Synthèse 4 (08/04) - (16/04)

Structure of the Multi-Objective Local Search algorithm

– Start with a random population of size 1.
– Choose solution from the Population.
– Generate the associated neighborhood.
– Update the archive by adding the new non-dominated solutions (domination in the objective space).
– When no improve movement can be done, filter the archive so it only stores non-dominated solutions (in the **decision space**).

Pareto Local Search algorithm :

– A local filtering is applied (pre-filtering) when generating the neighborhood of a solution. It keeps only the non-dominated ones.
– Right after, the archive is updated.

PLS

```
Data: init_population
Population ← init_population
for alt ∈ Archive do
 | Archive ← Archive ∪ {alt}
end
while Population is not empty do
 | alt ← Population.pop_front()
 | neighborhood ← alt→ get_neighborhood()
 | for neighbor : neighborhood do
 | | //pre-filter
 | | if alt does not dominates neighbor then
 | | | Update_Archive(neighbor, Local_front)
 | | end
 | end
 | //update Archive after each exploration
 | for new_alt : Local_front do
 | | if Update_Archive(new_alt, Archive ) then
 | | | Population ← Population ∪ {new_alt}
 | | end
 | end
 | Local_front.clear()
end
```

Figure 1: PLS approach

Update_Archive(p, set_Sol) : Add new alternative p to the set *set_Sol* if it is not dominated and also remove the dominated ones by p.

get_neighborhood() : give all the neighbors of the alternative.

Neighborhood generation :

Generation of a neighborhood's alternative is done by removing one object and fill the backpack with the remaining items ordered in a decreasing order of the ratio $r_c(.) = \frac{f_c(x}{w(x)}$.
$f_c(.)$ is an aggregation WS function with the set of weights c. We tried to evaluate different variation of c using:

1. a random generation of weights.
2. a random feasible weights.
3. the exact (optimal) weights.

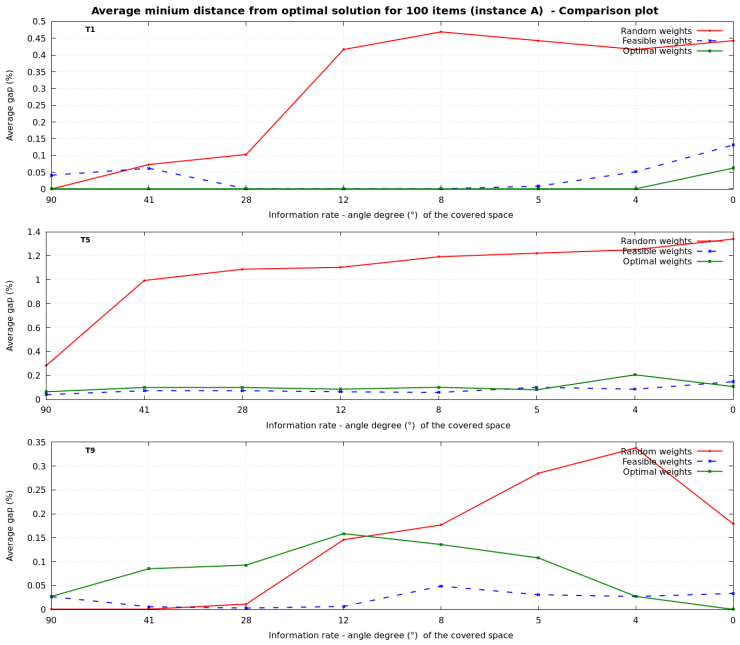
We measure the impact of this changes, by measuring the average minimum distance to the optimal solution using random instances A of size 100.

AVG minimum distance

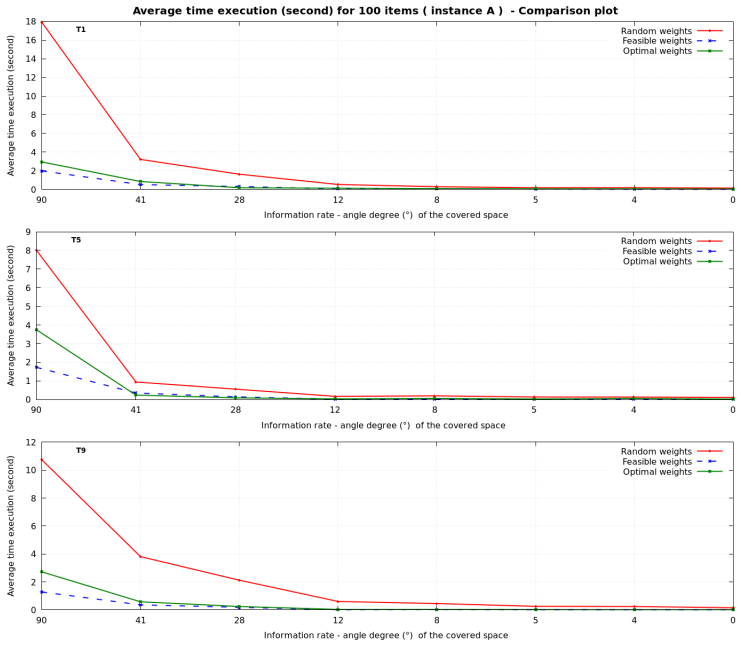
	T1			**T5**		
angle degree	**Random**	**Feasible**	**Optimal**	**Random**	**Feasible**	**Optimal**
90	**0**	0.0411996	**0**	0.191015	**0.0405006**	0.064801
42	0.0730011	0.0617994	**0**	0.294025	**0.0755598**	0.10227
28	0.102999	**0**	**0**	0.308698	**0.0755598**	0.10227
12	0.416516	**0**	**0**	0.408084	**0.0645525**	0.08607
8	0.469146	**0**	**0**	0.570922	**0.058714**	0.10227
0	0.442831	0.131421	**0.062283**	0.818039	0.148488	**0.107588**

AVG time execution (sec)

	T1			**T5**		
angle degree	**Random**	**Feasible**	**Optimal**	**Random**	**Feasible**	**Optimal**
90	17.9343	**1.97908**	2.93483	11.216	**1.73094**	3.74744
42	3.20504	**0.50407**	0.835641	3.08918	0.354667	**0.241344**
28	1.61661	**0.302227**	0.168313	1.94912	0.148462	**0.094583**
12	0.515898	**0.0399829**	0.120573	0.38225	**0.0248546**	0.039162
8	0.279968	**0.0455232**	0.077687	0.327977	**0.0231983**	0.0642616
0	0.131031	**0.0184169**	0.0289452	0.12592	**0.0158243**	0.026677



Average gap using different set of weight for neighborhood generation - Random instances A (T1,T5,T9) of size 100



Average time execution using different set of weight neighborhood generation - Random instances A (T1,T5,T9) of size 100

