Introduction to Multi-Objective Optimization and its Applications

Laurent.Lemarchand@univ-brest.fr

Lab-STICC
labsticc.univ-brest.fr/~lemarch

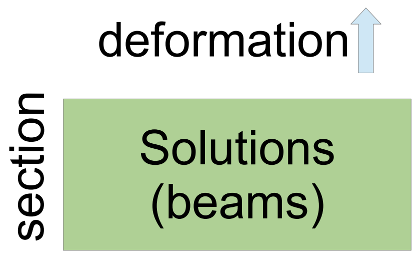
Europska unija
"Zajedno do fondova EU"

EUROPSKI STRUKTURNI
I INVESTICIJSKI FONDOVI

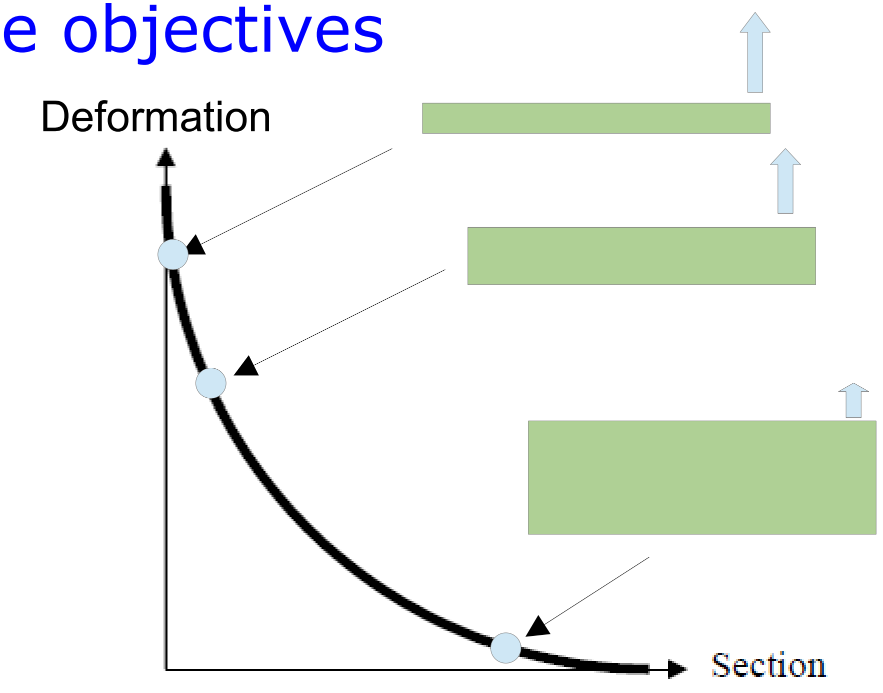
ESF
UČINKOVITI
LJUDSKI
POTENCIJALI

Projekt je financirala Europska unija iz Europskog Socijalnog fonda

Introduction to MOO



Single vs (simultaneous) multiple objectives

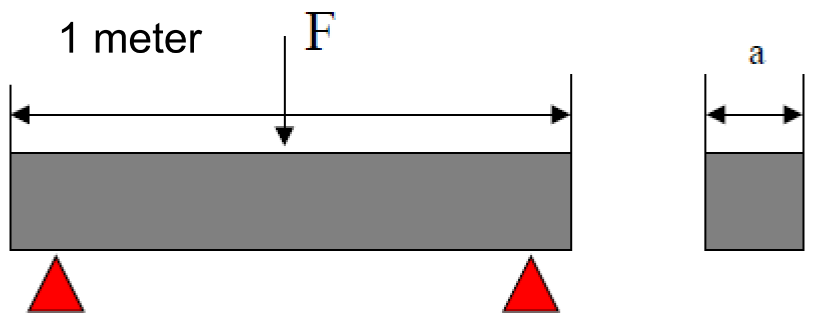


- Many optimization methods (with constraints and single or multiple optimization objectives)
- A beam : section (→ weight) vs deformation

$$S(a)=a^2$$

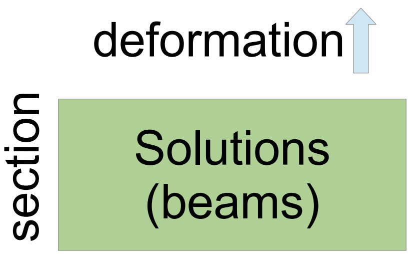
$$d(a)=1000+\frac{1.10^{-2}}{192+2.10^5+\frac{a^4}{12}}$$

$$a\leq 0.1$$

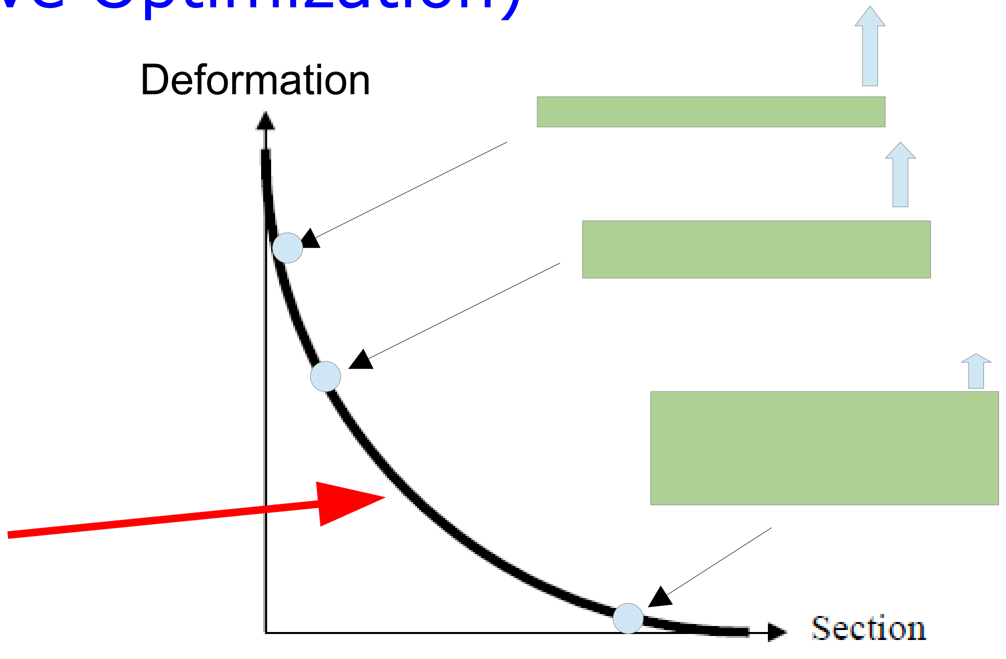


MOO

Goal : find trade-offs



SOO ≠ MOO (Multiple Objective Optimization)

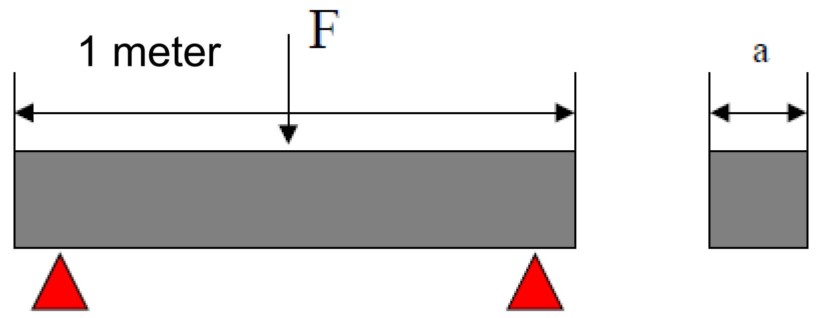


- Many objective fonctions
  - antagonism
- No best solution
  - set of solutions

$$S(a)=a^2$$

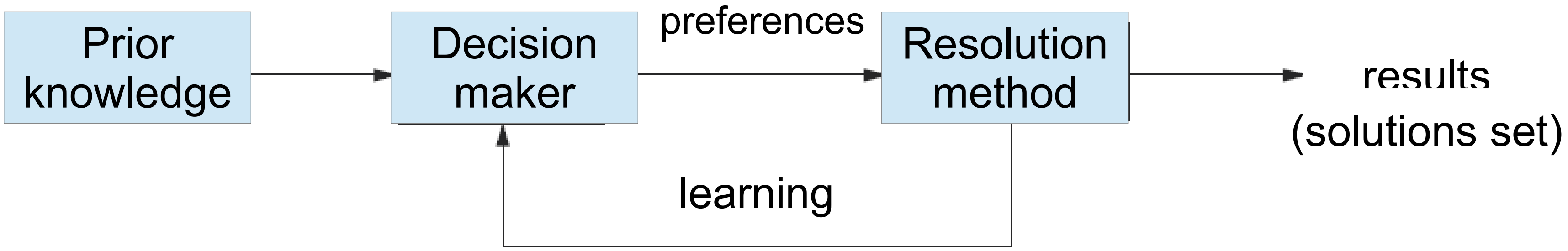
$$d(a)=1000+\frac{1.10^{-2}}{192+2.10^5+\frac{a^4}{12}}$$

$$a\leq 0.1$$



Decision process

Our goal is not to choose/decide ...



Prior knowledge

Decision maker

preferences

Resolution method

results (solutions set)

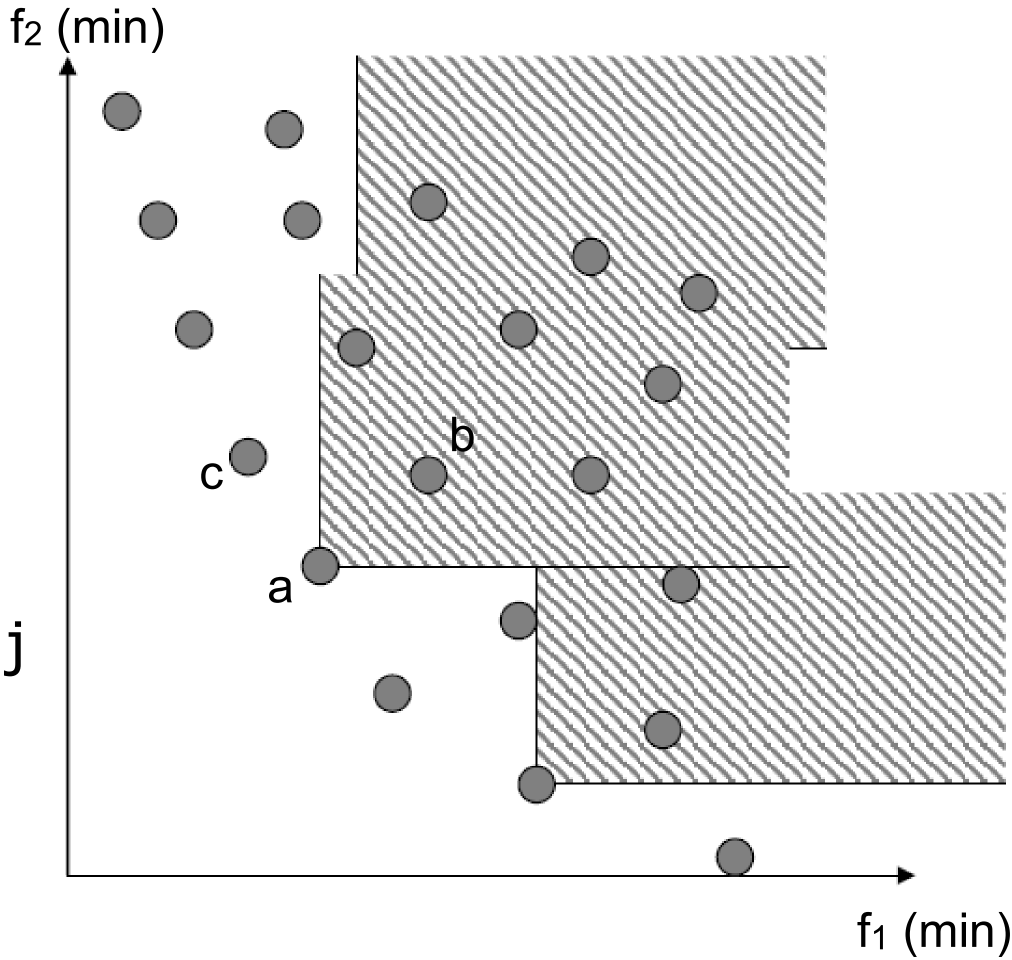
learning

- a priori search
  - Priorizing bias (eg. Aggregation method)
- a posteriori search → get whole set of solutions
  - Maybe difficult to analyze
- Interactive search
  - ... helps the decision process

Dominance

Our goal is to find good trade-offs

- How to compare solutions to each other ?
- Solution a *dominates* solution b if
  - a is as good as b for all of the optimization criteria i :
- $\forall$ i, $f_i(a) \lfloor f_i(b)$
  - There is at least one criterium j where a is better than b :
- $\exists$ j, $f_j(a) < f_j(b)$



MOO

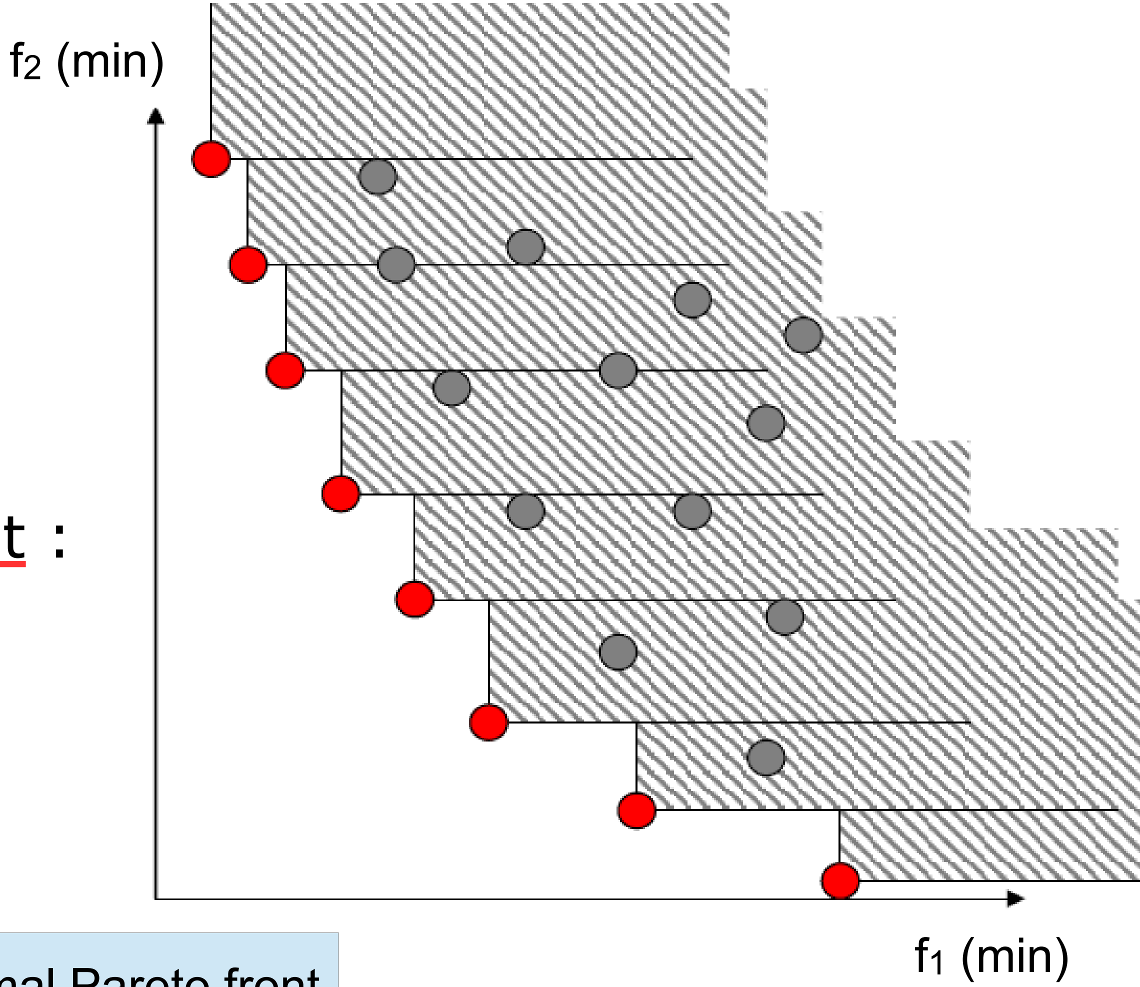
Pareto front

V. Pareto (economist): *in some cases, you can not improve someone income without degrading somebody else*

- Non-dominated solutions set
  - Optimal solutions according to Pareto dominancy relationship

    → Pareto Set

  Mapping from decision to objective space → Pareto Front :
  - maximal/minimal : all of/a single solution(s) for a given objective function vector



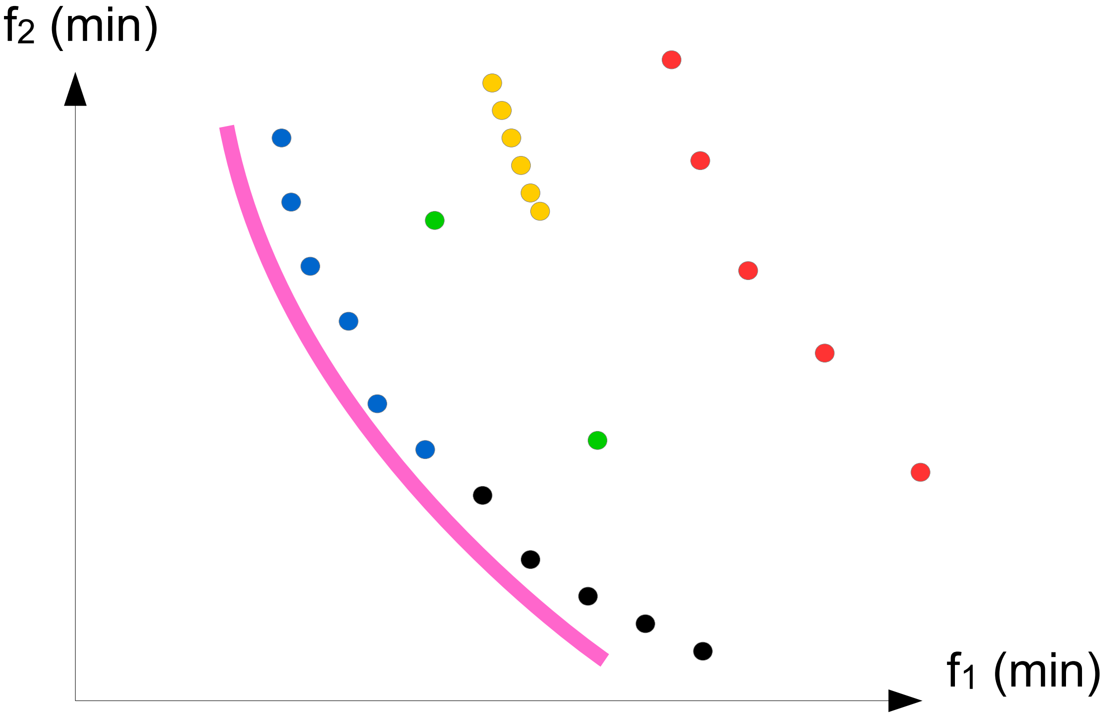
Generally, MOO algorithms look for a minimal Pareto front

Properties of Fronts

Many metrics for comparing fronts with each others or with (exact) Pareto front. Must take care of:

- Density → number of solutions ●●●
- Accurracy → close to Pareto front ●●●
- Sparsity → diversity of solutions ●●●

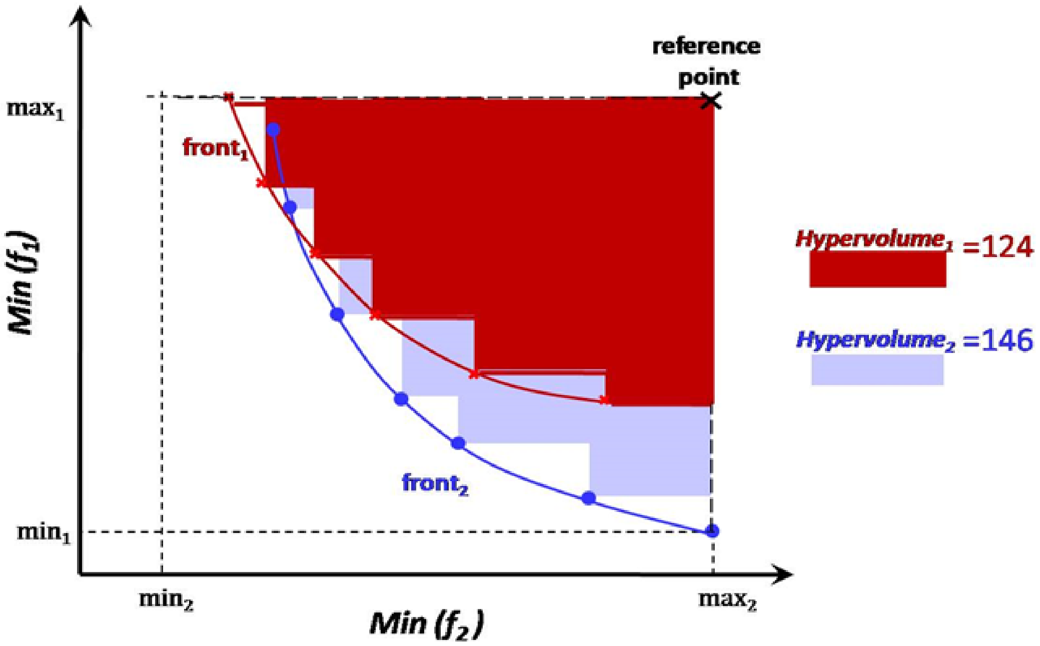
the best front
● ● ●

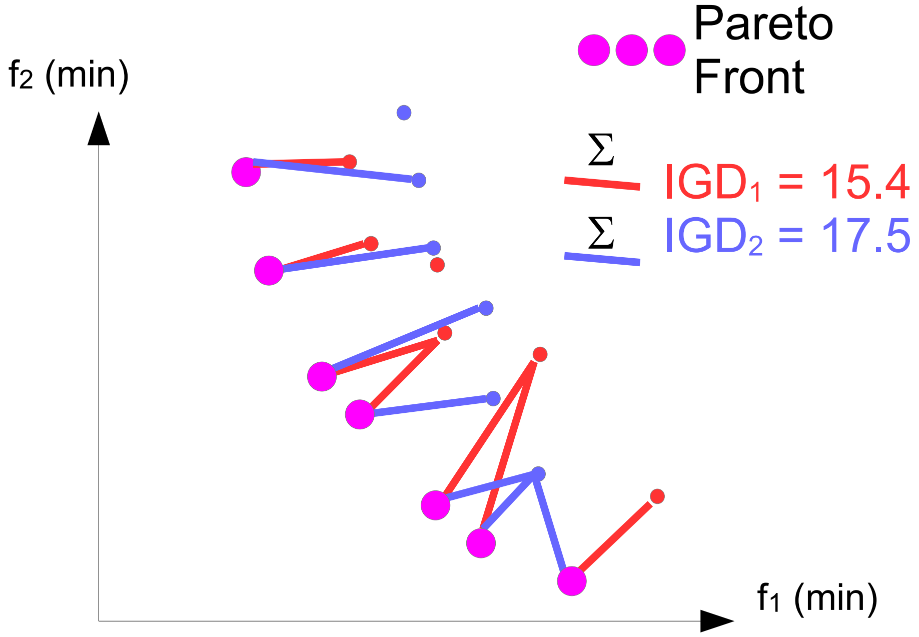


Comparison metrics

Many metrics for comparing fronts which each other or with (exact) Pareto front.

