *automatic configuration of multi-objective solvers!*

Experimental comparison



Applying MAX-MIN ant system to the TSP

Improving anytime behavior of SCIP



Applying SCIP to Winner determination problem for combinatorial auctions; 1000 training, 1000 test instances, 300 secs CPU time; 5000 budget

Conclusions on configuring anytime algorithms

- Same or better results than humans in less time
- Improving the anytime behaviour of metaheuristics becomes *much easier*

We can use offline configuration of online strategies for improving anytime behaviour

Automatic design of hybrid SLS algorithms








Automatic design of hybrid SLS algorithms

[Marmion, Mascia, Lópes-Ibáñez, Stützle, 2013][Lópes-Ibáñez, Marmion, Stützle, 2017]

Approach

- decompose single-point SLS methods into components
- derive generalized SLS algorithms structure
- component-wise implementation of SLS algorithms part

Implementation of automatic design (2013 – today)

- present possible algorithm compositions by a grammar
- instantiate grammer using a parametric representation
  - allows use of standard automatic configuration tools
  - shows good performance when compared to, e.g., grammatical evolution [Mascia, Lópes-Ibáñez, Dubois-Lacoste, Stützle, 2014]

General Local Search Structure: ILS

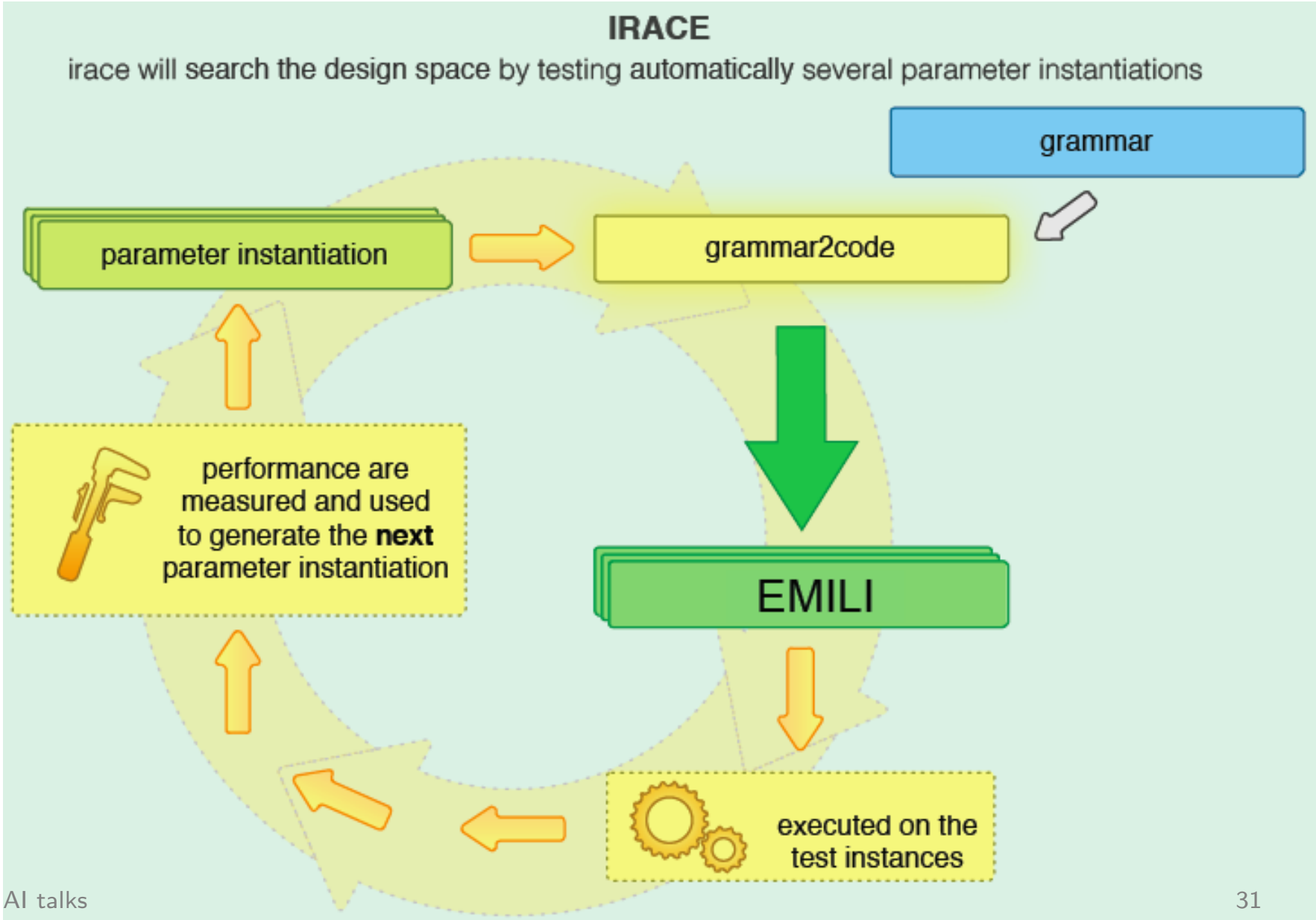
s_0 :=initSolution
s^* := ls(s_0)
repeat
s' :=perturb(s^*,*history*)
$s^{*\prime}$:=ls(s')
s^* :=accept(s^*,$s^{*\prime}$,*history*)
until termination criterion met

