findBestHyperparam

July 15, 2022

```
[NbConvertApp] WARNING | pattern '#' matched no files
[NbConvertApp] WARNING | pattern 'pdf,' matched no files
[NbConvertApp] WARNING | pattern 'html,' matched no files
[NbConvertApp] WARNING | pattern 'latex' matched no files
[NbConvertApp] Converting notebook findBestHyperparam.ipynb to pdf
[NbConvertApp] Support files will be in findBestHyperparam_files\
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Making directory .\findBestHyperparam_files
[NbConvertApp] Writing 61926 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 821559 bytes to findBestHyperparam.pdf
```

0.1 1. Datasets insight

We used a corpus made of 16 datasets: 2 public (pX datasets), 4 synthetic sampled for agile projects (aX datasets), and 10 synthetic sampled for plan-driven projects (cX and dX datasets): - *pX datasets*: 2 small datasets which have been used in several works, as in 'Ant Colony Optimization for the Next Release Problem'. P1 has very few requirements (20) and also very few dependencies (7). P2 makes a bit more effort in providing more realistic project datasets, althouth dependencies are requirements are not too large.

- *aX datasets*: commonly agile projects have a lower number of stakeholders actively managed, althouth their buy-in in the development of the product if more constant. Thus, aX datasets present a lower number of stakeholders. Since requirements are not decided a priori with a long elicitation process, requirements are dependencies among them are not usually large for a given minor o functional release. Thus, we produced datasets with not a ver high number of dependencies and requirements. For the sake of completenees, two aX dataset have a large number of requirements. Effort estimations are computed using a fibonacci scale, similar to some agile estimation techniques.
- *cX datasets*: classic or plan-driven datsets tend to have a large number of requirements and, due to a long and expensive planning, algo a large number of dependencies. Also due to usual processes which deal with management of stakeholders interests, it is also common to identify more stakeholders than in agile datasets. Effort values were simulated by using Function Points values extracted from the 2015 version of the International Software Benchmarking Standards Group (ISBSG) dataset, using {A,B} values for "Unadjusted Function Points rating", "New development" for "Development type" and "IFPUG 4+" for "Count approach". This procedure is used to generate percentile 25,50,75 of total FPs of a classic project, in order to generate a realistic sample of classic estimation of requirements, done by selecting randomly, for a given number of requirements, a list of costs that sums up to the percentile value.
- *dX datasets*: following the same procedure than for *cX datasets*, here we simulate the most complex classical projects, with very large number of requirements (column #PBI) and dependencies (column #(PBI-->[PBI])). In fact, this is the case in which the MONRP might be of greater help for the decision maker.
- *eX|datasets*: again, derived from classical estimation of effort with Function Points from SBSG 2015, these datasets not only contain large number of requirements, and requirements which imply dependencies, but also the cardinality of these dependencies is also large (column Avg_len[PBI]); that is, when a requirement has a dependency X --> [list of requirements], this list is larger in the eX datasets compared to the others. ---> aún no añadidos en la experimentación, pendiente de que termine MIMIC.

	Dataset	#Stakeholders	#PBI	#(PBI-->[PBI])	%(PBI-->[PBI])	Avg_len[PBI]
0	p1	5	20	7	0.3500	1.8571
1	p2	5	100	29	0.2900	2.6897
2	a1	5	50	18	0.3600	2.2222
3	a2	15	50	18	0.3600	2.7222
4	a3	5	200	74	0.3700	1.9459
5	a4	15	200	75	0.3750	2.2533
6	c1	15	50	20	0.4000	2.4000

7	c2	100	50	17	0.3400	3.5294
8	c3	15	200	69	0.3450	1.9420
9	c4	100	200	75	0.3750	2.0933
10	d1	15	100	45	0.4500	2.8444
11	d2	50	100	39	0.3900	2.1026
12	d3	15	200	88	0.4400	3.3523
13	d4	50	200	88	0.4400	4.8523
14	d5	50	200	88	0.4400	3.4773
15	d6	15	300	131	0.4367	3.7710
16	d7	50	300	145	0.4833	3.6966

0.2 2. FEDA description:

Given an initial set of requirements dependencies in the form of
X1-->X2, FEDA uses this knowledge as a prefixed structure.
e.g: we can have an acyclic graph like this: G={0-->2, 1-->2, 3, 2-->4},
where requirements 0,1,3 do not have parents,
parents(2)={0,1} and parents(4)={2}.

Thus, learning is not structural and only applies to data.
Sampling is always performed following a topological (ancestral) order
([3,0,1,2,4] in the example above).

Algorithm is as follows:

1. Sampling of First generation:
-- If X does not have any parents, then sample using P(X)=1/num_requirements
-- If any Y in parents(X) is set to 1, then X=1, else use P(X)=1/num_requirements

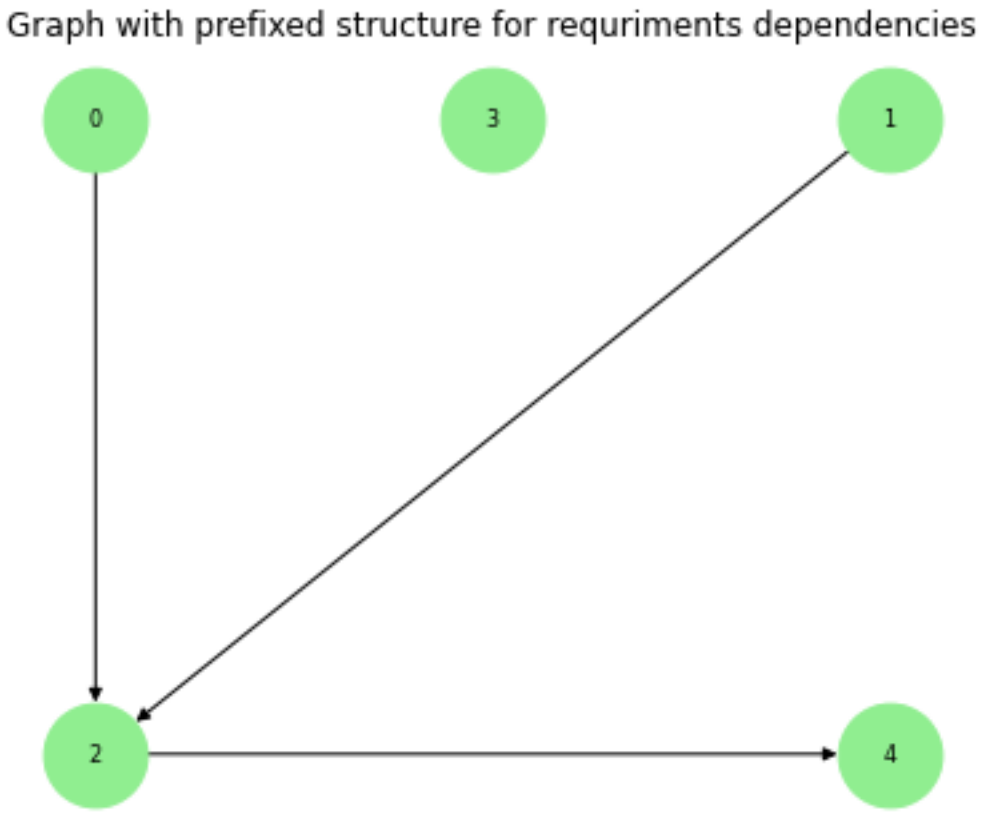
do

2. Learning
-- If X does not have any parents in graph structure, then learn
its marginal probability
-- If X does have parents in graph structure, learn Conditional:
P(X| all Y in parents(X)==0) In the example above, P(2| 0==0,1==0).
Thus, we only need to learn P(X|parents(X)) from requirements
whose parents Y are not selected.
That is, we do not need P(X | any parents(X)==1), just the
all parents(X)==0 case.
This means that conditional probability can be stored in a
unidimensional array,
using the same array to store either marginal or conditional
probability for each X.

3. Sampling
-- In the case of requirements without parents in graph structure,
use learned marginal probability

```
    -- In any Y in parents(X) is set to 1, then X=1,
    else use P(X|parents(X)==0)

while(!stop_criterion)
```



Ancestral order: [3,0,1,2,4]

a) Sampling of first generation: In the dependencies structure shown above, when sampling the first set of solutions, the requirement in each solution is selected given the following probabilites:

P(3) = 1/5

P(0) = 1/5

P(1) = 1/5

P(2) = 1 if requirement 0 or 1 has been selected in current solution; 1/5 otherwise.

P(4) = 1 if requirement 2 has been selected in current soluction; 1/5 otherwise.

Let us assume that we sample 6 individuals with the following result:

solutions =

```
        [ [0 0 1 0 1]
```

```
  [1 0 1 1 1]

  [0 1 1 0 1]

  [0 0 0 0 0]

  [0 1 1 0 1]

  [0 1 0 1 1]

  [0 0 0 1 0]
]
```

An impossible sampled individual would be, for example: [0 1 0 0 1] because requirement 2 should be selected since requirement 1 is. Thus, the dependencies graph structured is respected.

The whole population is evaluated, and the local NDS in current iteration set is identified. Let us assume this NDS:

nds_local =

```
[
 [1 0 1 1 1]

 [0 0 1 0 1]

 [0 1 0 1 1]

 [0 0 0 1 0]
]
```

nds_global = nds_local

b) Learning This step consists in updating the sampling probability of each requirement from the nds_local population.

P(0) = 1/4

P(1) = 1/4

P(2) = P_nds_local(2|requirement_0=0 and requirement_1=0) = 1/2

P(3) = 2/4

P(4) = P_nds_local(4|requirement_0=0) = 2/3

Thus, our probabilities vector is now:

probabilities =

```
[0.25, 0.25, 0.5, 0.5, 0.66]
```

Sidenote about learning

we could have special cases in which a requirement is never selected. For instance, let us imagine this local NDS:

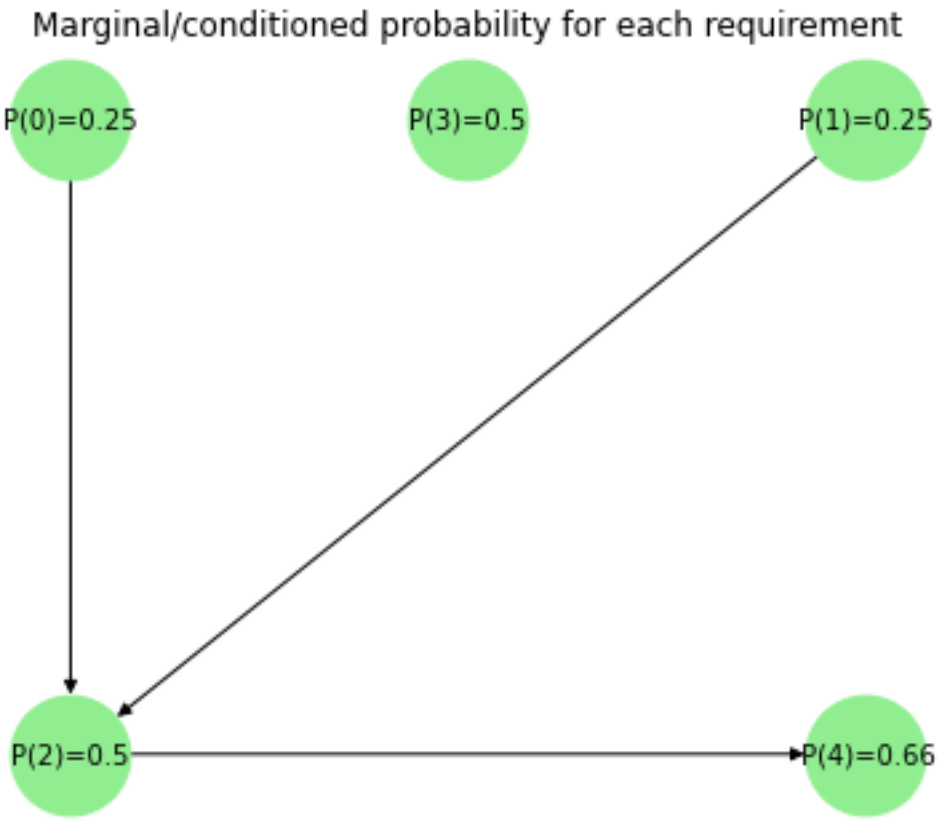
nds_local =

```
[
 [1 0 1 1 1]

 [0 0 1 0 1]

 [0 0 0 1 1]
]
```

P(1) = 0, so sampling of new individuals would not select requirement 1 anymore. In this case, we keep P(1) as its previous stored probability. Another possible alleaviations to this cases could be learning P(1) from the global NDS, or using some smooth.



c) Sampling Each new individual is sampled given the Ancestral order: [3,0,1,2,4], and using the following probabilities:

P(3) = 0.5

P(0) = 0.25

P(1) = 0.25

P(2) = 1 if requirement 0 or 1 is selected. 0.5 otherwise.