- Front → scalar value
- Hypervolume → compare two approximative fronts
- Inverse Generational Distance → compare to Pareto Front





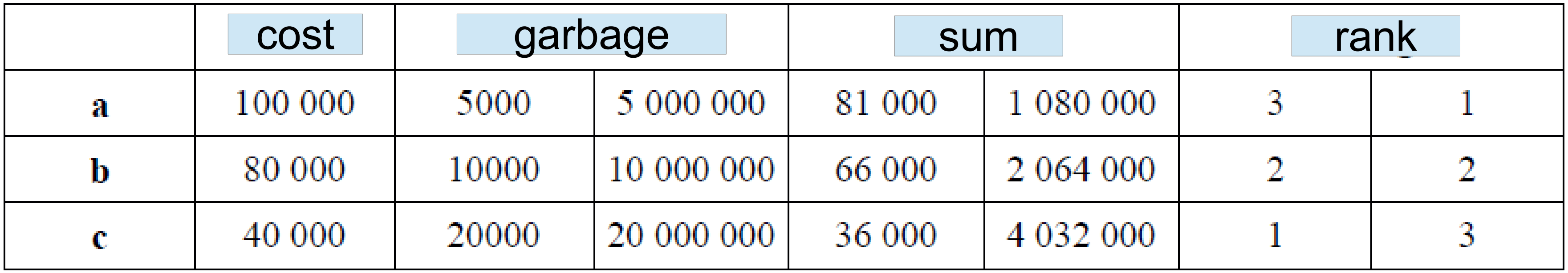
Comparison metrics

Impact of the different metrics on comparison results

- Scale / range of values for each metric
  - Normalization requested or
- Implicit bias toward one objective

- Example : Kilos → tons for a single objective → inversed dominance

f_1 **f_2** **$80\%f_1 + 20\%f_2$**



	cost	garbage		sum		rank	
a	100 000	5000	5 000 000	81 000	1 080 000	3	1
b	80 000	10000	10 000 000	66 000	2 064 000	2	2
c	40 000	20000	20 000 000	36 000	4 032 000	1	3

Algorithms

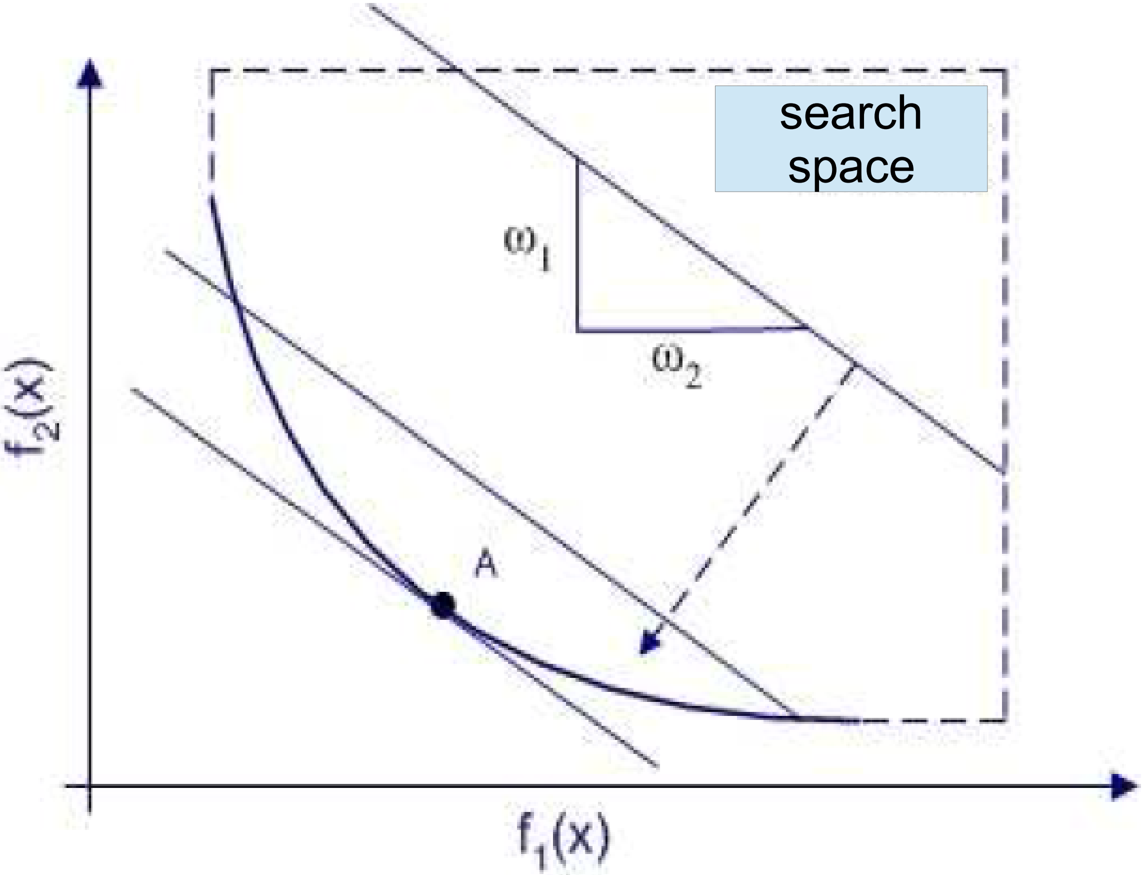
Only a few of them here

- Aggregation based methods → SOO
  - Weighted sum, Goal programming, Chebysheff, …
- A method based on Linear Programming
  - ε-constraints → transforms objective into constraints
  - Exact method for IP
- A non dominance based method
  - VEGA → Process objectives independently

SOO methods for MOO problems

Combine objectives in a weighted sum

- min f(x) = (f1(x), f2(x), …, fn(x))
- min f'(x) = ω_1.f1(x) + ω_2.f2(x) + … + ω_n.fn(x)
  with ω_1 + … + ω_n = 1
- If convex space, optimal point A tangeant to line of head $-\omega_1 / \omega_2$



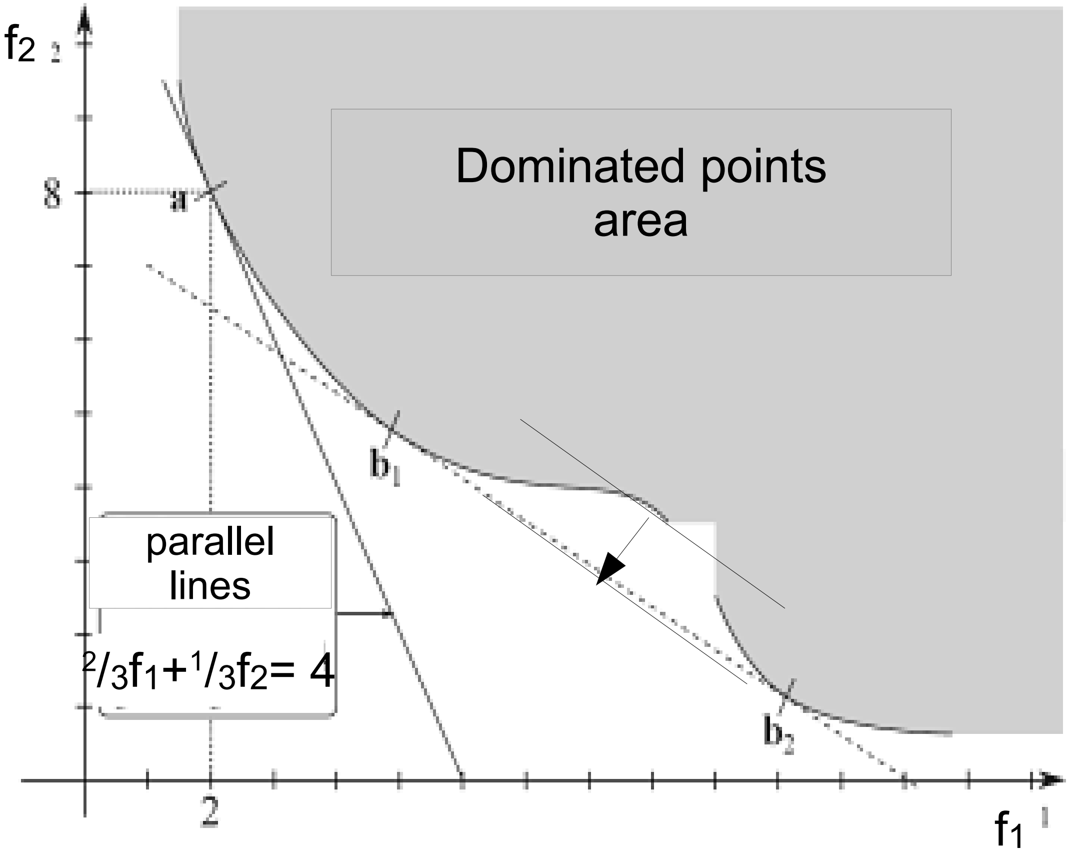
MOO

SOO methods for MOO problems

Combine objectives in a weighted sum

- Problem for non convex fronts
  - Non combination of weights ω_i for some points (unsupported solutions)

- For points between b1 and b2, you can shift the line to obtain a better value for the sum



SOO methods for MOO problems

Goal programming

- min $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$
- min $f'(x) = |f_1(x) - T_1| + |f_2(x) - T_2| + \ldots + |f_n(x) - T_n|$
- T_1, T_2, T_n are Target values for each objective
- Each objective can also be weighted
- Controlled bias

Lexicograph method

- Sort objectives by priority
- Optimize f_1. If a single solution at optimal value f_1^*, stop.
- Else, optimize f_2 for solutions with f_1^* value, and so on
- Controlled bias

EMOA

Evolution Based Multi-Objective Algorithms

- Mainly based on dominance property
  - Evolution of a population → neighborhood operators
  - Niching : fitness sharing/ crowding → how to keep diversity
  - Elistism (e.g with archive) → keep best individuals

- PAES : (1 + 1) + crowding + archive [Knowles 1999]
- NSGA2 : (μ, λ) + population + crowding [Deb 1994]
- IBEA : indicator driven evolution [Zitler 2004]

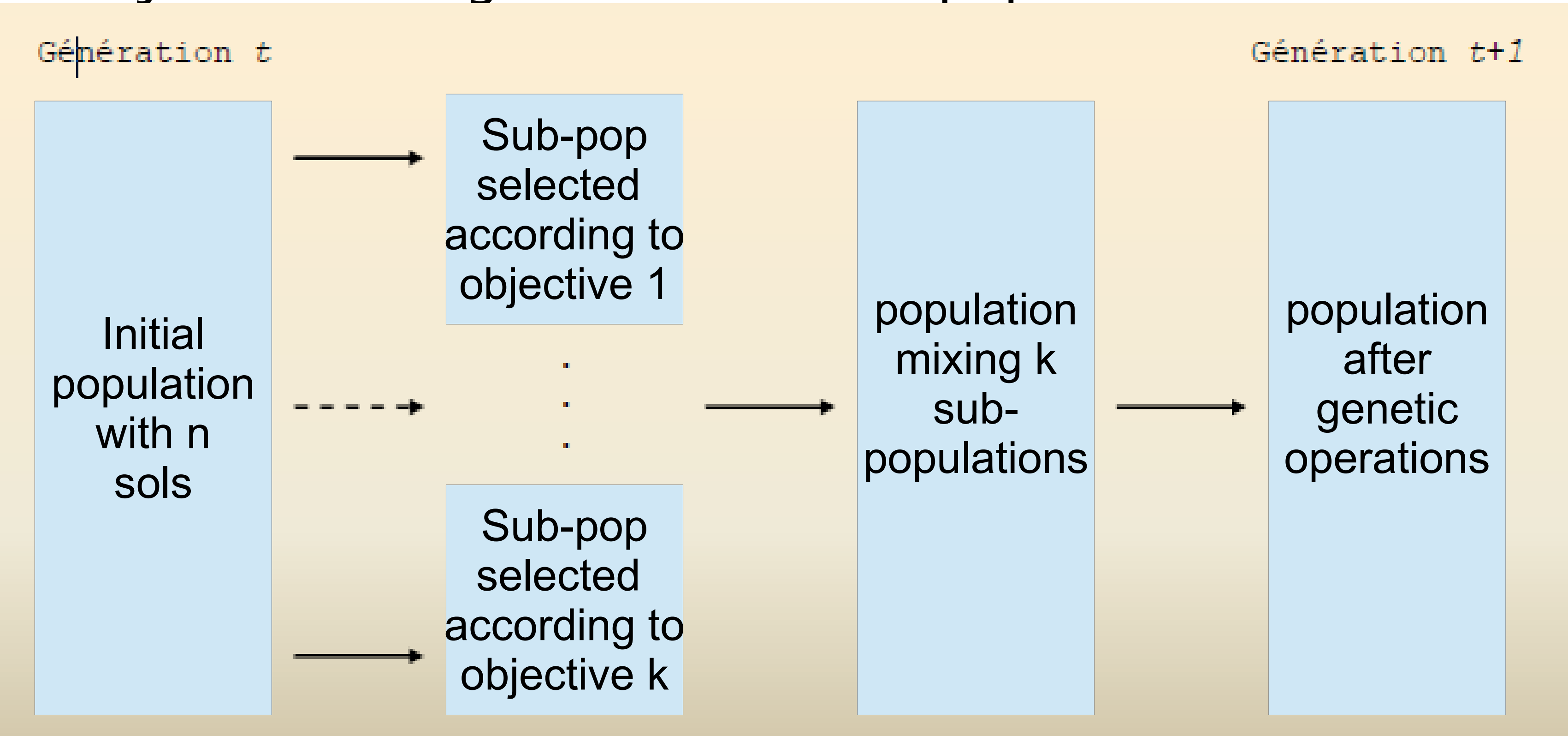
- Many others : SPEA2, MOGA, ...

MOO

EMOA

A non Pareto based method: VEGA

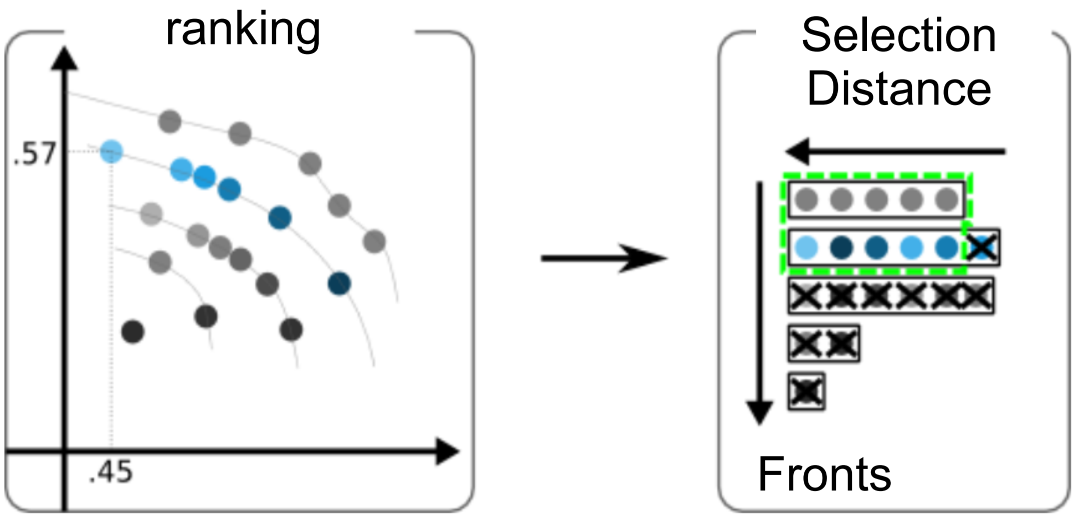
- Vector Evaluated GA
  - A Genetic Algorithm
  - Objective changes for each sub-population selection

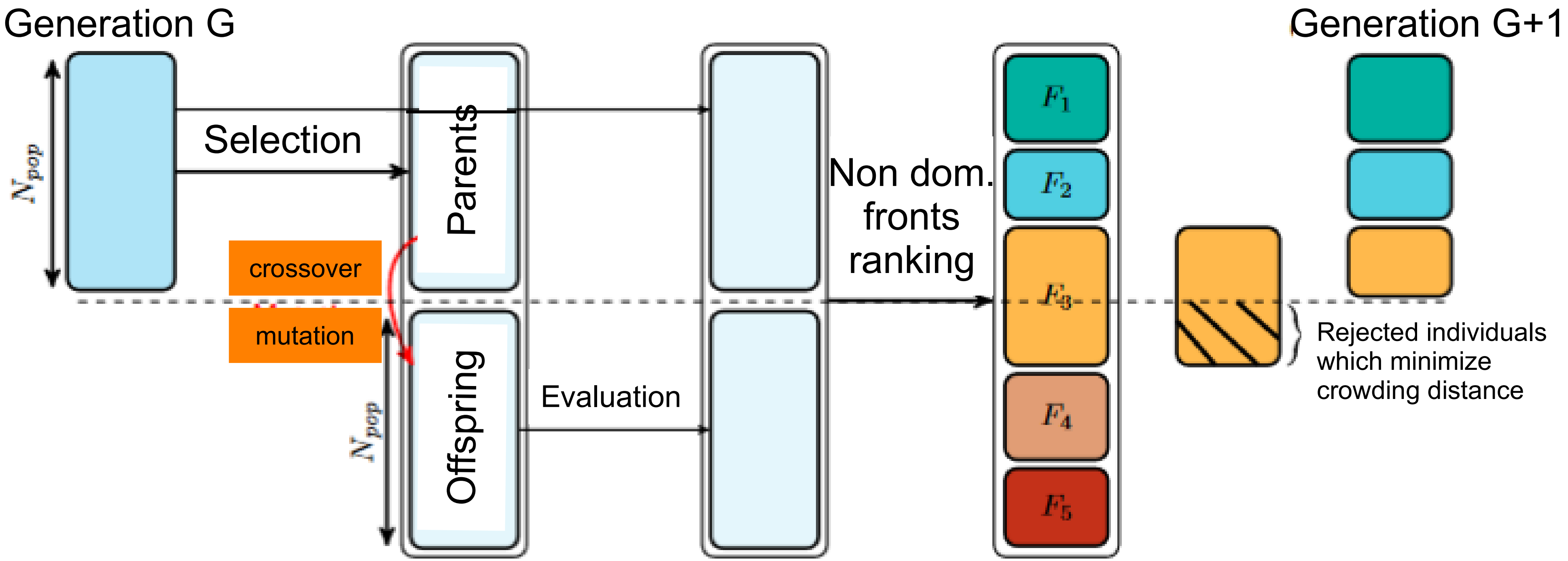

Génération t
Génération t+1
Initial population with n sols
Sub-pop selected according to objective 1
Sub-pop selected according to objective k
population mixing k sub-populations
population after genetic operations

EMOA

NSGA-II: sorting population by fronts

- Elitist reproduction
- Dominating fronts first
- Most isolated solutions of each front



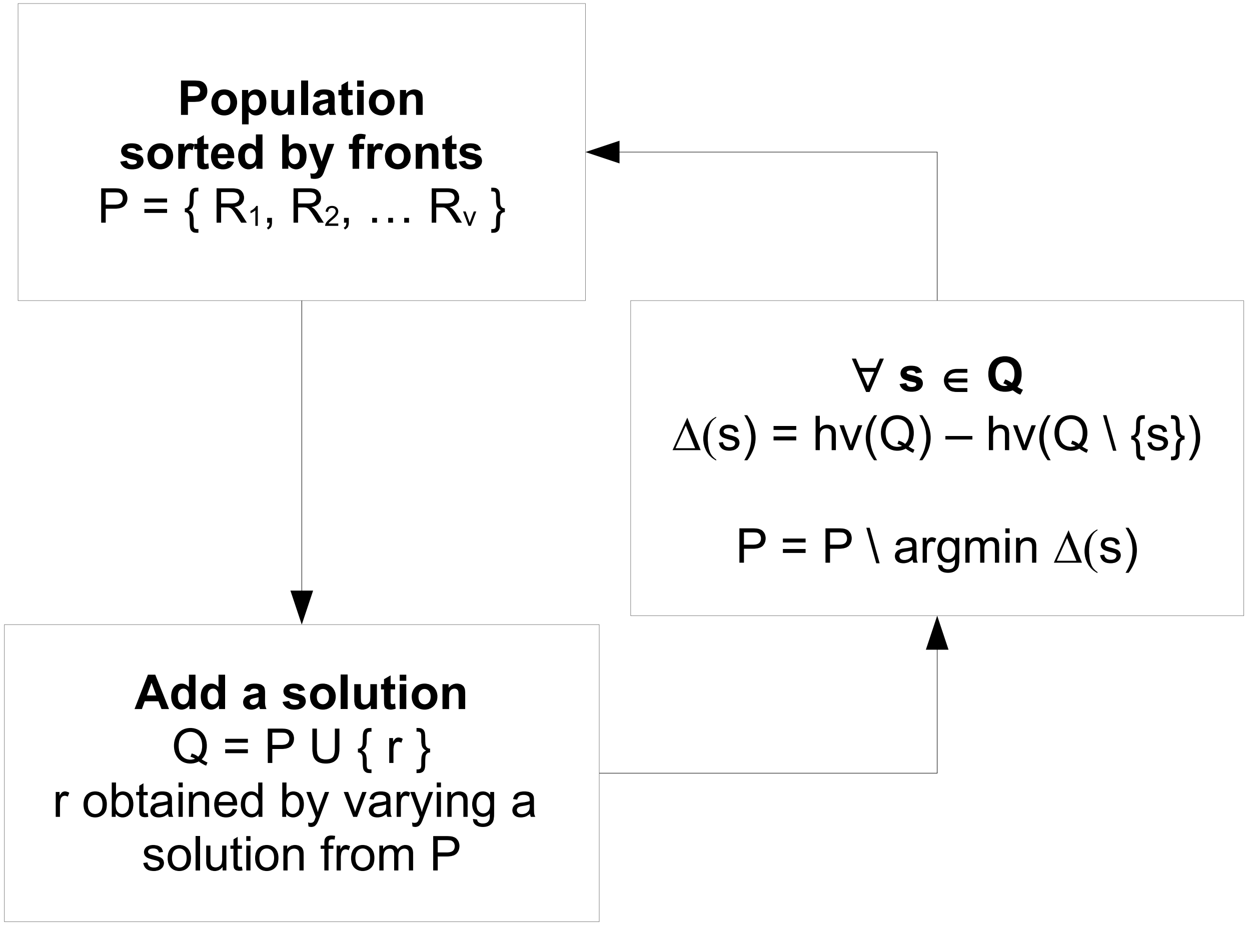


Rejected individuals which minimize crowding distance

EMOA

SMS-EMOA: guided by metrics on resulting front quality

- Example : hypervolume value obtained if you accept or reject a solution
- Remove s_1 or s_2 ?

