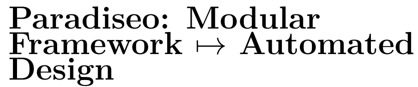


Paradiseo: Modular Framework $\mapsto$ Automated Design

22 years of Paradiseo

- Speaker: J. Dreo
- 2021-07-1X

Paradiseo

From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics

—22 years of Paradiseo—

- **Johann Dreo**
- Arnaud Liefooghe
- Sébastien Verel
- Marc Schoenauer
- Juan J. Merelo
- Alexandre Quemy
- Benjamin Bouvier
- Jan Gmys

Introduction

- Huge variety of problem landscapes $\mapsto$ no algorithm dominates.
- A framework should allow to explore the *algorithm design space*.
- Automated design $\mapsto$ synthetic benchmarking $\mapsto$ fast computation are needed.
- Paradiseo provides:
  - *modular* architecture
  - large set of *components*
  - *fast* computations
  - *automated design features*

Summary

01

History and Alternatives

02

Architecture

03

Features

04

Conclusion

Part 1

History and Alternatives

22 years of Paradiseo

- Contributions from more than 50 people: Maarteen Keijzer, Gustavo Romero, Jeroen Eggermont, Olivier König, Jochen Küpper, El-Ghazali Talbi, Thomas Legrand, Sébastien Cahon, Nouredine Melab, Jérémie Humeau, C. FC., Jean-Charles Boission, Caner Candan, ...
- With the support of the following institutions: Inria, University of Lille, University of the Littoral Opal Coast, Thales, École Polytechnique, University of Granada, Vrije Universiteit Amsterdam, Leiden University, French National Centre for Scientific Research (CNRS), French National Agency for Research (ANR), Fritz Haber Institute of the Max Planck Society, Center for Free-Electron Laser Science, University of Angers, French National Institute of Applied Sciences, Free University of Brussels, Pasteur Institute.

Started in 1999.

Some spin-off:

- JEO (EO in Java),
- EO in ActiveX
- GUIDE (graphical interface for assembling algorithms)
- EASEA (declarative language)

Alternatives

jMetal, ECJ, HeuristicLab, etc.

- open-source frameworks (*designing* algorithms)
- active since 2015
- more than 15 contributors

Name	Language	Update	License	Contributors	kloc	Evol.	EDAs	PSO	Local Search	Cluster	Multicore	GPGPU	Multiobjective	Landscapes	States	Auto. Design
ParadisEO	C++	2021	LGPLv2	33	82	Y	Y	Y	Y	Y	Y	~	Y	Y	Y	Y
jMetal	Java	2021	MIT	29	60	Y	N	Y	N	Y	N	N	Y	N	?	N
ECF	C++	2017	MIT	19	15	Y	N	Y	N	Y	N	N	N	N	Y	N
ECJ	Java	2021	AFLv3	33	54	Y	Y	Y	N	Y	Y	Y	Y	N	Y	N
DEAP	Python	2020	LGPLv3	45	9	Y	N	N	N	Y	Y	N	Y	N	Y	N
CIlib	Scala	2021	Apachev2	17	4	Y	N	N	N	N	N	N	N	N	?	N
HeuristicLab	C#	2021	GPLv3	20	150	Y	N	Y	Y	Y	Y	N	Y	~	Y	N
Clojush	Clojure	2020	EPLv1	17	19	Y	N	N	N	N	N	N	N	N	N	N

INSTITUT PASTEUR

Part 2

Architecture

INSTITUT PASTEUR

Main concepts

Encoding: The data structure modelling a solution to the optimization problem (which type is generally denoted `EOT`).

Evaluation: The process of associating a value to a solution, thanks to an objective function.

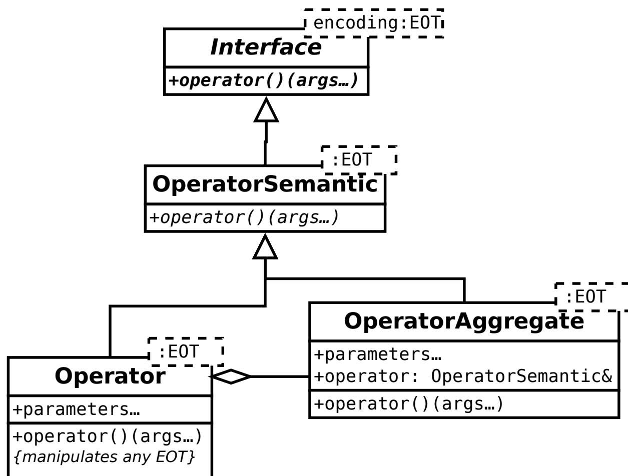
Fitness: The value of a solution as seen by the objective function.

