Comparative study of constraint-based causal structure learning for Bayesian networks

Yannis Elrharbi-Fleury
Hector Kohler

November 2021

Sorbonne University
Master ANDROIDE

1 Introduction

This project aims to reproduce the results of Li, H.; Cabeli, V.; Sella, N.; and Isambert, H. 2019, *Constraint-based Causal Structure Learning with Consistent Separating Sets*, Advances in Neural Information Processing Systems.

We implemented several PC-derived algorithms, which learn a Bayesian Network's causal structure based on a given data set. Because conditional probabilities are unknown, independence relations must be inferred using a test such as the *Chi-squared test* (or the *G-test*). The latter's output is dependant on a threshold value α.

In this project, we sought to study the comparative robustness and performances of each algorithm based on the α value and the size of the data set used for the independence test.

2 PC-derived algorithms

The PC-derived algorithms rely on a given data set and on an independence test that will infer the independence relations of the variables. Therefore, all the algorithms presented in the following section are fallibles in the sense that there could be errors in the statistical tests, or other unobserved variables that could affect the relations of the variables.

2.1 PC

Given a complete graph, the PC-algorithm ([1]) consists of three steps.

Firstly, PC finds the graph skeleton and separating sets of removed edges. Given a data set, it tests the pairwise and conditional independence of each node (here with the *Chi-squared test*), an edge is removed if a pair of nodes is independent.

Secondly, the v-structures are oriented based on the computed separating sets. For each unshielded triple, if a variable is not in the other two's separating set, it is the target of a v-structure.

Lastly, the orientations are propagated. All the remaining edges are oriented so that no new v-structures are built.

Let us note that while being fairly safe, this algorithm is susceptible to fail a learning process. Indeed, it is possible to build a skeleton where orientations can not be propagated without creating a new v-structure.

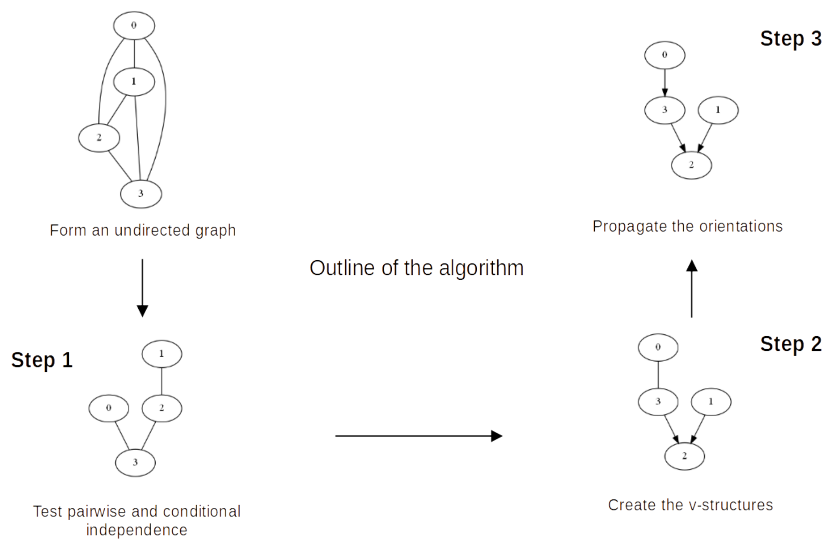

Figure 1: Outline of the PC algorithm from a complete graph and a given database

2.2 PC-stable

The previous implementation of the PC algorithm is order dependant. Indeed, in the skeleton learning phase, the edges are removed in an iterative fashion. Therefore, given the same data set, the output will depend on the lexicographic order of the graph's nodes.

A stable variant of the algorithm was proposed by Colombo and Maathuis ([2]), PC-stable. It consists of a simple change in the PC learning skeleton process : we stop adding to a pair's separating set the conditional independence if they are no longer adjacent in the graph.

2.3 PC-consistent separating sets

The main issue of iterative methods is that they lack robustness to sampling noise and can infer wrong conditional independences (false negatives). The uncovered separating sets may not remain consistent with the final graph ([3]).

The PC-consistent separating sets algorithm aims to reduce the number of false negatives for learnings on finite data sets (due to the slicing of available data when computing a conditional independence).

2.3.1 Skeleton inconsistency

A skeleton inconsistency, also known as a Type I inconsistency, corresponds to an independence relation $X \perp\!\!\!\perp Y|Z$ for which there is no path between X and Y going through Z. This inconsistency occurs in step 1 (figure 1) and is described in detail in [3].

