Synergies
Cooperation among metaheuristics

Álvaro Rodríguez

Universidad Autónoma de Madrid

Universidad Complutense de Madrid

Let's remember

To solve an optimization problem one should ask himself in which way a given function f s.t. $f : R^n \longrightarrow R$ could be optimized (i.e. find a position associated to its best fitness).

Let's remember

To solve an optimization problem one should ask himself in which way a given function f s.t. $f : R^n \longrightarrow R$ could be optimized (i.e. find a position associated to its best fitness).

During the subject we have seen different types of metaheuristics, and implemented some of them.

Let's remember

To solve an optimization problem one should ask himself in which way a given function f s.t. $f : R^N \longrightarrow R$ could be optimized (i.e. find a position associated to its best fitness).

During the subject we have seen different types of metaheuristics, and implemented some of them.

Metaheuristic

An experimental heuristic method for solving a general class of computational problems by combining user procedures in the hope of obtaining a more efficient or robust procedure.

Which one to choose?

Plenty of them

PSO, ABC, FA, WCA, DE, etc.

Which one to choose?

Plenty of them

PSO, ABC, FA, WCA, DE, etc.

Decision problem

When facing such problems one must consider which metaheuristic would have a better performance, or which of them could find the best solution.

Which one to choose?

Plenty of them

PSO, ABC, FA, WCA, DE, etc.

Decision problem

When facing such problems one must consider which metaheuristic would have a better performance, or which of them could find the best solution.

New problem

Metaheuristic has been selected, now we face a new question: which parameters should one use after selecting a metaheuristic?

Approximation

We need to find suitable parameters. Even tiny variations could significantly improve the results in some situations.

Approximation

We need to find suitable parameters. Even tiny variations could significantly improve the results in some situations.

We could run, for example, multiple instances of the same metaheuristic varying the parameters and select the position associated with the best fitness (the best among all instances).

Approximation

We need to find suitable parameters. Even tiny variations could significantly improve the results in some situations.

We could run, for example, multiple instances of the same metaheuristic varying the parameters and select the position associated with the best fitness (the best among all instances).

This leads to an unwanted situation, where a lot of hand-crafted code is expected to be wrote just for doing something that could be easily automated.

A better solution

Instead of selecting a metaheuristic and then its parameters, it's better to let a class do the work for us: comparison and information exchange to share best fitnesses.

No need to consider further comparisons:

- Create different instances of $\neq$ metaheuristics
- Create different instances of $=$ metaheuristic but with $\neq$ parameters

Synergy

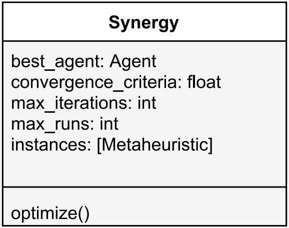
For implementing this idea of cooperation we can consider a class ”Synergy” with the following parameters:

- *metaheuristics*, a list of instances.
- *convergence criteria*, a float value that establishes the value from which we would think the difference is from now negligible.
- *max iterations*, the times each instance will work per run of the algorithm.
- *max runs*, the number of times synergy will execute each instance for the number of iterations given.

Each metaheuristic will run for *max iterations* for a total number of *max runs*, iff convergence criteria is not satisfied.

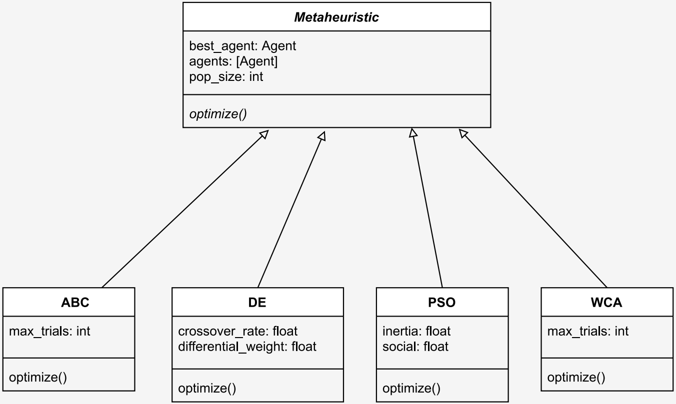
Synergy

Such class can be represented in this way, with parameters regarding the best agent (i.e. best fitness) found and others already mentioned.