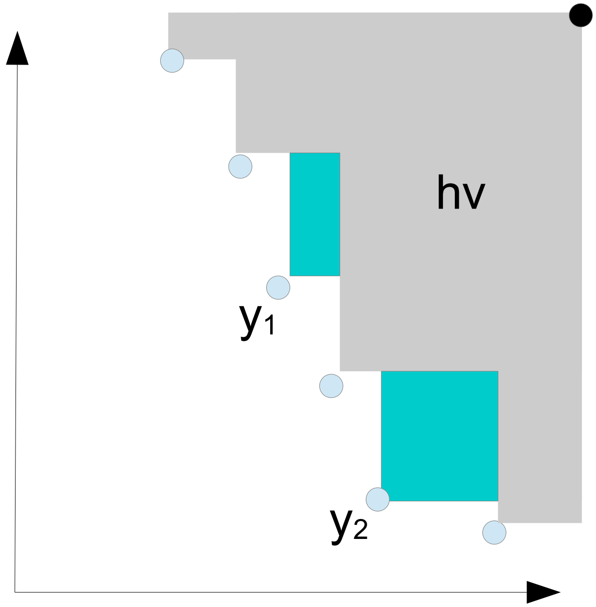


Population sorted by fronts
$P = \{ R_1, R_2, \dots R_v \}$

Add a solution
$Q = P \cup \{ r \}$
r obtained by varying a solution from P

$\forall \mathbf{s} \in \mathbf{Q}$
$\Delta(s) = hv(Q) - hv(Q \setminus \{s\})$

$P = P \setminus \text{argmin } \Delta(s)$

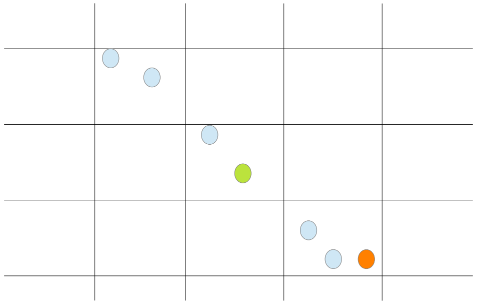


MOO

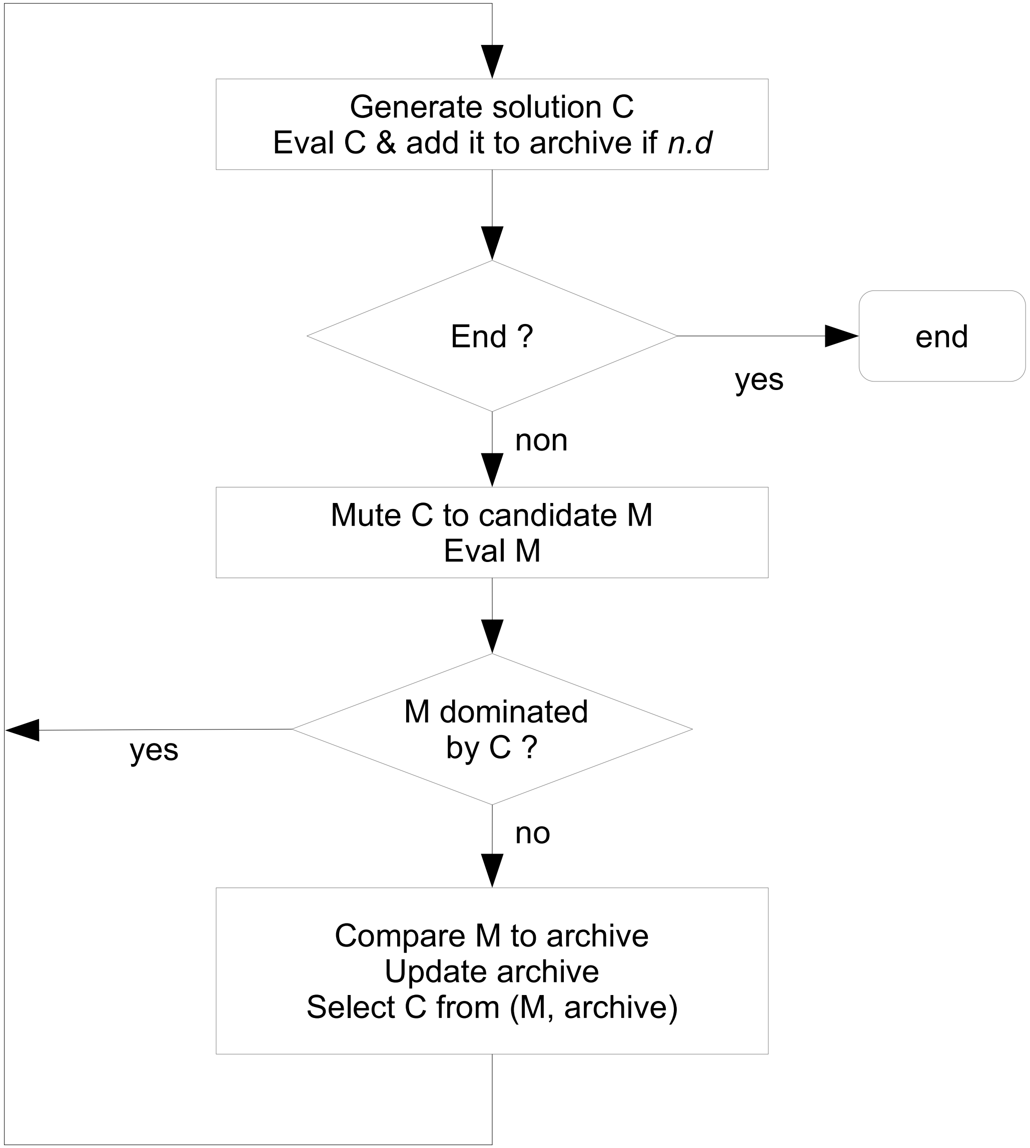
EMOA

Pareto archived evolution Strategy

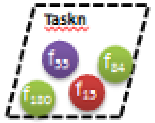
- Individual evolution by mutation
- Fixed size archive of ND individuals
  - New individuals checked against archive
- Grid based crowding



|A|=6



Applications



- Real time scheduling : preemptions vs laxity vs blocking resource – Parallel PAES

- Flash memory driver configuration : wearing vs latency vs mapping table size – Parallel PAES

- Weather routing : time to destination vs hardware and human stress PAES + heuristic
- Cloud federation storage : storage vs latency vs migration costs – Matheuristic – NSGA2 + CPLEX

Real-Time task scheduling

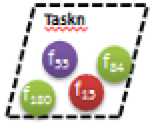
Critical applications with timing constraints

- Definition and characteristics [Stankovic 1988]:

  Processing inputs within a specified time
  Correct behavior: functional correctness + timing correctness
  Failures lead to severe damages
  Limited resources
  etc.

- Design and development challenges

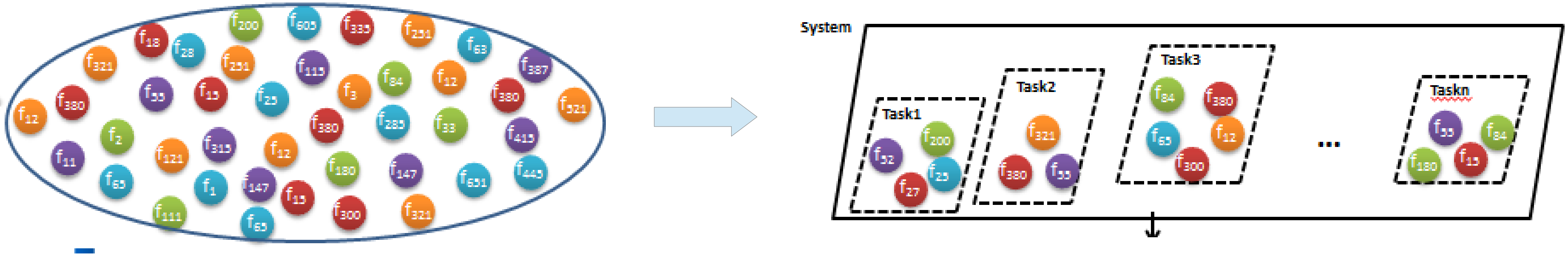
  Increasing **size**
  Increasing **complexity:** timing constraints, concurrency, resources sharing, etc.
  Important **non-functional requirements**: predictability, cost, response-time, resources consumption, etc.
  Multiple orthogonal **performance criteria**: improving one criterion may lead to the degradation of another