P(4) = 1 if requirement 2 is selected. 0.66 otherwise.

Let us assume that we sample 6 individuals with the following result:

new_solutions =

```
[ [0 0 0 1 1]

  [0 0 1 1 1]

  [1 1 1 0 1]

  [0 0 1 1 1]

  [0 0 0 1 0]

  [0 0 0 0 1]
]
```

We evaluate the solutions to: - update the global NDS given the new_solution population - set local NDS to the nds found in new_soluction population

Repeat Learning+Sampling until stop criterion.

0.3 3. Search of the best hyperparameters configuration for each algorithm.

3.1 Best configuration for: FEDA

```
These are the different values used to set hyperparameters in FEDA, for each
dataset:
population_length: [ 100  200  500  700 1000]
max_generations: [ 50 100 200 300 400]
max_evaluations: [0]
selection_scheme: ['nds']
In total, 25 configuration per dataset.

Counts of best configurations found in 17 datasets. Please notice that those
with less than the  maximum possible #iterations or #solutions_per_iteration
have converged sooner, and they obtain the same HV with higher configurations,
thus the least configuration possible is shown. In conclusion: 'best
configuration' can be interpreted as 'minimum configuration to converge'.
```

	population_length	max_generations	max_evaluations	selection_scheme
0	1000	300	0	nds
1	1000	50	0	nds
2	1000	400	0	nds
3	1000	200	0	nds

```
4              1000             100                0              nds

                   datasets                                                HV  \
0  [a1, a2, c2, c3, d2, d4]  [0.8728, 0.9435, 0.932, 0.9061, 0.8702, 0.808]
1      [a3, a4, d1, d5, d7]          [0.8244, 0.8047, 0.9126, 0.812, 0.781]
2                  [c1, c4]                                 [0.8999, 0.8521]
3                  [d3, p1]                                 [0.8099, 0.8789]
4                  [d6, p2]                                 [0.8011, 0.7891]

    wins
0 6.0000
1 5.0000
2 2.0000
3 2.0000
4 2.0000


Best hyperparameter configuration for FEDA is:
population_length:1000
max_generations:300
max_evaluations:0
selection_scheme:nds
```

3.2 Best configuration for: GRASP It takes too long. In the few datasets in which results are available, GRASP's HV is better than all algorithms. However, in one week experiments did not finish for datasets with a large number of PBIs.

3.3 Best configuration for: GeneticNDS

```
These are the different values used to set hyperparameters in GeneticNDS, for
each dataset:
population_length: [ 100  200  500  700 1000]
max_generations: [ 50 100 200 300 400]
selection_candidates: [2]
crossover_prob: [0.8]
mutation_prob: [0.1 0.3]
mutation: ['flip1bit']
replacement: ['elitismnds']
In total, 50 configuration per dataset.

Counts of best configurations found in 17 datasets. Please notice that those
with less than the  maximum possible #iterations or #solutions_per_iteration
have converged sooner, and they obtain the same HV with higher configurations,
thus the least configuration possible is shown. In conclusion: 'best
configuration' can be interpreted as 'minimum configuration to converge'.

  population_length max_generations selection_candidates crossover_prob  \
0              1000              50                    2            0.8
1              1000             300                    2            0.8
```

```
2              1000             400                    2            0.8
3              1000             100                    2            0.8
4              1000             200                    2            0.8
5              1000             400                    2            0.8

  mutation_prob  mutation replacement                 datasets  \
0           0.3  flip1bit  elitismnds                 [a1, d1]
1           0.3  flip1bit  elitismnds              [a2, c4, d7]
2           0.3  flip1bit  elitismnds  [a3, a4, c3, d4, d5, d6]
3           0.3  flip1bit  elitismnds                 [c1, p2]
4           0.3  flip1bit  elitismnds              [c2, d2, d3]
5           0.1  flip1bit  elitismnds                      [p1]

                                              HV   wins
0                               [0.8163, 0.7937] 2.0000
1                          [0.8916, 0.724, 0.6546] 3.0000
2  [0.7022, 0.6978, 0.808, 0.6969, 0.7019, 0.6686] 6.0000
3                                [0.8493, 0.685] 2.0000
4                        [0.8285, 0.7663, 0.7074] 3.0000
5                                      [0.8837] 1.0000


Best hyperparameter configuration for GeneticNDS is:
population_length:1000
max_generations:400
selection_candidates:2
crossover_prob:0.8
mutation_prob:0.3
mutation:flip1bit
replacement:elitismnds
```

3.4 Best configuration for: UMDA

```
These are the different values used to set hyperparameters in UMDA, for each
dataset:
population_length: [ 100  200  500  700 1000]
max_generations: [ 50 100 200 300 400]
selection_scheme: ['nds']
replacement_scheme: ['elitism']
In total, 25 configuration per dataset.

Counts of best configurations found in 17 datasets. Please notice that those
with less than the  maximum possible #iterations or #solutions_per_iteration
have converged sooner, and they obtain the same HV with higher configurations,
thus the least configuration possible is shown. In conclusion: 'best
configuration' can be interpreted as 'minimum configuration to converge'.

  population_length max_generations selection_scheme replacement_scheme  \
0              1000              50              nds            elitism
```

```
1              1000             400              nds            elitism
2              1000             300              nds            elitism
3              1000             200              nds            elitism
4              1000             100              nds            elitism

                   datasets                                                HV  \
0  [a1, a3, c3, c4, d3, p1]  [0.8639, 0.8031, 0.8779, 0.8204, 0.7945, 0.8783]
1              [a2, d2, d7]                            [0.9303, 0.831, 0.7516]
2              [a4, c1, d1]                            [0.7942, 0.8819, 0.865]
3                  [c2, d4]                                    [0.8631, 0.802]
4              [d5, d6, p2]                          [0.7844, 0.7648, 0.7713]

    wins
0 6.0000
1 3.0000
2 3.0000
3 2.0000
4 3.0000


Best hyperparameter configuration for UMDA is:
population_length:1000
max_generations:50
selection_scheme:nds
replacement_scheme:elitism
```

3.5 Best configuration for: PBIL

```
These are the different values used to set hyperparameters in PBIL, for each
dataset:
population_length: [ 100  200  500  700 1000]
max_generations: [ 50 100 200 300 400]
max_evaluations: [0]
learning_rate: [0.1]
mutation_prob: [0.1]
mutation_shift: [0.1]
In total, 25 configuration per dataset.

Counts of best configurations found in 17 datasets. Please notice that those
with less than the  maximum possible #iterations or #solutions_per_iteration
have converged sooner, and they obtain the same HV with higher configurations,
thus the least configuration possible is shown. In conclusion: 'best
configuration' can be interpreted as 'minimum configuration to converge'.

  population_length max_generations max_evaluations learning_rate  \
0         1000.0000        400.0000          0.0000        0.1000
1          700.0000        400.0000          0.0000        0.1000
2          200.0000        300.0000          0.0000        0.1000
```

```
  mutation_prob mutation_shift  \
0        0.1000         0.1000
1        0.1000         0.1000
2        0.1000         0.1000

                                            datasets  \
0  [a1, c1, c2, c3, c4, d1, d2, d3, d4, d5, d6, d…
1                                       [a2, a3, a4]
2                                               [p1]

                                                  HV    wins
0  [0.8968, 0.903, 0.9421, 0.7414, 0.7336, 0.9033… 13.0000
1                          [0.9736, 0.6568, 0.6583]  3.0000
2                                          [0.8948]  1.0000


Best hyperparameter configuration for PBIL is:
population_length:1000.0000
max_generations:400.0000
max_evaluations:0.0000
learning_rate:0.1000
mutation_prob:0.1000
mutation_shift:0.1000
```

3.6 Best configuracion for: MIMIC ---> pendiente de terminar los experimentos.

```
These are the different values used to set hyperparameters in MIMIC, for each
dataset:
population_length: [ 100  200  500  700 1000]
max_generations: [ 50 100 200 300 400]
max_evaluations: [0]
selection_scheme: ['nds']
selected_individuals: [ 50 100]
In total, 50 configuration per dataset.

Counts of best configurations found in 23 datasets. Please notice that those
with less than the  maximum possible #iterations or #solutions_per_iteration
have converged sooner, and they obtain the same HV with higher configurations,
thus the least configuration possible is shown. In conclusion: 'best
configuration' can be interpreted as 'minimum configuration to converge'.

  population_length max_generations max_evaluations selection_scheme  \
0              1000             200               0              nds
1              1000             300               0              nds
2              1000             400               0              nds
3              1000             100               0              nds
4              1000             200               0              nds
5              1000              50               0              nds
```

```
  selected_individuals                                        datasets  \
0                   50                                  [a1, e2, e5, e6]
1                   50                                  [a2, c1, d3, d4]
2                   50  [a3, a4, c2, c3, c4, d1, d2, d5, d6, d7, p2]
3                   50                                       [e1, e4]
4                  100                                           [e3]
5                   50                                           [p1]

                                          HV    wins
0                  [0.9106, 0.8031, 0.7809, 0.7826]  4.0000
1                  [0.9701, 0.9313, 0.8293, 0.8286]  4.0000
2  [0.8458, 0.8355, 0.9412, 0.9203, 0.8679, 0.912… 11.0000
3                                [0.7979, 0.8093]  2.0000
4                                        [0.7831]  1.0000
5                                        [0.9134]  1.0000


Best hyperparameter configuration for MIMIC is:
population_length:1000
max_generations:400
max_evaluations:0
selection_scheme:nds
selected_individuals:50
```

All algorithms find their best results when using a Population Size = 1000, the maximum value among the 5 given for this hyperparameter.

Respect to the number of generations, all algorithms but UMDA are slow to converge, needing the maximum (400) almost the maximum (300 in FEDA) value among the possible values. UMDA seems to converge really soon, since in 6 out of 17 datasets it finds its best results with just 50 generations, in both agile and classic projects.