Operator: A function which reads and/or alters a (set of) solutions.

Population: A set of solutions.

Main Design Patterns

High-level view



INSTITUT PASTEUR

Main Design Patterns

Modular assembling of components

Functor: callable *operators* with a *state*, possibly holding reference to other operators.

Strategy: operators *composition*, honoring a semantic *interface*.

Generic Typing: everything defined over a *solution encoding* template, exposing its interface anywhere.

Factory: collections of pre-instantiated operators.

Part 3

Features

Modules

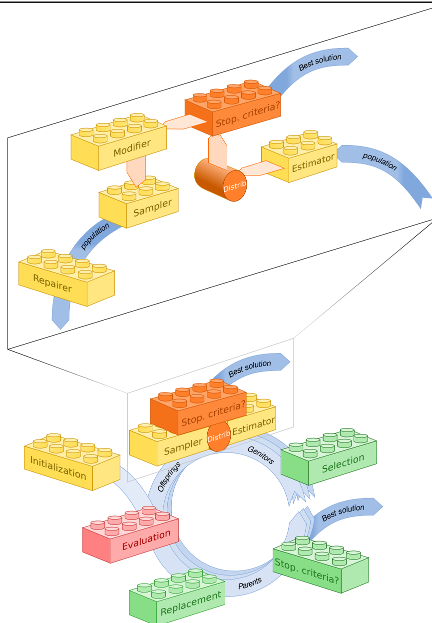
- EO: Evolutionary Algorithms & Particle Swarm Optimization,
- MO: Local Search & Landscape Analysis,
- MOEO: Multi-objective Optimization,
- EDO: Estimation of Distribution.
- SMP Shared Memory Parallelization.

Key Features

Modular Algorithms

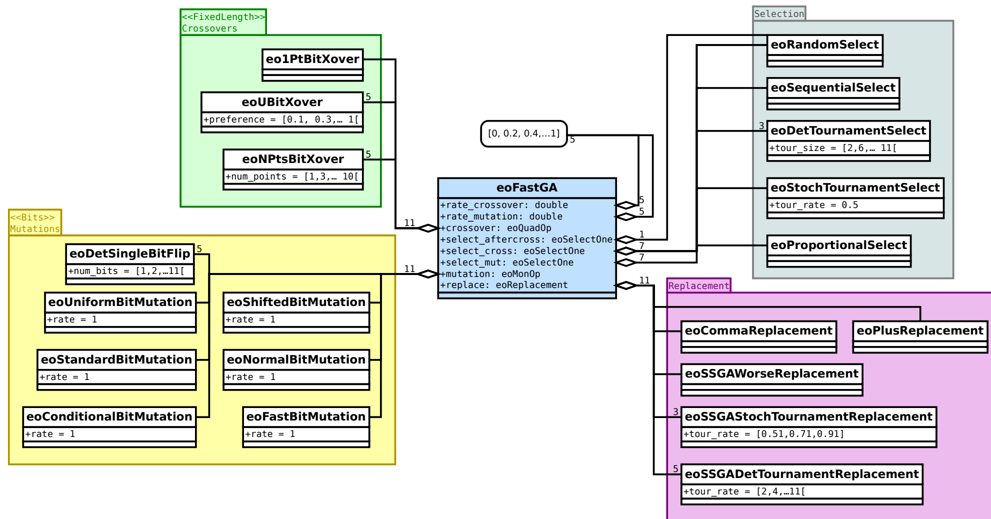
- Large set of ”algorithm” templates.
- Large set of operators.
- Extensibility through interoperability.
- Examples:
  - Using a local search as a mutation operator.
  - Extending evolutionary algorithm as estimation of distribution.