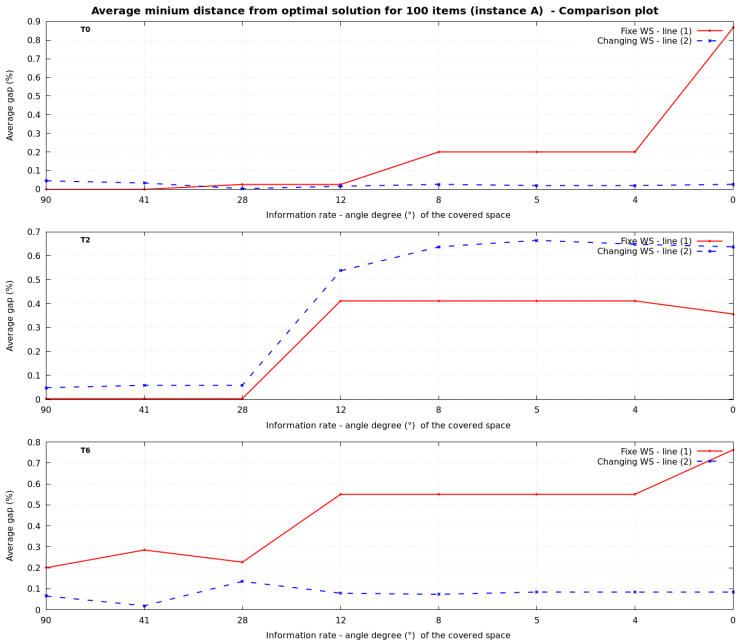
The next figure presents a brief description of *get_neighborhood* method that generates a neighborhood's solution:

get_neighborhood

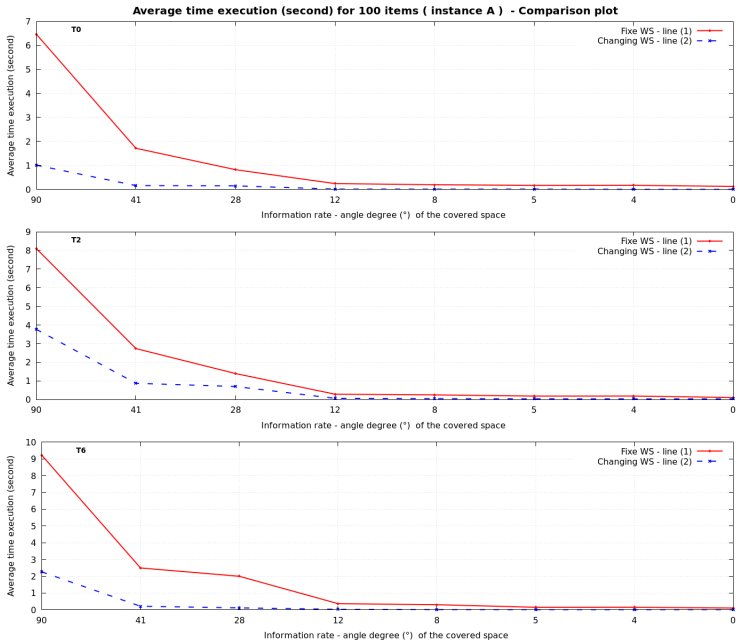
Data: S : picked_items
Result: neighborhood
c ← *generate_random_WS* (1)
for *item : S* **do**
  new_neighbor ← $\mathcal{S}/\{item\}$
  c ← *generate_random_WS* (2)
  order the remain items in decreasing order of r_c
  new_neighbor ← *fill the backpack with the remain ordered items*
  neighborhood ← *neighborhood* ∪ {new_neighbor}
end

Figure 2: Generation of neighborhood

We also measured the fact of using the same set of weight c at each deletion of an item (line **(1)** only) and when constantly changing the sets of weights (line **(2)**).



Average minimum distance from optimal solution - Random instances A (T1,T5,T9) of size 100



Average time execution - Random instances A (T0,T2,T6) of size 100

Note : No comment.

11 Controlling the population size :

For the random benchmarks of 100 items, we change the population size from 10 to 210 by keeping only **non-dominated** solutions **randomly selected** (from the objective space).
Note : It may not reach the fixed limit.

	AVG minimum distance from opt point					AVG time execution				
Instance/Population size	**10**	**50**	**150**	**210**	**∞**	**10**	**50**	**150**	**210**	**∞**
T4	0.23	0.21	**0.012**	**0.012**	0.07	1.83	4.76	8.89	8.82	**0.91**
T5	0.70	0.33	0.29	0.22	**0**	**1.84**	4.43	7.35	7.85	3.33
T6	0.35	0.28	0.238	0.19	**0.065**	2.39	5.63	9.69	10.58	**2.26**

Figure 3: Total uncertainty

	AVG minimum distance from opt front					Proportion of optimal solutions				
Instance/Population size	**10**	**50**	**150**	**210**	**∞**	**10**	**50**	**150**	**210**	**∞**
T4	557.19	581.73	**325.61**	357.29	344.13	9.37	15	**42.67**	42.5	40.35
T5	462.66	317.78	278.46	234.02	**52.8372**	0.44	1.76	1.98	2.20	**76.32**
T6	408.33	276.35	228.38	235.33	**135.21**	0	0	0.51	0.64	**19.871**

Figure 4: Average minimum distance from the optimal front and Proportion references Indicators

