# Parallel Multiobjective Optimization Algorithms

Sébastien Verel

LISIC
Université du Littoral Côte d'Opale
Equipe OSMOSE

verel@univ-littoral.fr
http://www.lisic.univ-littoral.fr/~verel

Master informatique WeDSci, ULCO,

2022, version 0.1

# Expensive optimization

When the computation of objective values are expensive,
3 main strategies can be used:

- Clever algorithm to speed up the convergence using relevant variables (sub-search space)
- Parallelism : increase and benefit of parallel system
- Surrogate model : learn to computation efficient model of objectives

# Parallel algortihms

## "full" distributed algorithms

Cons:

# Parallel algortihms

## "full" distributed algorithms

Cons:

Sharing information

Pros:

# Parallel algortihms

## "full" distributed algorithms

Cons:

Sharing information

Pros:

Balance of the work load (but algo design...)

Robust to failure

## Maste/slaves algorithms

Cons:

# Parallel algortihms

## "full" distributed algorithms

Cons:

- Sharing information

Pros:

- Balance of the work load (but algo design...)
- Robust to failure

## Maste/slaves algorithms

Cons:

- Heterogeneous on computation time on Slaves
- Overflow the master (time and data)

Pros:

# Parallel algortihms

## "full" distributed algorithms

Cons:

- Sharing information

Pros:

- Balance of the work load (but algo design...)
- Robust to failure

## Maste/slaves algorithms

Cons:

- Heterogeneous on computation time on Slaves
- Overflow the master (time and data)

Pros:

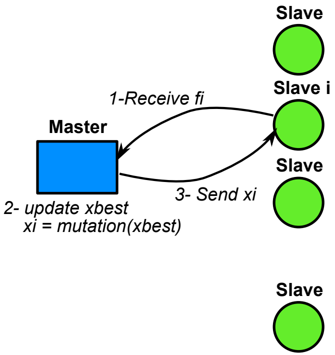
- Centralized information
- Easy to implement

# Asynchronous Master-Workers MOEA/D

Drouet, V., S. Verel, and J-M. Do.
"Surrogate-assisted asynchronous multiobjective algorithm for nuclear power plant operations."
In Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 1073-1081. 2020.

# Asynchronous distributed $(1 + \lambda)$-Evolution Strategy

## Master-slaves architecture




### Algorithm on Master

$\{x_1, \ldots, x_\lambda\} \leftarrow$ **Initialization**()
**for** $i = 1..\lambda$ **do**
  **Send (Non-blocking)** $x_i$ to slave $S_i$
**end for**
$x_{best} \leftarrow \emptyset$, and $f_{best} \leftarrow \infty$
**repeat**
  **if** there is a pending mess. from $S_i$ **then**
    **Receive** fitness $f_i$ of $x_i$ from $S_i$
    **if** $f_i \leqslant f_{best}$ **then**
      $x_{best} \leftarrow x_i$, and $f_{best} \leftarrow f_i$
    **end if**
    $x_i \leftarrow \texttt{mutation}(x_{best})$
    **Send (Non-blocking)** $x_i$ to slave $S_i$
  **end if**
**until** time limit

# Quick analysis of parallel algortihms

The execution time of a job on slave is a random variable $T$ with average $\mu$, and standard deviation $\sigma$.
$n$ slaves are available.

When the jobs are supposed independent (asynchronous), what is the average time between 2 jobs on master?

Application with $\mu = 10\text{min}$, $\sigma \approx \mu/2$, and $n = 1000$.

Conclusion?

# Asynchronous Master-Workers MOEA/D

```
for i ← 1 to N_proc do
    D_i ← Attribute direction;
    x_i ← Initialize using Sobol numbers;
    Send x_i to process i;  // Non blocking communication
end
z* ← Initialize reference point;
f*_D ← Initialize best fit for each direction D;
x*_D ← Initialize best solution for each direction D;
S ← ∅ ;                        // Set of evaluated solutions
while time left do
    Receive Msg from process i w.r.t. direction D_i;
    (f, x) ← Msg; // Receive obj. vector and solution
    Normalize f;                               // See 2.4
    S ← S ∪ {(x, f)};
    for k ← 1 to d do
        if f_k < z*_k then
            z*_k ← f_k;          // Update reference point
        end
    end
    for D ∈ N(D_i) do
        if g(f|w_D, z*) < g(f*_D|w_D, z*) then
            x*_D ← x;      // Update best known solution
            f*_D ← f;
        end
    end
    Train surrogate models M with S ;
    if ♯S < N_start then
        x′ ← Mutate x*_{D_i};
    else
        Select x′ using surrogate models M;  // See Alg. 2
    end
    Send x′ to process i;  // Non blocking communication
end
```