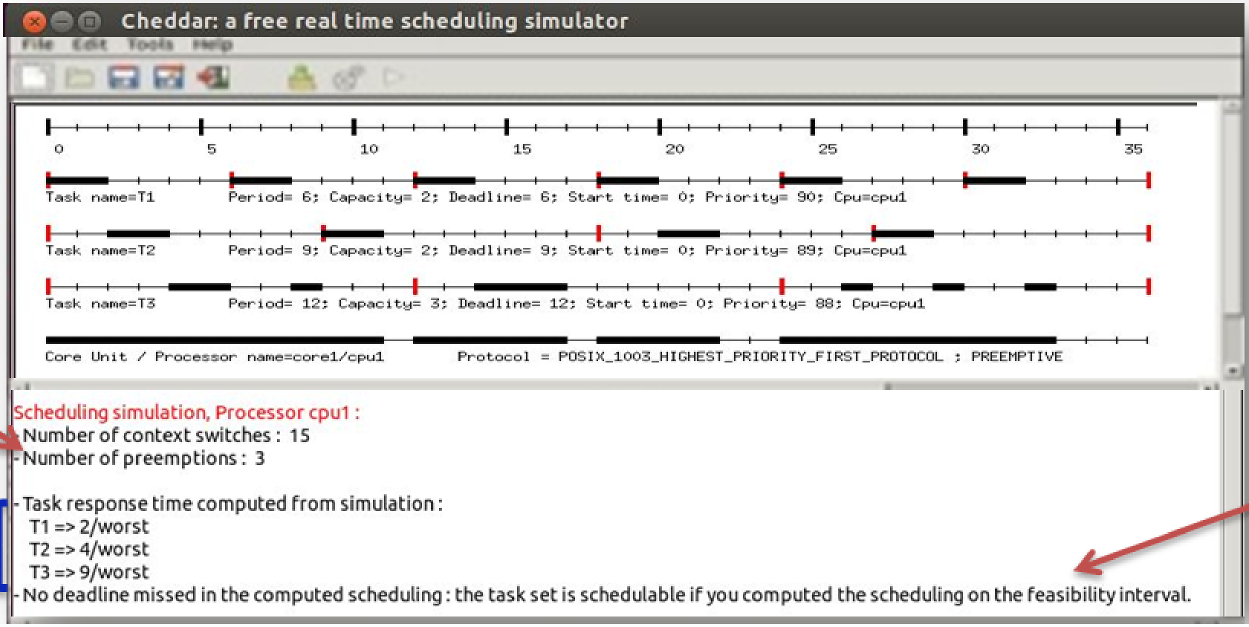
Real-Time task scheduling

Mapping functions into tasks

- One solution = One mapping



- Scheduling tasks and analysing results



```
Task name=T1          Period= 6; Capacity= 2; Deadline= 6; Start time= 0; Priority= 90; Cpu=cpu1
Task name=T2          Period= 9; Capacity= 2; Deadline= 9; Start time= 0; Priority= 89; Cpu=cpu1
Task name=T3          Period= 12; Capacity= 3; Deadline= 12; Start time= 0; Priority= 88; Cpu=cpu1
Core Unit / Processor name=core1/cpu1        Protocol = POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL ; PREEMPTIVE
```

```
Scheduling simulation, Processor cpu1 :
- Number of context switches :  15
- Number of preemptions :  3

- Task response time computed from simulation :
  T1 => 2/worst
  T2 => 4/worst
  T3 => 9/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```

Number of preemptions

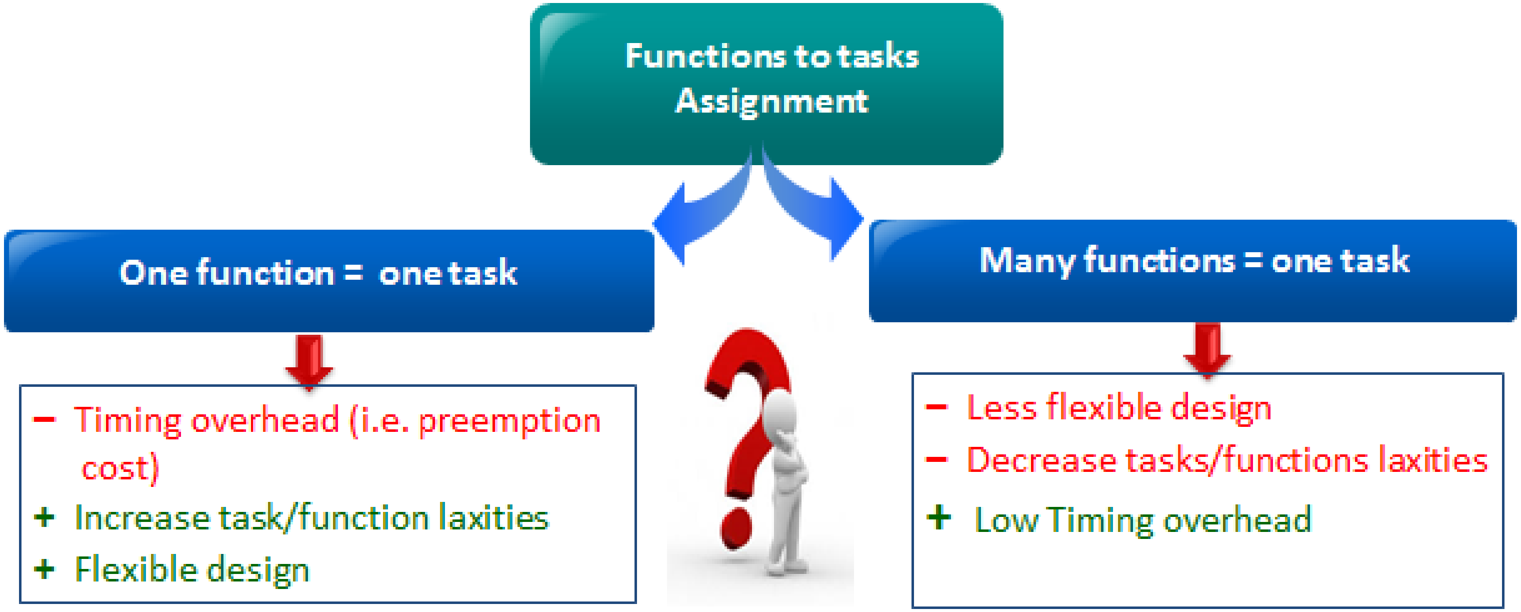
WCRT of tasks

Schedulability (yes/no)

Real-Time task scheduling

Trade offs

- Laxity : capability to schedule additional functions without violating timing constraints
- Preemptions : # of interruptions of tasks by higher priority ones



Functions to tasks Assignment

One function = one task

- Timing overhead (i.e. preemption cost)
+ Increase task/function laxities
+ Flexible design

Many functions = one task

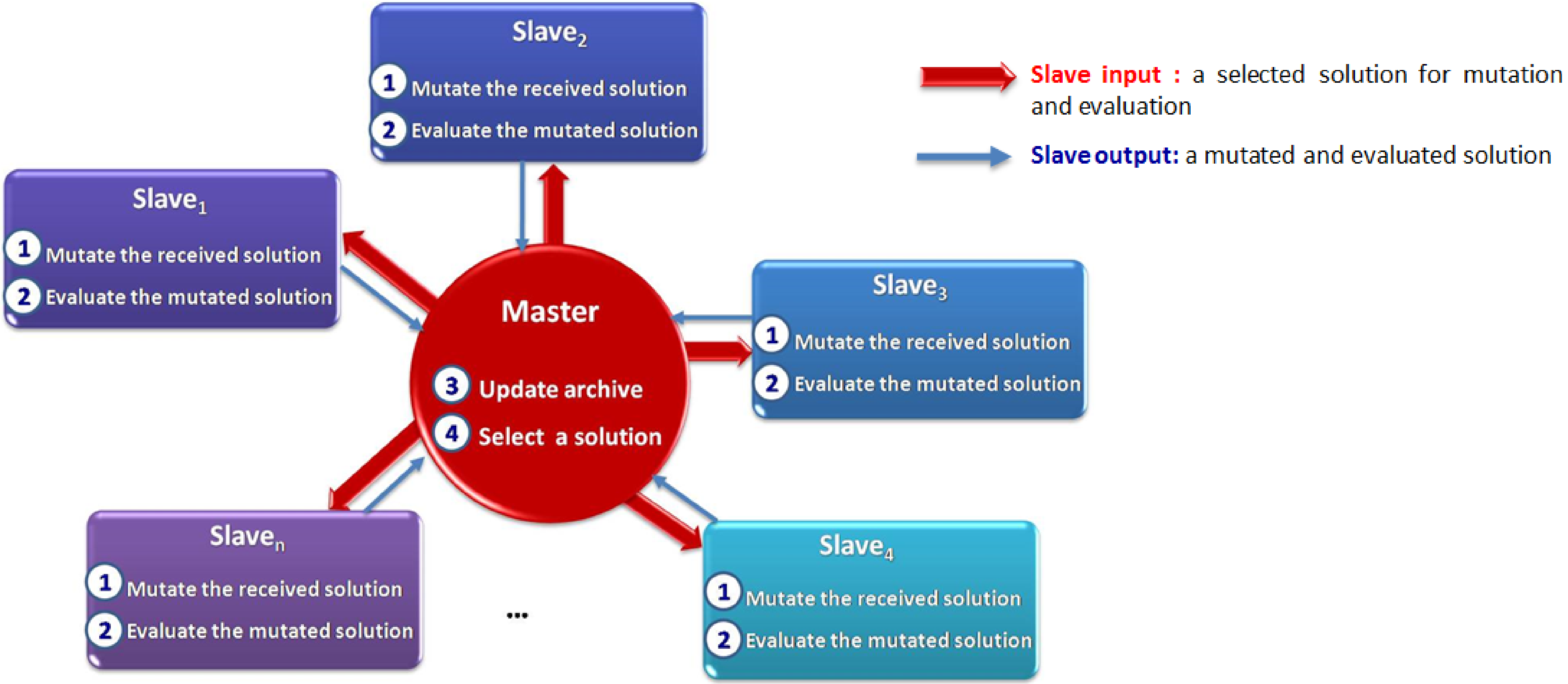
- Less flexible design
- Decrease tasks/functions laxities
+ Low Timing overhead

[Architecture Exploration of Real-time Systems Based on Multi-Objective Optimization, Bouaziz et al, ICECCS 2015]

Real-Time task scheduling

Simulation is time consuming

- Parallel asynchronous PAES with modified selection



Slave input : a selected solution for mutation and evaluation

Slave output: a mutated and evaluated solution

Master

3 Update archive

4 Select a solution

Slave$_1$

1 Mutate the received solution

2 Evaluate the mutated solution

Slave$_2$

1 Mutate the received solution

2 Evaluate the mutated solution

Slave$_3$

1 Mutate the received solution

2 Evaluate the mutated solution

Slave$_4$

1 Mutate the received solution

2 Evaluate the mutated solution

Slave$_n$

1 Mutate the received solution

2 Evaluate the mutated solution

...

[Efficient Parallel Multi-objective Optimization for Real-Time Systems Software Design Exploration, Bouaziz et al, Rapid System Prototyping Symposium 2016]

Real-Time task scheduling

Ongoing : more rich models

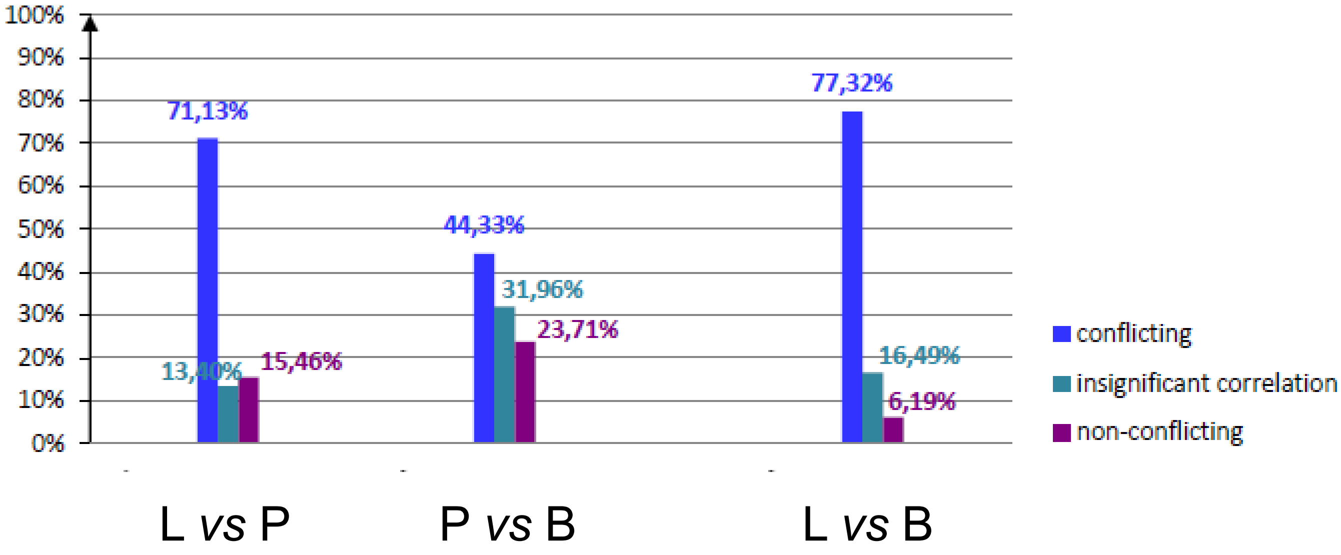
- Shared ressources
- Multi-processor scheduling (partionned scheduling)

More possible objective functions

#preemptions, #context switches, Σ laxity, Σ blocking-time, #shared ressources, #tasks, Σ response-times, …

- Correlated ?

  correlation for
  3 objectives,
  100 testcases
  L : Σ (laxities)
  P : #preemptions
  B : Σ (blocking times)



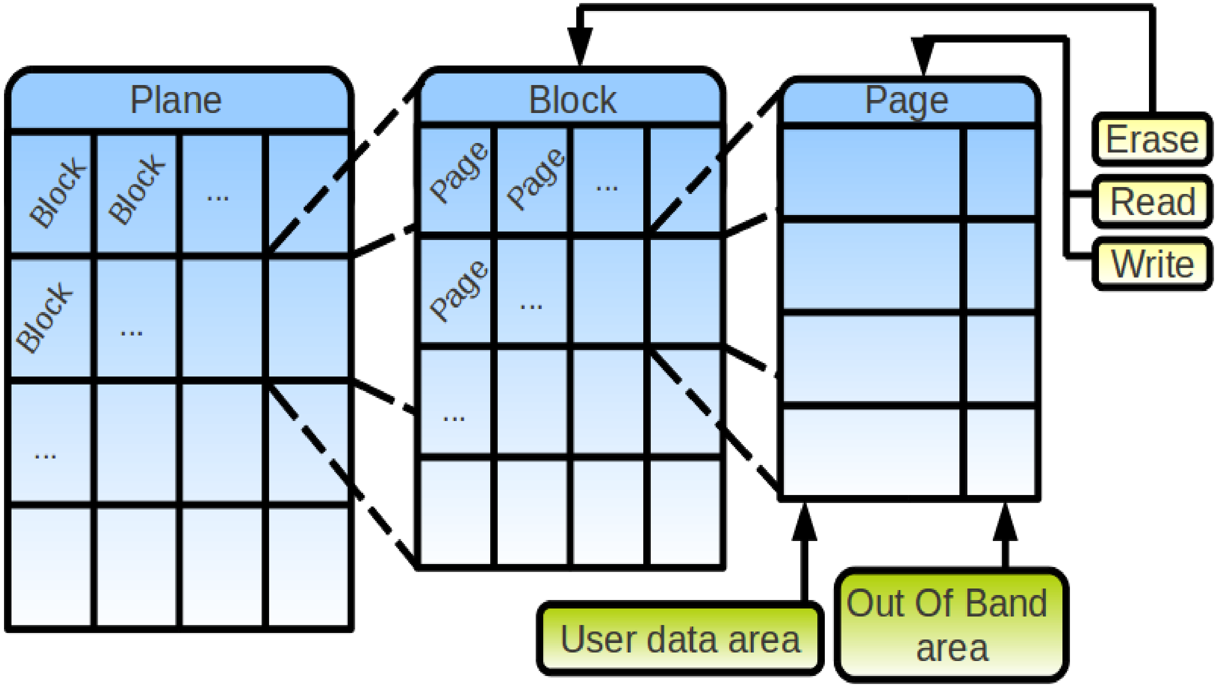
- Many objectives: reduce dynamically #objectives

[Multi-Objective Design Exploration Approach for Ravenscar Real-time Systems, Bouaziz et al, JRTS 2018]

Flash Memory Driver Configuration

Operations

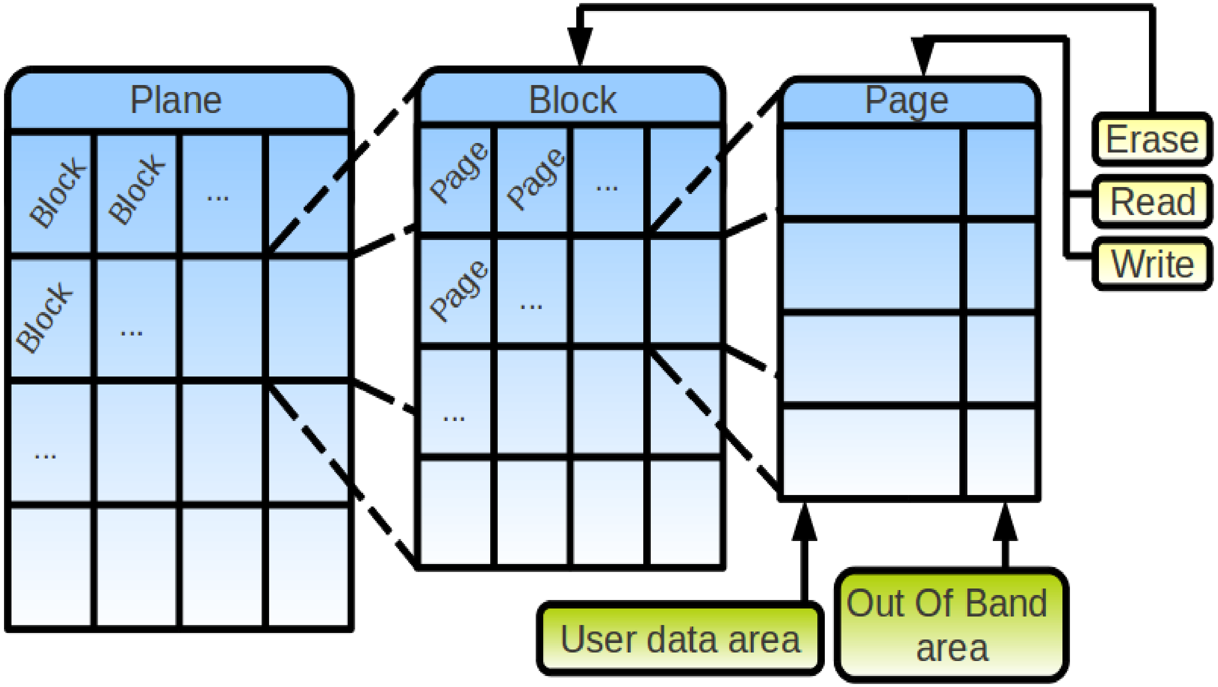
- Operations E/R/W
- E on blocks (wear)
- R/W on pages
- E before W



Good Flash Memory Configuration

Operations

- Operations E/R/W
- E on blocks (wear)
- R/W on pages
- E before W



@ mapping

- By page (PM) → RAM cost
- By block (BM) → #E cost
- Hybrid → %PM

BM vs PM choice for W

- depends on #pages to be written

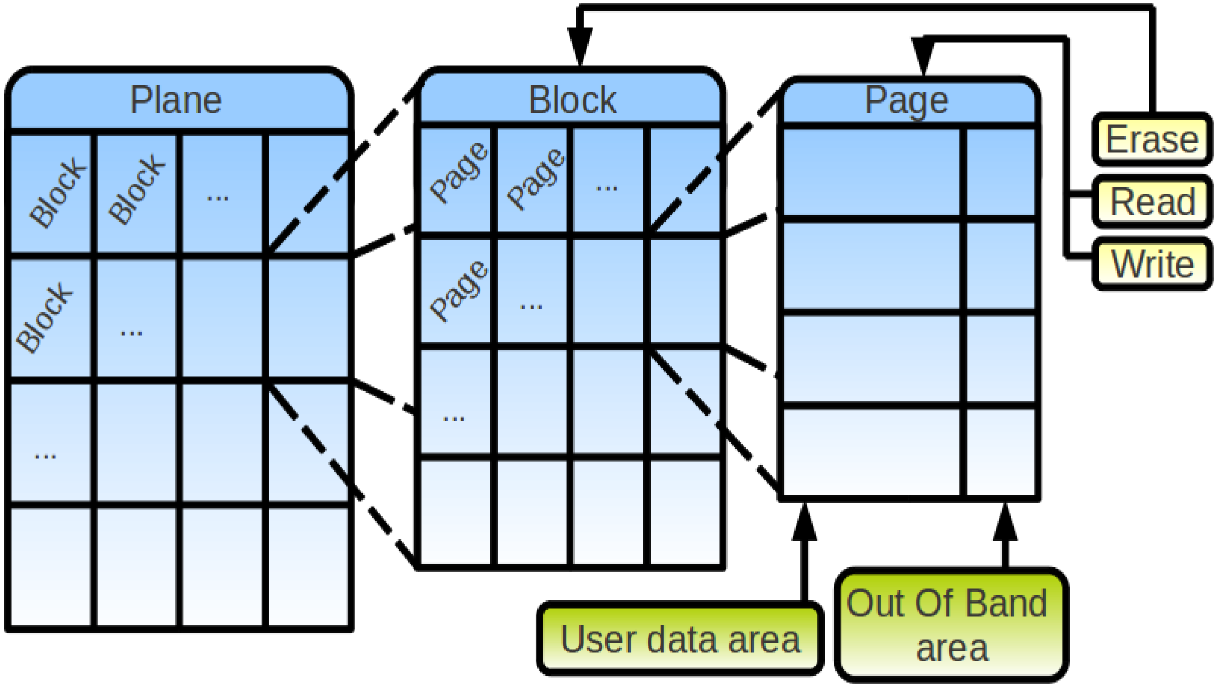
→ PM < threshold < BM

Good Flash Memory Configuration

Operations

- Operations E/R/W
- E on blocks (wear)
- R/W on pages
- E before W

R/W response time



@ mapping

- By block (BM) → #E cost
- By page (PM) → RAM cost
- Hybrid → %PM %PM

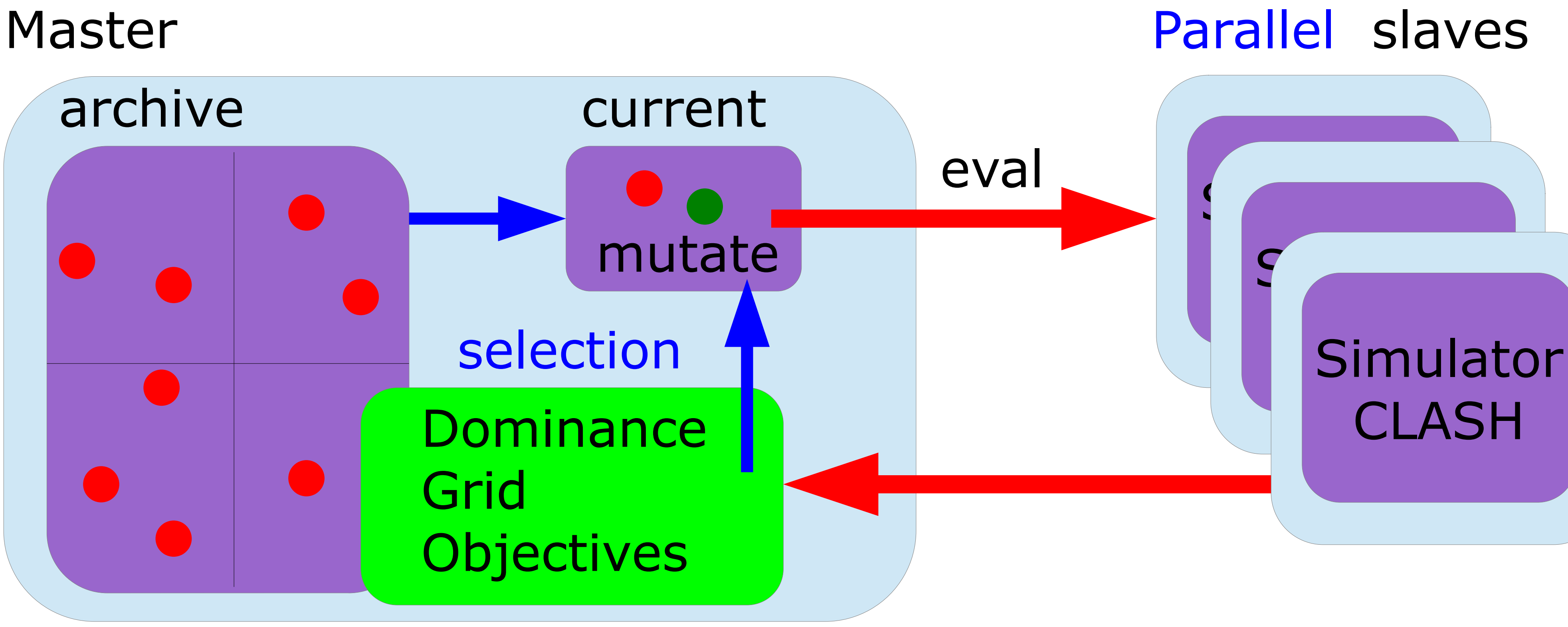
BM vs PM choice for W

- depends on #pages to be written

→ PM < threshold < BM

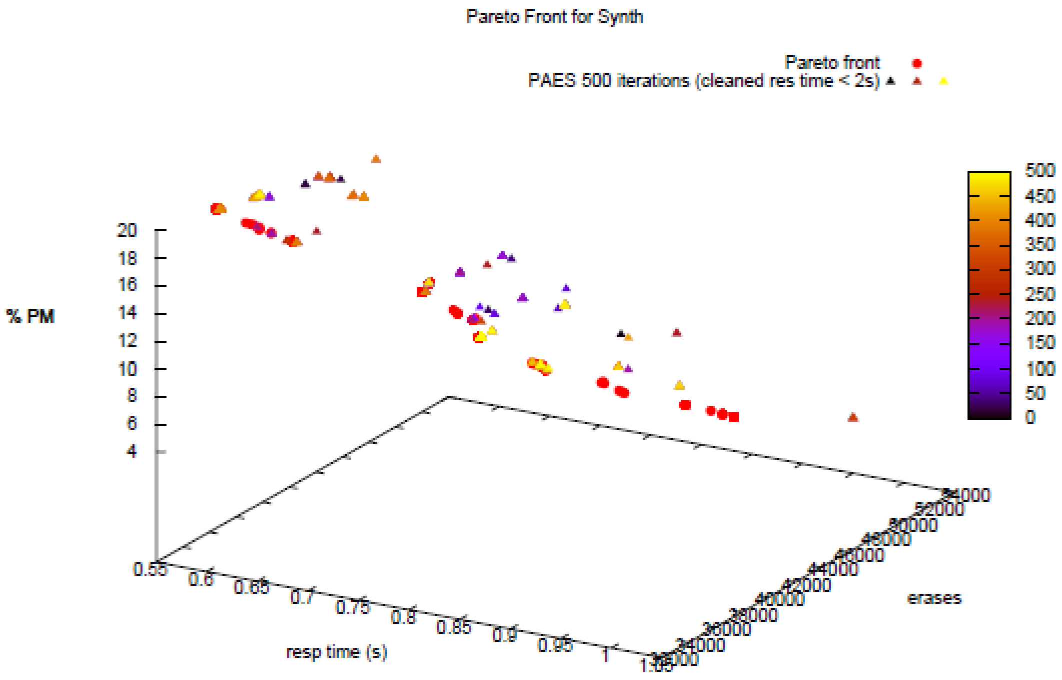
Flash Memory Driver Configuration

Parallelized Pareto Archived Evolution Strategy



[MaCACH: An adaptive cache-aware hybrid FTL mapping scheme using feedback control for efficient page-mapped space management, Boukhobza et al, *Journal of Systems Architecture,* 2015]

Flash Memory Configuration



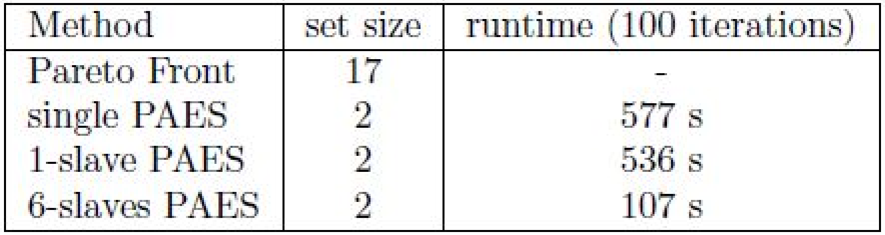
Fronts

- convergency
- dispersion

  4 – 6 % PM
  30 – 54K erases
  0.55 – 1.05ms RT
- Design maker problem

Parallel version

- Same results, linear speedups



Method	set size	runtime (100 iterations)
Pareto Front	17	-