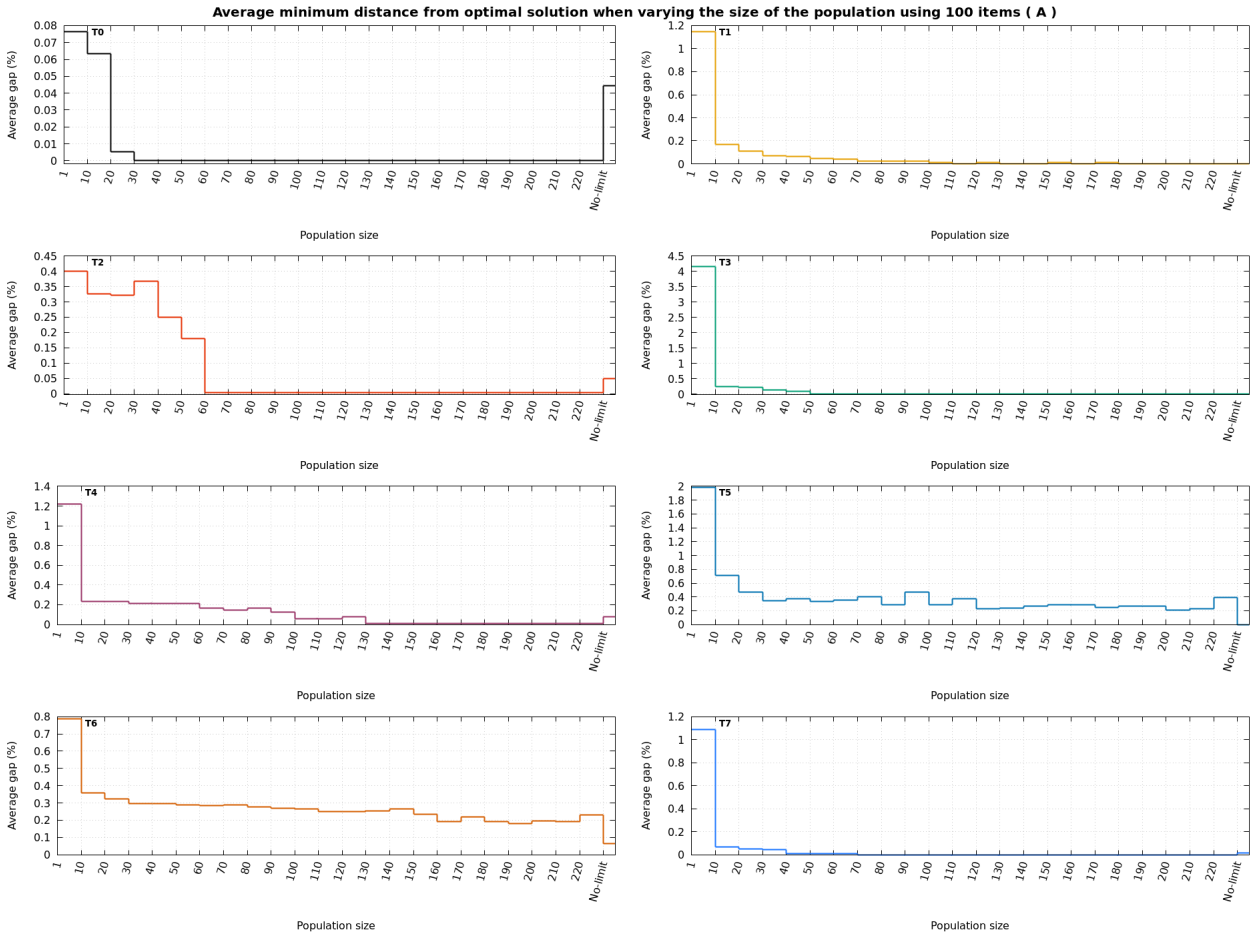


Figure 5: Average gap - Random instances A size 100

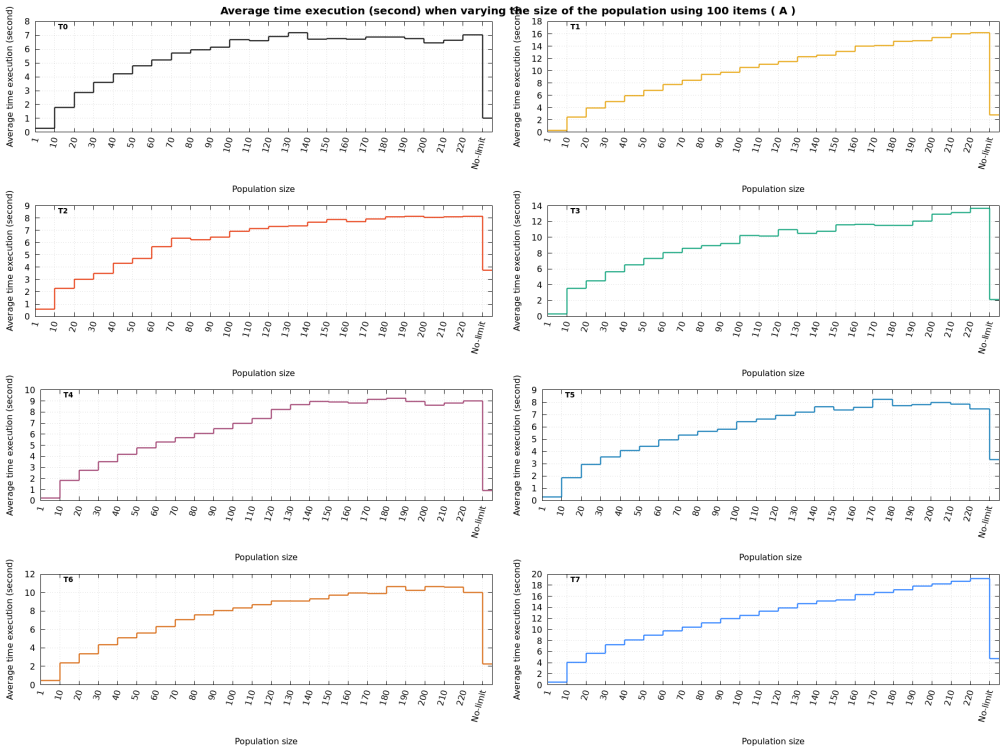


Figure 6: Average time execution - Random instances A size 100

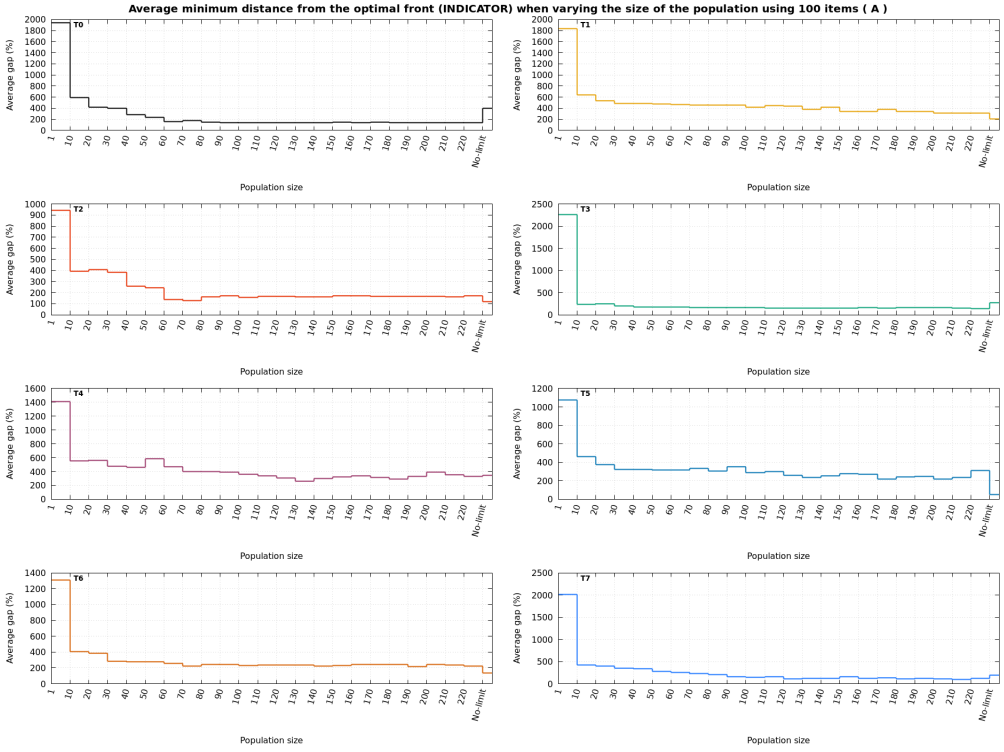


Figure 7: Average minimum distance from optimal front (INDICATOR) - Random instances A size 100

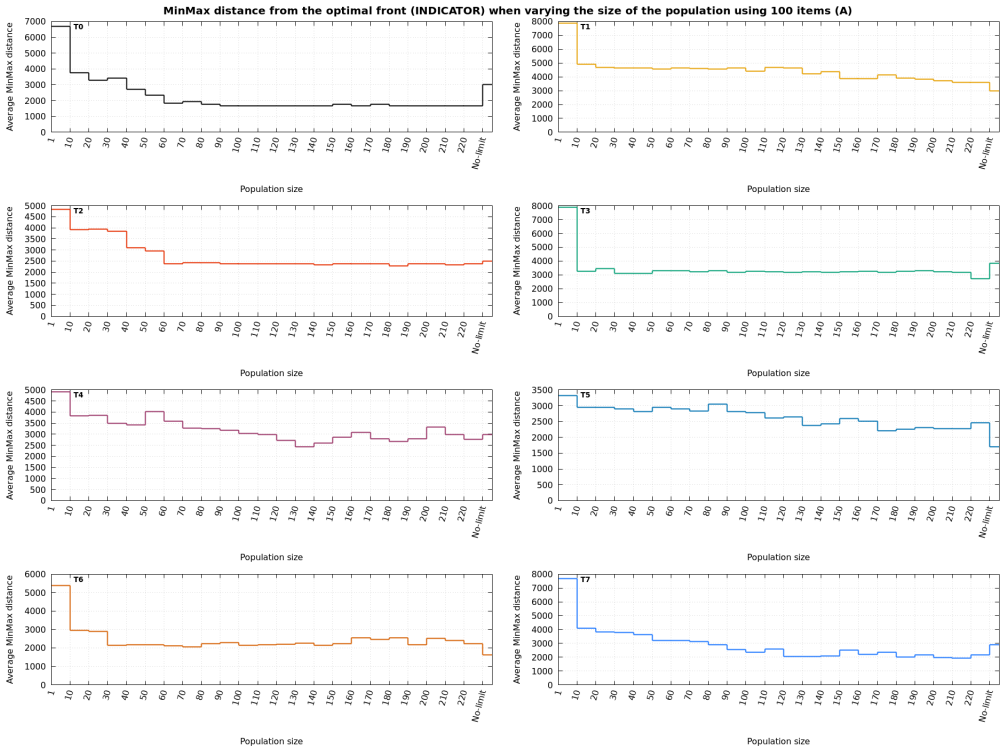


Figure 8: Average MaxMin distance form optimal front (INDICATOR) - Random instances A size 100