0.4 4. Pareto plots for each dataset, setting each algorithm with its best configuration found (wins among all datasets).

Given the most frequently best configuration (over all datasets), we plot the pareto for each dataset given that configuration. That is, the configuration for a given algorithm is the same acrooss all datasets, concretely the one which performed the best more times (more wins).

We show a plot for each dataset. In each plot, for each algorithm, we show the pareto front found in all the executions (commonly 30). Since the solutions subset size is commonly 10, thus for each algorithm we plot 300 points. Since each algorithm has 30 paretos, please note that such paretos are not non-dominated among them, which can be seen in the shapes they create.

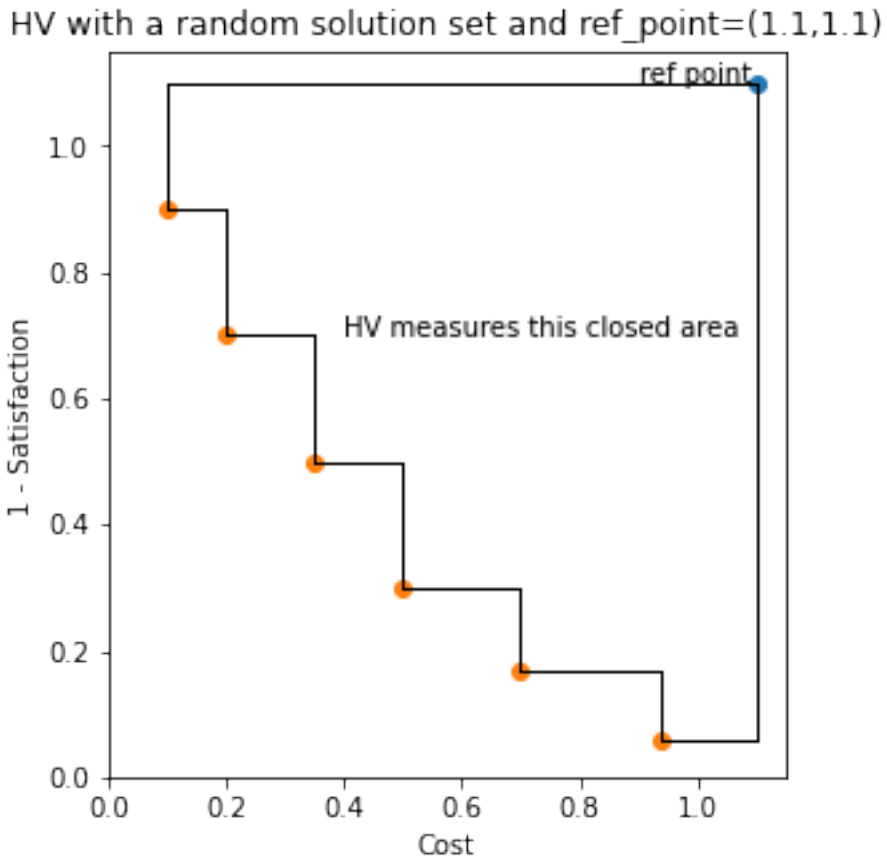
4.1 Metrics We show the average results over 30 executions for each algorithm. Each execution produces a NDS, from wich we keep the subset of 10 solutions which maximize HV, as suggested in 'Difficulties in Fair Performance Comparison of Multi-Objective Evolutionary Algorithms'. Such a subset is constructed by following an incremental forward greedy search.

Metrics shown are: 4 Quality Indiciators (HV, gd+, UNFR, spread), time and cardinality of the

global NDS found by the algorithm during execution.

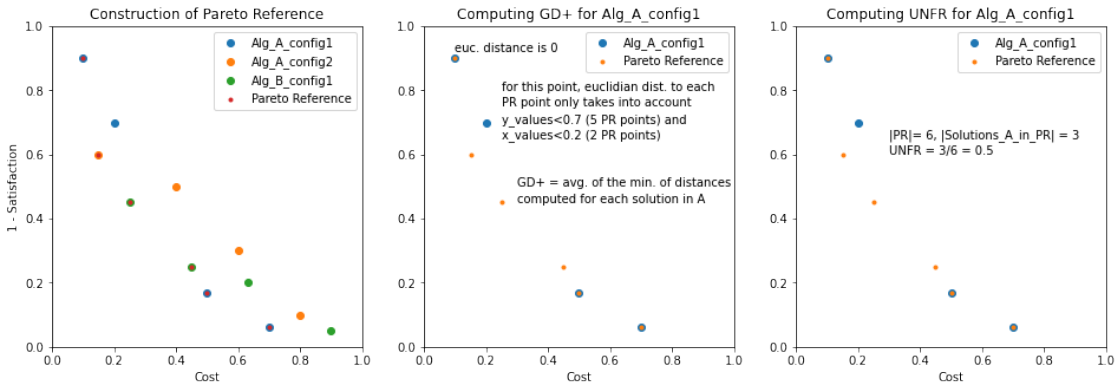
- HV (Hypervolume): this is the most widely used metric to assess paretos solutions for multi-objective problems and, concretly, in SBSE. It summarizes the four aspects of a solution set (convergence, spread, uniformity and cardinality); that is, this metric can be used as a compliance metric representing how good a Pareto front is. HV tends to be greater as knee points in the pareto are neareer the optimal point (0,0), thus it is preferable when Decision Makers prefer balanced solutions. In order to compute it, a reference point is needed, and this should be the same for all algorithms under comparison. Following the results and suggestions in ("How to evaluate solutions in pareto-based search-bases software engin."), we set the following reference point in our bi-objective problem: $ref_x = nadir_x + range_x/10$ $ref_y = nadir_y + range_y/10$ The nadir point is the worst point found by algorithms during search. Since we normalize both cost and satisfaction, our worst point is 1 for both metrics (satisfaction is plotted as $1 - satisfaction$). Range is the difference between the best and worst point found. Best point is 0, so clearly the value of the reference point for both goals is $1+(1-0)/10 = 1.1$. HV is pareto compliance, so $HV_a > HV_b$ means that, visually, the pareto front of algorithm a dominates algorithm b. A great advantage of HV is that it does not need an ideal Pareto Reference, so its computation and fair comparison with other algorithm only needs a shared reference point which, thanks to goals normalization, is known a priori.

```
Text(0.4, 0.7, 'HV measures this closed area')
```



- GD+: General Distance (GD) covers the convergence aspect of the quality of solution set, measuring the Euclidian distance of such solution set to the ideal Pareto Reference. For each solution, its GD is the minimum of the distances to each point in the PF. In order to become GD compliant with Pareto dominance, GD+ enhances GD by measuring distances between points using only the goal coordinates which are superior in the Pareto Reference than those from the solutions set being measured. This metric is to be minimized, and a key point is the computation of the Pareto Reference, which needs to be done after execution of search algorithms. In our experiments, Pareto Reference is constructed by finding the Non Dominated Solutions set among all solutions sets found by all algorithm, under all hyperparameters configurations explained in Section 2.
- UNFR: Unique Non Dominated Front Ratio. It measures the ratio of solutio points in the PR which belong to the solution set of the evaluated algorithm. That is, it measures the contribution (from to 0 to 1) of an algorithm to the PR. Of curse, a point in the PR might be present in the solution sets of several algorithms. In our case, since the PR is constructed from such a large number of algorithms and configuration combinations, it presents a high cardinality of solutions. Furthermore, since each algorithm is evaluated using only using a selected subset of 10 points from its solution set, the UNFR value for each algorithm tends to be quite low, and the maximum possible is never 1, since the PR contais further more than 10 points. Anyway, greater UNFR values is desiderable.

```
Text(0.3, 0.6, 'UNFR = 3/6 = 0.5')
```

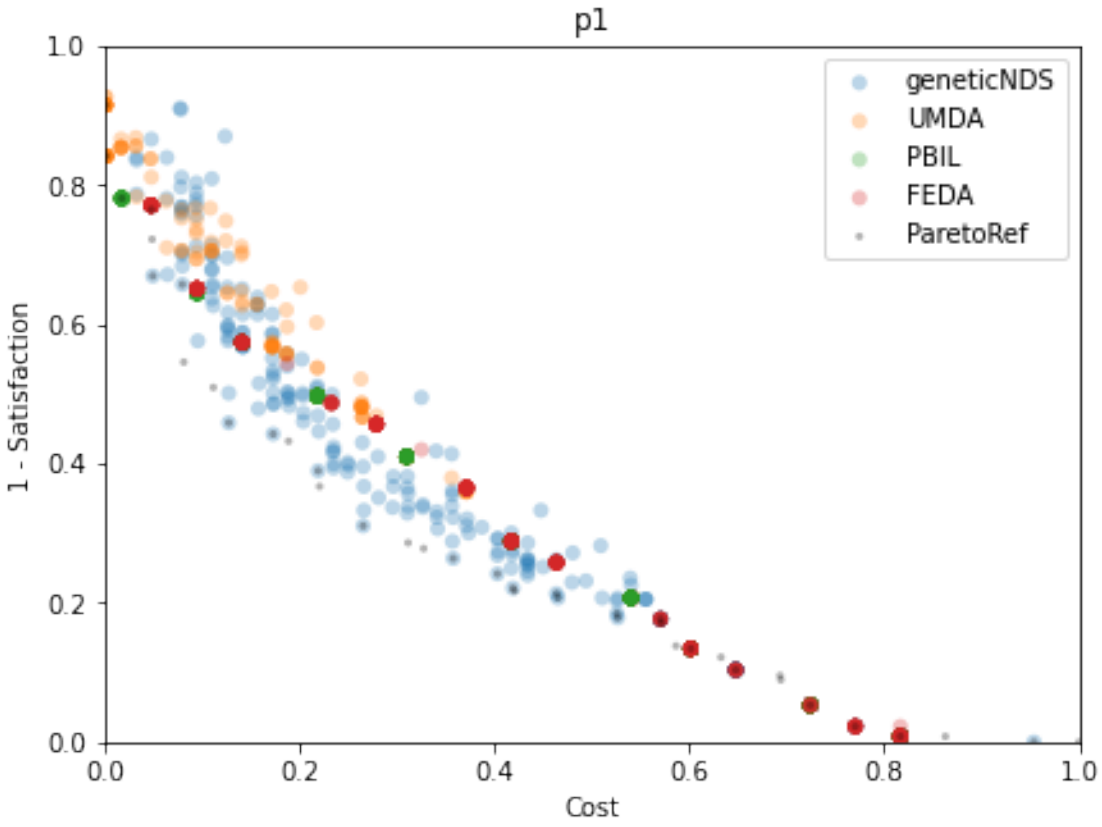


4.2 Results Here we show one plot pero datasets, containing the result of all algorithms with their best configuration. As mentioned before, each algorithm is run 30 times; afeter each execution, we keep a subset of the NDS constructed. This subset contains the 10 solutions from NDS which maximize HV in a forward greedy search over the NDS. Thus, for each algorithm, we plot 10*30=300 points. This way, it is easy to identify the common pareto shape relative to the algorithm. After each plot, all metrics are tabbed to show the average over the 30 executions in the dataset.