- many SLS methods instantiable from this structure
- abilities
  - hybridization
  - recursion
  - problem specific implementation at low-level

Example instantiations of some SLS algorithms

	perturb	*ls*	*accept*
SA	$\emptyset$	random move	Metropolis
PII	$\emptyset$	random move	Metropolis, fixed T
TS	$\emptyset$	TS	$\emptyset$
ILS	any	any	any
IG	destruct/construct	any/$\emptyset$	any
GRASP	rand. greedy sol.	any	$\emptyset$

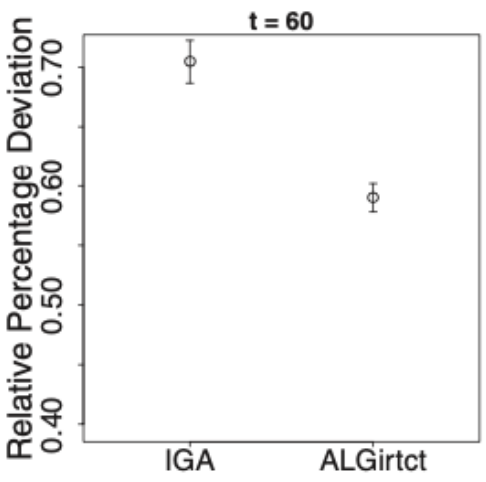
System overview

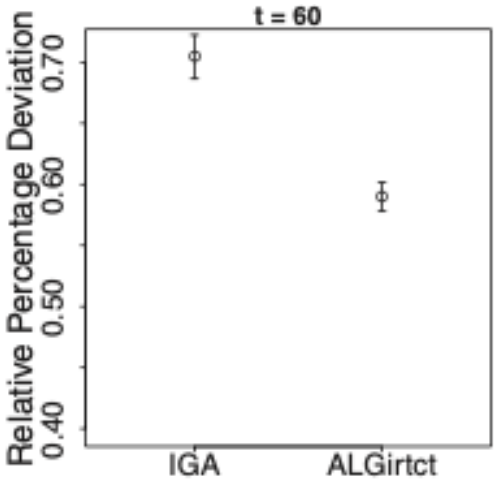


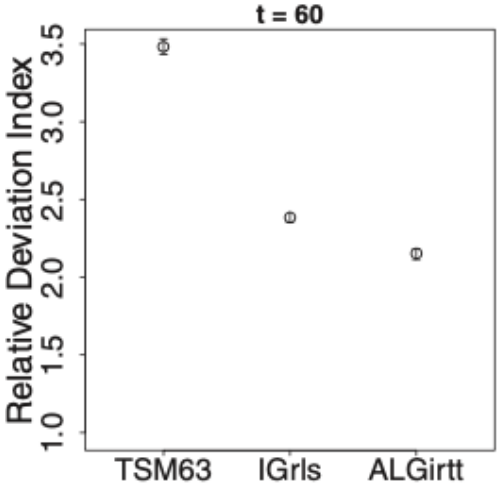
Flow-shop problem with various objectives

[Pagnozzi, Stützle, 2019]

- Automatic configuration with ≈ 500 parameters
- budget: 200 000 trials of $n \cdot m \cdot 0.06$ seconds