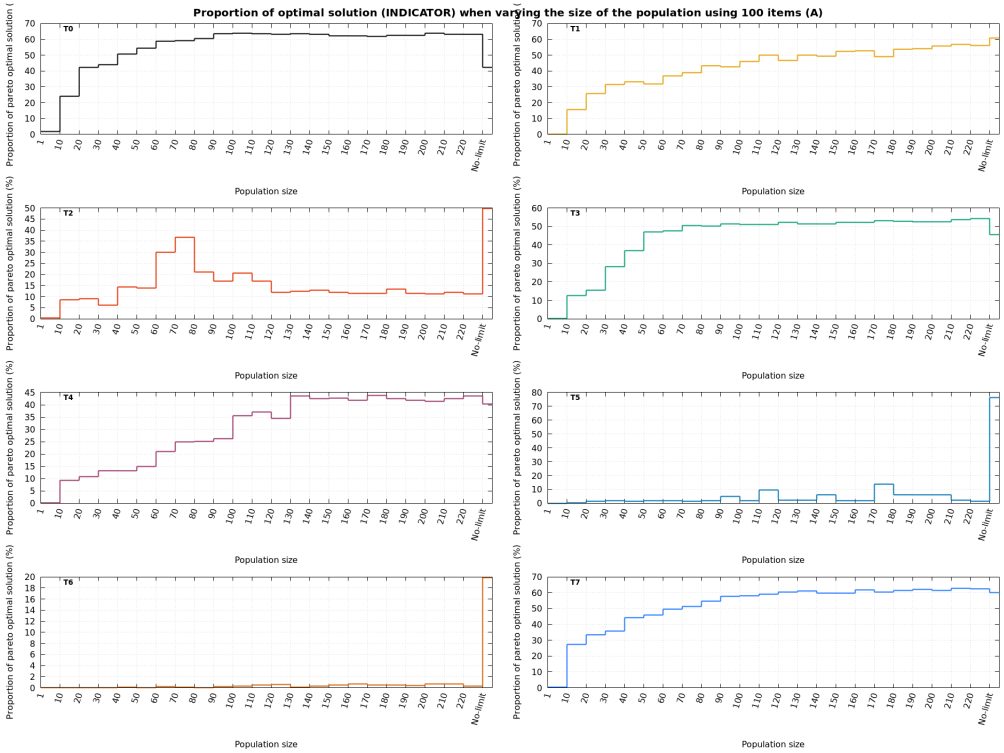


Figure 9: Average Proportion of optimal solution (INDICATOR) - Random instances A size 100

– **MaxMin Indicator (to minimize)** : Computes the max-min euclidean distance of reached solutions χ with the optimal front $\mathcal{OPT}$: $D_2(\chi, \mathcal{OPT}) = max_{r \in \mathcal{OPT}} \, min_{x \in \chi} \, dist(r, x)$

Evolution of the search space

Figures bellow show the evolution of the search space when varying the population size.
A group of solutions (points) that belong to the same population (generation) have the same color plot.

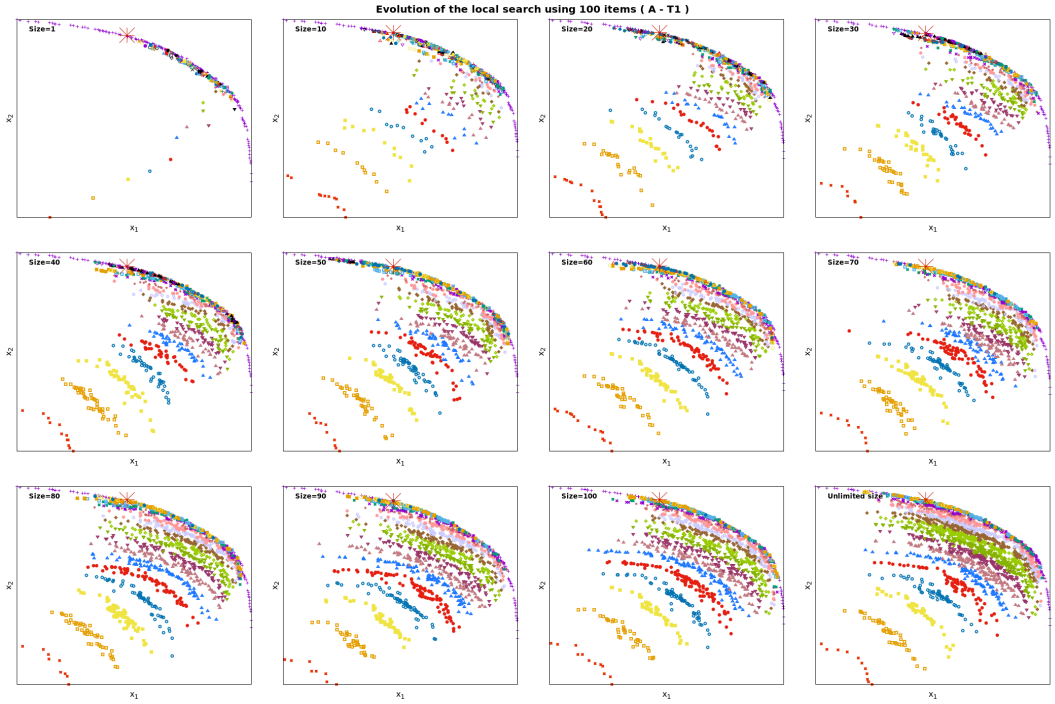


Figure 10: Evolution of search space - random instance A T1 with 100 items (100% UNCERTAINTY)

