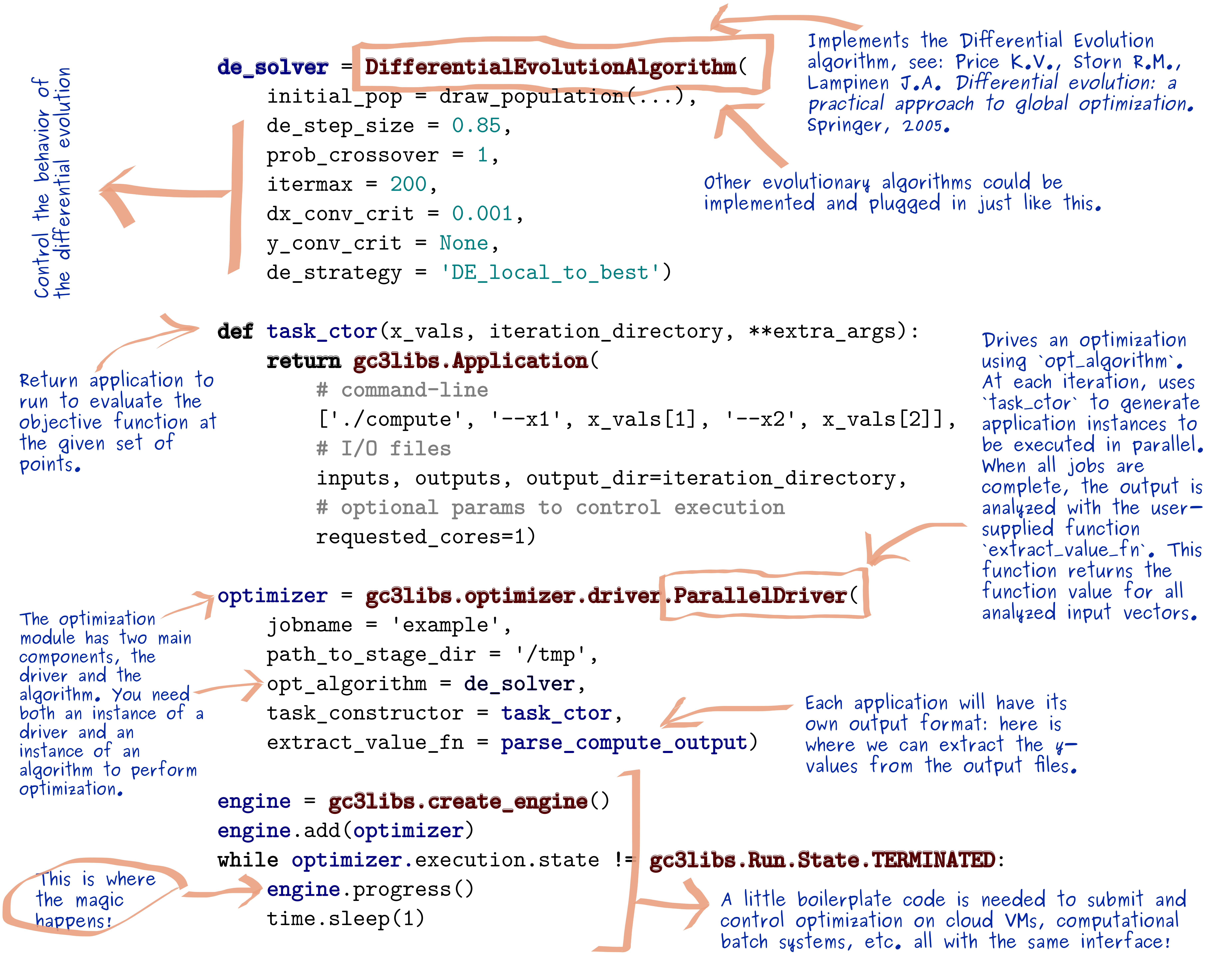
Global Optimization with GC3Pie

http://gc3pie.googlecode.com   gc3pie@googlegroups.com

The GC3Pie Optimizer module

GC3Pie can run a large number of application instances in parallel. The idea of this optimization module is to use these core capabilities to perform optimization, which is particularly effective for optimization using evolutionary algorithms, as they require several independent evaluations of the target function.

