Randomized Optimization Framework tailored for Benchmarking & Auto-Design

Johann Dreo

Randomized Optimization Framework *tailored for* Benchmarking & Auto-Design

Johann Dreo

Institut Pasteur & Université Paris Cité ⟩ Computational Biology dept. & Bioinformatics and Biostatistics hub ⟩ Computational Systems Biomedicine lab.

Paradiseo is an open-source ***full-featured evolutionary computation framework*** which main purpose is to help you ***write your own stochastic optimization algorithms***, insanely fast:

Choose an algorithm template, select operators of interest, plug a benchmark and let it design the algorithm for you!

It focus on the efficiency of the implementation of solvers, by providing:
- a ***modular design*** for several types of paradigms,
- the ***largest codebase*** of existing components,
- tools for ***automated design and selection*** of algorithms,
- a focus on ***speed*** and several ***parallelization*** options.

WHY PARADISEO?

Full-featured

Paradiseo provides the ***largest mature codebase*** of state-of-the-art algorithms, and focuses on (automatically) find the ***most efficient solvers***.

The most classical impediment to the use of Paradiseo is that you just want to check if your problem can actually be solved with heuristics. You feel that it would be a loss of time to learn complex stuff if it ends being useless.
However, you should keep in mind that:
- Metaheuristics do seem very easy to implement in textbooks, but the state-of-the art versions of efficient algorithms can be a lot more complex.
- It is usually easy to get something to actually run, but it is far more difficult to get an efficient solver.
- Metaheuristics performances on a given problem are very sensitive to small variations in the parameter setting or the choice of some operators. Which render large experimental plans and algorithm selection compulsory to attain peak efficiency.

Fortunately, Paradiseo have the largest codebase of the market, hardened along 20 years of development of tens of solvers.
Additionally, it provides the tools to rapidly search for the best combination of algorithms to solve your problem, even searching for this combination automatically.

Fast Benchmarking

Paradiseo is ***the fastest*** framework on the market, which is a crucial feature for modern and robust approach to solver ***design and validation***.

To give an order of magnitude:
- If you use the "official" vanilla implementation of CMA-ES in Python/Numpy solving the BBOB problem suite through the COCO plateform, running the whole benchmark will take approximately 10 minutes on a single Intel Core i5 @ 2.50GHz with a solid state disk.
- The same experiment, running the Paradiseo implementation using the seamless binding to the IOHprofiler BBOB implementation, will take 1 minute.

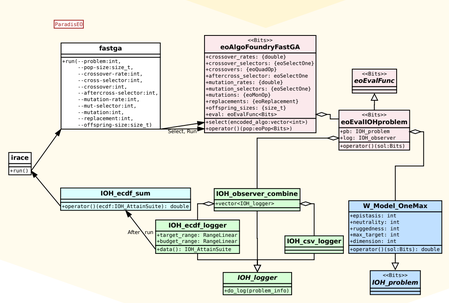
Name	Language	Update	License	Contributors	kloc	Evol.	EDAs	PSO	Local Search	Cluster	Multicore	GPGPU	Multiobjective	Landscapes	States	Auto. Design
ParadisEO	C++	2021	LGPLv2	33	82	Y	Y	Y	Y	Y	Y	~	Y	Y	Y	Y
jMetal	Java	2021	MIT	29	60	Y	N	Y	N	Y	N	N	Y	N	?	N
ECF	C++	2017	MIT	19	15	Y	N	Y	N	Y	N	N	N	N	Y	N
ECJ	Java	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
Cllib	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

82 kloc
50 Contributors
LGPL/CeCILL

10× Faster Benchmarking

BENCHMARKING

- ***IOH experimenter*** binding
  - COCO, PBO, Wmodels, NKL
  - IOH analyzer Logger
  - Performance Logger
- Parallelization
- Container-compatible
- Auto. Command Line Interface
- Suspend on state