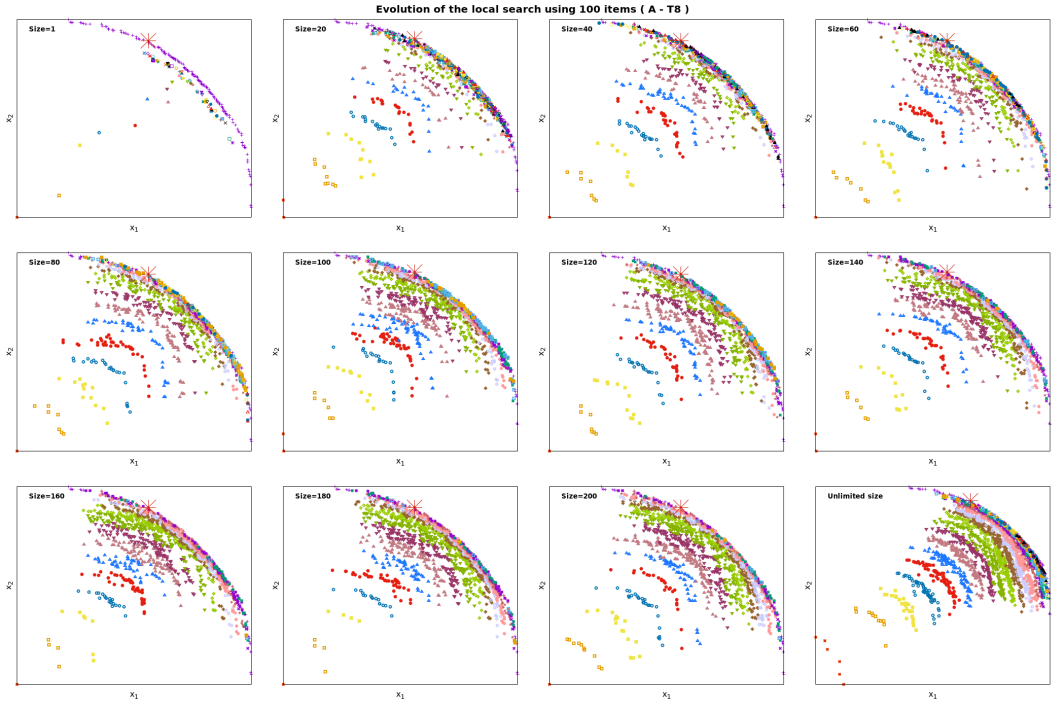


Figure 11: Evolution of search space - random instance A T8 with 100 items (100% UNCERTAINTY)

Note : ONLY SIZE = 10 did not reach the timeout (90sec).

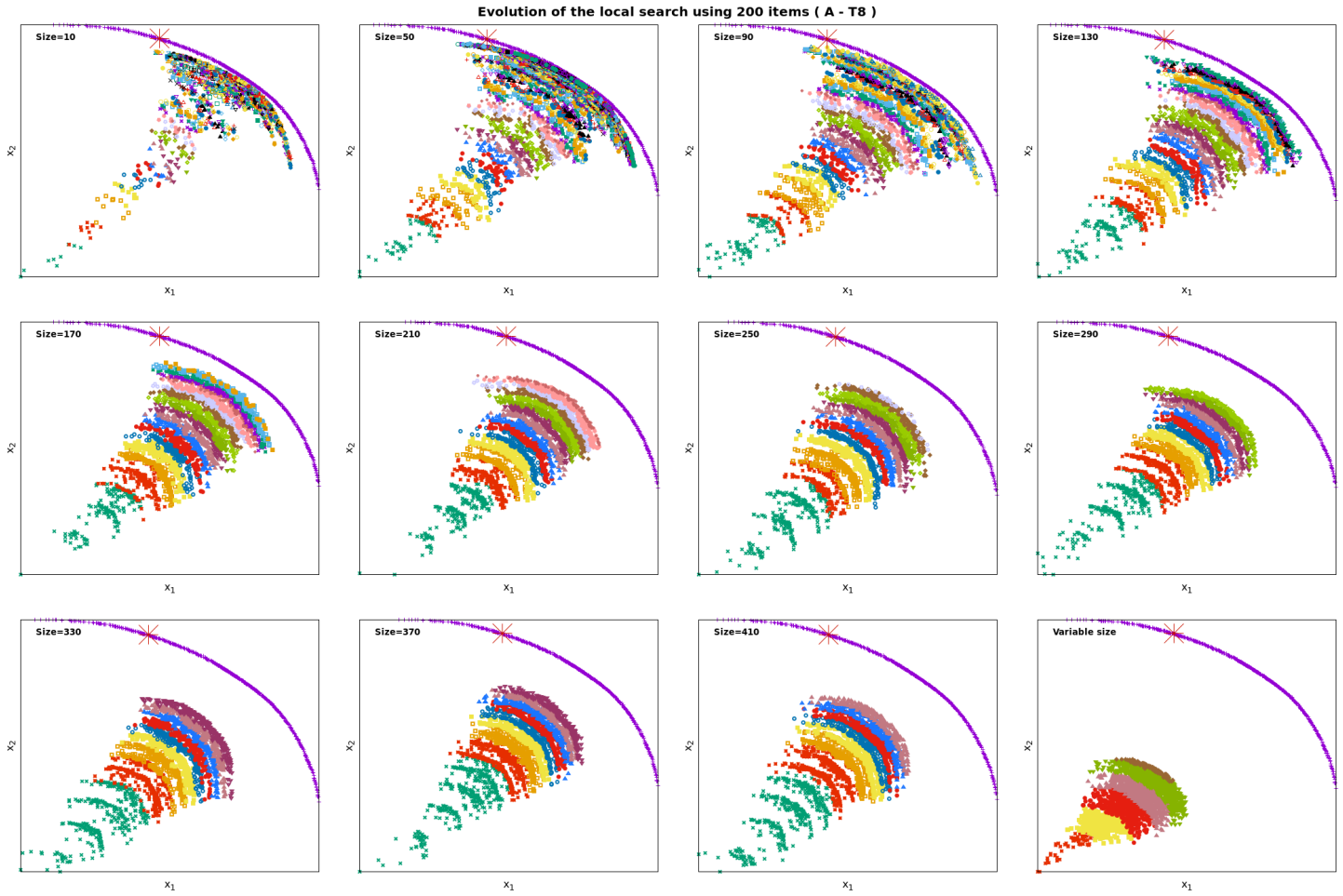


Figure 12: Evolution of search space - random instance A T8 with 200 items TIMEOUT 4 min (100% UNCERTAINTY)

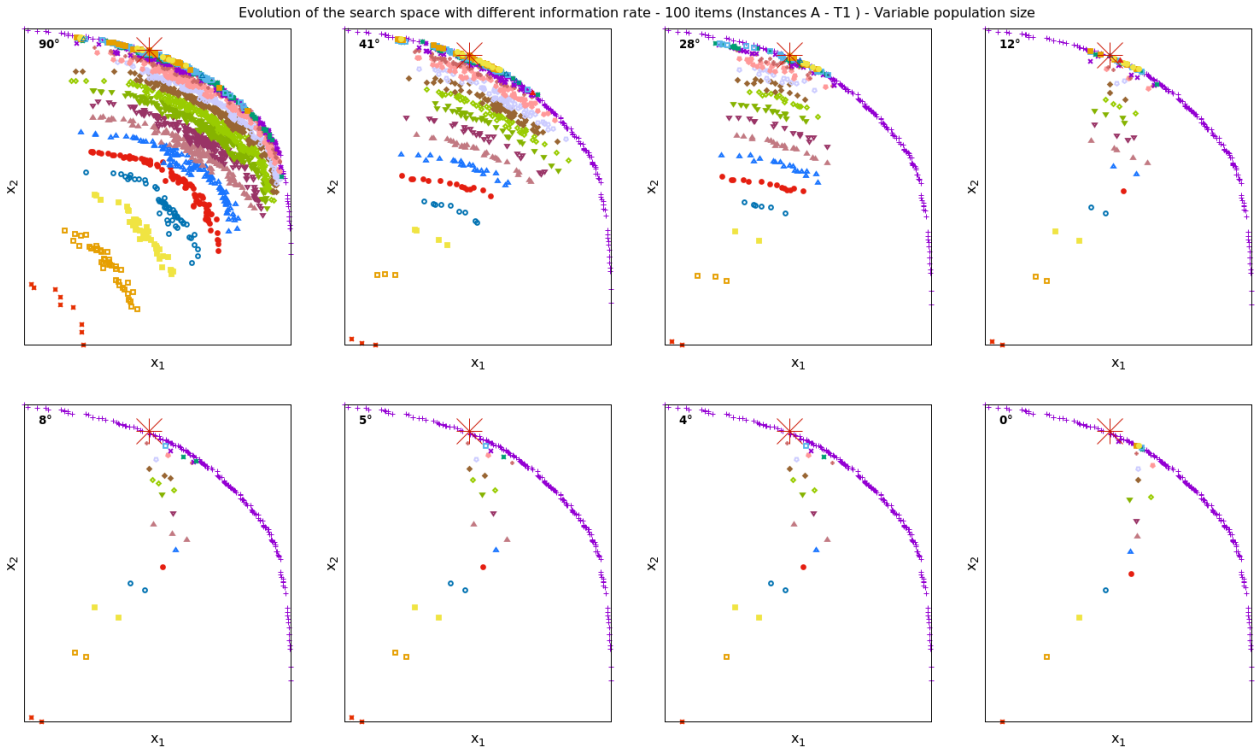


Figure 13: Evolution of search space - random instance A T1 with 100 items (Decreasing uncertainty)