It is worth to mention that half of the execution time in algorithms is due to updating NDS after each iteration. Thus, although after each execution we select a subset of 10 solutions in the final

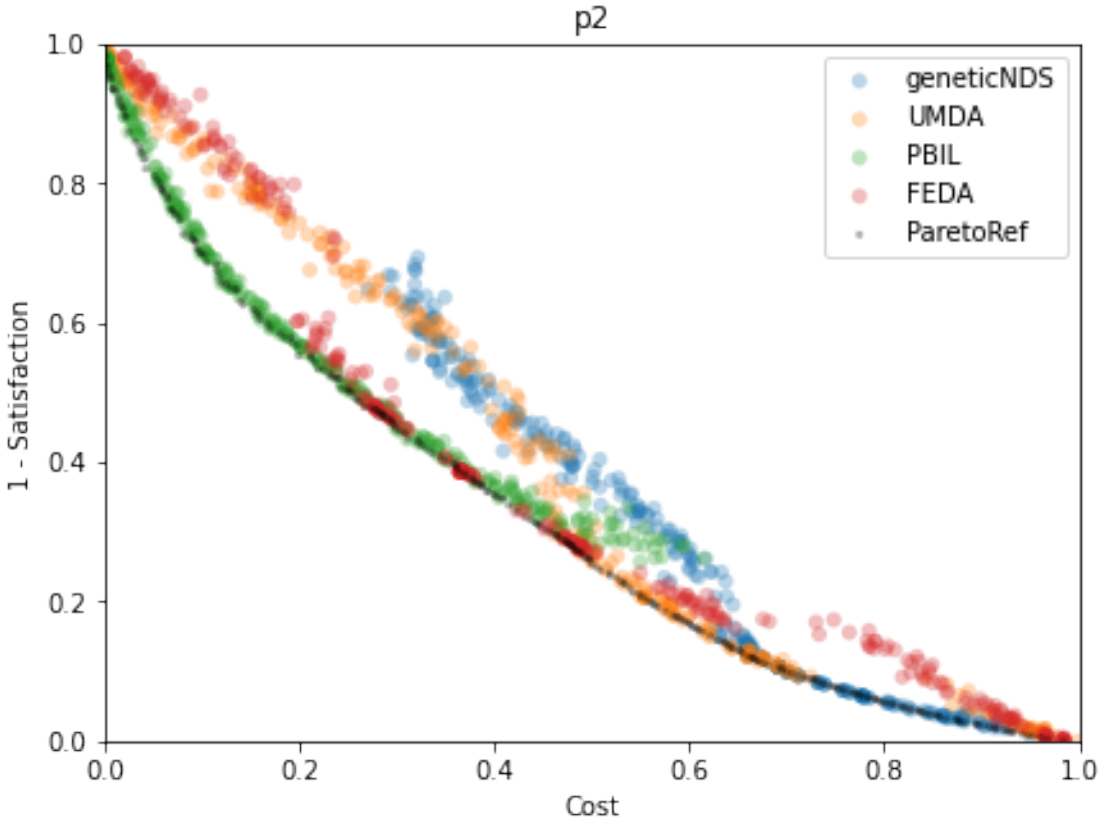
NDS, more time consuming algorithms are usually those which tend to develop a NDS with higher cardinality suring their search.

```
Pareto Reference has 40 points
Maximum UNFR possible is 10/40=0.2500
```



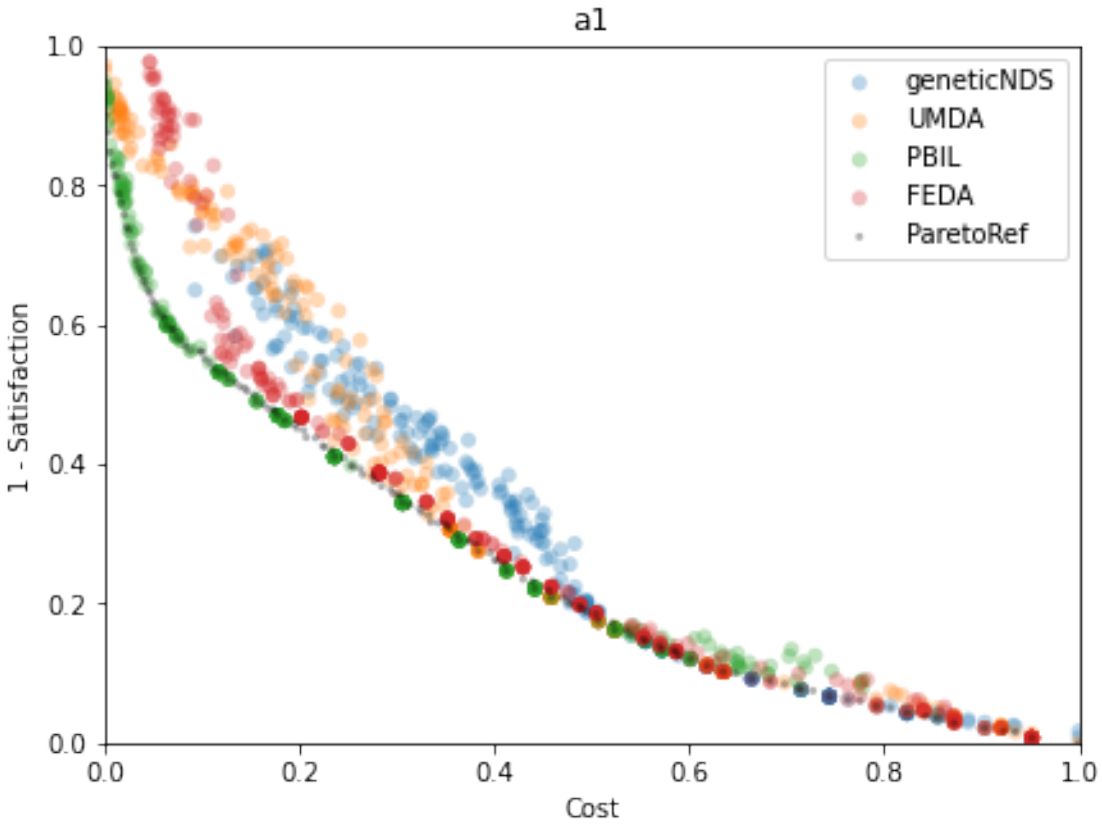
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8793	0.1142	0.0277	0.6453	800.8662	47.0667
1	UMDA	0.8783	0.0967	0.0374	0.6236	84.4473	34.8667
2	PBIL	0.8948	0.1000	0.0316	0.6038	452.2194	42.8333
3	FEDA	0.8787	0.0733	0.0373	0.6795	554.5325	31.9667

```
----------------------------------------------------------------
Pareto Reference has 283 points
Maximum UNFR possible is 10/283=0.0353
```



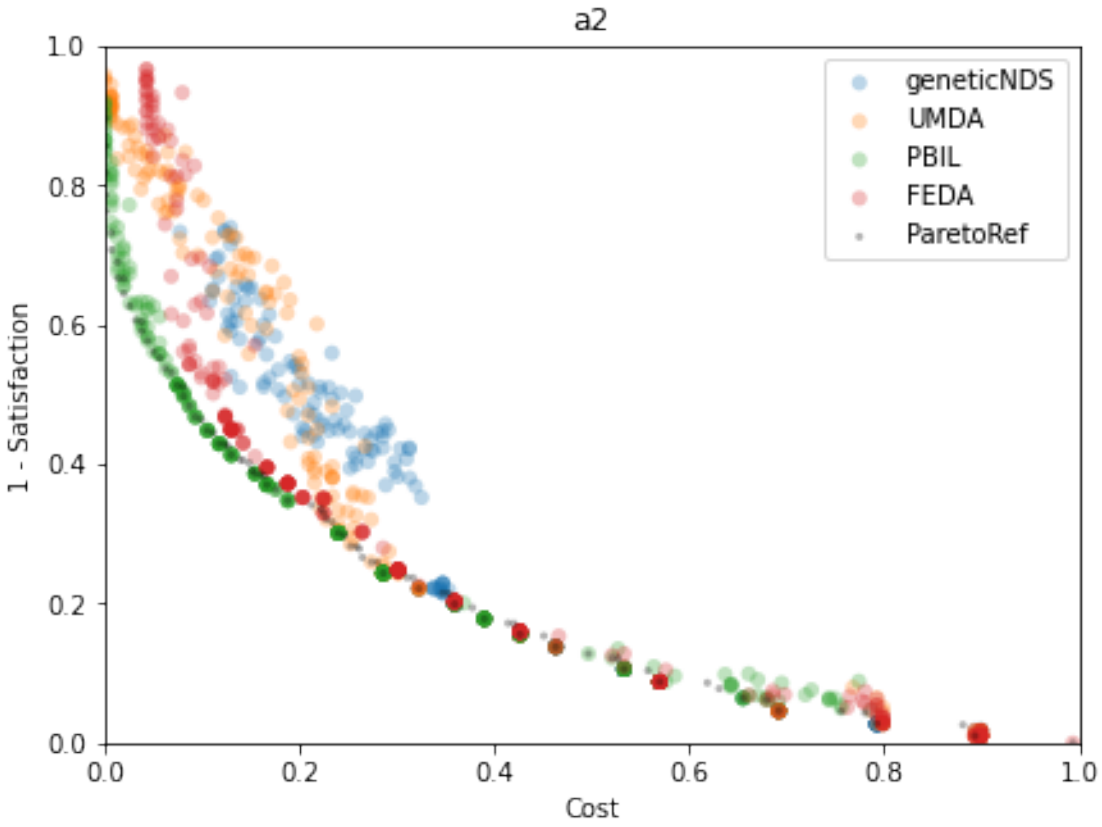
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.6844	0.0068	0.0561	0.6608	2213.7621	114.0000
1	UMDA	0.7712	0.0033	0.0458	0.5759	335.5445	151.1667
2	PBIL	0.7390	0.0045	0.0113	0.5571	1024.6246	107.2667
3	FEDA	0.7886	0.0025	0.0299	0.6018	3037.3493	217.2667