Example: Minimal CMA-ES

```cpp
#include <eo>
#include <edo>
#include <es.h>
#include <do/make_pop.h>
#include <do/make_run.h>
#include <do/make_continue.h>
#include <do/make_checkpoint.h>

using R = eoReal<eoMinimizingFitness>;
using CMA = edoNormalAdaptive<R>;

R::FitnessType sphere(const R& sol) {
    double sum = 0;
    for(auto x : sol) { sum += x * x; }
    return sum;
}

int main(int argc, char** argv) {
    eoParser parser(argc, argv);
    eoState state;

    size_t dim = parser.createParam<size_t>(10,
            "dimension", "Dimension", 'd',
            "Problem").value();

    size_t max_eval = parser.getORcreateParam<size_t>(100 * dim,
            "maxEval", "Maximum number of evaluations", 'E',
            "Stopping criterion").value();

    edoNormalAdaptive<R> gaussian(dim);

    auto& obj_func = state.pack< eoEvalFuncPtr<R> >(sphere);
    auto& eval     = state.pack< eoEvalCounterThrowException<R> >(obj_func, max_eval);
    auto& pop_eval = state.pack< eoPopLoopEval<R> >(eval);

    auto& gen  = state.pack< eoUniformGenerator<R::AtomType> >(-5, 5);
    auto& init = state.pack< eoInitFixedLength<R> >(dim, gen);
    auto& pop = do_make_pop(parser, state, init);
    pop_eval(pop,pop);

    auto& eo_continue  = do_make_continue(  parser, state, eval);
    auto& pop_continue = do_make_checkpoint(parser, state, eval, eo_continue);
    auto& best = state.pack< eoBestIndividualStat<R> >();
    pop_continue.add( best );
    auto& distrib_continue = state.pack< edoContAdaptiveFinite<CMA> >();

    auto& selector  = state.pack< eoRankMuSelect<R> >(dim/2);
    auto& estimator = state.pack< edoEstimatorNormalAdaptive<R> >(gaussian);
    auto& bounder   = state.pack< edoBounderRng<R> >(R(dim, -5), R(dim, 5), gen);
    auto& sampler   = state.pack< edoSamplerNormalAdaptive<R> >(bounder);
    auto& replacor  = state.pack< eoCommaReplacement<R> >();

    make_verbose(parser);
    make_help(parser);

    auto& algo = state.pack< edoAlgoAdaptive<CMA> >(
         gaussian , pop_eval, selector,
         estimator, sampler , replacor,
         pop_continue, distrib_continue);

    try {
        algo(pop);
    } catch (eoMaxEvalException& e) {
        eo::log << eo::progress << "STOP" << std::endl;
    }

    std::cout << best.value() << std::endl;
    return 0;
}
```

git clone https://github.com/nojhan/paradiseo.git

sudo apt install g++-8 cmake make libeigen3-dev libopenmpi-dev doxygen graphviz libgnuplot-iostream-dev

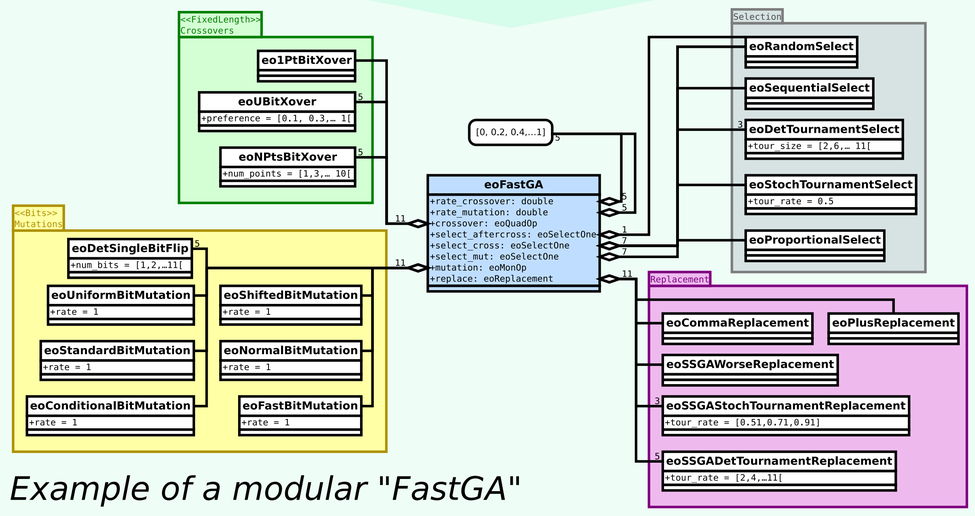
mkdir build ; cd build ; cmake -DEDO=ON .. && make -j