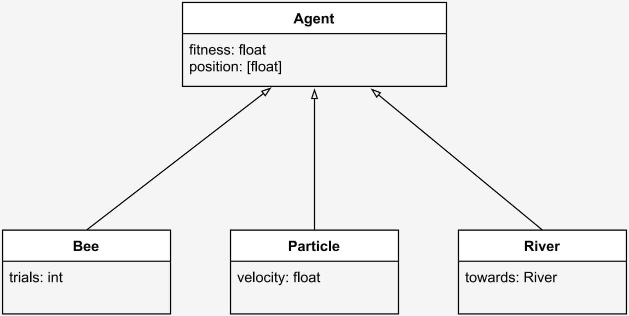
Metaheuristic

Such synergy class requires class that can be used as a black box, from which each particular metaheuristic will extend.

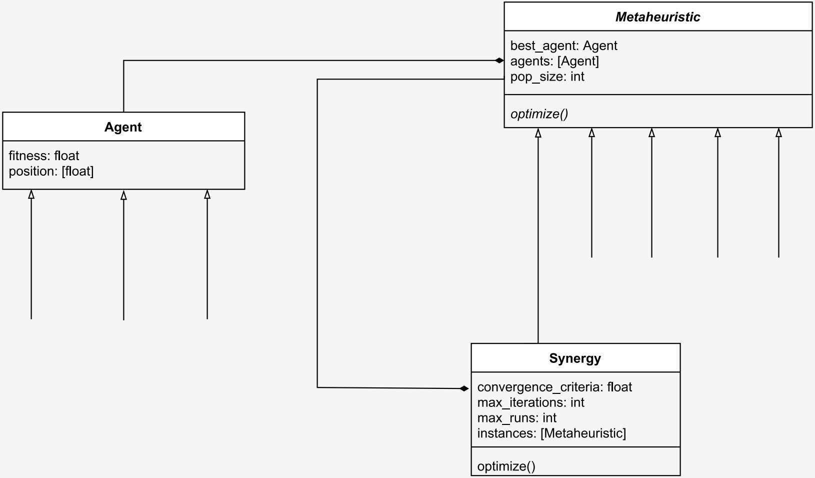


Agents

A general class for agents should include only fitness and position, and then each particular agent (bees, particles, etc) add some extra information required.



All together



Pseudocode

```
Input: instances, max_iterations, max_runs, criteria
Output: best fitness
for i in range(max runs) do
    for instance in instances do
        for j in range(max iterations) do
            if instance.optimize() improves best_agent then
                best_agent = instance.best_agent;
            end
        end
    end
    if criteria satisfies then
        break;
    end
end
```

Optimizing Rastrigin

$$f(X) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$$

Bounds

$-5.12 \leq x_i \leq 5.12, i = 1, 2, ..., n$ where n is the number of dimensions, here $n = 10$.

We know

$$f_{min}(X*) = 0$$
$$x_i^* = 0$$

Now, some experiments done. First, we could consider a basic configuration with one running instance, which is possible, although not of the upmost interest.

Now, some experiments done. First, we could consider a basic configuration with one running instance, which is possible, although not of the upmost interest.

The time taken to optimize a function will depend on different variables, so different strategies, although they could take more time to compute, typically they will be able to explore more space and indeed find a better solution.

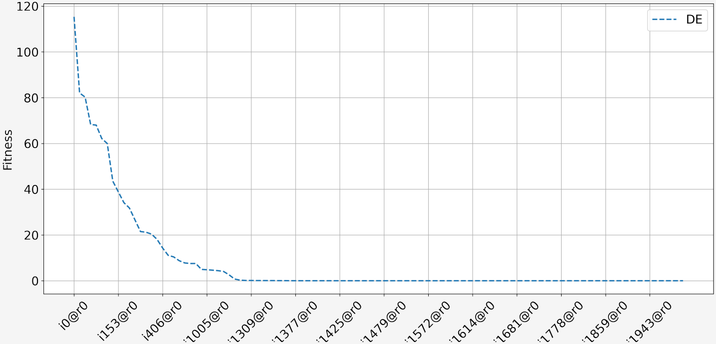
Now, some experiments done. First, we could consider a basic configuration with one running instance, which is possible, although not of the upmost interest.

The time taken to optimize a function will depend on different variables, so different strategies, although they could take more time to compute, typically they will be able to explore more space and indeed find a better solution.

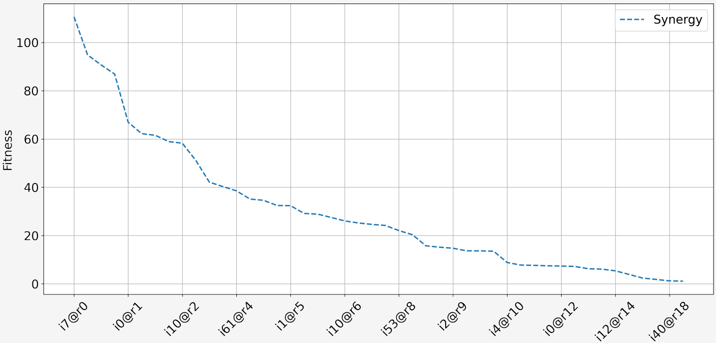
Consider:

- **runs**: 1 (DE) and 20 (DE + ABC + WCA + PSO)
- **iterations**: 2000 both
- Tuning parameters adjusted following research done for each.