Pareto Reference has 142 points
Maximum UNFR possible is 10/142=0.0704



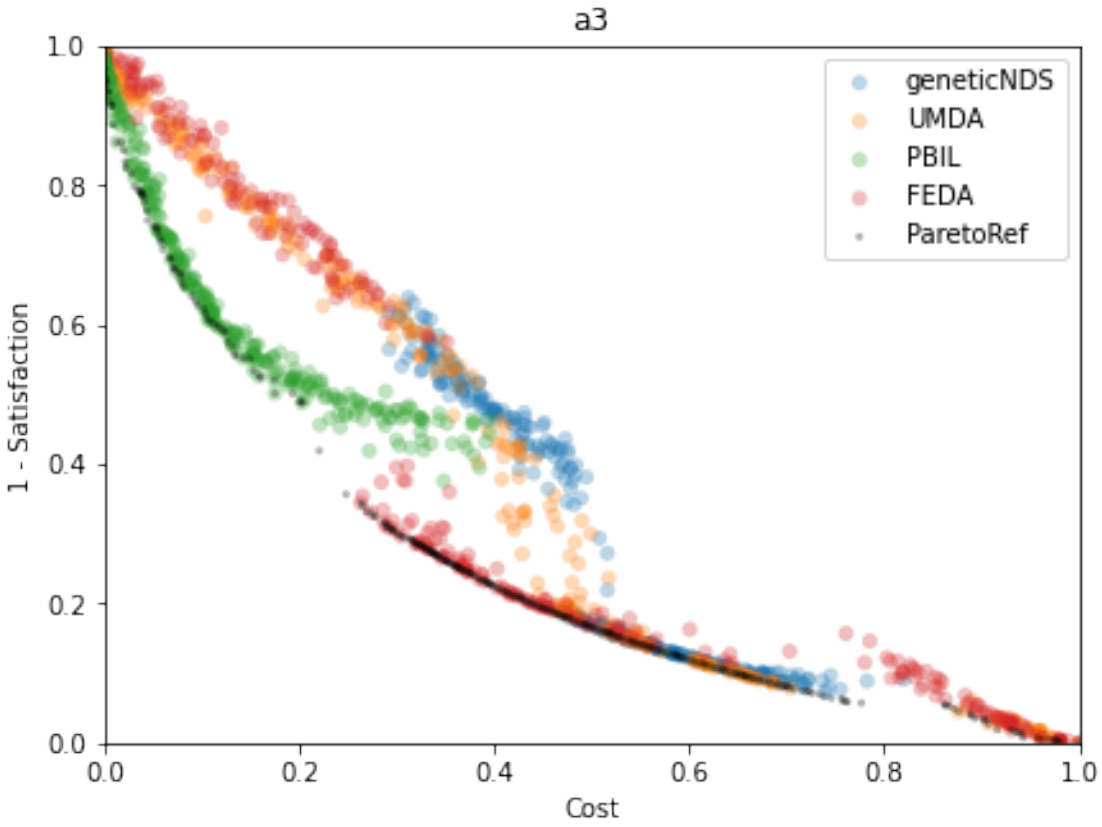
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8150	0.0279	0.0369	0.6649	1193.1049	64.5667
1	UMDA	0.8639	0.0279	0.0299	0.5756	174.7565	67.0000
2	PBIL	0.8968	0.0491	0.0031	0.6078	698.1781	110.7667
3	FEDA	0.8728	0.0120	0.0179	0.5812	1427.4162	94.9333

Pareto Reference has 86 points
Maximum UNFR possible is 10/86=0.1163



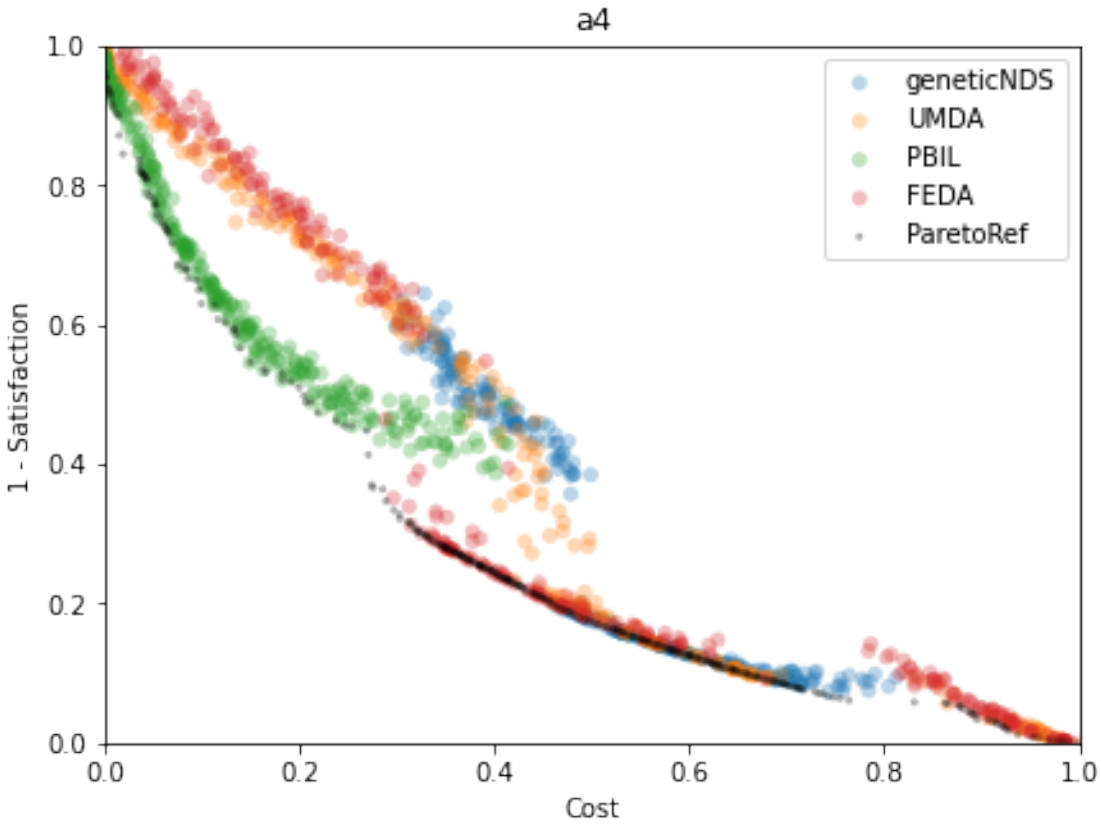
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8877	0.0717	0.0338	0.6420	806.1419	50.5000
1	UMDA	0.9290	0.0488	0.0296	0.5923	136.8646	47.1667
2	PBIL	0.9736	0.0795	0.0029	0.6097	548.6406	73.6000
3	FEDA	0.9435	0.0143	0.0187	0.6313	662.6650	50.9667

Pareto Reference has 306 points
Maximum UNFR possible is 10/306=0.0327



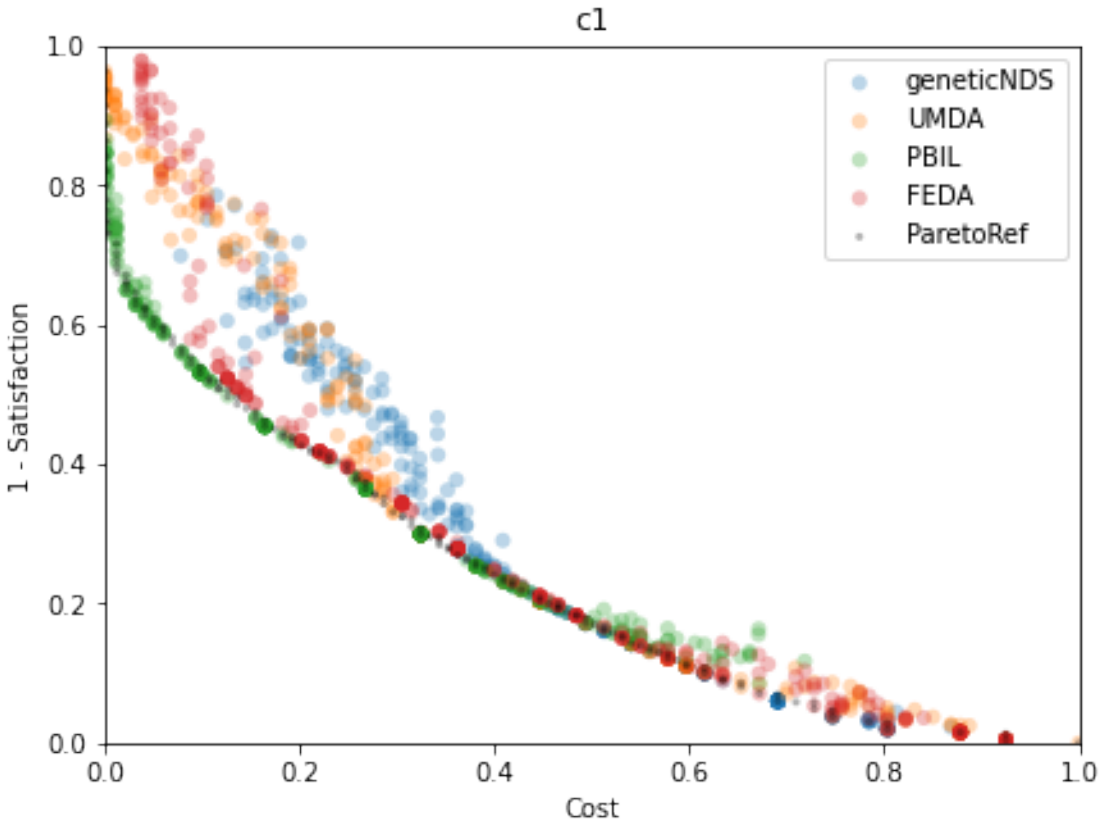
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7022	0.0020	0.0805	0.7520	1504.1177	86.9333
1	UMDA	0.8031	0.0032	0.0663	0.5809	344.3830	126.4667
2	PBIL	0.6558	0.0022	0.0275	0.5703	828.3615	62.8667
3	FEDA	0.8238	0.0041	0.0454	0.5921	2268.7561	153.9000

Pareto Reference has 305 points
Maximum UNFR possible is 10/305=0.0328



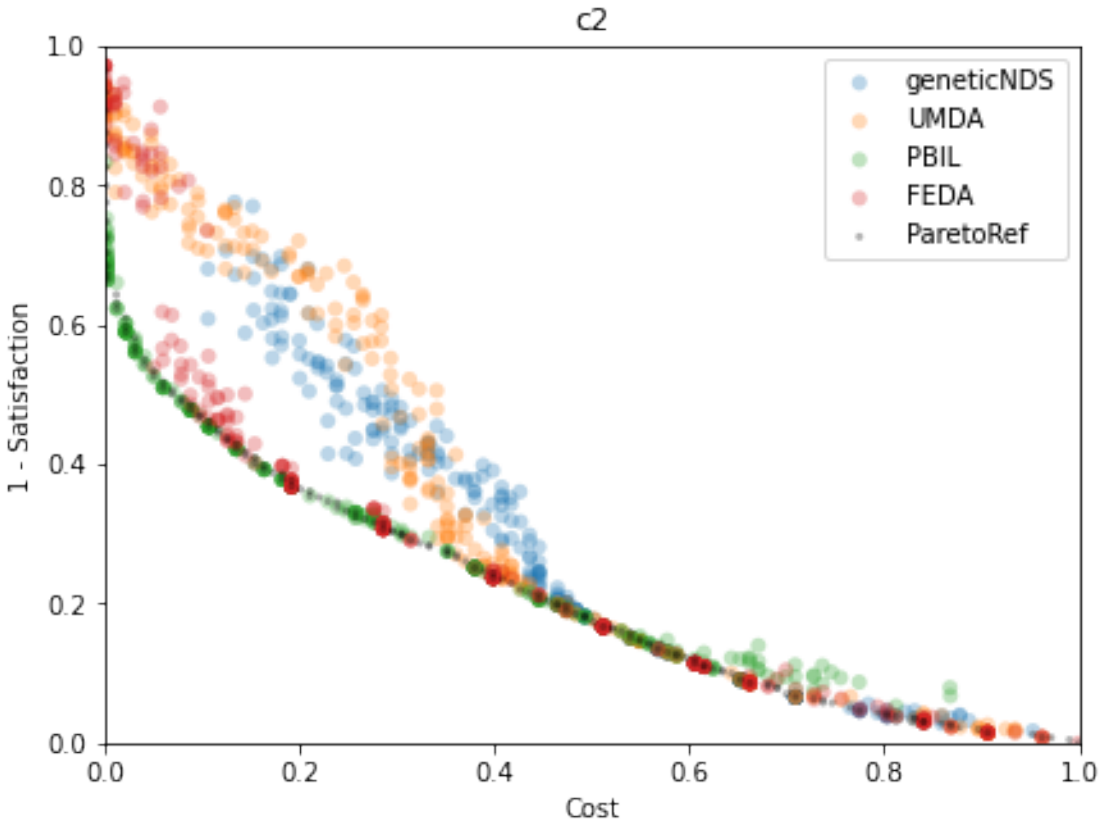
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.6978	0.0049	0.0637	0.7291	1761.2607	99.8333
1	UMDA	0.7923	0.0023	0.0636	0.5875	375.8887	143.3667
2	PBIL	0.6576	0.0013	0.0237	0.5685	874.9545	64.6333
3	FEDA	0.8022	0.0035	0.0493	0.6169	2775.3945	170.3000