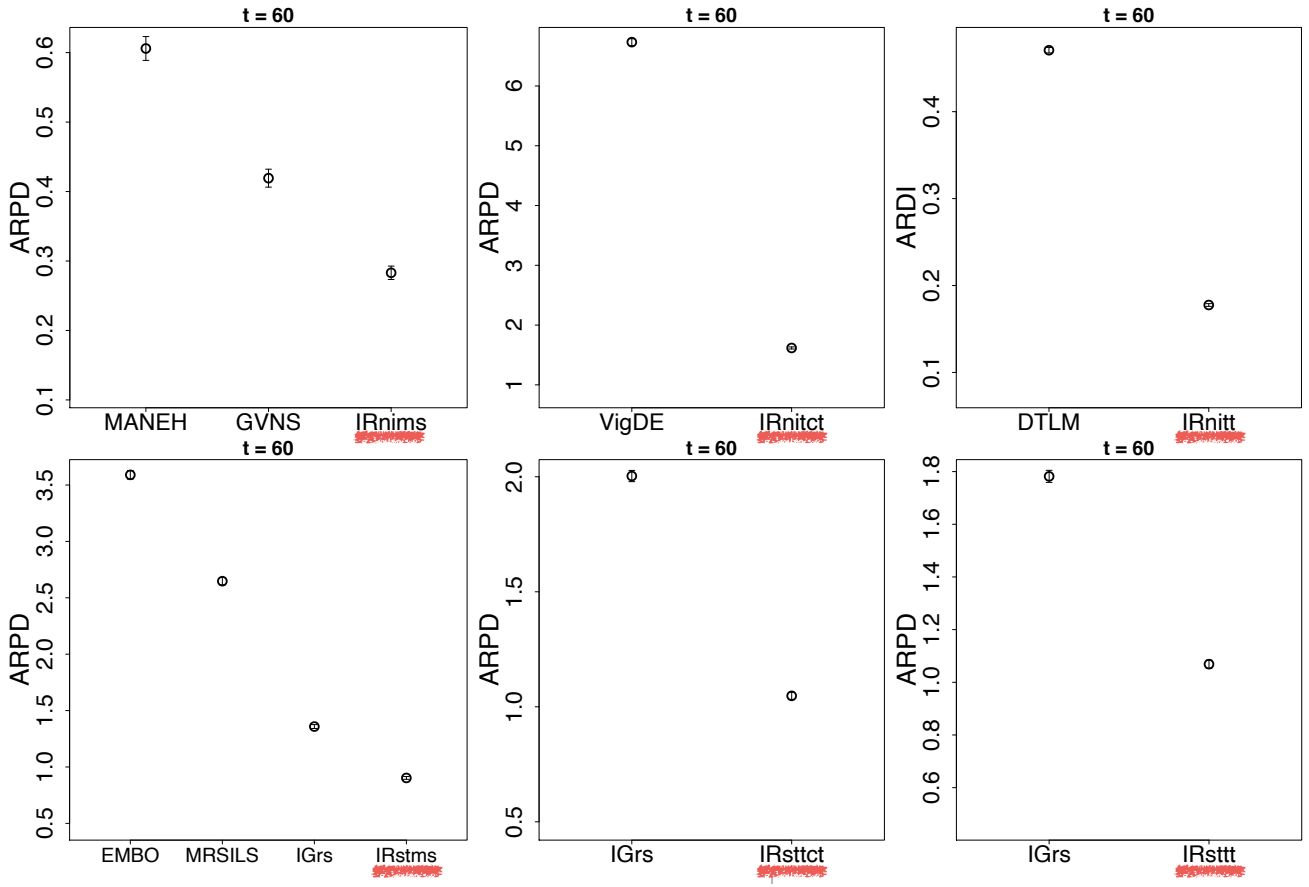


Results for all three objectives are superior to state-of-the-art

Additional constraints

[Pagnozzi, Stützle, 2021]

- we tackled the same three objectives with no-idle and sequence dependent setup times



Results are clearly superior to state-of-the-art

Summary on automatic design of hybrid SLS algorithms

Contributions

- approach to automate design and analysis of (hybrid) metaheuristics
- not a silver bullet, but needs right components, especially low-level problem-specific ones if state-of-the-art is desired

Current/future work

- extensions to other methods and templates
- integration of other techniques
- increase generality, tackling wide problem classes

Some perspectives

Why LLMs (and other future things)?

- LLMs as algorithm and components generators
  - as generators for operators, components or full algorithms
- LLMs for supporting algorithm design
  - the LLM helps with analysis, experimentation, or knowledge extraction
- LLMs as part of the automatic design process
  - they define the search space, select promising variants and guide the design process

LLMs as generators

- own experience with automatic design of memetic algorithms
  - generate components of the memetic algorithm
  - manual versus LLM-generated code for many design choices
  - manual was better in first version than LLM-generated code
  - . . . but not after optimizing LLM-code (ChatGPT 5.1)
- the coding is much much faster than human-generated code
- it can be used for SLS algorithms and components of these
- iterative improvements of algorithms

LLMs as a support for design process

- performance analysis and interpretation
- analysis of problem and instance charateristics
- extraction of algorithmic ideas from literature
- automatic experimental design

LLMs as design process

- supporting automatic algorithm configuration (e.g. warm starts)
- construction of the algorithmic design space
- automated problem modeling and solver design
- automatic selection and portfolios
- LLMs as controllers of the automatic design process

Concluding remarks

- I think for metaheuristic very important is anytime configuration

- I think for metaheuristic very important is anytime configuration
- SLS methods are algorithmic templates but they are replaced by automatic design of SLS algorithms

- I think for metaheuristic very important is anytime configuration
- SLS methods are algorithmic templates but they are replaced by automatic design of SLS algorithms
- Automatic configuration and automatic design had and have a major impact in transforming the field

- I think for metaheuristic very important is anytime configuration
- SLS methods are algorithmic templates but they are replaced by automatic design of SLS algorithms
- Automatic configuration and automatic design had and have a major impact in transforming the field
- LLMs are there (and more to come) and they help a lot in automatic configuration and design of algorithms

Acknowledgements

IRIDIA



External collaborators



Research funding

F.R.S.-FRNS, Projects ANTS (ARC), Meta-X (ARC), Comex (PAI), MIBISOC (FP7), COLOMBO (FP7), FRFC, Metaheuristics Network (FP5)