2.3.2 Orientation inconsistency

An orientation inconsistency, also known as a Type II inconsistency ([3]), corresponds to an independence relation $X \perp\!\!\!\perp Y|Z$ such that Z is a common descendant of X and Y in the final graph.

2.3.3 Consistent separating-sets

To deal with either types of inconsistency, one can use a PC-derived algorithm that computes consistent separating-sets instead of base separating-sets ([3]). Consistent separating-sets are separating-sets with two additional conditions each dealing with one type of inconsistency. Given a graph G, the consistent separating set of X and Y are the variables Z among $adj(X)\backslash Y$ such that:

- There is a path between X and Y passing through Z in G (Type I inconsistency).
- Z is not a child of X in G (Type II inconsistency).

3 Implementation

The implementation of the algorithms was made in `Python` using the `PyAgrum` ([4]) library.

3.1 Algorithms

An algorithm is represented by a class, which posses a *learn* method. The latter takes as input a data set and learns a Bayesian network, this process is itself divided into several private methods.

This code structure enables us (and future users) to easily implement our own algorithms, or to modify what's already implemented through inheritance, by overloading the private methods that make the learning process. This is what we did for the implementation of PC-stable and the PC-css algorithms, which all inherit each other.

Let us note that, in order to highlight the lexicographic dependence of the basic PC implementation, we shuffle the list of edges that will be iterated upon during the learning process.

3.2 Benchmarks

A Benchmark is represented by a class, it takes as input the number of networks to generate and their parameters. Its methods allow it to generate and sample Bayesian networks, and to run user-defined tests on a given algorithm.

A Benchmark is able to save and load itself in a file to save the user time in computing his tests with different parameters. The file format is LZMA, which is slow in compression but fast in decompression (hence saving time during the loading phase).

We defined tests that allow to run a given algorithm on the benchmark. The former can be used on any algorithm that posses a `learn` and a `reset` method.

Defined tests consists in computing the Hamming and Skeleton scores of an algorithm on the benchmark, and in computing these scores while changing the α value.

A use example of the class is provided in the `main.py` file, which was used to compute our results.

4 Results

To reproduce the results of [3], we wanted to use the classical benchmarks *Insurance*, *Hepar2* and *Barley*. However, while our code allows us to define such benchmarks, we lack the computational power to generate them.

Therefore, the following results were computed on smaller networks of only 25 nodes, with 4 possible values for each variable. We distinguish three kind of networks : those with weak interactions (an average of 1 arc per node), medium interactions (1.6 arc per node) and strong interactions (2.2 arcs per node).

Hence we have 3 benchmarks, that consist of a 100 of such randomly generated networks. We emphasize on the fact that the latter were only generated and sampled once per benchmark, letting us test each algorithm on the same networks and samples.

The performance of the implementations were evaluated using a precision-recall curve. The values are defined as follows :

$$Precision = \frac{True\ positives}{True\ positives + False\ positives}\ Recall = \frac{True\ positives}{True\ positives + False\ negatives}$$

In our case, these values are helpful as we are not particularly interested in the True negatives, we rather seek to make correct prediction of the minority class, the independent variables ([5]). Furthermore, the ideal point corresponds to a precision and a recall equal to 1.

We used the same α values than in ([3]): 1e-25, 1e-20, 1e-17, 1.0e-15, 1.0e-13, 1.0e-10, 8.7e-09, 7.6e-07, 6.6e-05, 5.7e-03, 5.0e-02, 5.0e-01. They consist of a logarithmic discretization of [1.0e-25, 5.0e-01], which allows for a regular repartition of points on the obtained curves (a standard value for α is 5.0e-02).