Pareto Reference has 155 points
Maximum UNFR possible is 10/155=0.0645



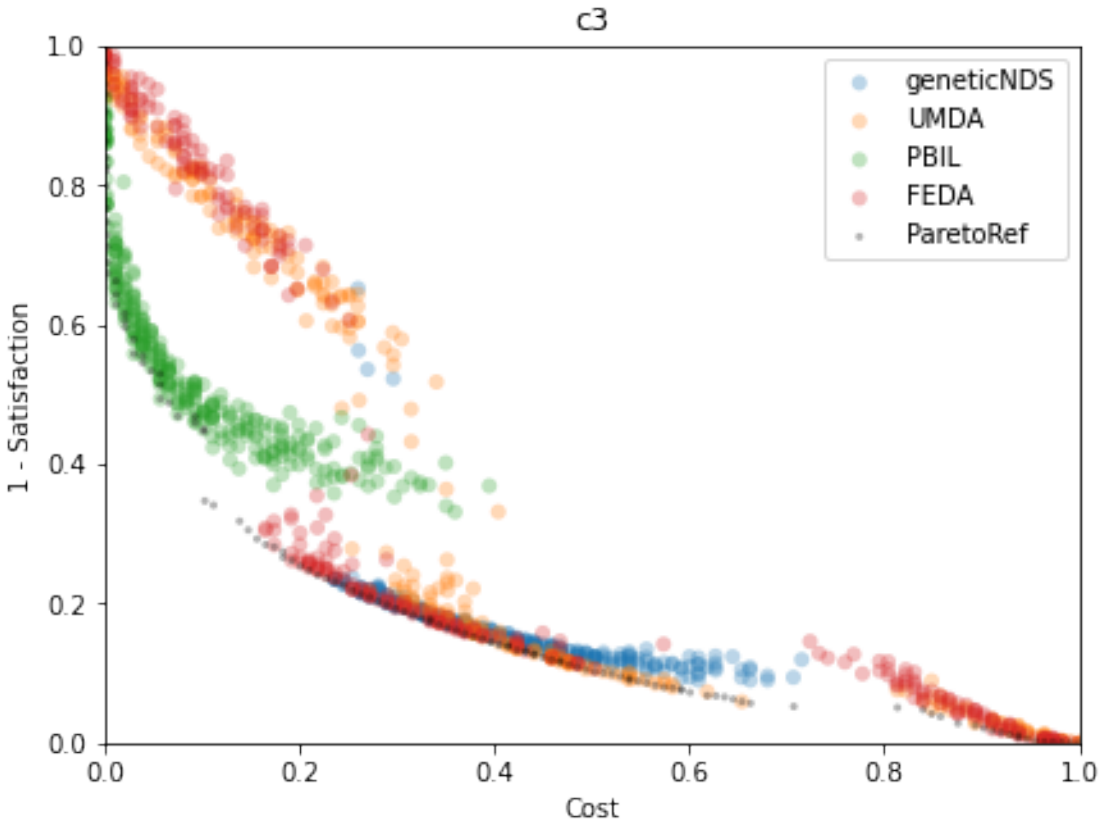
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8468	0.0318	0.0338	0.6418	1369.9759	86.2667
1	UMDA	0.8817	0.0273	0.0301	0.6068	223.7486	79.3000
2	PBIL	0.9030	0.0443	0.0037	0.6150	743.3746	105.7333
3	FEDA	0.8967	0.0157	0.0170	0.6054	1202.8190	90.1667

Pareto Reference has 162 points
Maximum UNFR possible is 10/162=0.0617



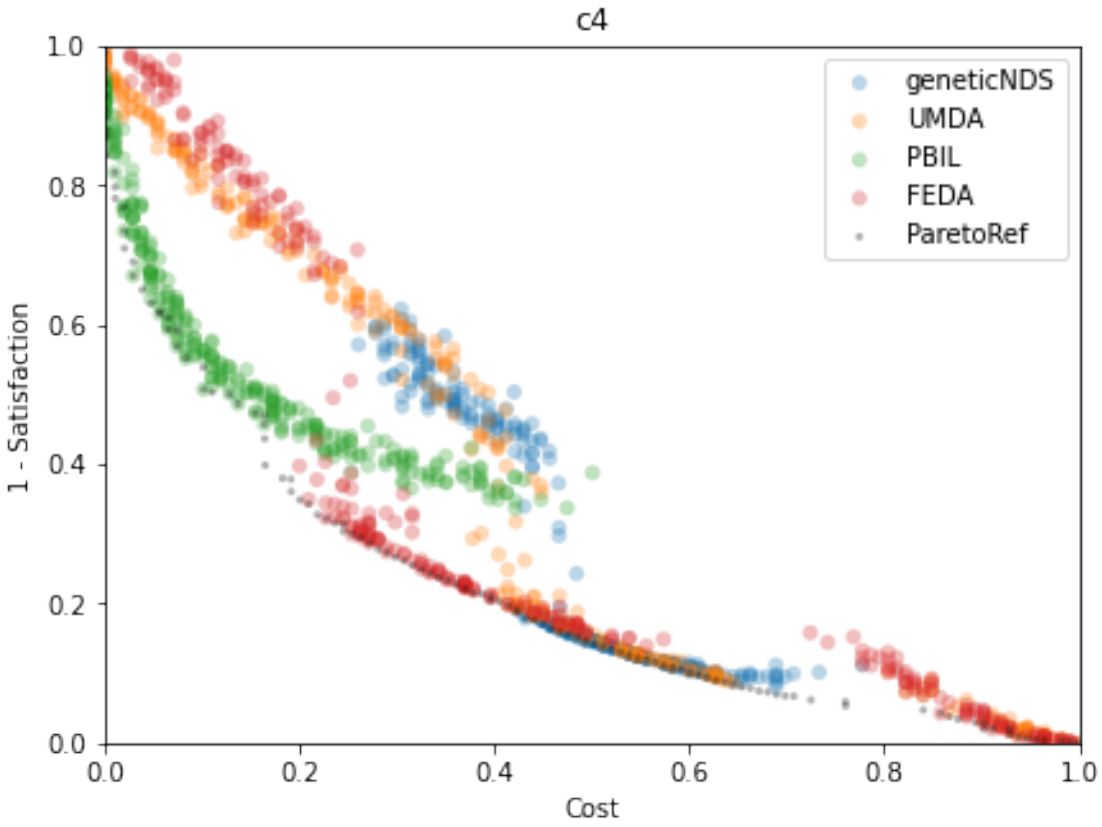
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8252	0.0249	0.0509	0.6707	1099.2675	81.3000
1	UMDA	0.8584	0.0198	0.0505	0.5627	185.9247	69.6333
2	PBIL	0.9421	0.0465	0.0030	0.6031	716.0132	103.4667
3	FEDA	0.9320	0.0383	0.0072	0.5564	1342.6053	98.3000

Pareto Reference has 126 points
Maximum UNFR possible is 10/126=0.0794



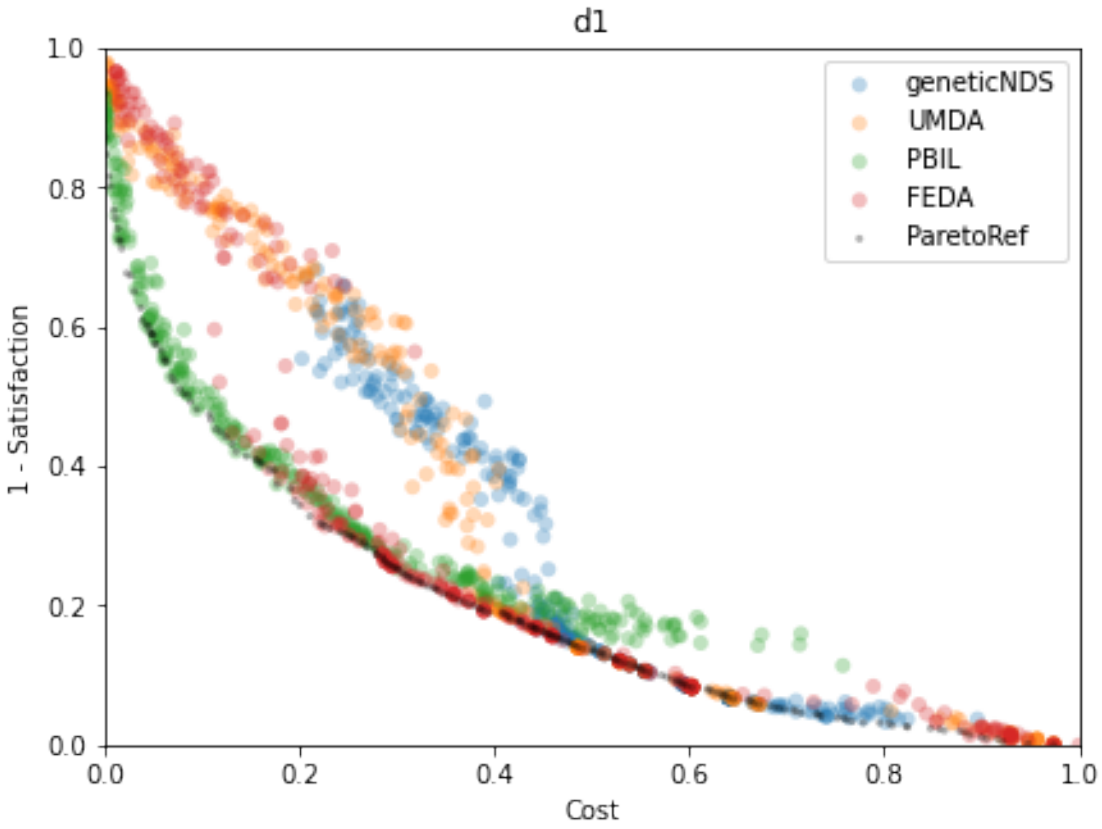
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.8080	0.0016	0.0173	0.7739	874.2606	38.1333
1	UMDA	0.8779	0.0037	0.0608	0.5699	197.7094	65.7333
2	PBIL	0.7414	0.0042	0.0391	0.6520	531.1763	36.9000
3	FEDA	0.9061	0.0050	0.0403	0.6381	1115.9604	57.4333

