PyDGGA: Distributed GGA for Automatic Configuration

Carlos Ansótegui [1] Josep Pon [1] Meinolf Sellmann [2]
Kevin Tierney [3]

LOG group, University of Lleida, Spain
carlos.ansotegui@diei.udl.cat
josep.pon@udl.cat

General Electric, USA
meinolf@ge.com

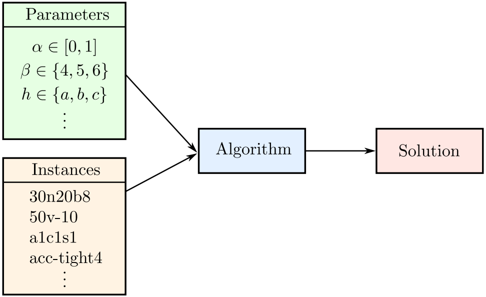
Decision and Operation Technologies Group, Bielefeld University, Germany
kevin.tierney@uni-bielefeld.de

June 21, 2021

Parameters

Parameters

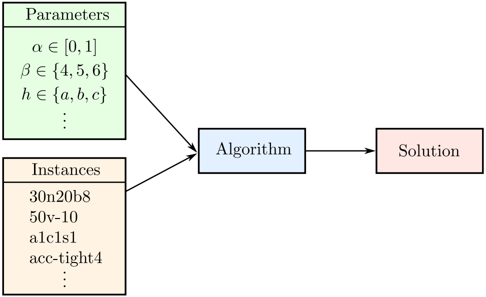
Algorithms have many parameters that influence their performance.



Parameters

Parameters

Algorithms have many parameters that influence their performance.



- With good parameter settings (configurations), problems can be solved faster (or better).

Parameter

What are parameters?

Parameters are settings (configurations) of an algorithm (or solver) that change the way an algorithm solves a problem.

Parameter Types

- Continuous: e.g. $[1.0, 5.0]$
- Discrete: e.g. $[1, \ldots, 10]$
- Categorical: e.g. $\{a, b, c, d\}$
- Ordinal: e.g. $\{low, medium, high\}$ (Ordered set)

SparrowToRiss parameters example

- prUips $\in \{-1, 0, 1\}$
- prDouble $\in \{yes, no\}$
- prProbleL $\in [500000, 7500000]$
- firstReduceDB $\in [2000, 8000]$
- incReduceDB $\in [300, 450]$
- rlevel $\in \{0, 1, 2\}$
- quickRed $\in \{yes, no\}$
- rndFreq $\in [0.00, 0.01]$
- minLBDMinimizingClause $\in [4, 9]$
- minSizeMinimizingClause $\in [3, 50]$
- incLBD $\in \{yes, no\}$
- xor $\in \{yes, no\}$
- … the list extends to about 222 parameters

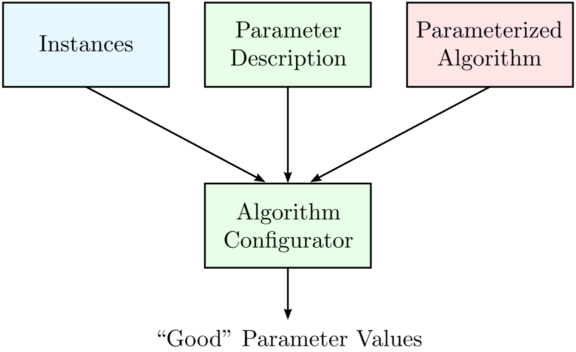
SparrowToRiss parameters example

- prUips $\in \{-1, 0, 1\} \rightarrow \mathbf{0}$
- prDouble $\in \{yes, no\} \rightarrow$ **yes**
- prProbleL $\in [500000, 7500000] \rightarrow \mathbf{5000000}$
- firstReduceDB $\in [2000, 8000] \rightarrow \mathbf{4000}$
- incReduceDB $\in [300, 450] \rightarrow \mathbf{300}$
- rlevel $\in \{0, 1, 2\} \rightarrow \mathbf{1}$
- quickRed $\in \{yes, no\} \rightarrow$ **yes**
- rndFreq $\in [0.00, 0.01] \rightarrow \mathbf{0.005}$
- minLBDMinimizingClause $\in [4, 9] \rightarrow \mathbf{6}$
- minSizeMinimizingClause $\in [3, 50] \rightarrow \mathbf{30}$
- incLBD $\in \{yes, no\} \rightarrow$ **no**
- xor $\in \{yes, no\} \rightarrow$ **no**
- ... the list extends to about 222 parameters