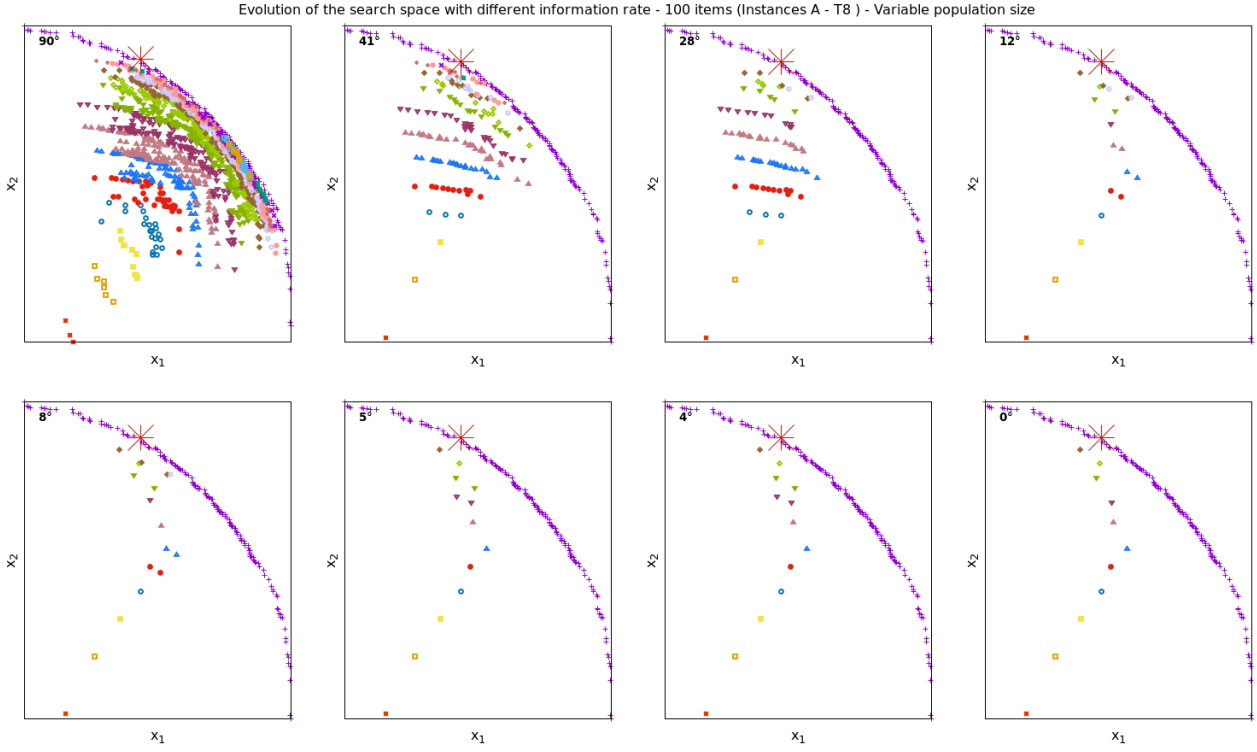


Figure 14: Evolution of search space - random instance A T8 with 100 items (Decreasing uncertainty)

12 Diversify the population :

– At each exploration of an alternative, we save the dominated neighbors in a local list.
– After updating the Archive, we choose with a probability D to accept or not alternatives from this list.
– Different approaches can be tested to diversify the population :

Limited number accepting N

Keep at most N random dominated solutions.

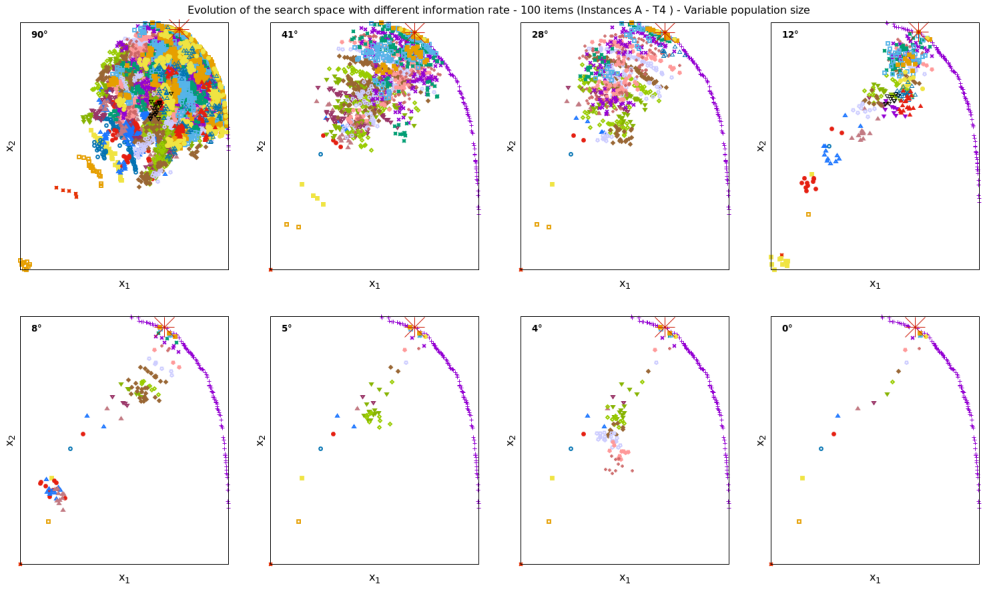


Figure 15: Evolution of search space - random instance A T4 with 100 items D=0.06, N=10

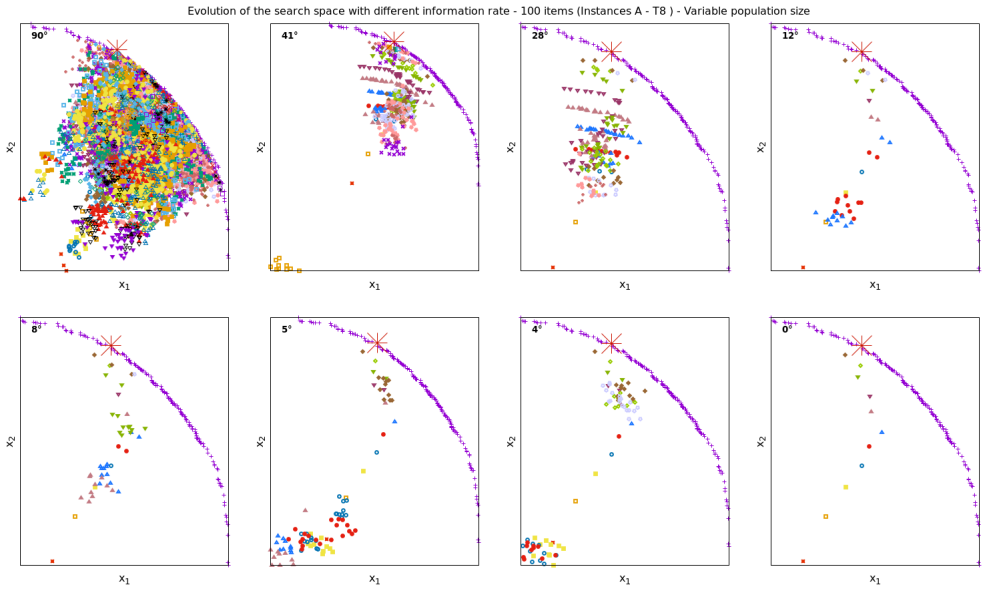


Figure 16: Evolution of search space - random instance A T8 with 100 items D=0.08, N=10

Note : Less efficient when the search space is being smaller.