Pareto Reference has 120 points
Maximum UNFR possible is 10/120=0.0833



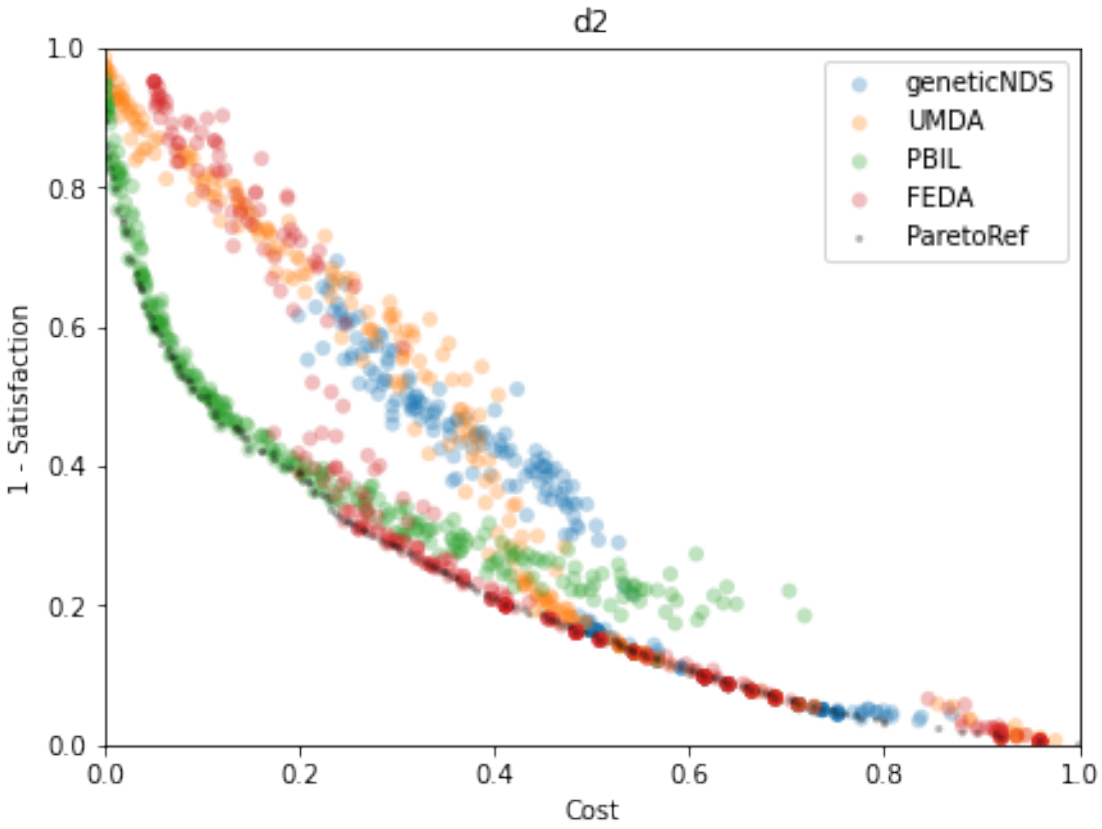
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7231	0.0025	0.0771	0.7743	820.1618	32.4667
1	UMDA	0.8204	0.0014	0.0751	0.5832	204.6724	61.0000
2	PBIL	0.7336	0.0047	0.0408	0.6020	656.5295	48.4333
3	FEDA	0.8497	0.0058	0.0474	0.6148	1172.4405	59.4667

Pareto Reference has 204 points
Maximum UNFR possible is 10/204=0.0490



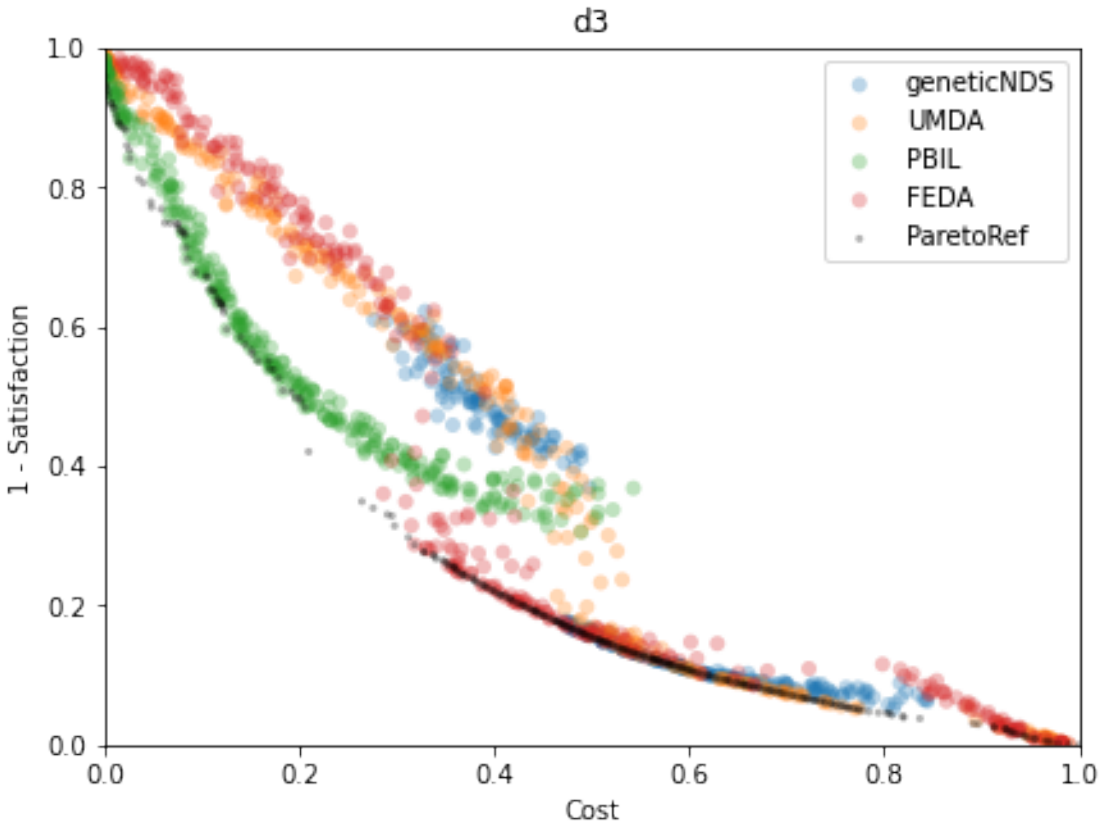
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7906	0.0132	0.0697	0.7176	1346.1592	74.6667
1	UMDA	0.8612	0.0101	0.0665	0.5579	239.4590	90.2667
2	PBIL	0.9033	0.0026	0.0201	0.6027	662.4540	72.0667
3	FEDA	0.9060	0.0132	0.0270	0.6000	1907.2975	121.5333

Pareto Reference has 167 points
Maximum UNFR possible is 10/167=0.0599



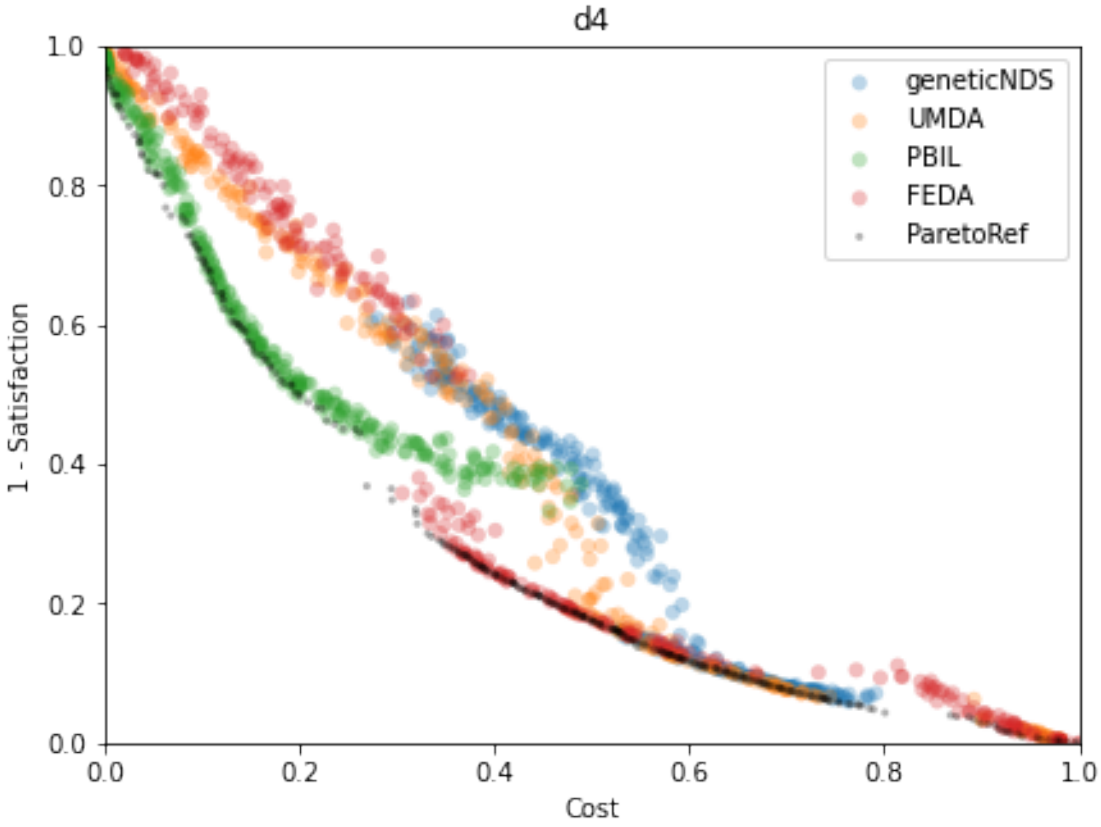
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7608	0.0220	0.0757	0.7147	939.1621	65.0000
1	UMDA	0.8304	0.0140	0.0732	0.5560	209.9717	71.2333
2	PBIL	0.8559	0.0044	0.0268	0.5932	671.5176	70.5667
3	FEDA	0.8702	0.0234	0.0337	0.6145	1414.7278	86.5000