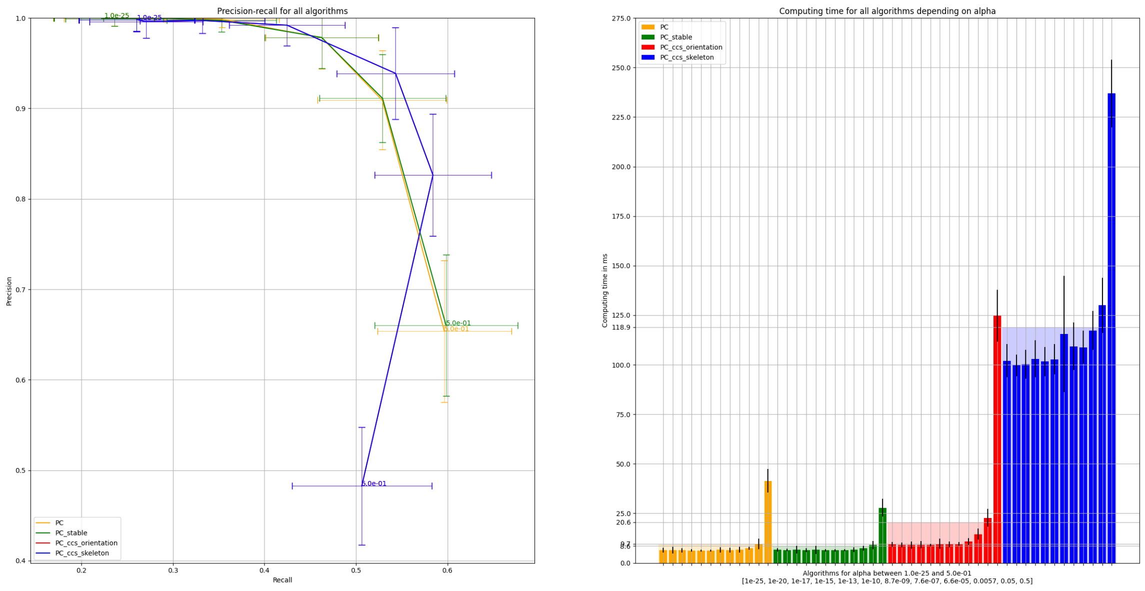

Figure 2: Example output of the `main.py` file, 25 nodes and 40 arcs and 4 values on 500 samples. Showing the averaged precision-recall curves and time complexity of each algorithm depending on α.

Figure 2 shows an example output of our benchmark, we see that the curves corresponding to the PC-css orientation and the PC-css skeleton algorithm are identical. Because PC-css skeleton inherits from PC-css orientation, and that the computing times are noticeably different, we concur that our implementation of PC-css skeleton is wrong. Hence PC-css skeleton will be discarded from the other outputs.

4.1 α dependency

Figure 3 shows the output for a medium interaction benchmark. Firstly we notice the subtle difference between PC and PC-stable due to the shuffling of variables during the learning process. Indeed, in figure 4, one can see that the fscore (harmonic mean of precision and recall) for PC holds a variance, whereas the fscore for PC stable is constant.

On the precision-recall graph, for any given algorithm, while low values of α yield a high precision, its increase leads to the discovery of other Pareto optimal points which posses a better compromise in precision-recall. Hence the α value has to be chosen as a compromise between precision and sensitivity (recall). Moreover, the increase in α causes a higher variance in precision, which becomes comparable to the variance in recall.

One can notice that there exists a range of values for α where PC ccs orientation is noticeably better (in the sens of Pareto) than the other algorithms.

In regards to the time complexity, we observe a linear increase with alpha (the logarithmic repartition of the values and an exponential curves yield a linear function). The slope of the curves are comparable for PC and PC stable (PC stable performs better since the iterative deletion of the edges breaks faster), but much higher for PC ccs orientation. This was expected, indeed as shown in figure 5, lower values of alpha tend to build Independency-maps. Therefore during the propagation of the orientations, the computation takes much more time.

Since the propagation of the orientations is linear in the size of the separating sets, the computation time increases. PC ccs orientation is even more dependant in the separating sets' size, hence the higher slope.

Let us note that we didn't study the increase in computation time due to the node complexity, as its increase is adamant to the change of complexity yielded by an increase of α.

4.2 Sample size dependency

When drawing the precision-recall curves for all our benchmarks (see figures 6, 7, 8), we notice that the sample size leads to a drastic change in overall precision and recall performances. Indeed, for a given alpha, the more values, the surer an independence test can be.

As the sample size increases, the gap in average performance between PC css orientation and the PC algorithms narrows. As previously stated, one of the main shortcomings of the basic PC algorithms are their weakness to noise on finite data sets. By increasing the data sets' size, this weakness fades.

Furthermore we notice an overall longer computational time. Given that the computational time curves' shape remain the same for each data set, we concur that this difference isn't due to the algorithms themselves, but rather to the use of *PyAgrum*'s data sets learners that take more time to compute the independence tests.

In between each benchmark, there seems to be no significant increase in computing time. While the true Bayesian networks are more complex for weak interactions, to medium interactions and strong interactions, this complexity is unknown to the algorithms. Rather, it is the α value that dictates the amount of complexity that it retains. Therefore most of the computing time potential expenditure is due to the increase of α.