This module implements a generic framework for evolutionary algorithms, and one particular type of global optimization algorithm called *Differential Evolution* is worked out in full. Other Evolutionary Algorithms can easily be incorporated by subclassing class `EvolutionaryAlgorithm`. (Different optimization algorithms, for example gradient based methods such as quasi-newton methods, could be implemented.)



```
de_solver = DifferentialEvolutionAlgorithm(
    initial_pop = draw_population(...),
    de_step_size = 0.85,
    prob_crossover = 1,
    itermax = 200,
    dx_conv_crit = 0.001,
    y_conv_crit = None,
    de_strategy = 'DE_local_to_best')

def task_ctor(x_vals, iteration_directory, **extra_args):
    return gc3libs.Application(
        # command-line
        ['./compute', '--x1', x_vals[1], '--x2', x_vals[2]],
        # I/O files
        inputs, outputs, output_dir=iteration_directory,
        # optional params to control execution
        requested_cores=1)

optimizer = gc3libs.optimizer.driver.ParallelDriver(
    jobname = 'example',
    path_to_stage_dir = '/tmp',
    opt_algorithm = de_solver,
    task_constructor = task_ctor,
    extract_value_fn = parse_compute_output)

engine = gc3libs.create_engine()
engine.add(optimizer)
while optimizer.execution.state != gc3libs.Run.State.TERMINATED:
    engine.progress()
    time.sleep(1)
```

Implements the Differential Evolution algorithm, see: Price K.V., Storn R.M., Lampinen J.A. *Differential evolution: a practical approach to global optimization.* Springer, 2005.

Other evolutionary algorithms could be implemented and plugged in just like this.

Control the behavior of the differential evolution

Return application to run to evaluate the objective function at the given set of points.

Drives an optimization using `opt_algorithm`. At each iteration, uses `task_ctor` to generate application instances to be executed in parallel. When all jobs are complete, the output is analyzed with the user-supplied function `extract_value_fn`. This function returns the function value for all analyzed input vectors.

The optimization module has two main components, the driver and the algorithm. You need both an instance of a driver and an instance of an algorithm to perform optimization.

Each application will have its own output format: here is where we can extract the y-values from the output files.

This is where the magic happens!

A little boilerplate code is needed to submit and control optimization on cloud VMs, computational batch systems, etc. all with the same interface!

Differential Evolution Optimizer: how does it work?

The differential evolution optimizer is used to find the global minimum of a generic function. The GC3Pie implementation works as follows:

1- A 'population' of initial *x*-points is supplied by the user (the `draw_population` convenience function is provided to generate a random one).

2- The function to minimize is then evaluated for each member of the population (./compute). Evaluation at distinct points happens in parallel: GC3Pie manages the task distribution to the available compute resources, supervises execution and retrieves results.

3- Once all computations have finished, the user-supplied function `extract_value_fn` is applied to each application's output file to get the computed *y*-value.

4- The optimizer (`de_solver`) then evaluates the convergence conditions. In the example, either all x-points are within a radius of 0.001 (`dx_conv_crit`), or the maximum number of iterations is reached.

5- If convergence is not reached, a new population is generated and evaluated starting from the current one.

What is GC3Pie?

GC3Pie is a library of Python classes to execute and control applications on distributed computing resources: e.g., cloud-based VMs, batch-queueing clusters, or a bunch of machines that you can SSH into.

GC3Pie is object-oriented: basic classes abstract the generic and repetitive part of application scripting, and let you focus on coding what is specific to your use case. Generic services provided by GC3Pie include: asynchronous job execution, programmatic generation of template files, checkpoint/restart workflow execution.

GC3Pie is a toolkit: it provides the building blocks to write Python scripts to run large computational campaigns (e.g., to analyze a vast dataset or explore a parameter space), and to combine several tasks into a dynamic workflow.

How is GC3Pie different?

Unlike other Python frameworks for distributing computation, e.g., Celery or Pyro, GC3Pie is designed to coordinate the execution of independent Applications (often pre-existing and written in another language): with GC3Pie you write Python code to steer the computation, not to perform it.

Universität Zürich^UZH

Riccardo Murri, Sergio Maffioletti,
Antonio Messina, Tyanko Aleksiev,
Benjamin Jonen

Grid Computing Competence Center
University of Zurich
http://www.gc3.uzh.ch/