single PAES	2	577 s
1-slave PAES	2	536 s
6-slaves PAES	2	107 s

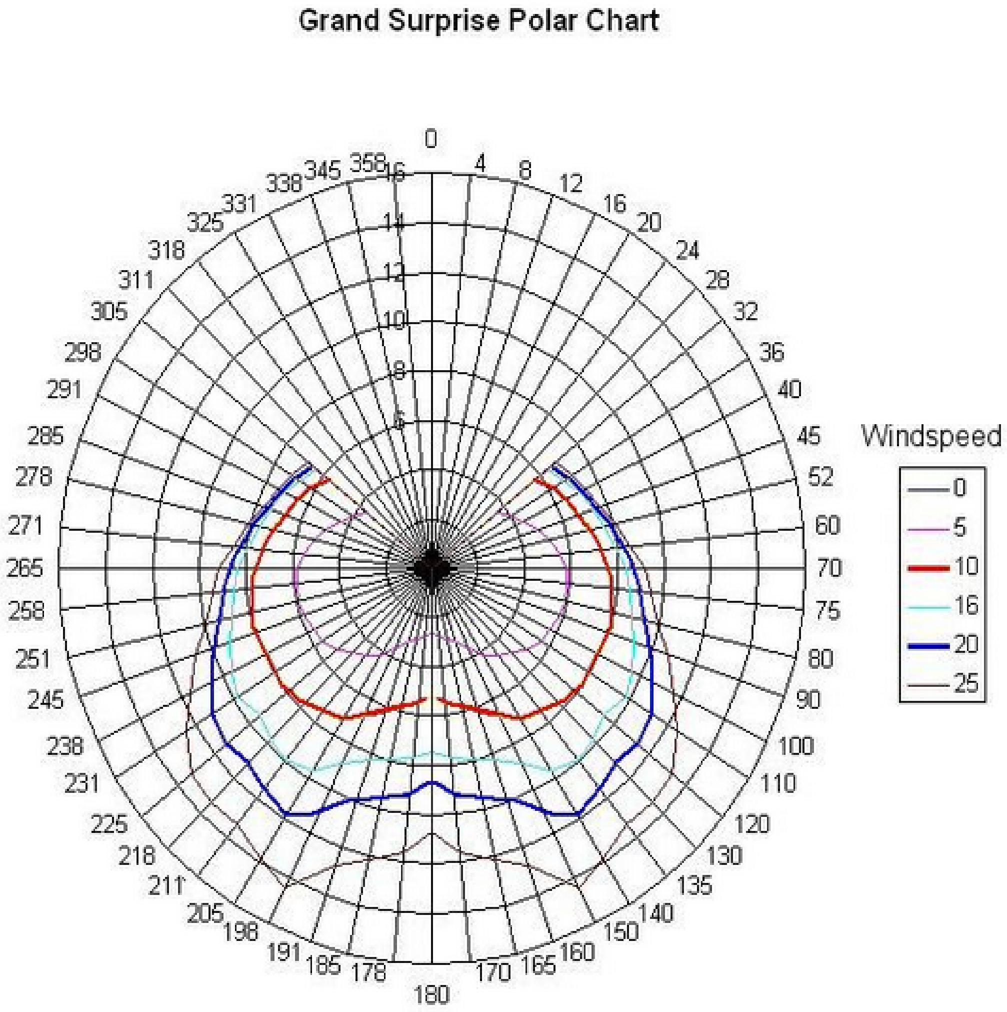
Yacht weather routing

Find the *best* route for a yacht

- Boat speed depends on
  - TWA : true wind angle
  - TWS : true wind speed

- Weather

  wind (and waves …)
  characteristics over the time



Grand Surprise Polar Chart

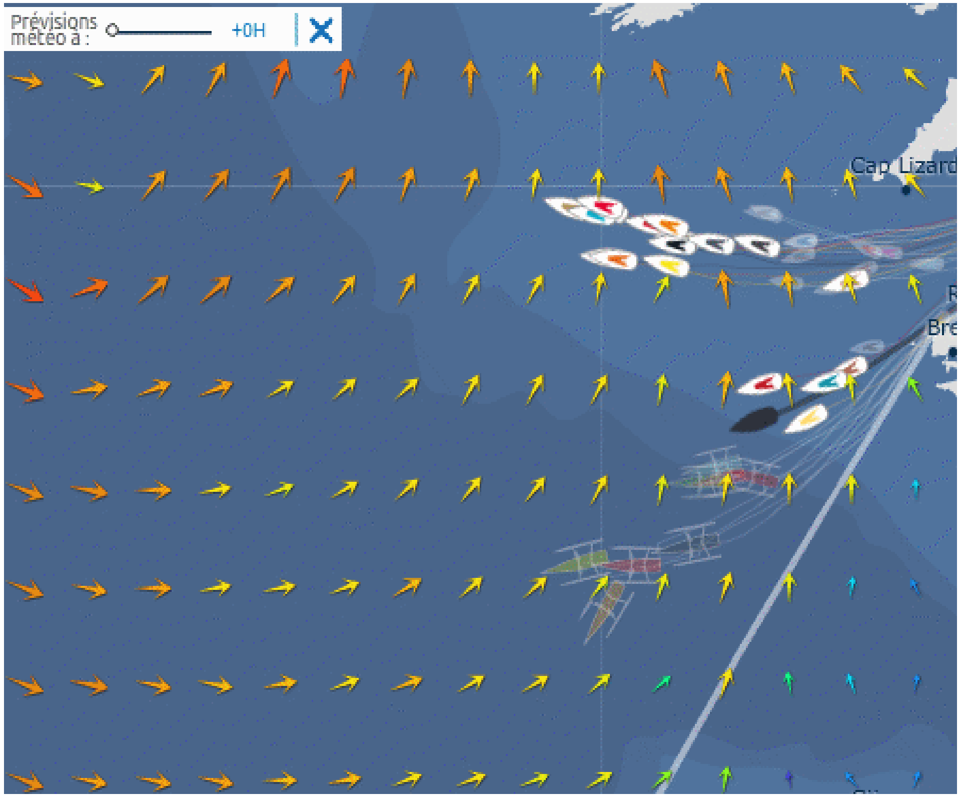
Yacht weather routing

Find the *best* route for a yacht

- Boat speed depends on
  - AWA : apparent wind angle
  - Wind speed

- Weather

  wind (and waves …)
  characteristics over the time

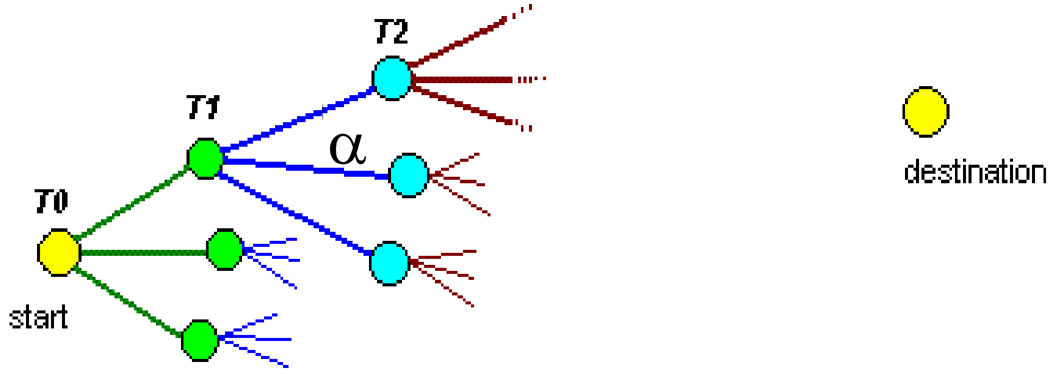


Basic weather routing: isochrones

Algorithm

- Time discretization
- Starting at point (x,y, t), compute all points reached at time t + 👎t
- Following direction (angle step 👎a)

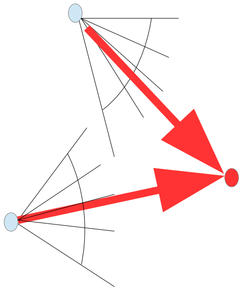
```
✂ boatDir = k.👎a
    (windDir, winSpeed) = weather(x,y,t)
    boatSpeed = polar(windSpeed, windDir, boatDir)
    (x',y', t'=t + 👎t) = addVector(xy, boatDir, boatSpeed*👎t)
```



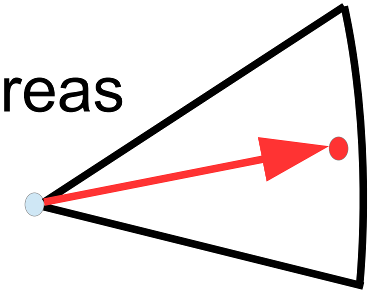
Basic Weather routing: isochrones

Cuts in the search tree → heuristics

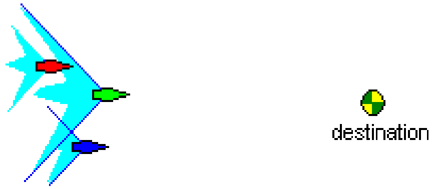
- Possible angles

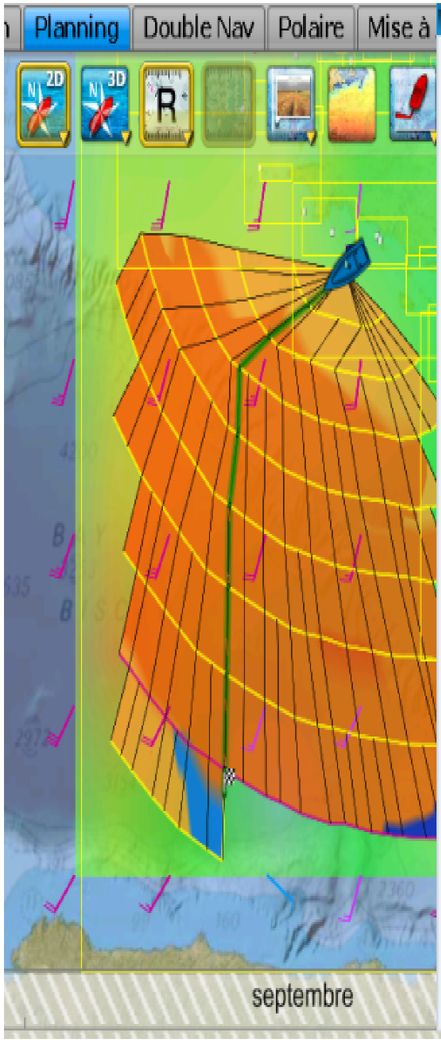


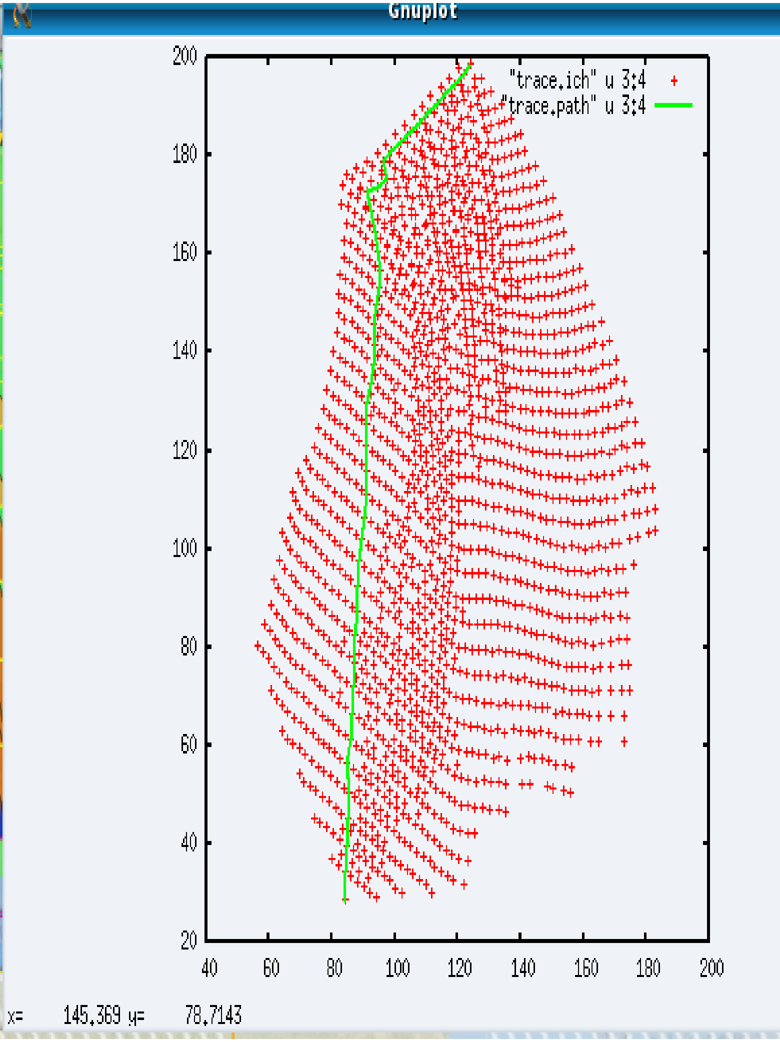
- Possible areas



- Lateness :
  in the wake







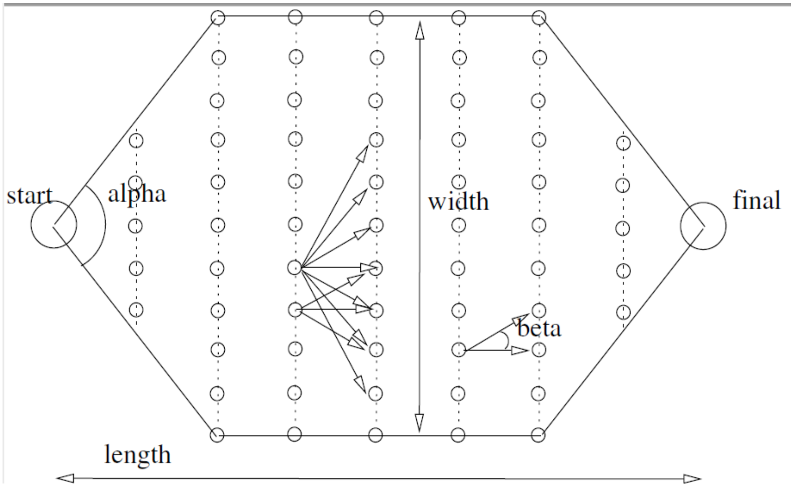
x= 145,369 y= 78,7143

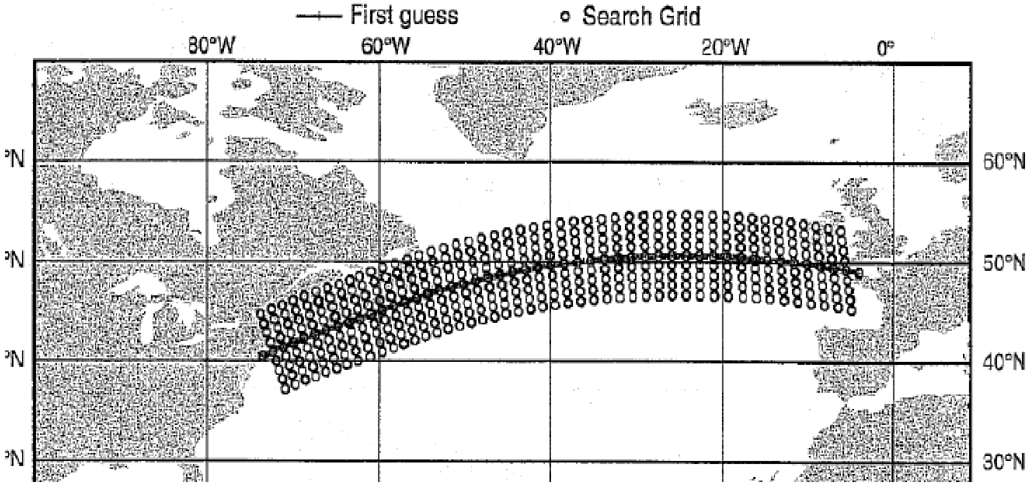
Basic Weather routing: mesh

Grid model

- Space discretization
- Dynamic Programming

  → shortest path





MOO Yacht Weather routing

Classical boat routing objectives

- Main **: Time to destination** *min* f_1(route, polar, weather)
- Fuel consomption
- Risk (strong waves, icebergs)

Yacht routing

- Power management (windweel power plant)
- Boat wearing (e.g. Distance)
- **Maneuvers effort (jibes, tacks, sail changes, …)**
- **Human stress (difficulties related to weather)**

min f_2(route, 👎wind, strongwind, lightwind, jibes, tacks, …)

MOO Yacht Weather routing

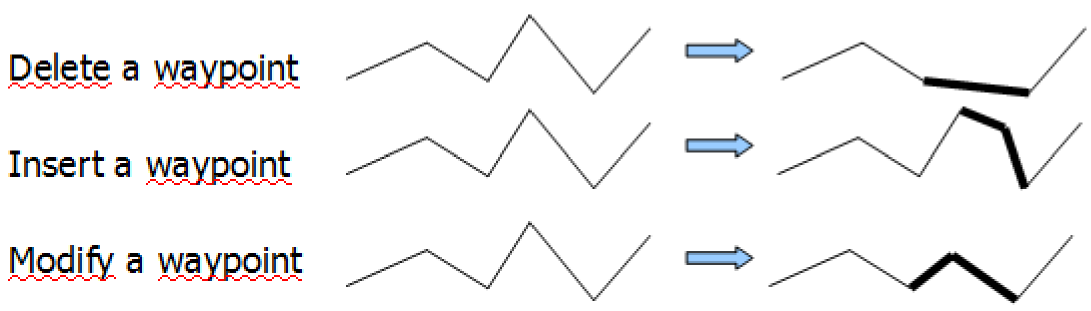
SOO (time) weather routing

- Basics of MOO algorithm
- MaxSea vs Isochrones vs Grid routing

route	time		
	MaxSea	Isochrone-based routing	Grid-based routing
#1	1d03h00	0d21h44 ★	0d21h58
#2	0d20h52	0d18h36	0d18h35 ★
#3	1d14h40	1d13h10 ★	1d13h21
#4	1d01h52	1d00h43	1d00h36 ★
#5	1d06h45	1d07h22	1d06h35 ★
#6	0d18h53	0d18h28	0d18h19 ★

MOO (time & stress) weather routing

- Multiple EMOAs
- Way-points based chromosome



- 6 testcases
- Kruskall-Wallis non parametric test

beats ↱	PAES	IBEA$_\epsilon$	NSGA2	SPEA2
PAES	-	0.0000	0.0000	0.0000
IBEA$_\epsilon$	1.0000	-	0.2375	0.2588
NSGA2	1.0000	0.7625	-	0.4701
SPEA2★	1.0000	0.7412	0.5299	-

SOO IP problem solving

Mathematical formulation of an optimization problem (Linear or Integer or Binary Programming)

- A carpenter can make at most 6 seats and 3 tables by day (8 hours of work)
  - He sells a table $90 (working 1h15)
  - A seat, $50 (working 45mn)
- How to maximize his benefit ?

$$\begin{cases} 90t & + & 50c & = & f(s) \\ 75t & + & 45c & \leq & 480 \\ 0 & \leq & \mathrm{t} & \leq & 3 \\ 0 & \leq & \mathrm{c} & \leq & 6 \end{cases}$$

CPLEX
solving
tool

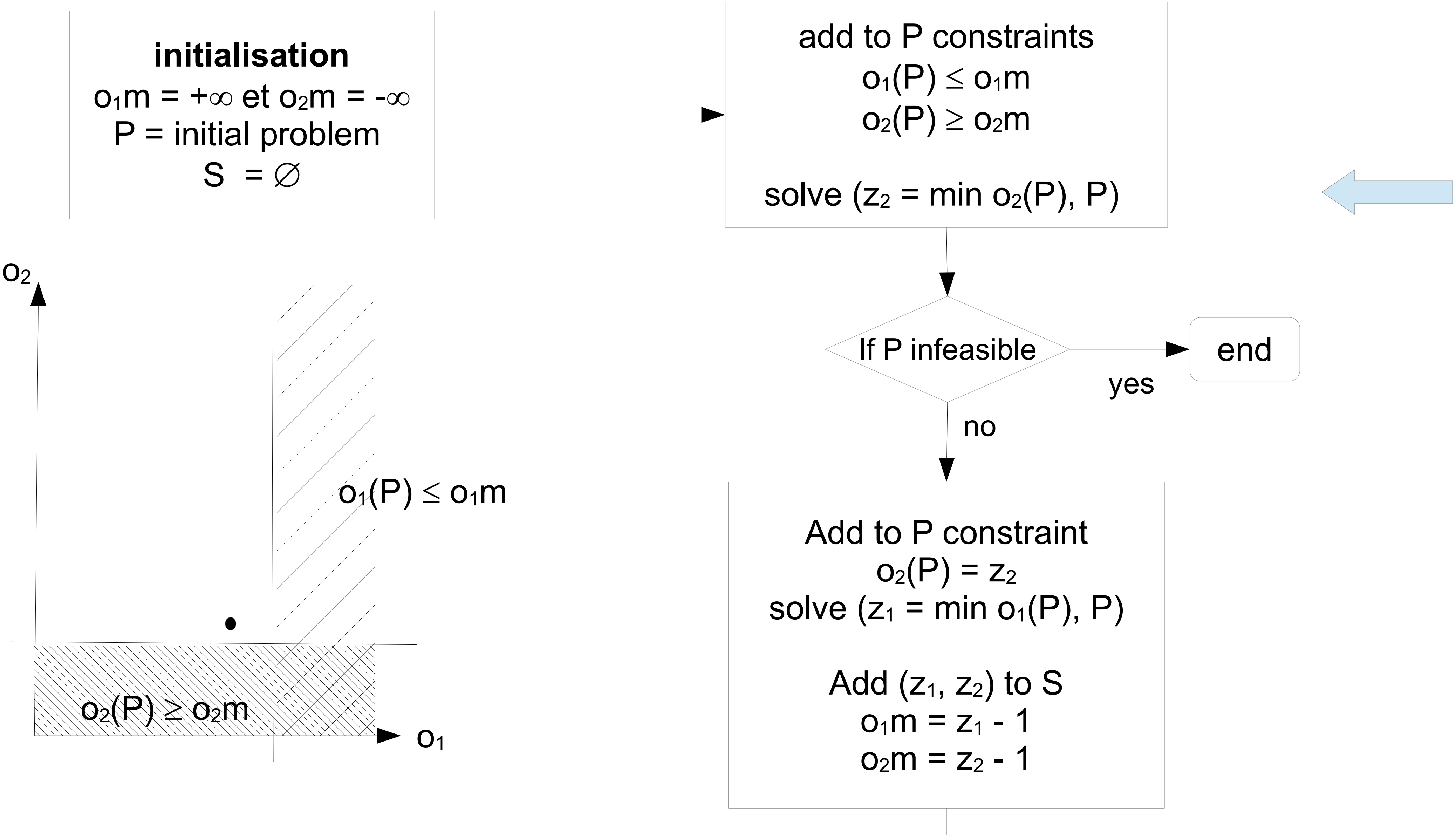
- *Linear pro*gramming : simplex method with O(2n) complexity – Branch&Bound for IP/BP resolution

MOO

MOO IP problem solving

Resolution with an ε-constraints technique

(for 2 objectives min)



initialisation
$o_1m = +\infty$ et $o_2m = -\infty$
P = initial problem
$S = \varnothing$

add to P constraints
$o_1(P) \leq o_1m$
$o_2(P) \geq o_2m$

solve (z_2 = min $o_2(P)$, P)

If P infeasible

yes

end

no

Add to P constraint
$o_2(P) = z_2$
solve (z_1 = min $o_1(P)$, P)

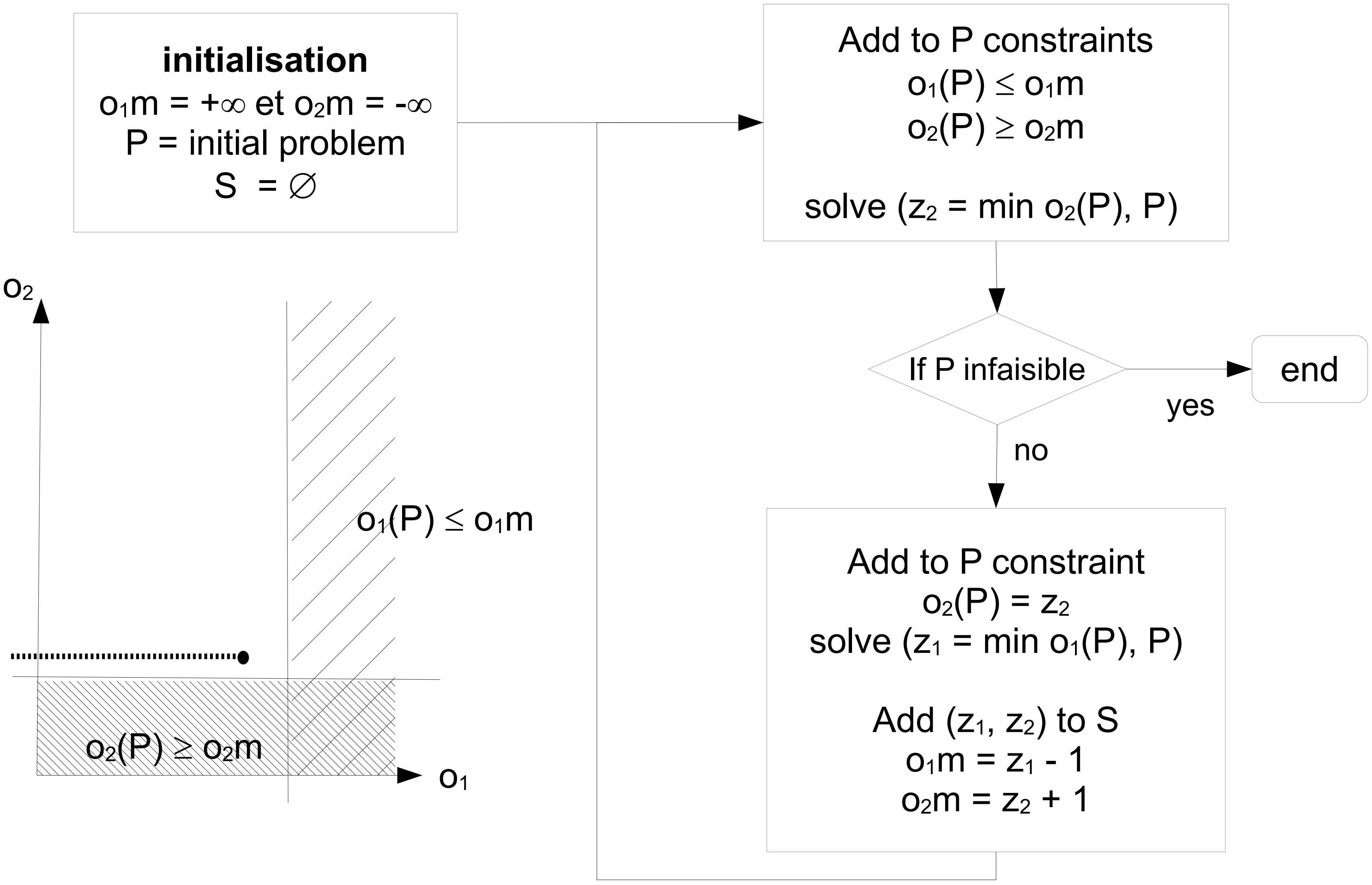
Add (z_1, z_2) to S
$o_1m = z_1 - 1$
$o_2m = z_2 - 1$

MOO

MOO IP problem solving

Resolution with an ε-constraints technique

(for 2 objectives min)



initialisation
$o_1m = +\infty$ et $o_2m = -\infty$
P = initial problem
$S = \varnothing$

Add to P constraints
$o_1(P) \leq o_1m$
$o_2(P) \geq o_2m$

solve ($z_2 = \min o_2(P)$, P)

If P infaisible

yes

end

no

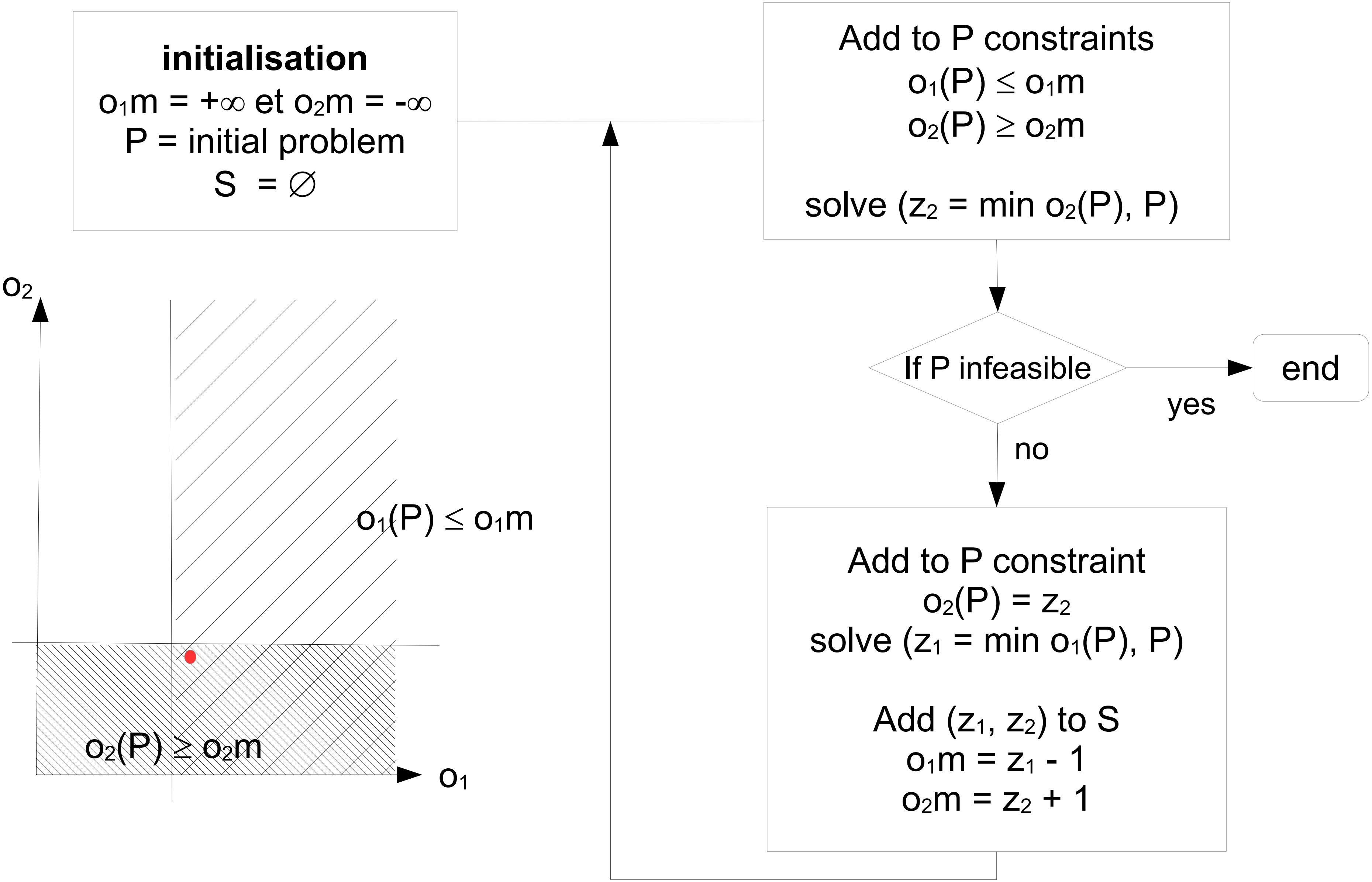
Add to P constraint
$o_2(P) = z_2$
solve ($z_1 = \min o_1(P)$, P)

Add (z_1, z_2) to S
$o_1m = z_1 - 1$
$o_2m = z_2 + 1$

MOO IP problem solving

Resolution with an ε-constraints technique

(for 2 objectives min)



initialisation
$o_1m = +\infty$ et $o_2m = -\infty$
P = initial problem
$S = \varnothing$