DE


120
100
80
60
40
20
0
Fitness
DE
i0@r0
i153@r0
i406@r0
i1005@r0
i1309@r0
i1377@r0
i1425@r0
i1479@r0
i1572@r0
i1614@r0
i1681@r0
i1778@r0
i1859@r0
i1943@r0

DE + PSO + WCA + ABC



What about *runs* and *iterations*?

- **runs** controls where an instance stops and another starts.
- **iterations** controls for how much steps an instance will execute before letting another start.

If we want each instance to run for N times:

$$N = R \times I,$$

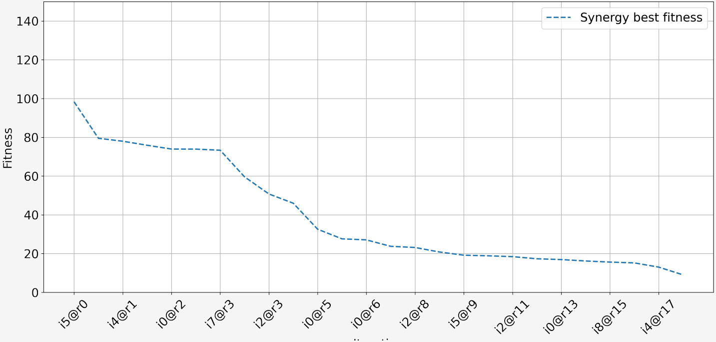
with R number of *runs* and I number of **iterations**.

Let's test it

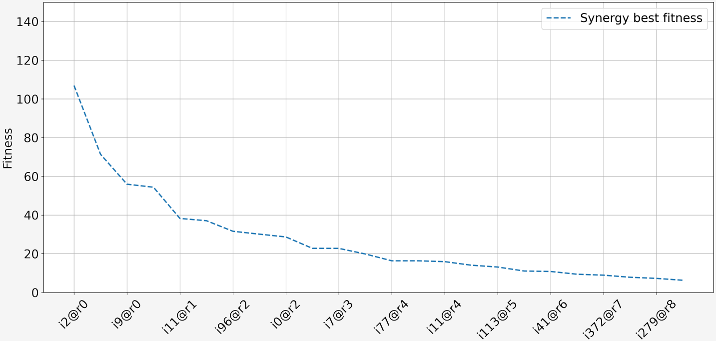
$\uparrow\downarrow \implies$ 10, 1000 in each case

$\sim \implies$ 100 of each

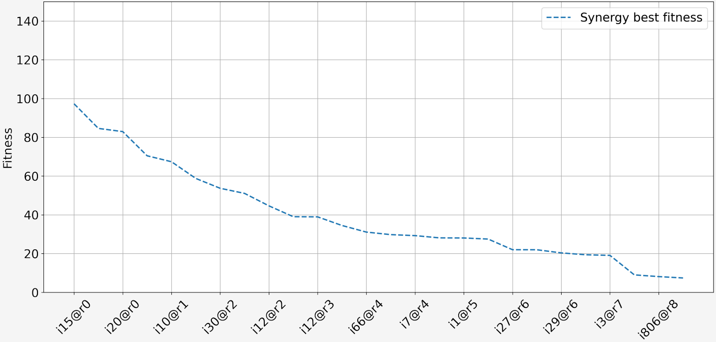
↑ runs, ↓ iterations



runs ~ iterations



↓ runs, ↑ iterations



And what?

- Strategy in the end reaches near values (negligible difference if we consider random starts)
- The way best fitness is found varies in the way it decays
- It's safe to keep an average between both parameters: high runs with low iterations will update too quickly the values and won't allow a strategy succeed in the short time $\longrightarrow$ need to wait for the rest to complete a run and the could continue.

Conclusions

Synergies...

- Can use different strategies without considering which one would fit better for a problem.
- Can communicate with almost every metaheuristic than can be implemented
- They can run easily in parallel: each run is independent, as global best is updated after such instance finishes.

Future Work

Some topics still need to be addressed:

- Update the synergy class presented to be compatible with parallel implementations
- Further research in what metaheuristics would collaborate better with each other
- Include different metaheuristics, as for now it just gives support to 4 of them.