Automatic Algorithm Configuration

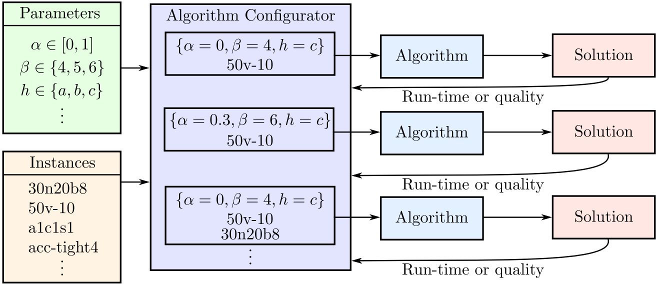
- Instead of experimental design (or parameter tuning by hand), algorithms can be automatically configured.
- Non-model-based algorithm configurators:
  - CALIBRA: Fractional factorial design with local search (Adenso-Diaz and Laguna, 2006)
  - ParamILS: Iterated Local Search (Hutter, Hoos und Stützle 2007/2009)
  - GGA: Genetic algorithms (Ansotegui, Sellmann and Tierney 2009)
  - (Iterated) F-Race: Racing with statistical tests (Birattari, et al. 2002/2010)
- Model based algorithm configurators:
  - SMAC: Random forest learning (Hutter, Hoos, Leyton-Brown 2011)
  - GGA++: Genetic algorithms with random forest model crossover (Ansotegui, et al. 2015)

Automated Algorithm Configuration



Parameter Tuning

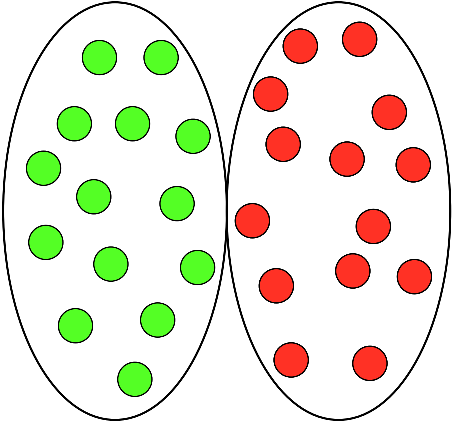
- Algorithm configurators try different parameter settings on selected instances and measure the performance of the target algorithm.



GGA

- Genetic algorithm based algorithm configurator
- "Gender-based GA" for diversification
- Intensification through strong selection procedure
- Supports all continuous, discrete and categorical parameters
- Can handle certain types of conditionality and forbidden parameter sequences
- First published in 2009

GGA: Population

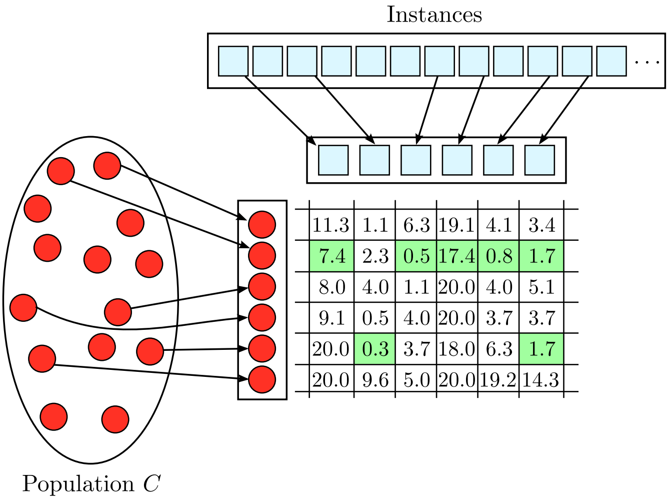


Partition: Population N and C

Population N: "Genome storage"

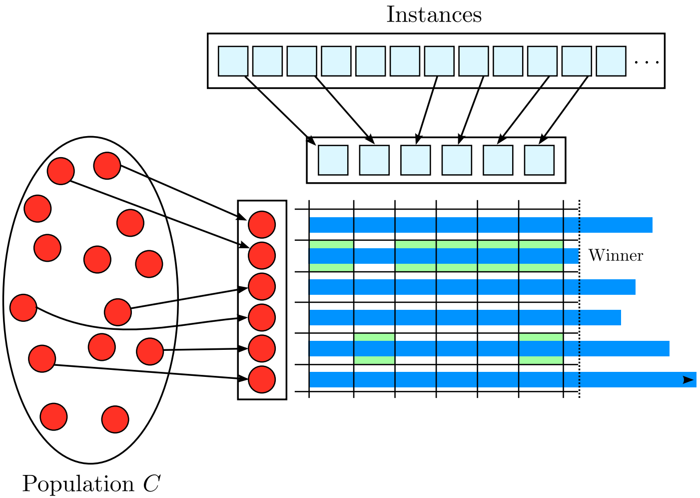
Population C: Competing against each other in competitions

Selection (mini-tournament)