Add to P constraints
$o_1(P) \leq o_1m$
$o_2(P) \geq o_2m$

solve ($z_2 = \min o_2(P)$, P)

If P infeasible

yes

end

no

Add to P constraint
$o_2(P) = z_2$
solve ($z_1 = \min o_1(P)$, P)

Add (z_1, z_2) to S
$o_1m = z_1 - 1$
$o_2m = z_2 + 1$

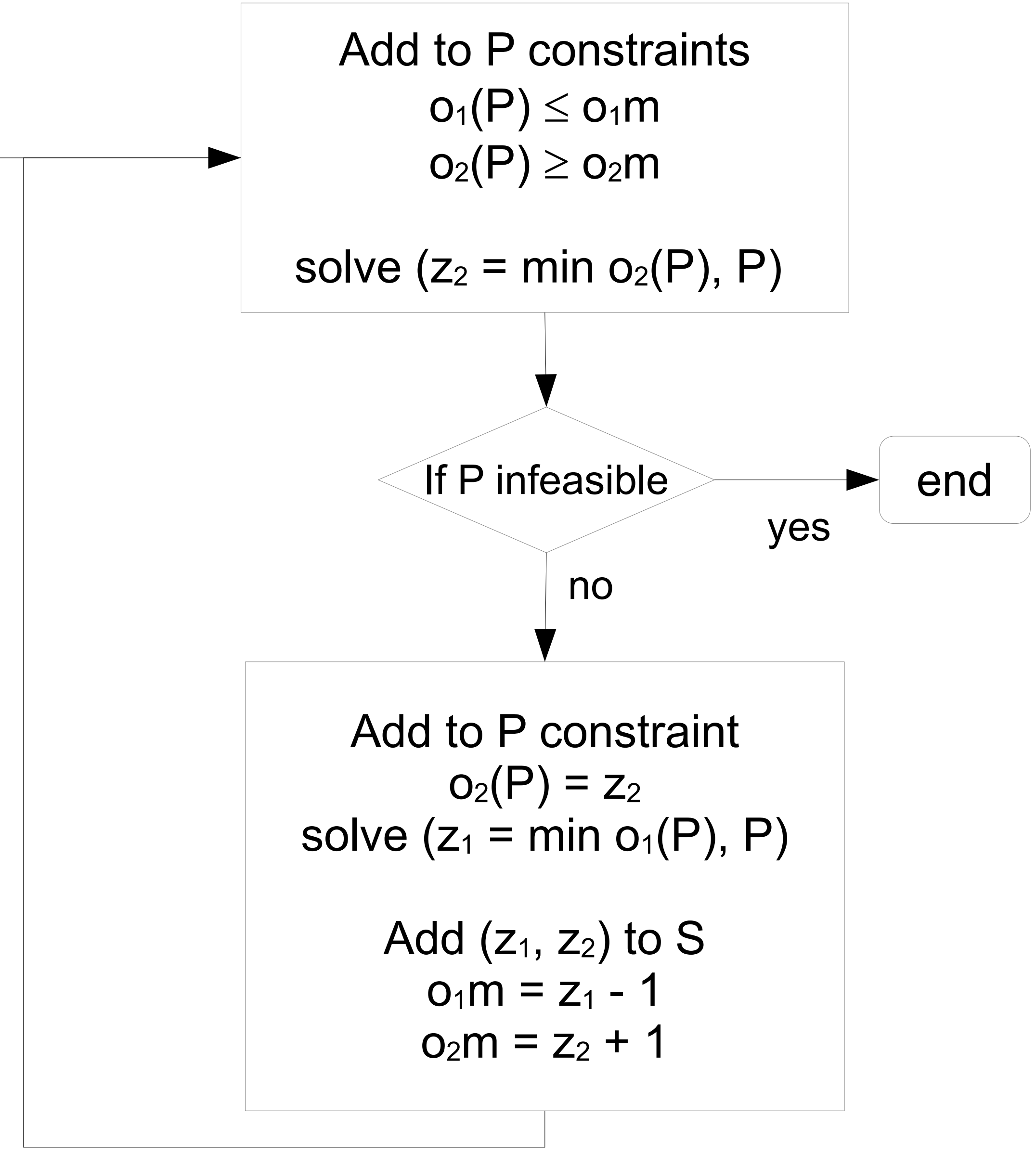
MOO

MOO IP problem solving

Resolution with an ε-constraints technique

(for 2 objectives min)

initialisation
$o_1m = +\infty$ et $o_2m = -\infty$
P = initial problem
S $= \varnothing$



Add to P constraints
$o_1(P) \leq o_1m$
$o_2(P) \geq o_2m$

solve ($z_2 = \min o_2(P)$, P)

If P infeasible

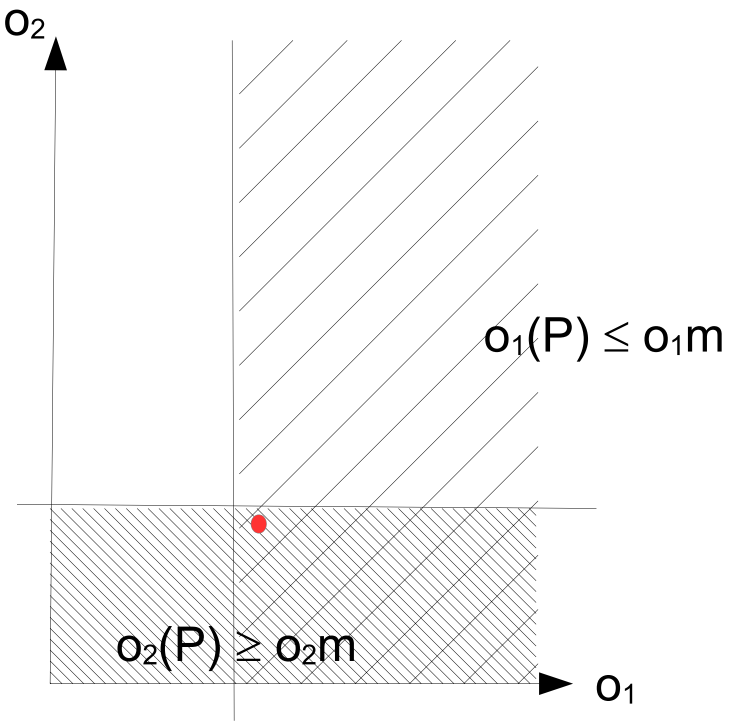
yes

end

no

Add to P constraint
$o_2(P) = z_2$
solve ($z_1 = \min o_1(P)$, P)

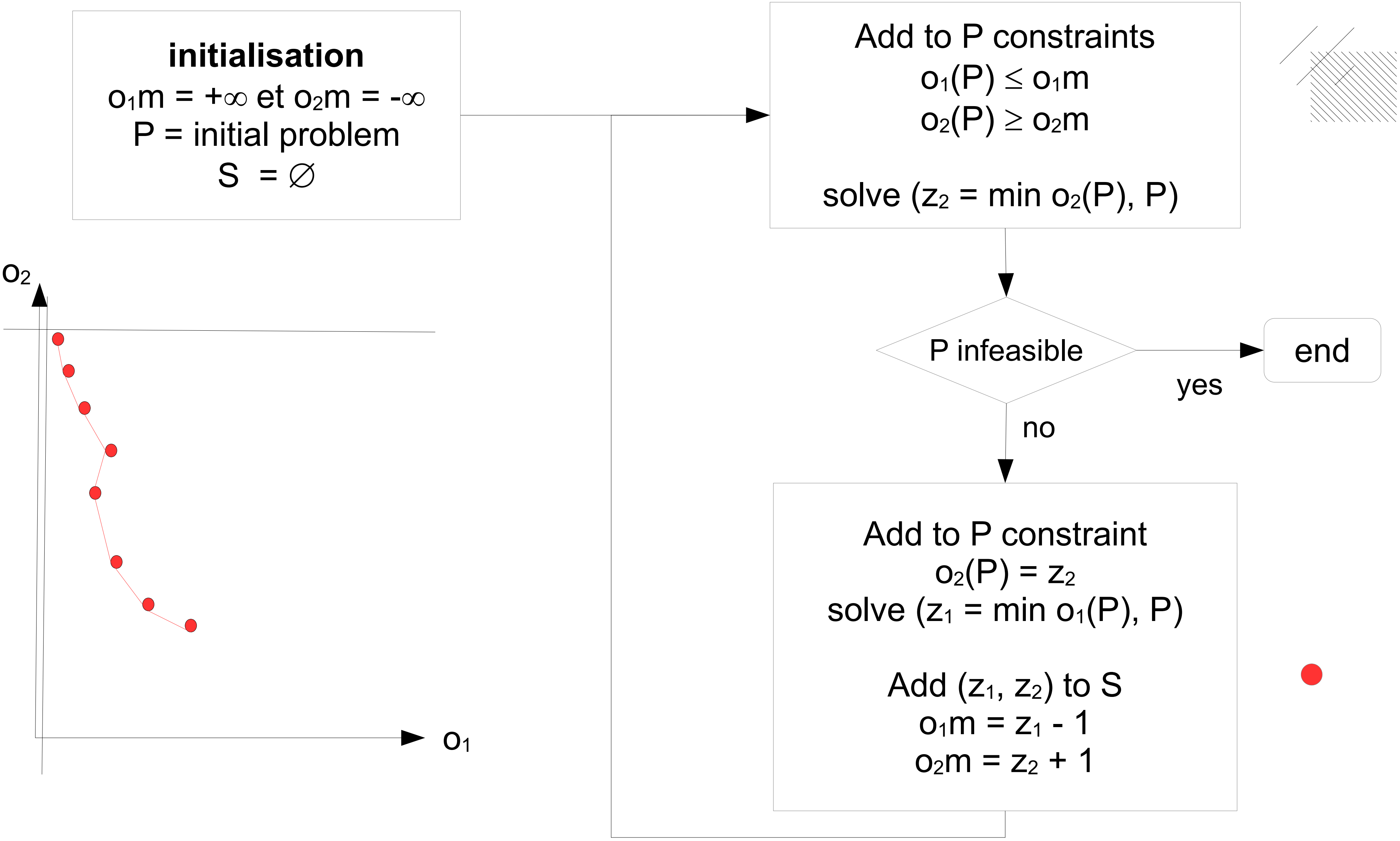
Add (z_1, z_2) to S
$o_1m = z_1 - 1$
$o_2m = z_2 + 1$



MOO IP problem solving

Resolution with an ε-constraints technique

(for 2 objectives min)



initialisation
$o_1m = +\infty$ et $o_2m = -\infty$
P = initial problem
S = ∅

Add to P constraints
$o_1(P) \leq o_1m$
$o_2(P) \geq o_2m$

solve ($z_2 = \min o_2(P)$, P)

P infeasible

yes

end

no

Add to P constraint
$o_2(P) = z_2$
solve ($z_1 = \min o_1(P)$, P)

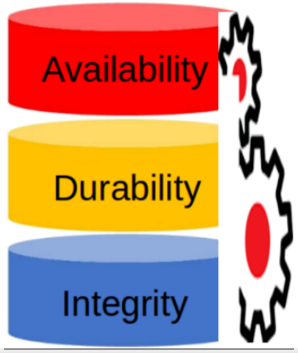
Add (z_1, z_2) to S
$o_1m = z_1 - 1$
$o_2m = z_2 + 1$

Matheuristics
Cloud storage

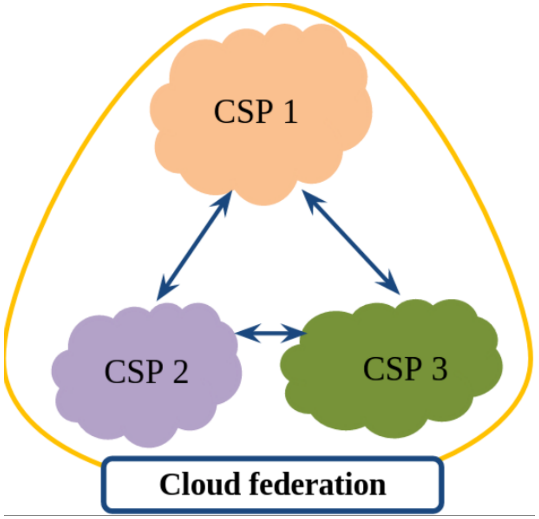
Mixing a MOIP with a MOEA

(for 2 objectives min)

- Solving all IP programs of e-constraint too much time consuming
- Good solutions with weighted sum (supported solutions) → input to MOEA
- Good solutions with MOEA → warmstart technique for MOIP



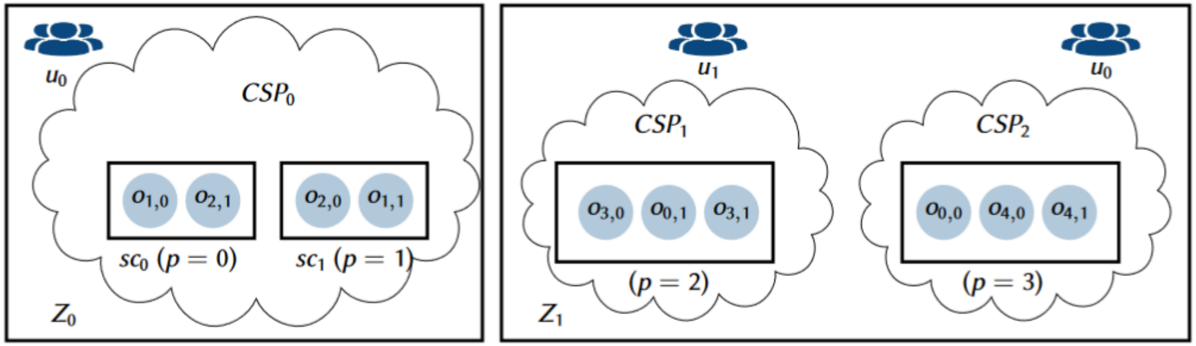
- Example of data storage for a federation of clouds



Cloud storage

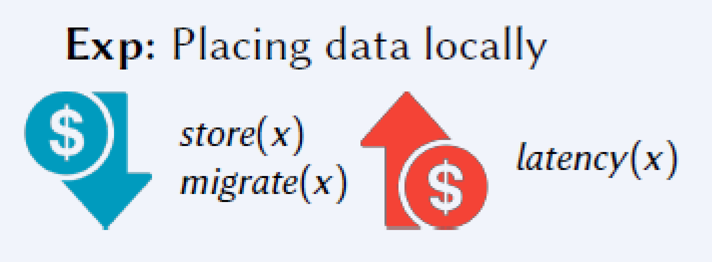
Placing clients' objets (3 copies each) on storage devices

- CSPs
  Optimize cost
  for CSP0



- Data inputs
  - Local storage (HDD, SSD, etc.), with capacity, wearing, perf, cost …
  - Remote storage (HDD, SSD, etc.), capacity, rental cost, migration cost …
  - Clients objects replicas, size, I/O workload, SLA, location

- Objective functions
  - Storage cost
  - Latency cost
  - Migration cost



- Constraints
  - Limited capacities
  - Limited IOPS
  - Clients' SLA

Cloud storage

Placing clients' objets (3 copies each) on storage devices

$$\min \begin{bmatrix} store(x) \\ migrate(x) \\ latecy(x) \end{bmatrix}$$

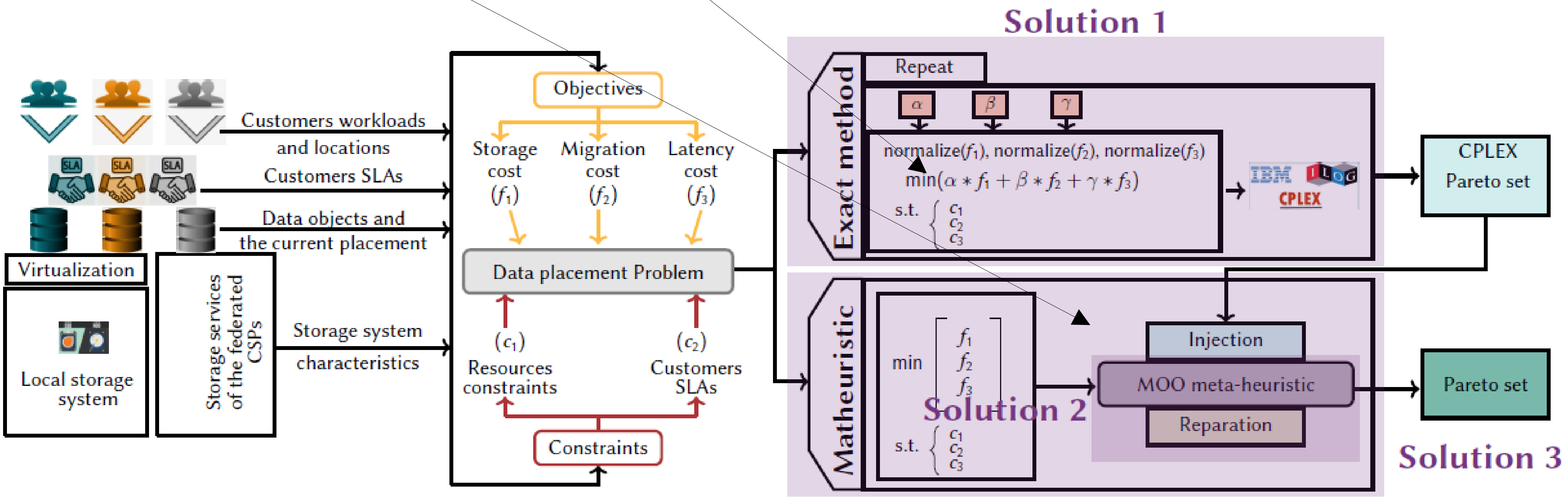
$$S.T. \quad \sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leqslant csc_j \qquad \forall j < J$$

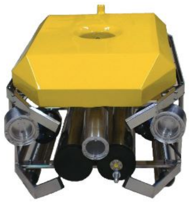
$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}} \leqslant css_d \qquad \forall d \geqslant J$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leqslant 1, \qquad \forall j < J$$

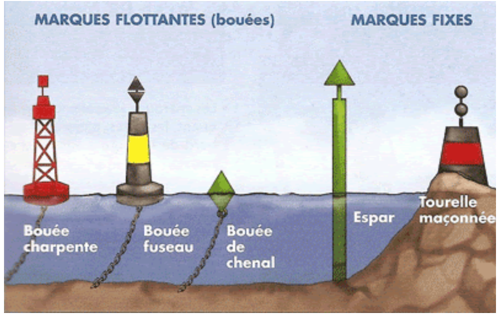
$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leqslant 1, \qquad \forall d \geqslant J$$

- MOIP
- Solve 10 times as MILP (agregate functions with different weights)
- Inject solutions as NSGA2 initial population





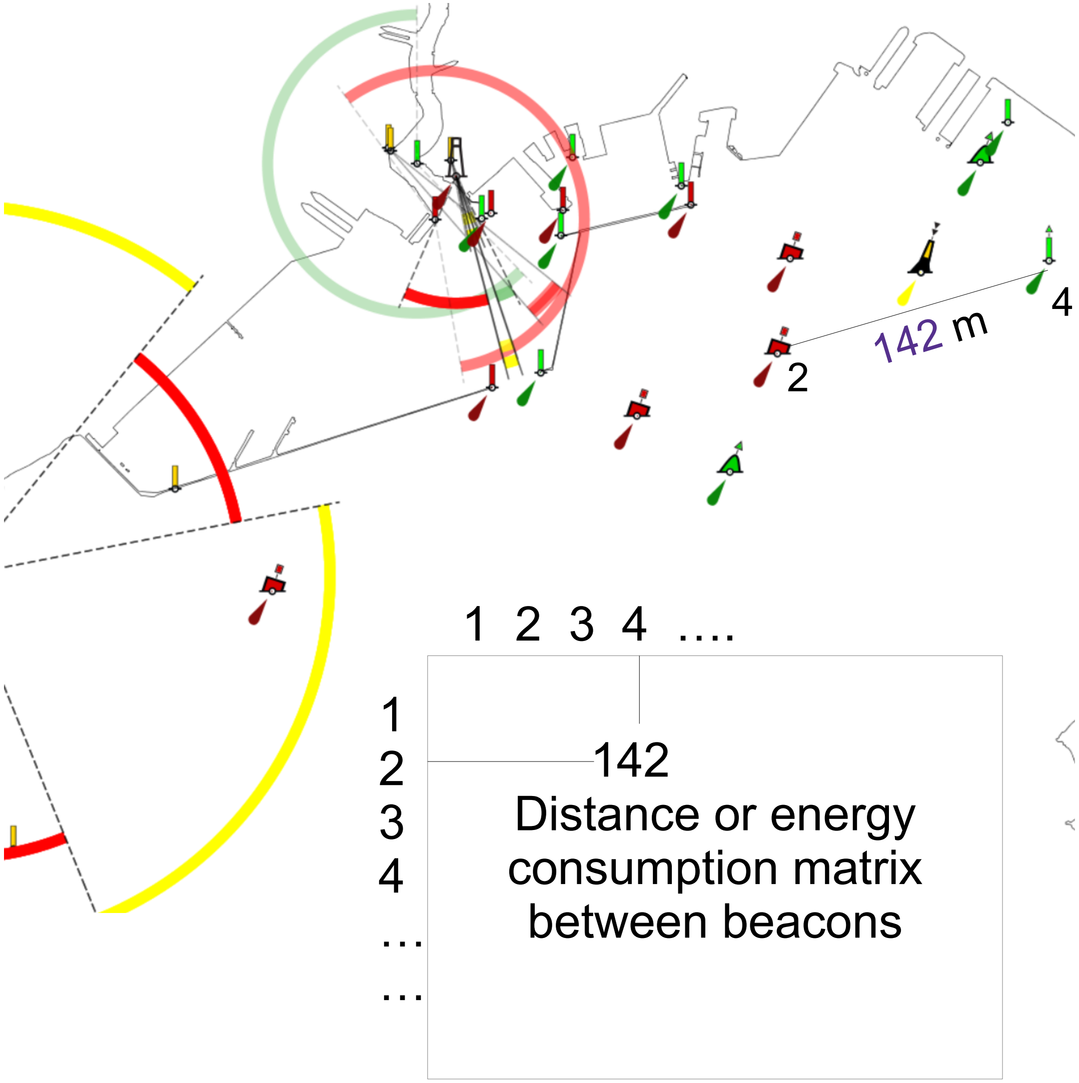
Lab : ROV mission



How to define the route for a ROV mission ?

- Define a route to a set of locations. Some ones may be ignored.
- Dive the ROV at any beacon
- Location have scores of interrest
- Brest Harbour beacons (buoys’ anchors) inspection
  - Red score 1
  - Green score 2
  - Others score 3

  (urgency of inspection)

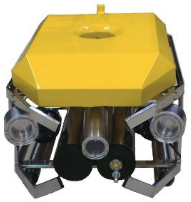


1 2 3 4 ....

1
2
3
4
…
…

142

Distance or energy consumption matrix between beacons



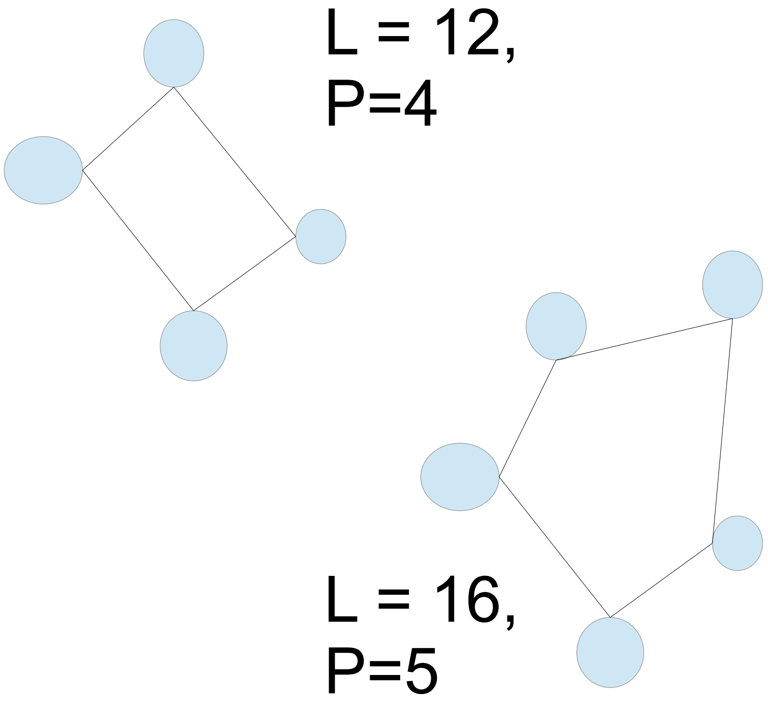
TSP SOO ≠ TSPP MOO :
All cities not mandatory

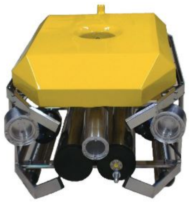
Lab
PAES
Travelling Salesman Problem with Profit

- Data
  - $G = (V, E)$
  - Lengths $v_i \rightarrow v_j$
  - Profits p_i
- Bi-objective MOO
  - min $\Sigma v_i \rightarrow v_j$ *vs* max Σp_i

    node v_1 must be included

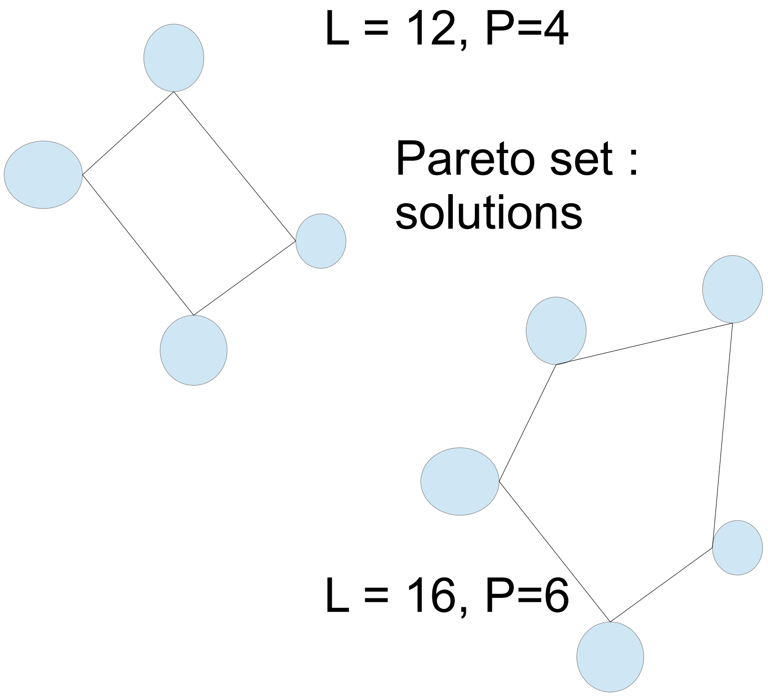
    (not for us, no particular starting point)

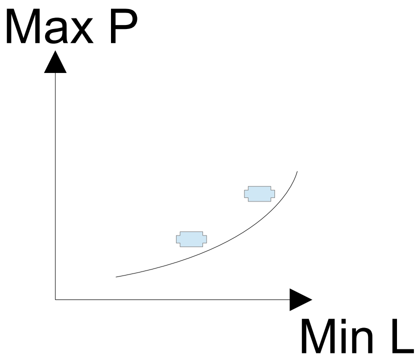


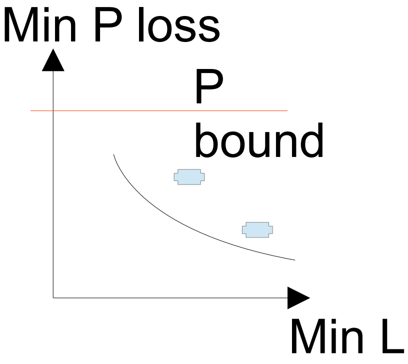


Lab
PAES
TSP : multi objective Travelling Salesman Problem

- How to solve a bi-objective problem with PAES ?
  evaluation functions :
- min Length (L)
- max Profit (P) → min loss of profit
- How to encode solutions ?
- How to mutate solutions ?
- (possible operations ?)

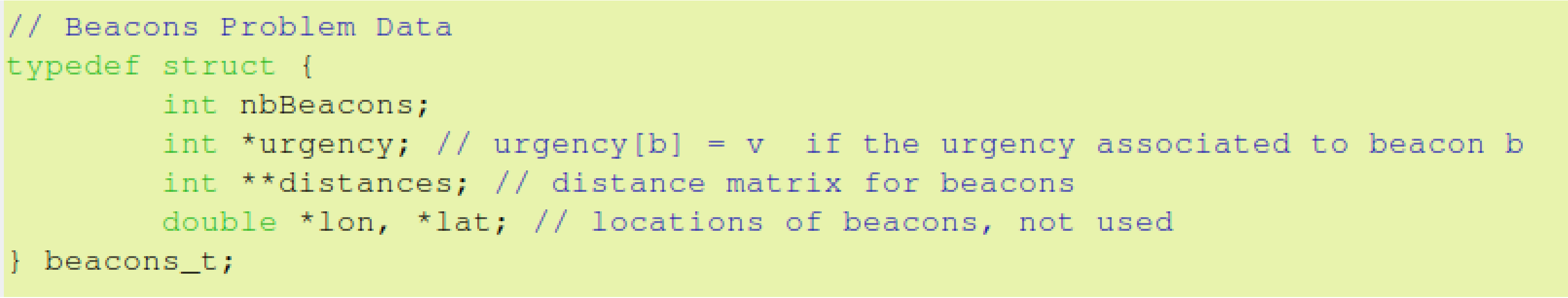




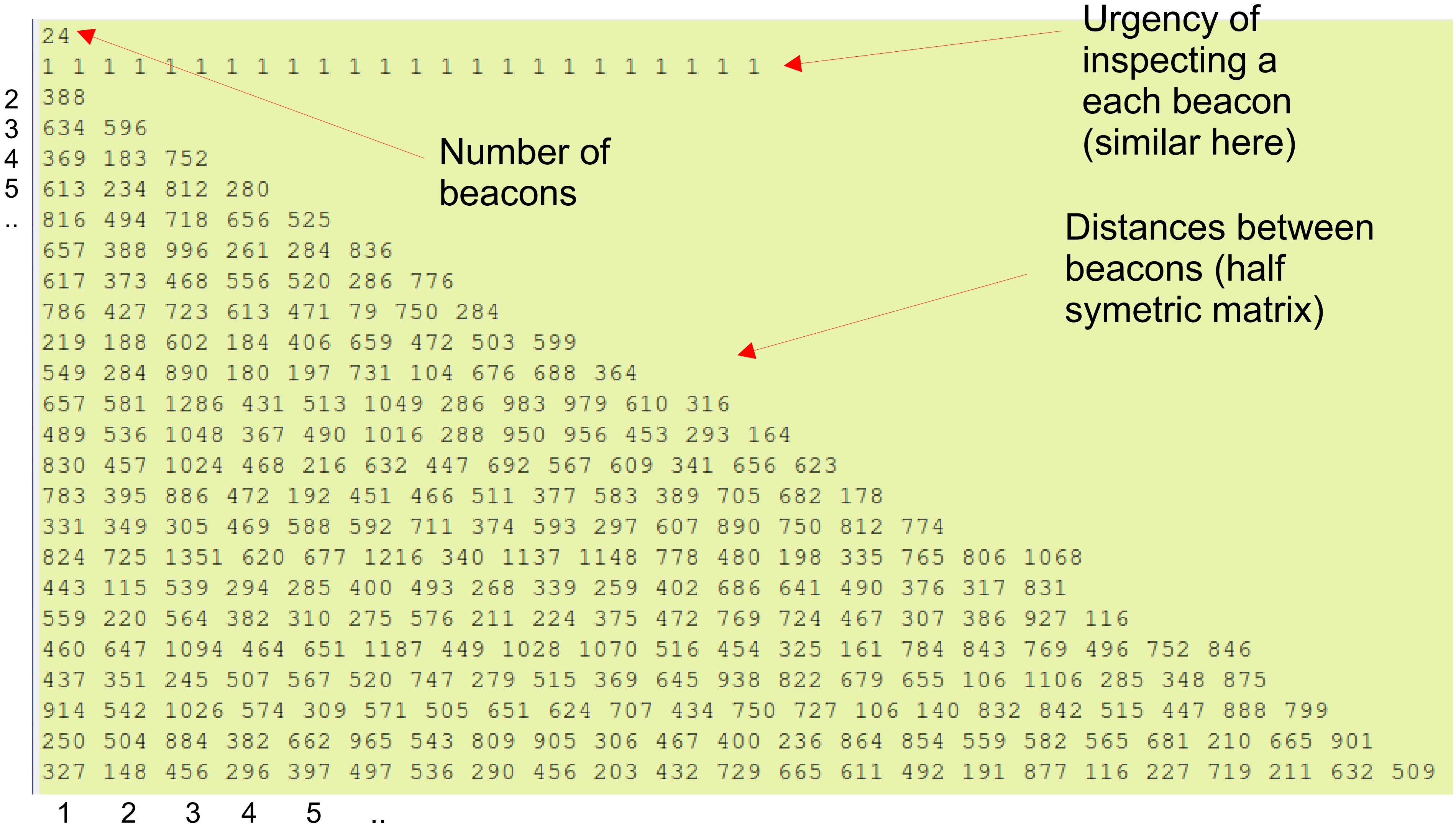


Pareto front :
Values of solutions

Lab
PAES
Input data