c++ cmaes.cpp -I../eo/src -I../edo/src -DWITH_EIGEN=1 -I/usr/include/eigen3 -std=c++17 -L./lib/ -leo -leoutils -les -o cmaes

./cmaes --help

MODULAR TEMPLATES

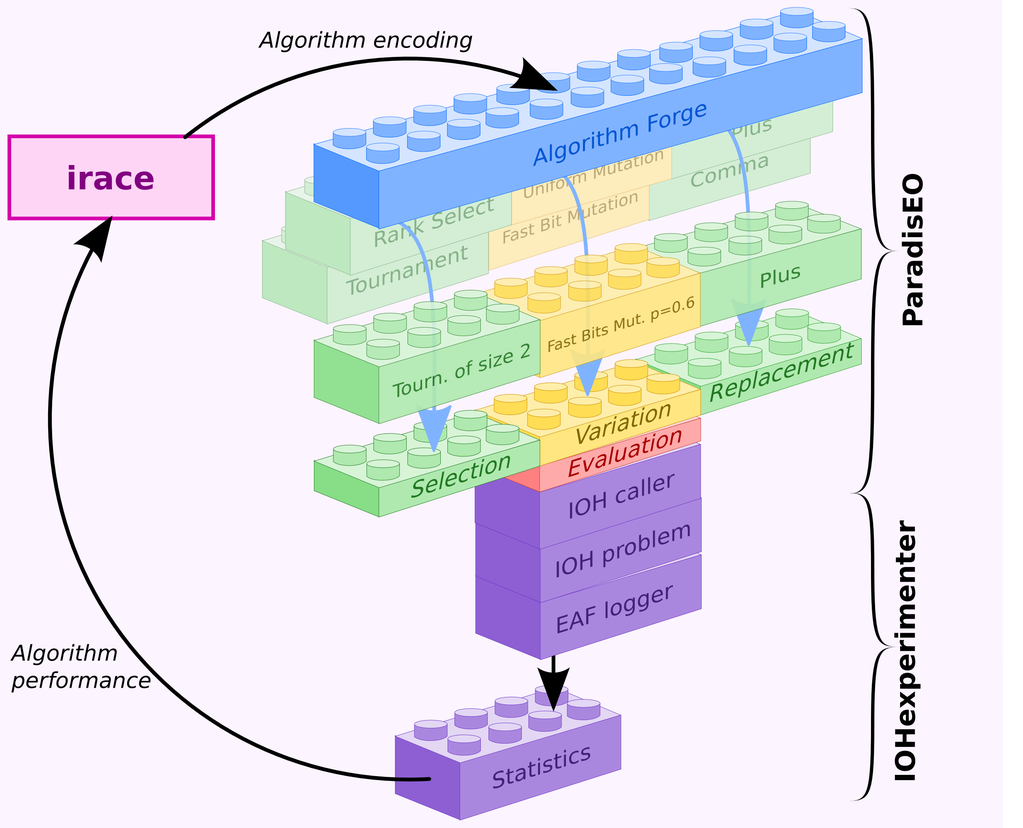
- Evolutionary Algorithms
- Local Searches
- Estimation of Distribution Algo.
- Particle Swarm Optimization
- Multi-Objective Algorithms



Example of a modular "FastGA"

AUTOMATED DESIGN

- Many Interoperable Operators
- On-the-fly Instantiation
- ***Millions of Algorithms*** Instances
- ***irace*** API automatic Export



https://nojhan.github.io/paradiseo

INSTITUT PASTEUR