Threshold Accepting $\mathcal{T}_\mathcal{A}$

Keep only solutions with a threshold accepting from a reference point(s) (parent generator, current population or archive...) that did not exceed $\mathcal{T}_\mathcal{A}$

– **Average threshold**

A dominated solution s' is added to the new Population for exploration if : $|\frac{\sum_{s \in Archive} f(s) - f(s')}{|Archive|}| \leq \mathcal{T}_\mathcal{A}$.

Keeping the same parameter value $\mathcal{T}_\mathcal{A}$ during the execution turns out to be time consuming and less efficient : It tends to explore distant solutions and not considering the search space evolution. More It evolves, more selective It should be.

We introduce a more sophisticated method to diversify the population as in a Simulated Annealing approach. It reduces the exploration rate at each iteration of the algorithm.
In our case, to refine the selection of degrading solutions, $\mathcal{T}_\mathcal{A}$ value will decrease all along the execution : $\mathcal{T}_\mathcal{A} = \mathcal{T}_\mathcal{A} * \alpha$. where α is close to 1.

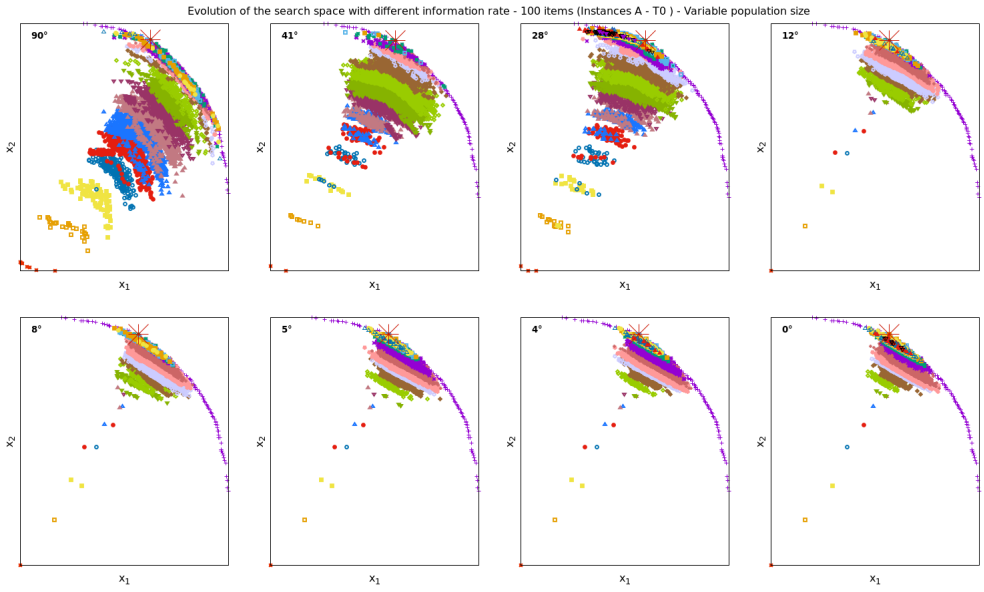
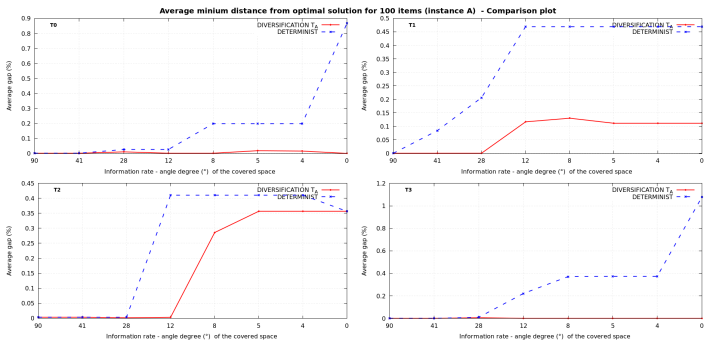
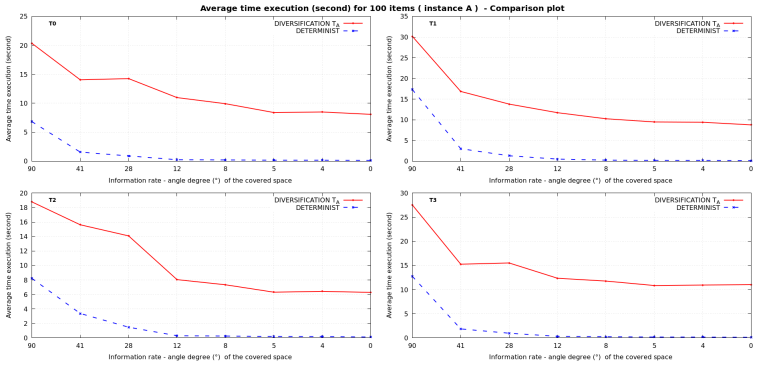


Figure 17: Evolution of search space - random instance A T8 with 100 items D=1, $\mathcal{T}_\mathcal{A}$=500 α=0.9999

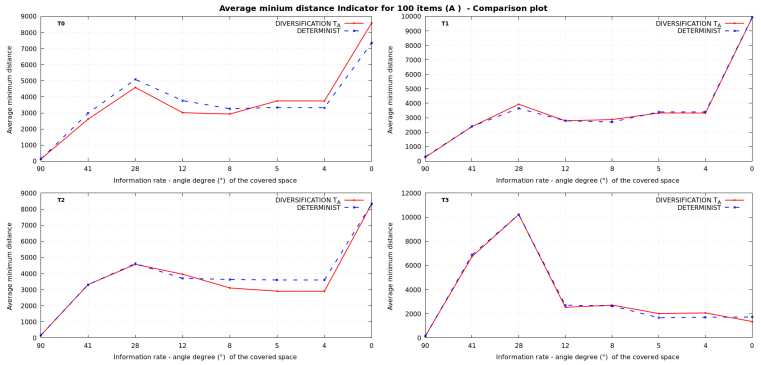
Experimental results when adding divers solutions :



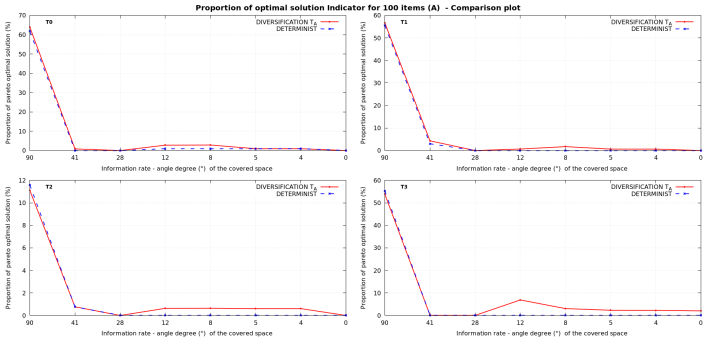
Average gap with/out diversification - Random instances A (T0,...,T3) of size 100



Average time execution with/out diversification - Random instances A (T0,...,T3) of size 100



Average minimum distance from optimal front with/out diversification - Random instances A (T0,...,T3) of size 100



Proportion reference with/out diversification - Random instances A (T0,...,T3) of size 100

Note : Take much more time but give better approximation.