```
// Beacons Problem Data
typedef struct {
        int nbBeacons;
        int *urgency; // urgency[b] = v  if the urgency associated to beacon b
        int **distances; // distance matrix for beacons
        double *lon, *lat; // locations of beacons, not used
} beacons_t;
```

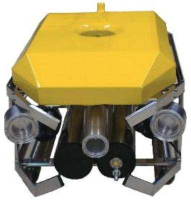


```
   24
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2  388
3  634 596
4  369 183 752
5  613 234 812 280
.. 816 494 718 656 525
   657 388 996 261 284 836
   617 373 468 556 520 286 776
   786 427 723 613 471 79 750 284
   219 188 602 184 406 659 472 503 599
   549 284 890 180 197 731 104 676 688 364
   657 581 1286 431 513 1049 286 983 979 610 316
   489 536 1048 367 490 1016 288 950 956 453 293 164
   830 457 1024 468 216 632 447 692 567 609 341 656 623
   783 395 886 472 192 451 466 511 377 583 389 705 682 178
   331 349 305 469 588 592 711 374 593 297 607 890 750 812 774
   824 725 1351 620 677 1216 340 1137 1148 778 480 198 335 765 806 1068
   443 115 539 294 285 400 493 268 339 259 402 686 641 490 376 317 831
   559 220 564 382 310 275 576 211 224 375 472 769 724 467 307 386 927 116
   460 647 1094 464 651 1187 449 1028 1070 516 454 325 161 784 843 769 496 752 846
   437 351 245 507 567 520 747 279 515 369 645 938 822 679 655 106 1106 285 348 875
   914 542 1026 574 309 571 505 651 624 707 434 750 727 106 140 832 842 515 447 888 799
   250 504 884 382 662 965 543 809 905 306 467 400 236 864 854 559 582 565 681 210 665 901
   327 148 456 296 397 497 536 290 456 203 432 729 665 611 492 191 877 116 227 719 211 632 509
    1   2   3   4   5   ..
```

Number of beacons

Urgency of inspecting a each beacon (similar here)

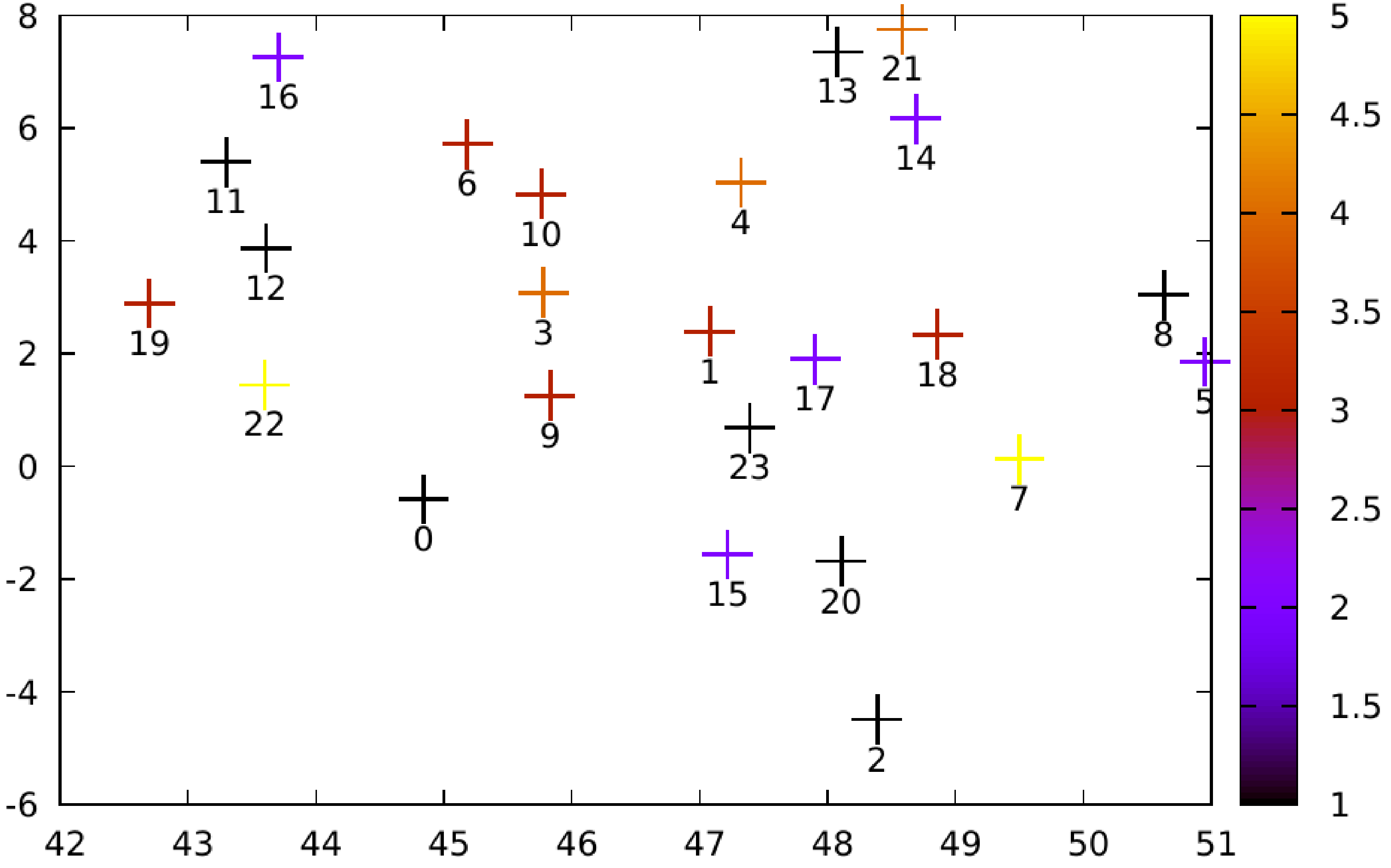
Distances between beacons (half symetric matrix)

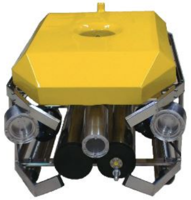


Lab
PAES
Input data

Urgency of inspecting beacon (between 1 and 5 here)

Drawing beacons at their location according to their urgency



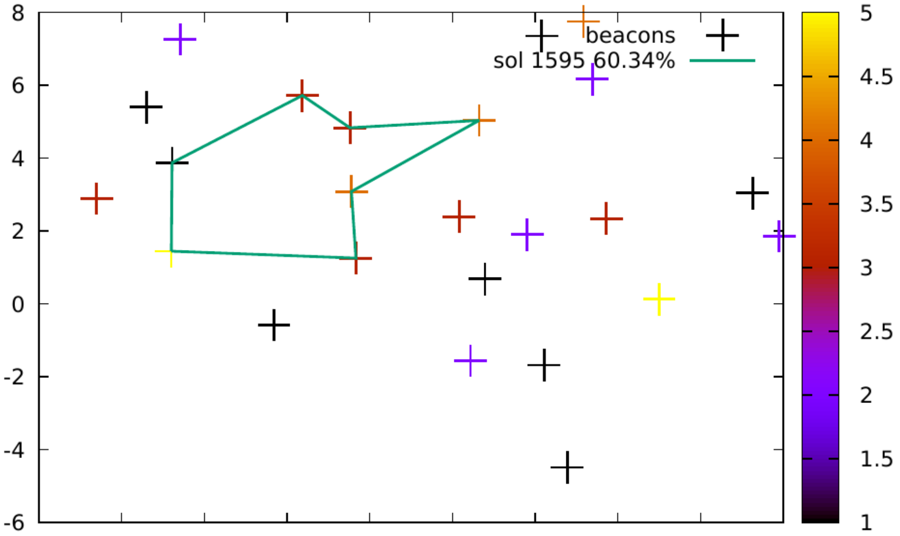


Lab
PAES
Coding a solution

- Chromosome of fixed size (nb beacons)

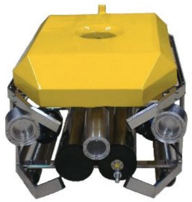
What kind of structure ? Give an example

- Sol 1595 60.34 % (length, urgency loss)



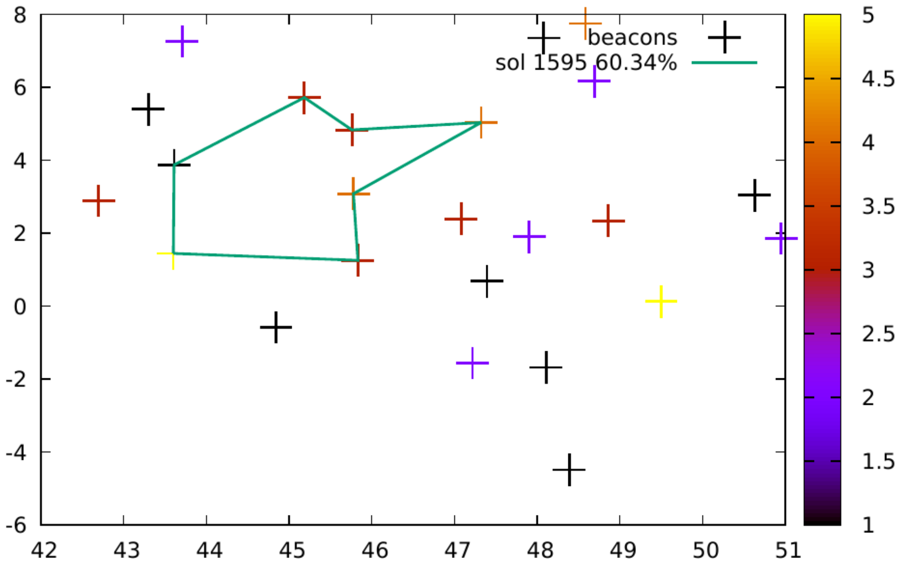
A solution

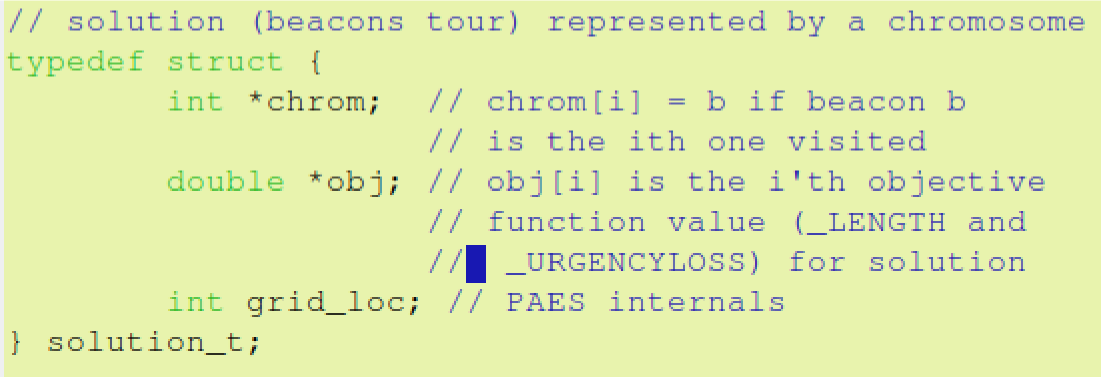
How to code it in C ?



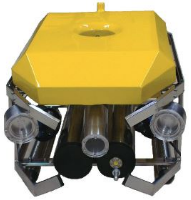
Lab
PAES
Coding a solution

- Chromosome of fixed size (nb beacons)
- Describe (in order) which beacons are visited (not all maybe)
  eg : 24 beacons, tour 8 → 5 → 3 → 0 → 7 → 12 → 8
  chrom[24] =
  { -1 -1 -1 -1 -1 -1 8 5 -1 -1 -1 3 -1 -1 -1 -1 -1 0 7 -1 -1 12 -1 -1 }
- Sol 1595 60.34 % (length, urgency loss)