Furthermore, over the benchmarks the average recall value decreases. We believe that this, on the contrary, is directly due to the increase in complexity (number of interactions) in the graph. Indeed, with more interactions there is a higher chance that at least one conditional independence test yields a positive results, hence increasing the amount of false negatives (variables wrongly assigned as independent) and decreasing the recall value.

Lastly, we observe that, in any case, an optimal value for α in the sense of the distance to the optimal point, is around 5%. This value is commonly used and recommended for learning algorithms.

References

[1] Spirtes, P,, and Glymour, C., *An algorithm for fast recovery of sparse causal graphs*, 1991.

[2] Diego Colombo, Marloes H. Maathuis, *Order-Independent Constraint-Based Causal Structure Learning*, 2014.

[3] Hervé Isambert et al., *Constraint-based Causal Structure Learning with Consistent Separating Sets*, 2019.

[4] Ducamp, Gaspard and Gonzales, Christophe and Wuillemin, Pierre-Henri, *aGrUM/pyAgrum : a Toolbox to Build Models and Algorithms for Probabilistic Graphical Models in Python*, 2020

[5] Jason Brownlee, *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*, machinelearningmastery.com, 2018

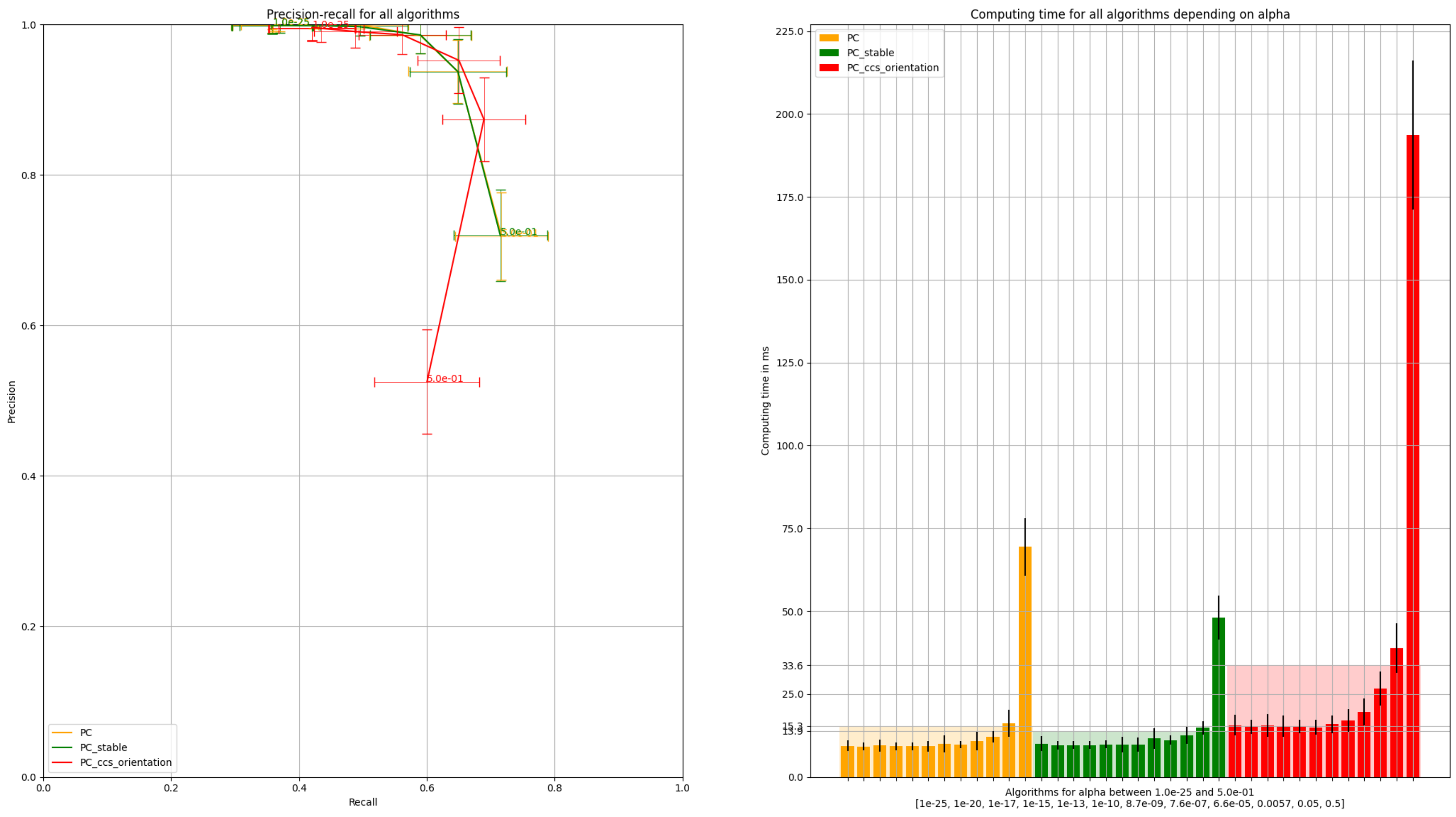

Figure 3: Averaged precision-recall and performance for the medium interactions benchmark, depending on α. Low values of alpha increase the precision, whereas higher values decrease the precision and increase the recall. Time complexity increases linearly with α.

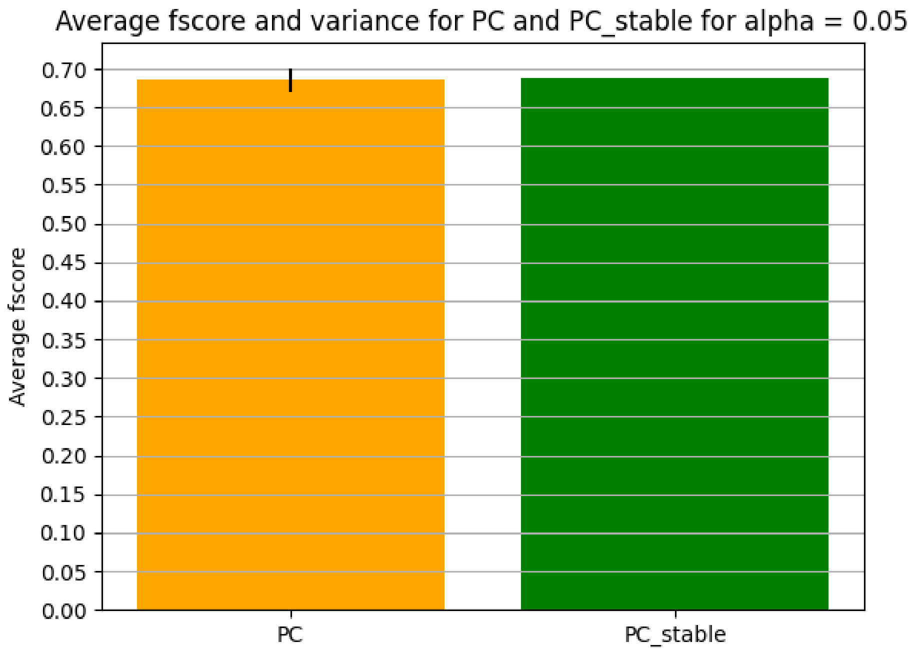

Figure 4: Average fscore for PC and PC stable on 10 learning runs of the same Bayesian network for $\alpha = 0.05$. We notice that while having similar performance, PC yields a variance to its fscore, highlighting the lexicographic dependence of the latter.