Hybrid approach (WS & PLS)

WS+PLS : the number of iteration (*step*) to explore the search space using a WS influences the direction of optimization. We can stuck into a local optima so PLS couldn't progress.

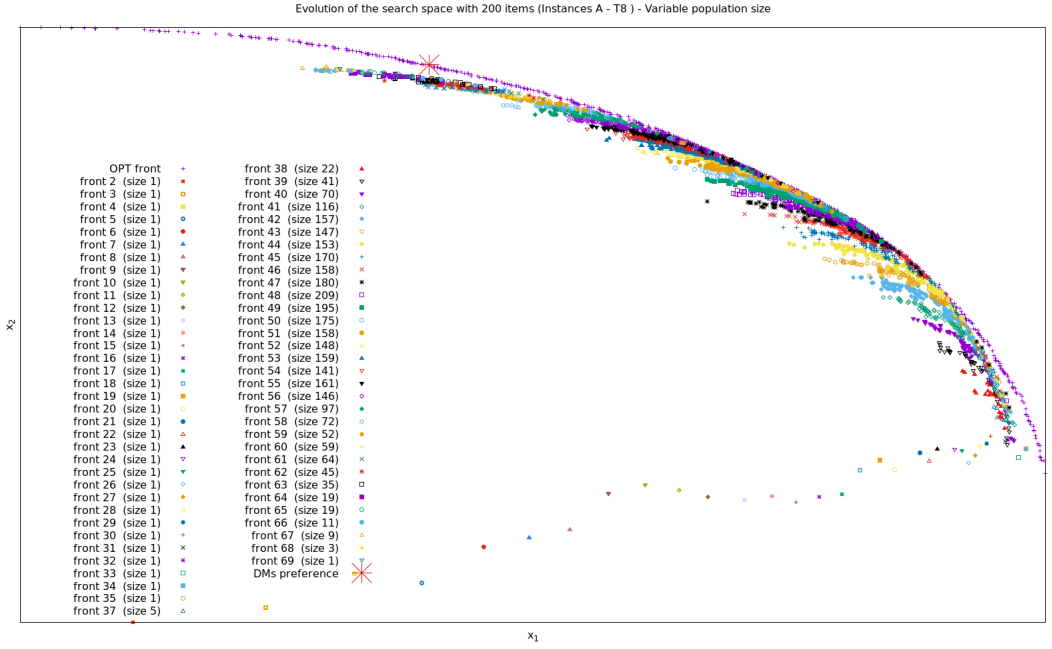


Figure 18: Evolution of search space - random instance A T8 of 200 items with 34 iterations of WS and no diversification

PLS+WS : Figure above shows the execution of PLS a number of time, next the objective is changed into a different random WS for each solution found so far.

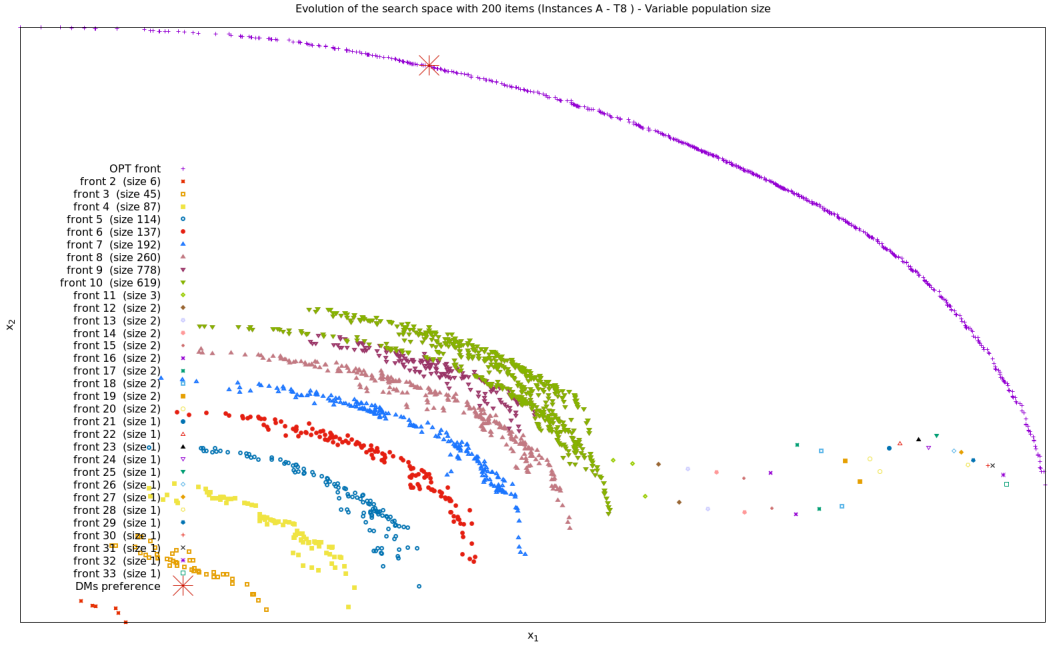


Figure 19: Evolution of search space - random instance A T8 of 200 items with 1000 iterations of PLS and no diversification

It also shows that all solutions found with PLS will be orientate to one direction. To this end, we can assign to each solution a local Archive it will be able to evolve in its own search space.

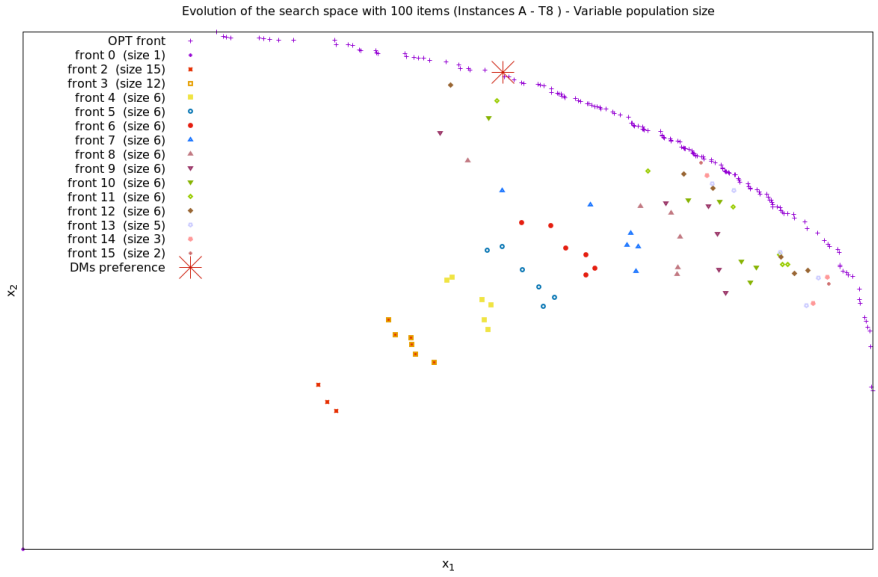


Figure 20: Evolution of search space - random instance A T8 of 100 items with 3 iterations of PLS and no diversification