```
// solution (beacons tour) represented by a chromosome
typedef struct {
        int *chrom;  // chrom[i] = b if beacon b
                     // is the ith one visited
        double *obj; // obj[i] is the i'th objective
                     // function value (_LENGTH and
                     //  _URGENCYLOSS) for solution
        int grid_loc; // PAES internals
} solution_t;
```



Lab
PAES
Algorithm

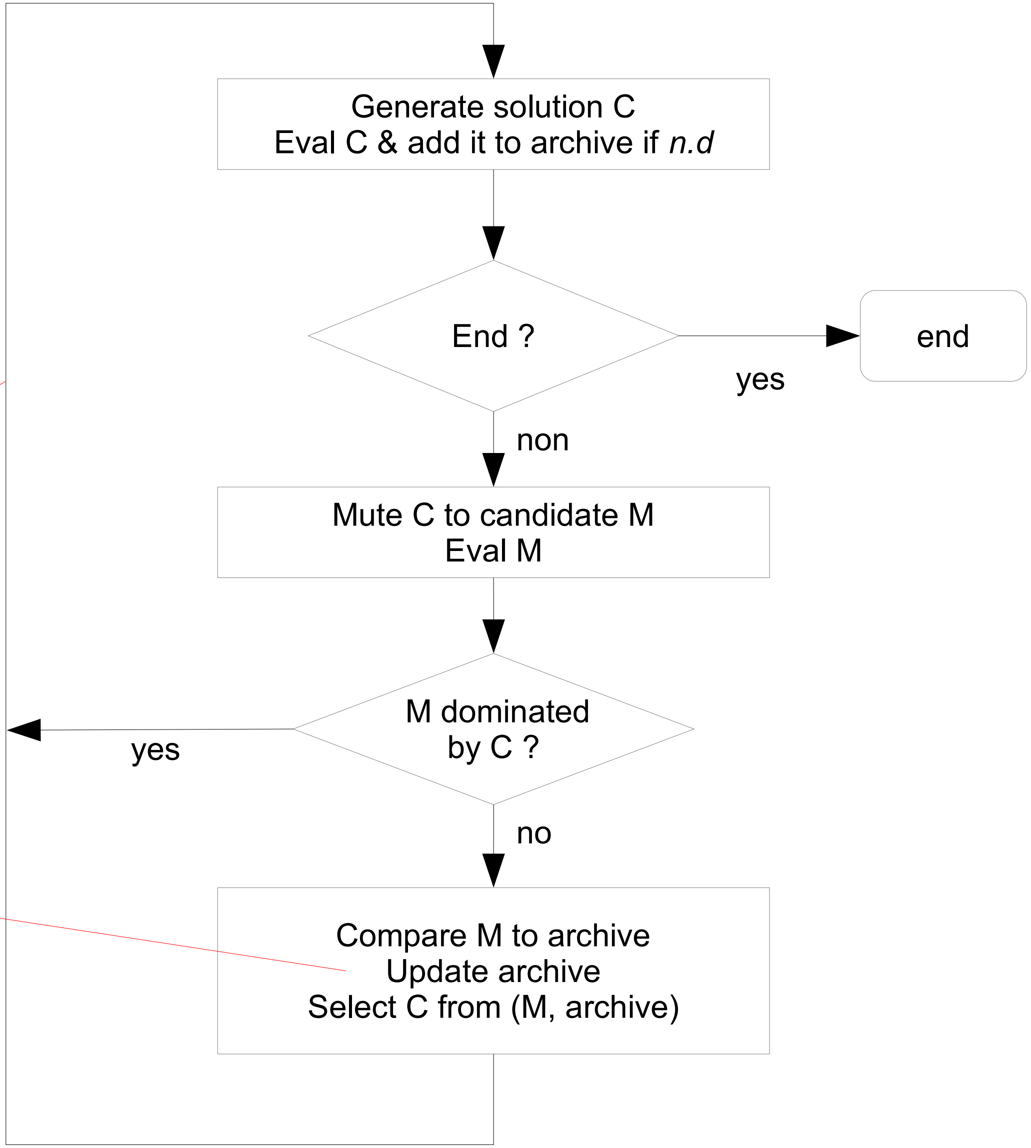
paes.dat params file

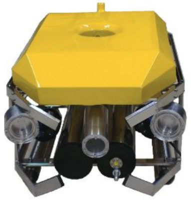
What are the input parameters
of the algorithm ?
(independant from testcase)

PAES C structure

What are the data maintained by
the algorithm ?
(independant from testcase)

How to code the structure ?

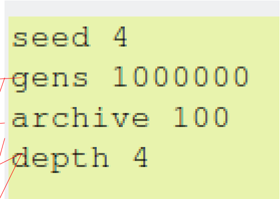




Lab
PAES
Algorithm

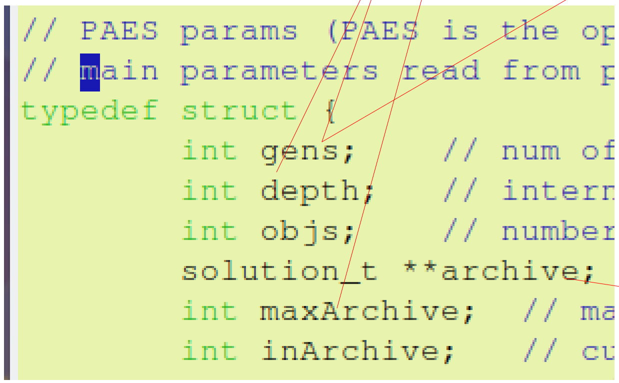
paes.dat params file

Random number seed
# generations
Size of archive
Crowding grid thickness

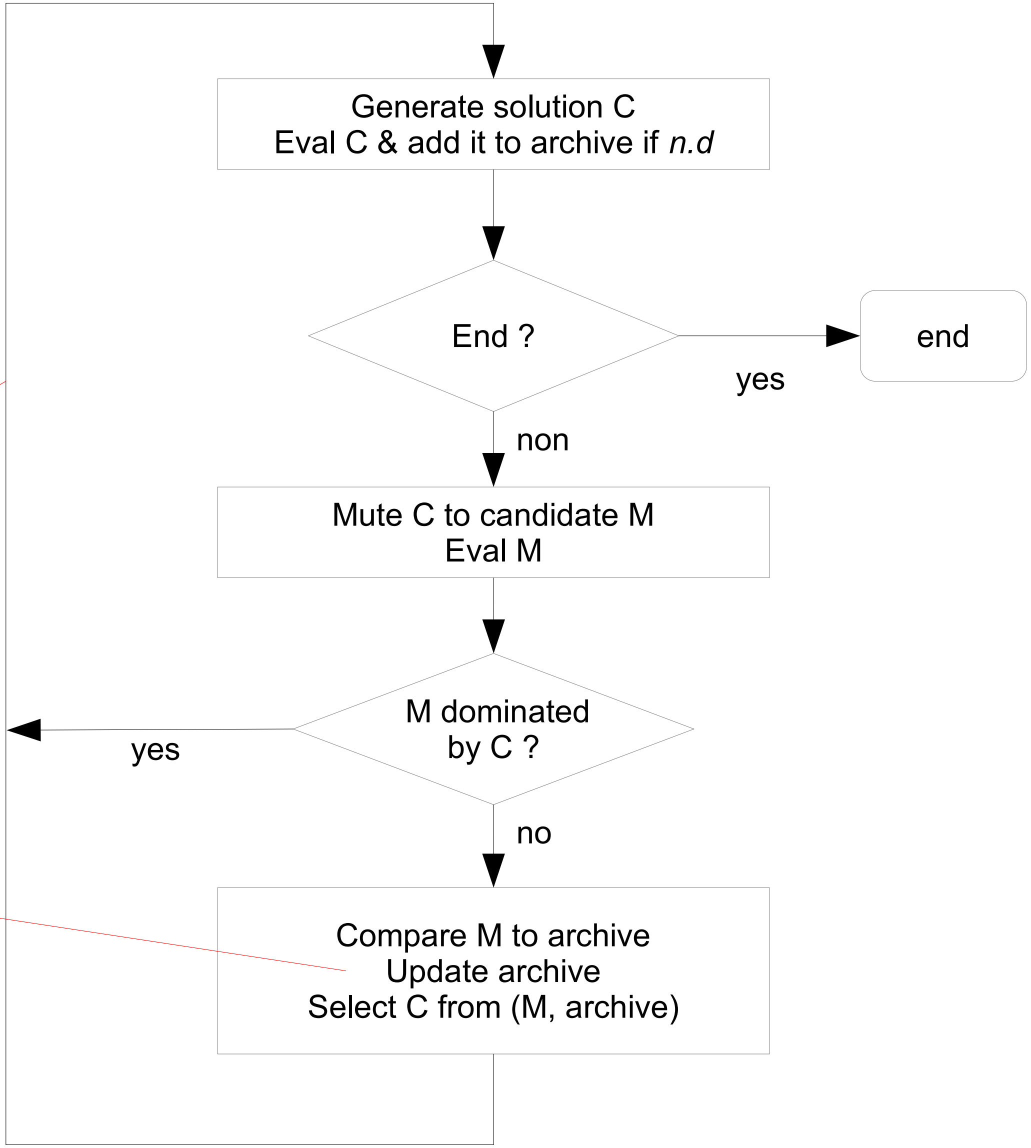


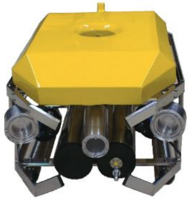
```
seed 4
gens 1000000
archive 100
depth 4
```

PAES C structure



```
// PAES params (PAES is the op
// main parameters read from p
typedef struct {
        int gens;     // num of
        int depth;    // intern
        int objs;     // number
        solution_t **archive;
        int maxArchive;   // ma
        int inArchive;    // cu
```





Lab
PAES
Algorithm

Usage : `./paesbeacons paes.dat beacons.dat`

C coding of algorithm

```
solution_t *c, *m;
beacons_t *beacons;
paes_params_t *paes;
```

```
first = new_sol(beacons, paes);
paes_update_grid(first, paes);
paes_archive_soln(first, beacons, paes);
```

```
beacons = read_beacons(argv[2]);
c = paes_init(paramfile, &paes, beacons);
```

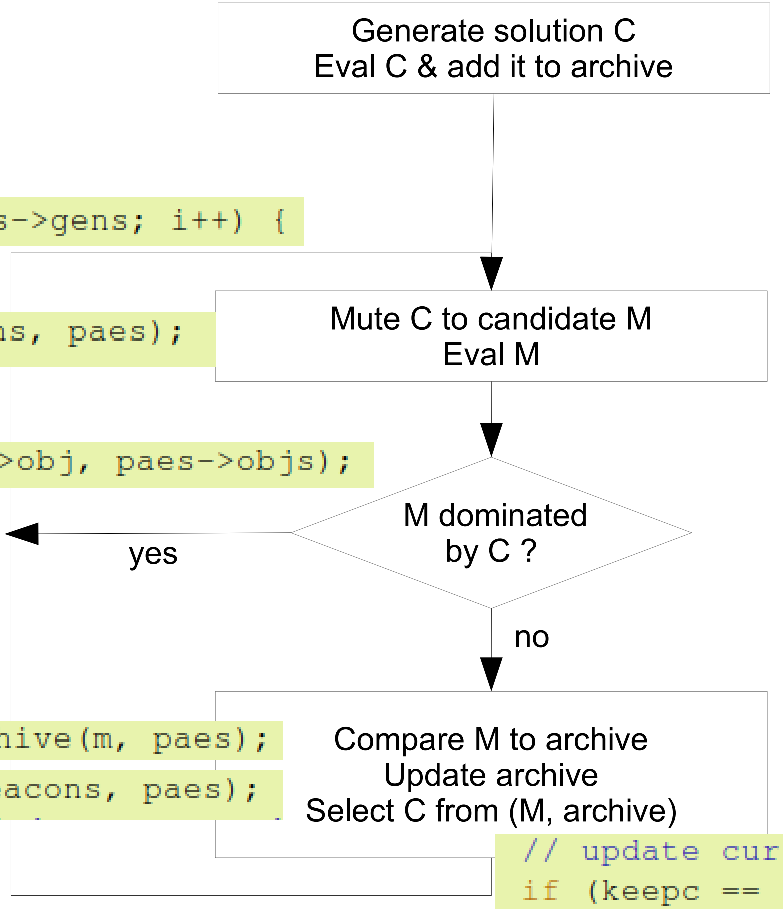
```
for (i = 0; i < paes->gens; i++) {
```

```
m = mutate_sol(c, beacons, paes);
```

```
result = compare_min(c->obj, m->obj, paes->objs);
```

```
paes_compare_to_archive(m, paes);
paes_archive_soln(m, beacons, paes);
```

```
// update current solution
if (keepc == 1)
        free_sol(m);
else {
        free_sol(c);
        c = m;
}
```



Generate solution C
Eval C & add it to archive

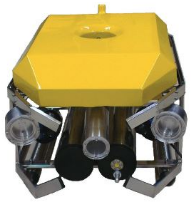
Mute C to candidate M
Eval M

M dominated
by C ?

yes

no

Compare M to archive
Update archive
Select C from (M, archive)



Lab
PAES
Todo list

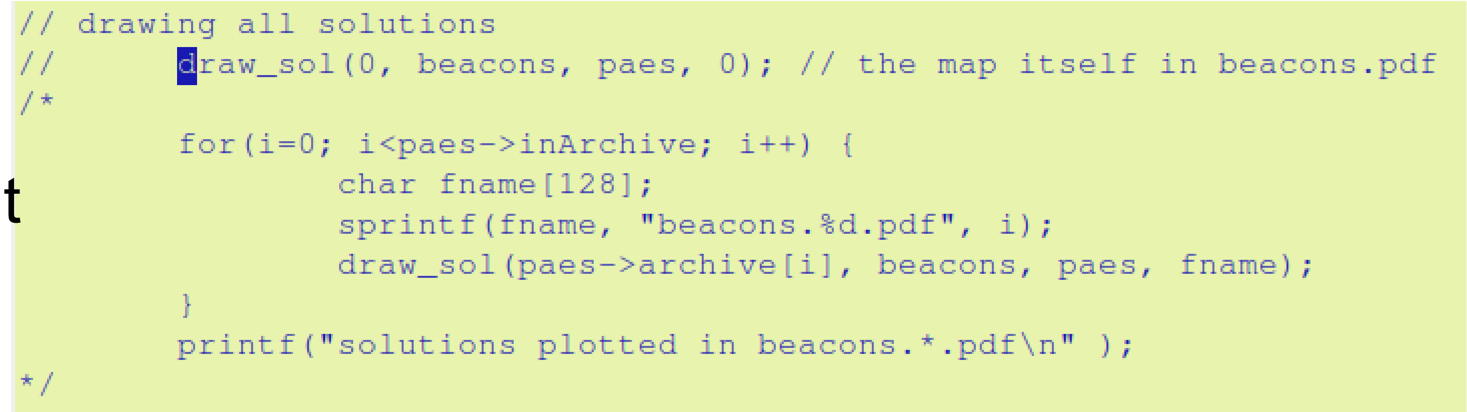
- Go to the PAES directory, and compile the achieved program corr_paes

```
cc corr_paesBeacons.c -o corr_paesBeacons -lm
```

- Play with it, modifying paes.dat, draw solutions, and beacon map

```
./corr_paesBeacons paes.dat beacons.dat
```

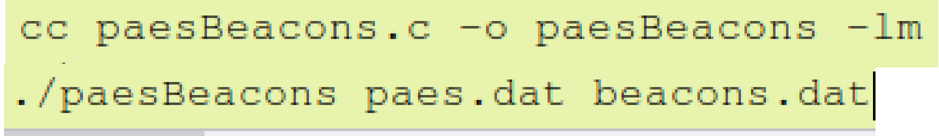
uncomment



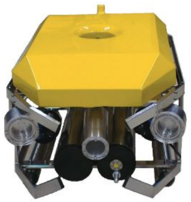
```
// drawing all solutions
//      draw_sol(0, beacons, paes, 0); // the map itself in beacons.pdf
/*
        for(i=0; i<paes->inArchive; i++) {
                char fname[128];
                sprintf(fname, "beacons.%d.pdf", i);
                draw_sol(paes->archive[i], beacons, paes, fname);
        }
        printf("solutions plotted in beacons.*.pdf\n" );
*/
```

- Now edit your own version and fill the evaluation function, looking for `// LAB:` tags in `paesBeacons.c`

- Compile and test

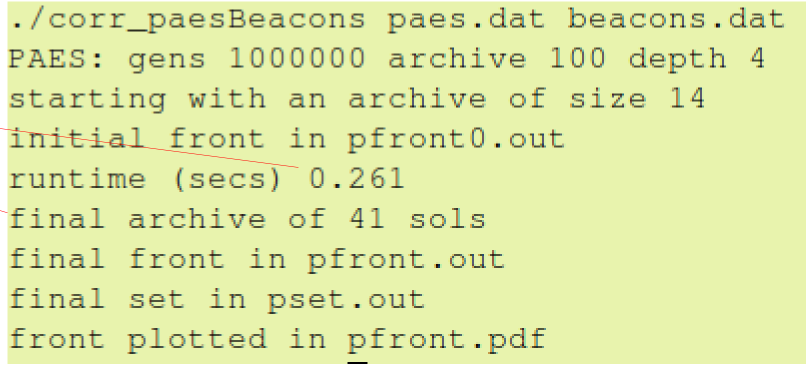


```
cc paesBeacons.c -o paesBeacons -lm
./paesBeacons paes.dat beacons.dat
```

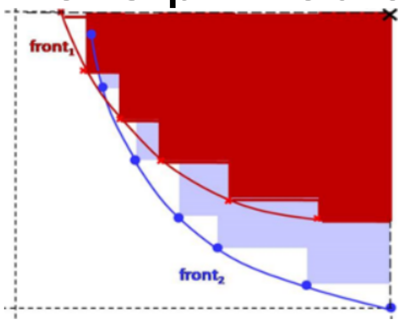


Lab
PAES
Todo list cont'd

- Use provided or your own version for testing the algorithm
- Plot time vs quality graph, varying #gens :
  - Quality is measured as hypervolume of solution front, front set is saved
  - Time is printed at each execution



```
./corr_paesBeacons paes.dat beacons.dat
PAES: gens 1000000 archive 100 depth 4
starting with an archive of size 14
initial front in pfront0.out
runtime (secs) 0.261
final archive of 41 sols
final front in pfront.out
final set in pset.out
front plotted in pfront.pdf
```



- In order to measure quality, use

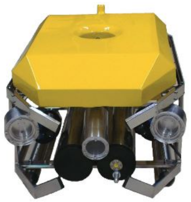
```
Tools/hyp2D 9000 100 pfront.out
```

  → it prints the HV

- To get the nadir point L value (U=100)

```
sort -n -r pfront.out | head -1
```

- Use same nadir value for all HV computations (you can put many fronts in the same pfront file, separated by blank lines)



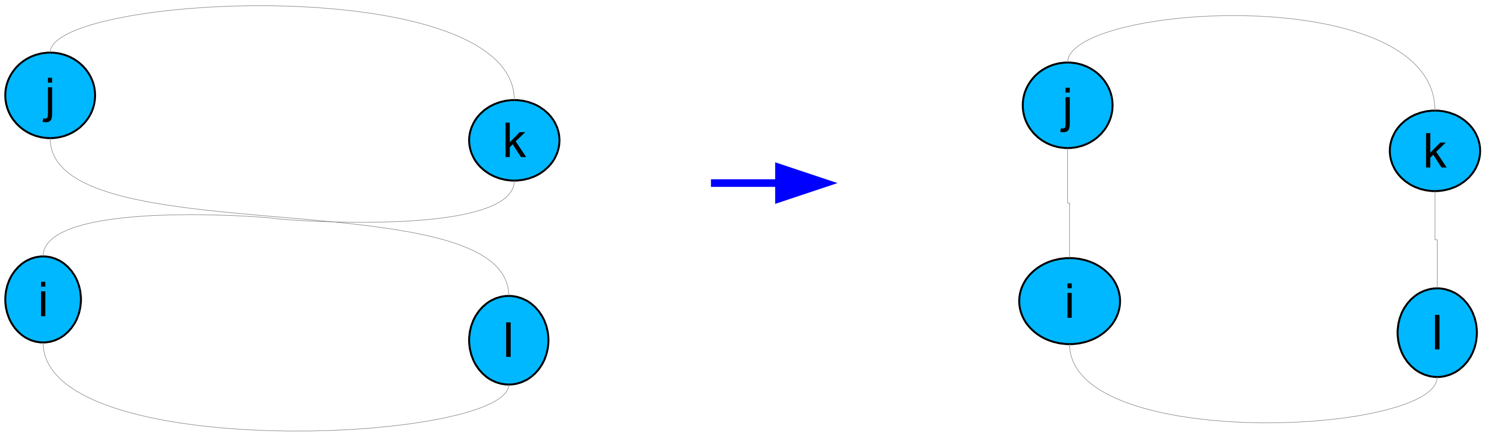
Lab
PAES
Todo list cont'd

- Use the same method for measuring convergency, computing average HV difference between initial and final fronts

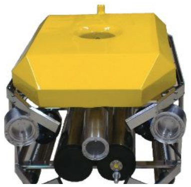
Bonus exercise

- If time, try to improve the code with a local search technique
  - Look at 2-opt search operator for TSP, (Lin, 1965, n(n – 3)/2 neighbours)

T' = T U { i→k, j→l } \ { i→j, k→l }



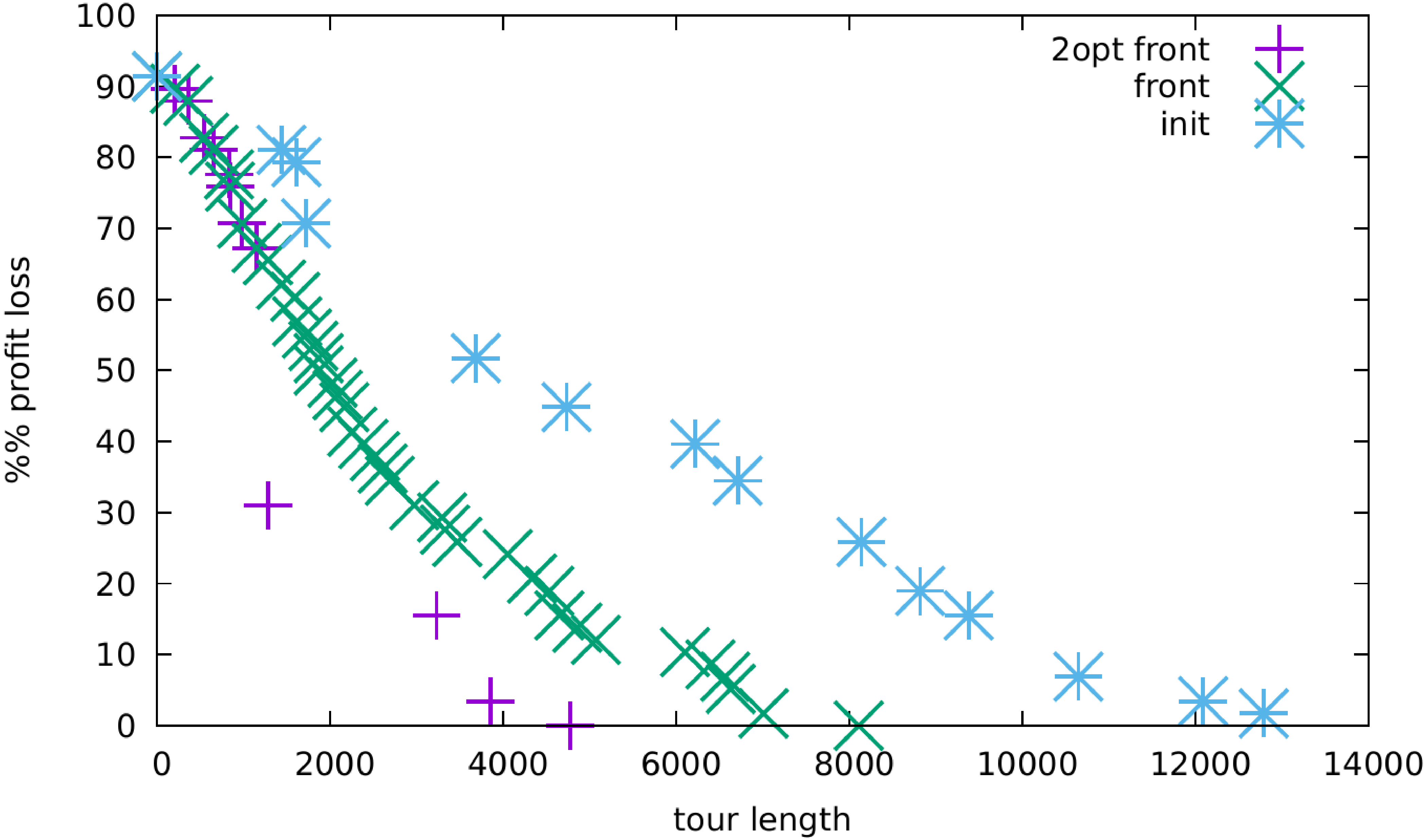
  - Propose a way to introduce it into the code, as a local search technique, as a post optimization and/or at each evaluation



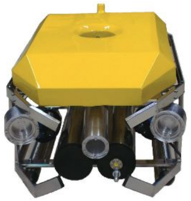
Lab

PAES

Todo list cont'd



technique, as a post optimization and/or at each evaluation

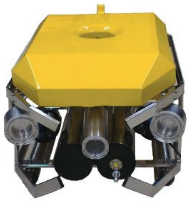


Lab : ROV mission extension

How to choose embedded equipment and route for a ROV mission ?

- Choose a motor version : the heaviest is the faster
- Choose a camera : the heaviest is the fastest (more performant)
- Choose a projector : the heaviest is the fastest
- Choose a route among a set of possible one, each has a length

→ minimize energy consumption, depending on total weight, and energy factors
→ minimize time depending on time factors



Lab : ROV mission

Data

Equipment	Weight	Time factor	Energy factor
Motor 1	12	1	0.3
Motor 2	14	0.7	0.4
Motor 3	20	0.4	1.0
Camera 1	3	1.0	0.5
Camera 2	5	0.3	1.0
Projector 1	1	1.0	0.5
Projector 2	2	0.2	1.0

Path order	Length	Time Factor
A → B → C → D	12	1.0
A → B → D → C	14	0.8
A → C → B → D	13	0.7
A → C → D → B	20	0.6
A → D → B → C	40	0.4
A → D → C → B	30	0.5

Use the TSP part of version 1

Energy consumption =
W(rov) * TF(motor) * EF(motor) * L + EF(camera) + EF(Projector)

Mission time =
time_0*TF(path) + time_1*TF(motor) + time_2*TF(camera) + time_3*TF(projector)

Questions ?

Brest 08
e internationale . Rendez-vous du 11 au 17 juillet 2008
Ouest-France