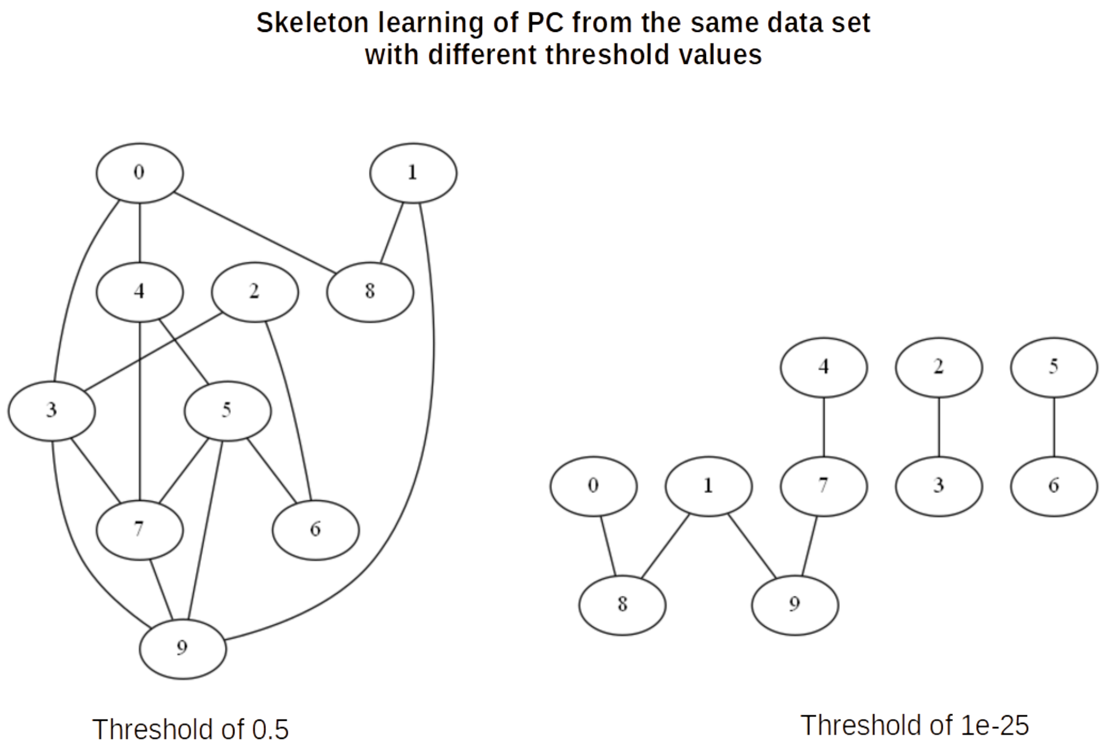
Skeleton learning of PC from the same data set with different threshold values
0
1
4
2
8
3
5
7
6
9
Threshold of 0.5
4
2
5
0
1
7
3
6
8
9
Threshold of 1e-25

Figure 5: Skeleton learning from the same data set with different α values (1e-25 and 0.5). An increase in α leads to a higher threshold for the independence test, therefore the learnt skeleton is more complex. Low values of alpha build a Independency-map whereas higher values tend to build Dependency-maps.