Example: EDO (interfaces)



INSTITUT PASTEUR

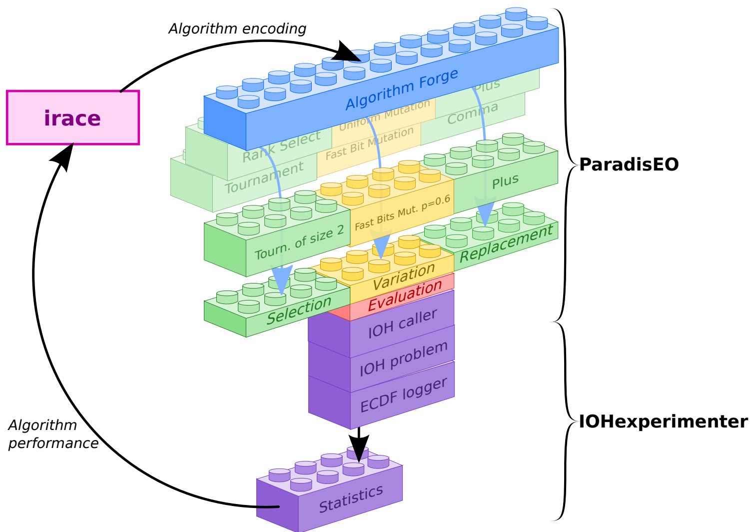
Example: GA (operators)



INSTITUT PASTEUR

Key Features

Automated Algorithm Design



INSTITUT PASTEUR

Automated Algorithm Design

Operators declaration

```
1   // Considering the FastGA modular algorithm,
2   // solving a problem with fixed initialization.
3   auto& foundry = store.pack< eoAlgoFoundryFastGA<Bits> >(
4     init, problem, max_eval_nb, /*max_restarts=*/1);
5   // Consider different crossover operators.
6   for(double i=0.1; i<1.0; i+=0.2) {
7     foundry.crossovers.add< eoUBitXover<Bits> >(i);
8     foundry.crossovers.add< eoNPtsBitXover<Bits> >(i*10);
9   }
10  // And different variation rates.
11  for(double i=0.0; i<1.0; i+=0.2) {
12      foundry.crossover_rates.add<double>(i);
13      foundry.mutation_rates.add<double>(i);
14  }
15  // etc.
16
```

Automated Algorithm Design

On-the-fly instantiation

```
1       // Decide which operators to use.
2       Ints encoded_algo(foundry.size());
3       encoded_algo[foundry.crossovers      .index()] = 2;
4       encoded_algo[foundry.crossover_rates.index()] = 1;
5       encoded_algo[foundry.mutation_rates  .index()] = 3;
6       // etc.
7       // Instantiate the operators, or use cached objects.
8       foundry.select(encoded_algo);
9       // Run the selected algorithm.
10      eoPop<Bits> pop; // [...]
11      foundry(pop);
12
```


INSTITUT PASTEUR

Automated Algorithm Design

IOH binding

```
// In-memory logger.
IOHprofiler_RangeLinear<size_t>
  target_range(0, max_target, buckets),
  budget_range(0, max_evals , buckets);
IOHprofiler_ecdf_logger<int,int,int> ecdf_logger(
  target_range, budget_range);
// Benchmark problem.
W_Model_OneMax w_model_om;
ecdf_logger.track_problem(w_model_om);
// The actual Paradiseo/IOH interface:
eoEvalIOHproblem<Bits> pb(w_model_om, ecdf_logger);
// [`pb` is plugged into an algorithm and ran...]
// The performance of the run is recovered:
IOHprofiler_ecdf_sum ecdf_sum;
long perf = ecdf_sum(ecdf_logger.data());

```


INSTITUT PASTEUR

Automated Algorithm Design

irace interface export

```
1      // Using Paradiseo parameters:
2      eoParser parser(argc, argv, "interface for irace");
3      auto crossover_p = parser.getORcreateParam<size_t>(
4          0, "crossover", "The crossover operator", 'c',
5          "Operator Choice", true);
6      // [...] assemble a foundry [...]
7      // Print the irace's configuration file for this binary:
8      std::cout << "# name\t switch\t type\t range\n";
9      // We only need the parameter(s) and the foundry itself:
10     print_irace(mutation_rate_p, foundry.mutation_rates);
11     print_irace(     crossover_p, foundry.crossovers     );
12     // Any other operator within the foundry [...]
13
```


INSTITUT PASTEUR

Part 4

Conclusion

Conclusion

Modern Framework

- Utility features (fine-grained parallelization, CLI, state suspension, etc.)
- Component-based architecture.
- Multiple, Modular Algorithms.

Conclusion

For future solvers

- Large sets of algorithms & operators.
- Large algorthms design space.
- Fast computations and bindings.
- Large Scale Automated Algorithm Design.
- *Focus on algorithmics, let the automation handle tedious tasks.*

Perspectives

- Improve design automation.
- Add more modular designs.
- Python interface.
- Overall user experience.

https://github.com/jdreo/paradiseo

Institut Pasteur
25–28, rue du Docteur Roux
75724 Paris Cedex 15 — France