Pareto Reference has 304 points
Maximum UNFR possible is 10/304=0.0329



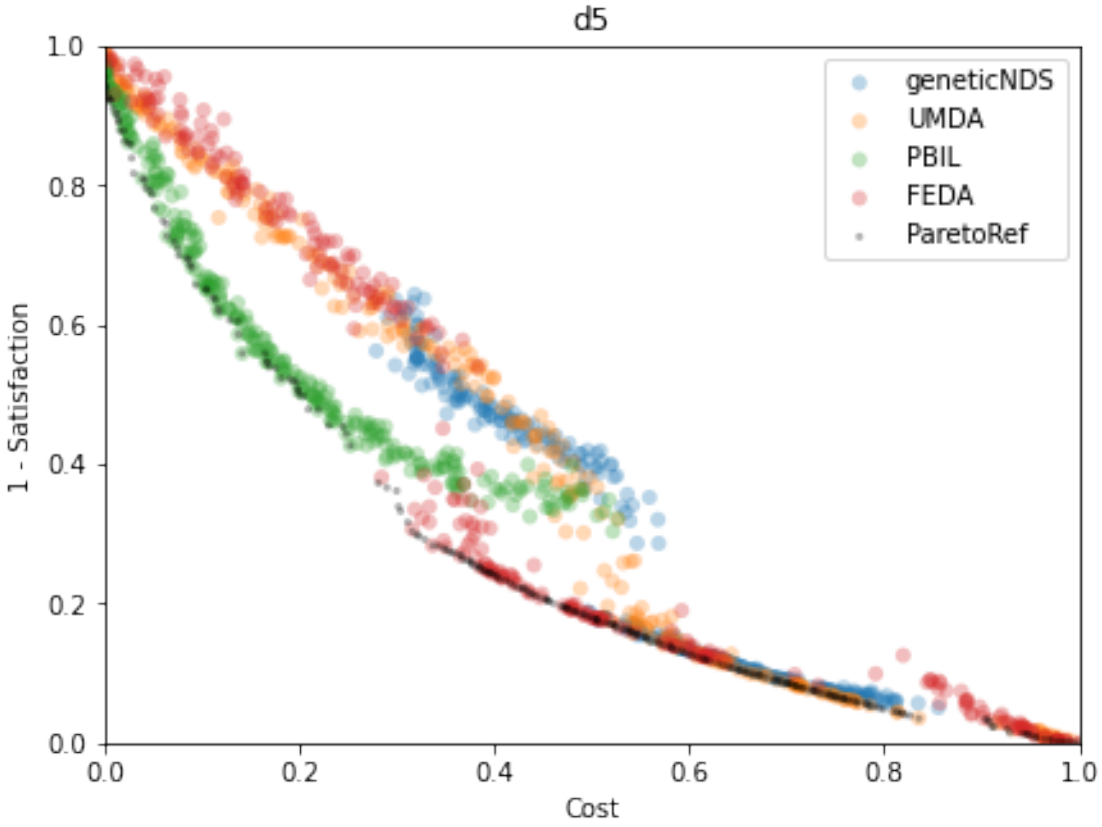
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7073	0.0065	0.0629	0.7581	1813.7526	86.8000
1	UMDA	0.7945	0.0035	0.0717	0.5627	338.0281	123.7667
2	PBIL	0.7197	0.0021	0.0388	0.5822	979.6848	71.1667
3	FEDA	0.8098	0.0041	0.0514	0.5935	2108.9927	141.5000

Pareto Reference has 294 points
Maximum UNFR possible is 10/294=0.0340



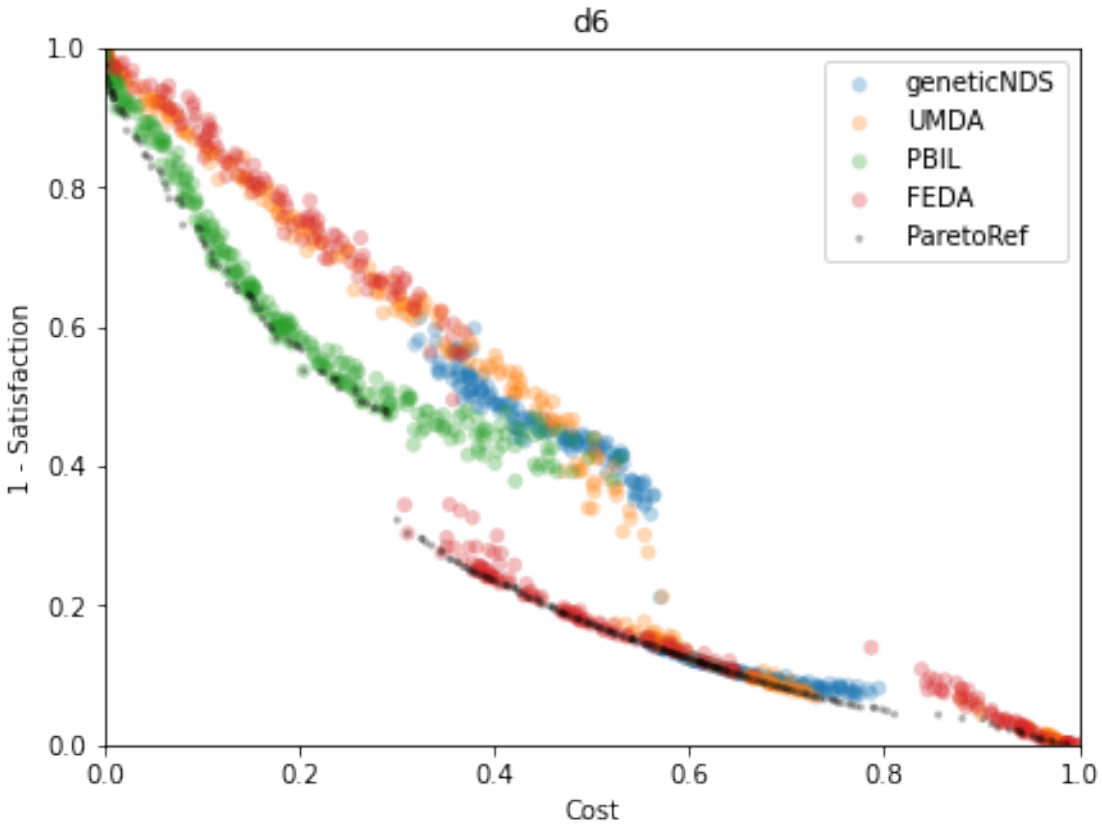
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.6969	0.0022	0.0820	0.7401	1775.7885	89.2667
1	UMDA	0.8006	0.0045	0.0558	0.5709	397.6439	147.8667
2	PBIL	0.6934	0.0015	0.0263	0.5751	1184.9404	79.5667
3	FEDA	0.8080	0.0027	0.0461	0.5963	3208.9304	200.3000

Pareto Reference has 263 points
Maximum UNFR possible is 10/263=0.0380



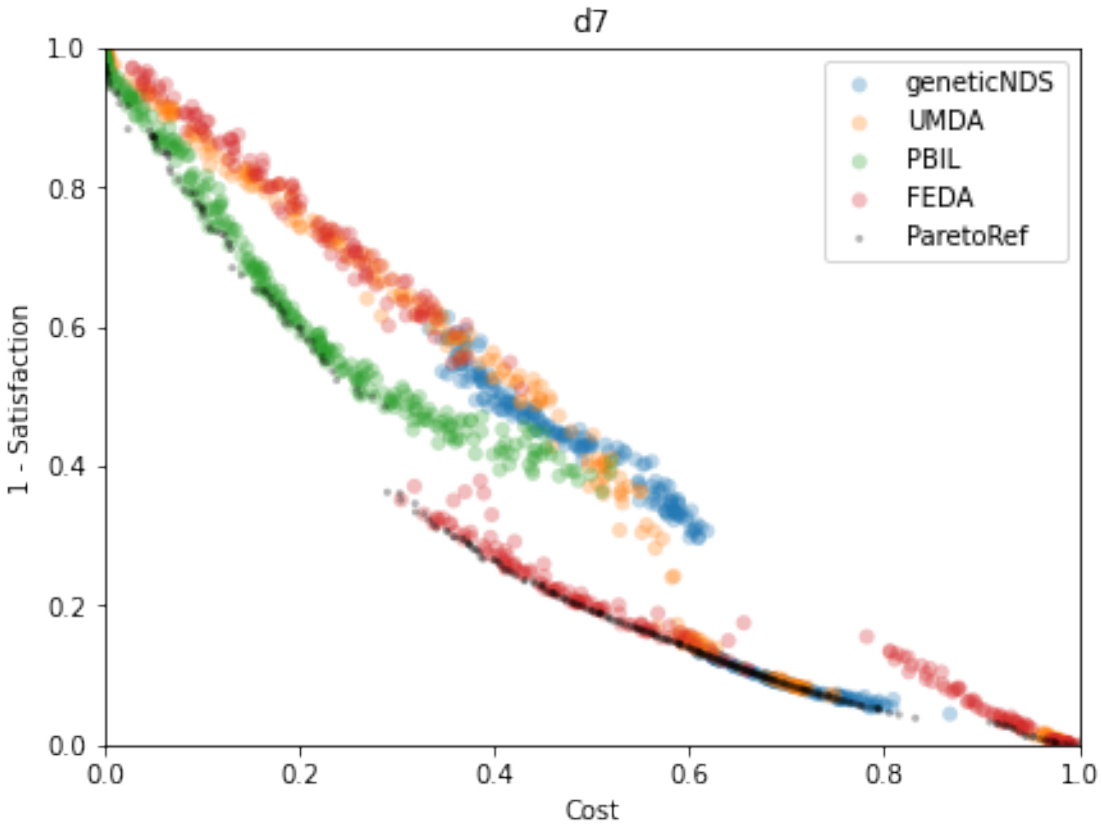
	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.7019	0.0004	0.0721	0.7259	1916.8350	94.4667
1	UMDA	0.7842	0.0035	0.0667	0.5800	328.0819	122.6667
2	PBIL	0.7197	0.0028	0.0307	0.5789	885.8085	67.7000
3	FEDA	0.8090	0.0038	0.0487	0.5953	2356.9579	149.2667

Pareto Reference has 277 points
Maximum UNFR possible is 10/277=0.0361



	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.6686	0.0030	0.0786	0.7611	1568.3957	78.3333
1	UMDA	0.7632	0.0014	0.0690	0.5703	359.8537	124.0667
2	PBIL	0.6441	0.0029	0.0327	0.5626	1200.1904	70.9667
3	FEDA	0.7980	0.0026	0.0429	0.6004	2437.1066	166.1000

Pareto Reference has 324 points
Maximum UNFR possible is 10/324=0.0309



	Method	HV	UNFR	gd+	spread	time(s)	|NDS|
0	geneticNDS	0.6545	0.0063	0.0896	0.7401	2144.0206	104.4333
1	UMDA	0.7503	0.0012	0.0657	0.5694	369.0416	148.0333
2	PBIL	0.6451	0.0024	0.0322	0.5723	1221.0485	80.3000
3	FEDA	0.7791	0.0019	0.0431	0.6063	3178.8110	192.3667

```
------------------------------------------------------------
Wins Counts:
{'geneticNDS': 0, 'UMDA': 0, 'PBIL': 5, 'FEDA': 12}
Wins in datasets:
{'geneticNDS': [], 'UMDA': [], 'PBIL': ['p1', 'a1', 'a2', 'c1', 'c2'], 'FEDA':
['p2', 'a3', 'a4', 'c3', 'c4', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7']}
```

Given the results, we see that PBIL behaves really well in datasets wich not a large number of requirements (aX and cX datasets). In the case dX datasets, with hundreds of requirements, FEDA obtains greater Hypervolumes than PBIL and the rest of algorithms. In some cases, FEDA obtains a very similar HV value than PBIL or UMDA; in such cases, in order to be sure that FEDA performs better we can take into account the UNFR value, which is pareto compliant, and when FEDA's HV is just slightly better, UNFR is clearly better than the other algorithm. \

A withdraw FEDA presents is that its execution time is much worse than the other algorithms. This is mostly due to the large number of Non Dominated Solutions it finds. \

Respect to gd+, FEDA is usually the second algorithm with best (lowest) mean general distance to the Pareto Reference, while PBIL commonly finds the closer solutions to the PR, although its HV covered is lower, as said, when the project presents hundreds of requirements.

0.4.1 5. Statistical tests of quality indicators

Antes de meterme en esto, a ver si veis alguna laguna en la experimentación. Por ejemplo me preocupa: - Ausencia de nsga-ii - qué hacer con GRASP - tiempos tan grandes por |NDS| y que ensombrece el tiempo real de learning+sampling - quitar algún dataset dX