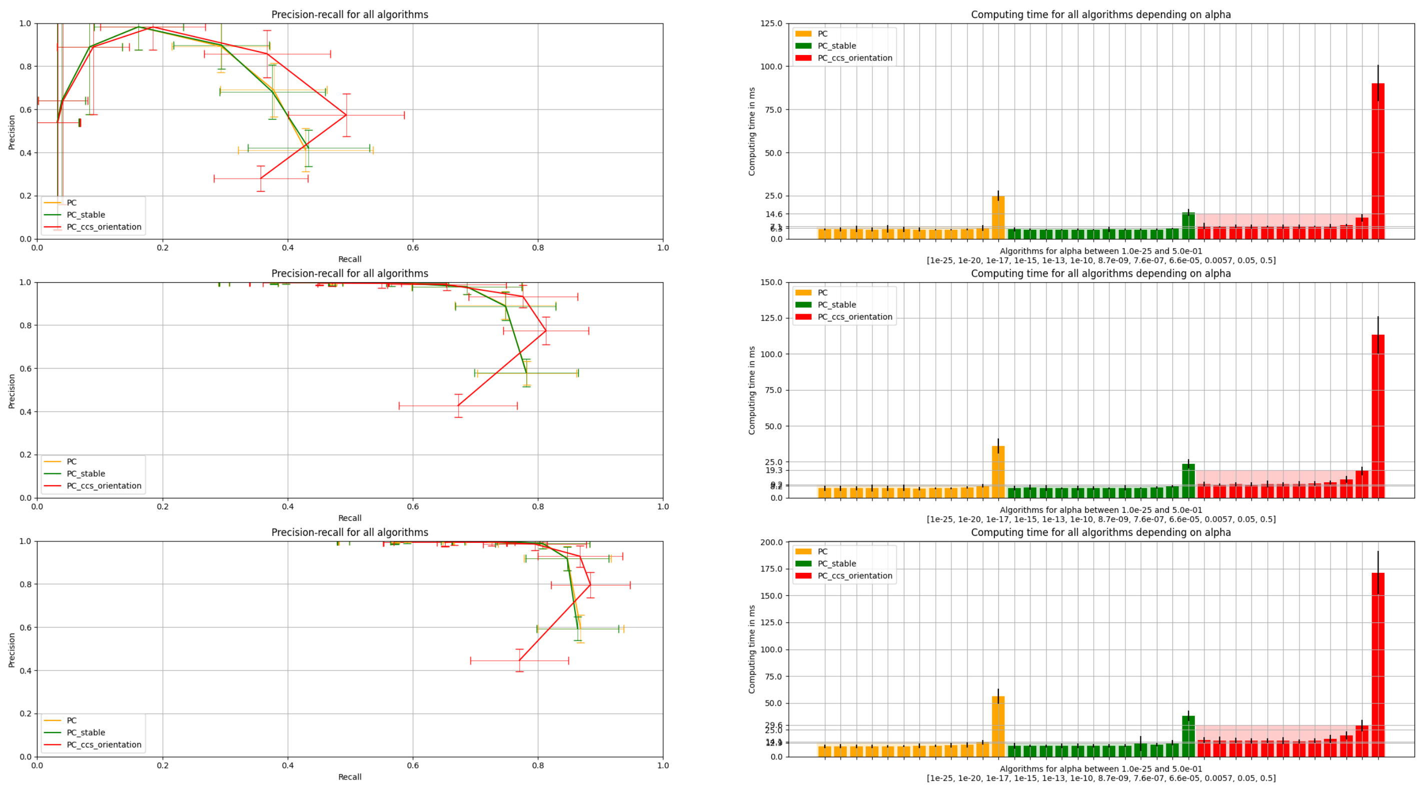

Figure 6: Precision-recall and performance of the algorithms on the weak interactions benchmark for 100, 500 and 1000 samples (top to bottom).