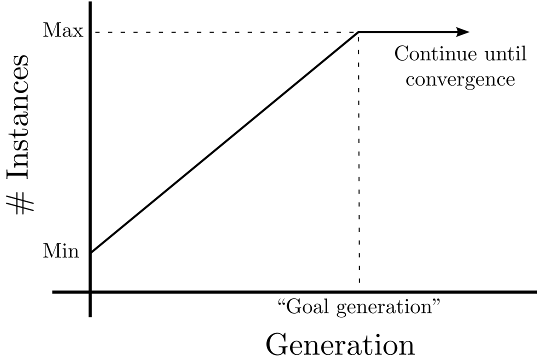
11.3	1.1	6.3	19.1	4.1	3.4
7.4	2.3	0.5	17.4	0.8	1.7
8.0	4.0	1.1	20.0	4.0	5.1
9.1	0.5	4.0	20.0	3.7	3.7
20.0	0.3	3.7	18.0	6.3	1.7
20.0	9.6	5.0	20.0	19.2	14.3

Selection (mini-tournament)

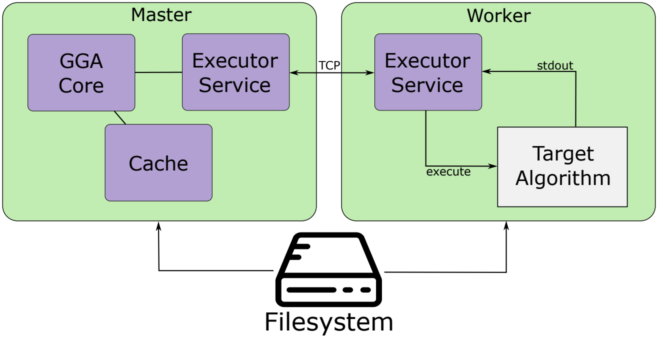


Instance selection

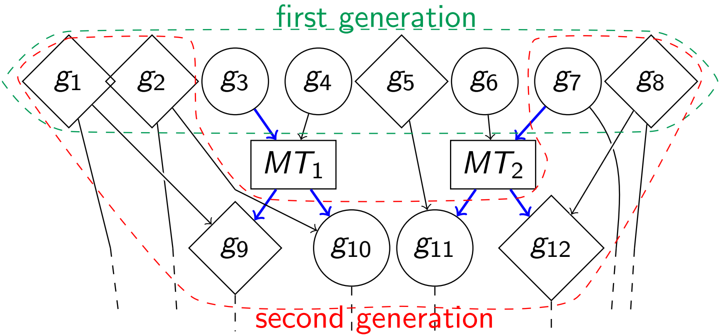
- GGA tunes a subset of instances that increases linearly in size according to the current generation.
- The subset of instances is chosen randomly at each generation



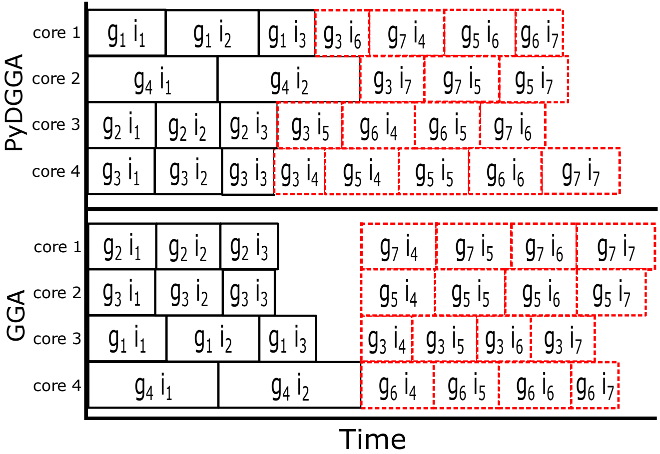
PyDGGA: Architecture



PyDGGA: Simulation



PyDGGA: Scheduler



Hands-on Demo

Elite mini-tournament, Stop/Resume, Improved Configuration constraints, ...

Available from https://ulog.udl.cat

Results

Table: SparrowToRiss PAR10 performance (# solved instances)

	BMC	CF	IBM	GI	N-Rooks
Default	346 (262)	297 (276)	113 (232)	247 (307)	116 (348)
PyDGGA	**171 (267)**	**89 (283)**	**10 (232)**	**91 (317)**	**6.3 (351)**

Table: CPLEX PAR10 performance (# solved instances)

	Assortment	CLS	COR-LAT	RCW
Default	2429 (52)	3.04 (50)	37 (999)	562 (844)
PyDGGA	**718 (58)**	**2.01 (50)**	**7 (1000)**	**344 (855)**

Conclusions & Future Work

PyDGGA is able to improve the performance of an algorithm, using the resources of a distributed computing environment efficiently.

- Integrate surrogate models
- Improve evaluations scheduling
- Simplify setup by means of technologies, such as Zero-configuration networking.