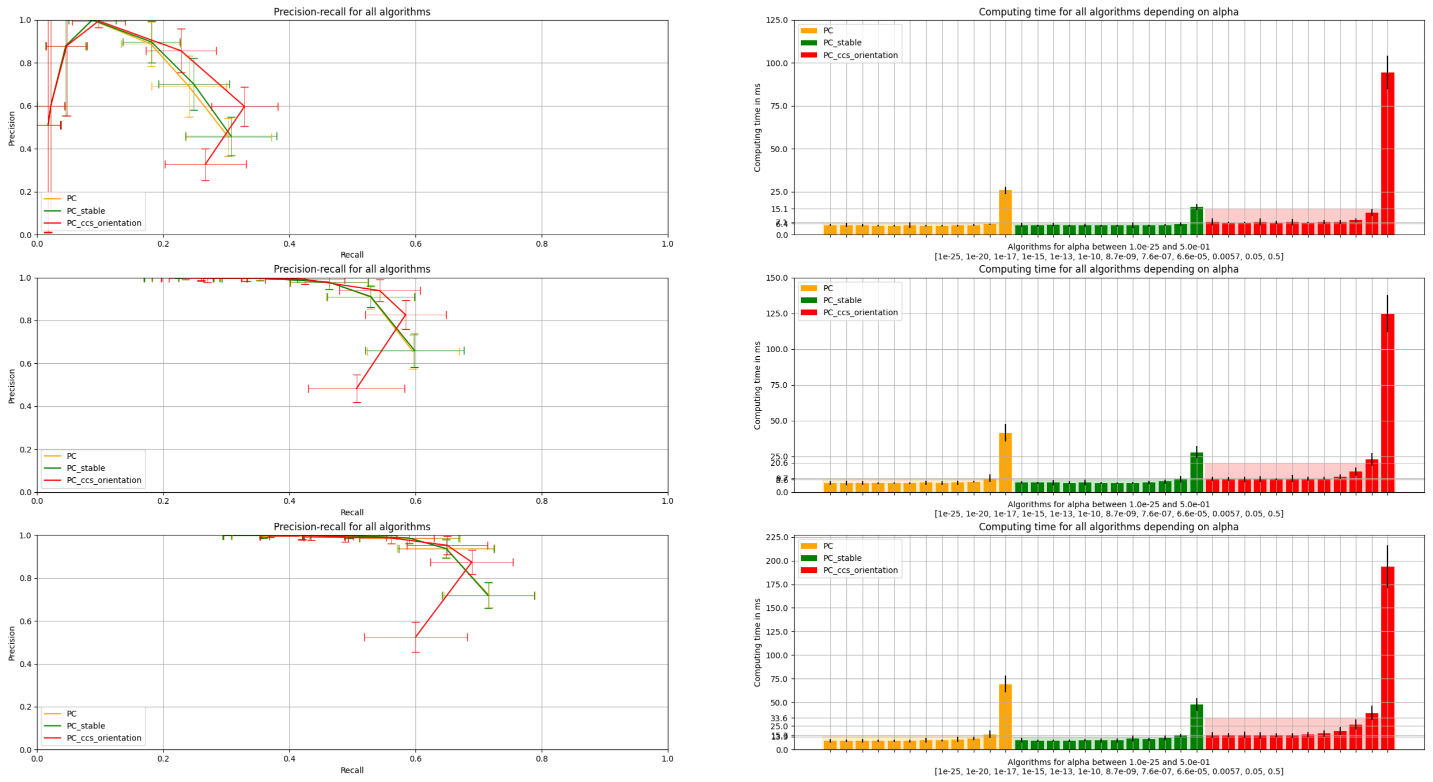

Figure 7: Precision-recall and performance of the algorithms on the medium interactions benchmark for 100, 500 and 1000 samples (top to bottom).

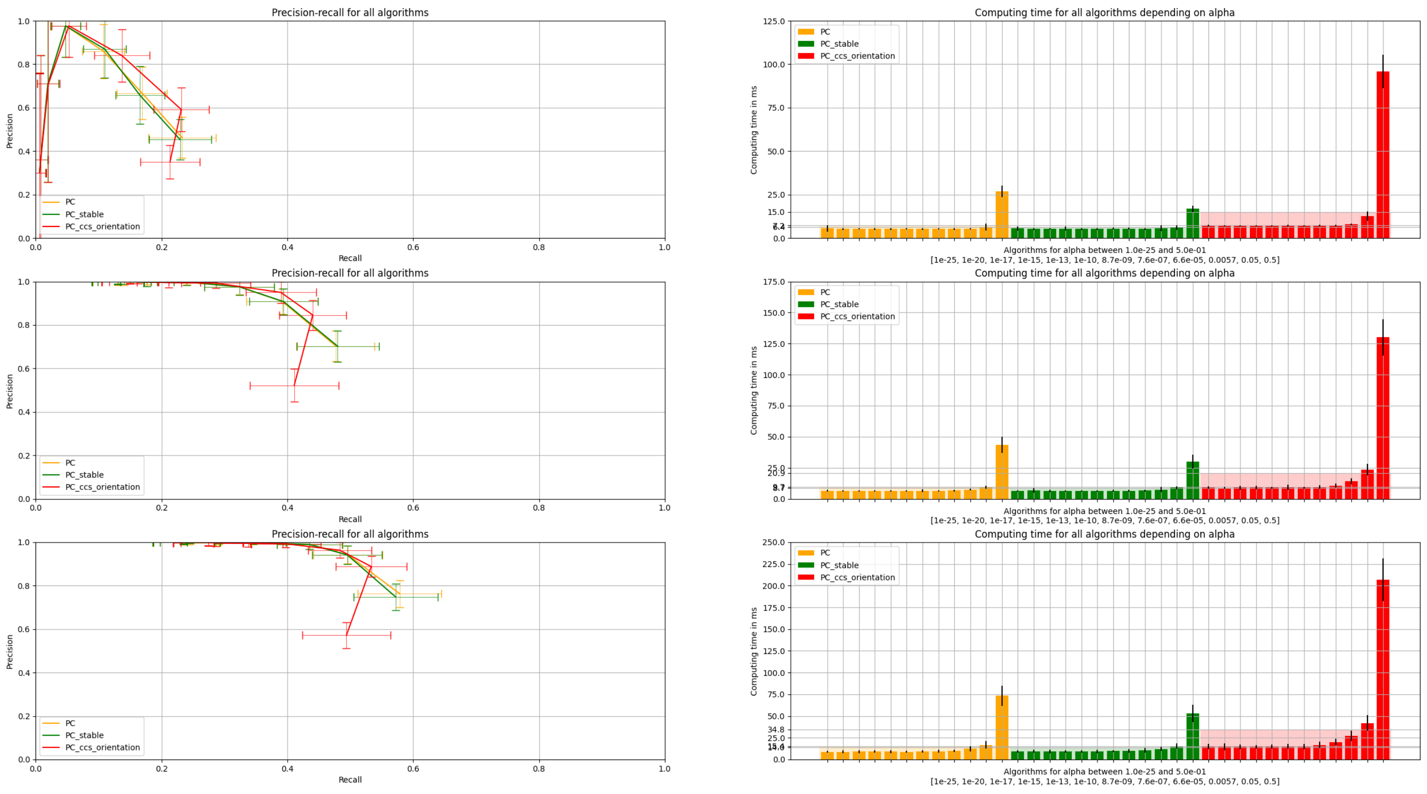

Figure 8: Precision-recall and performance of the algorithms on the strong interactions benchmark for 100, 500 and 1000 samples (top to bottom).