Experiment results log.

Planned order of scenarios

Scenarios	Parameters	Week
A1	N	March 23 - March 29
A2	B	March 30 - April 5
A3	Datasets	March 30 - April 5
B1	Chain Strength	April 6 - April 12
B2	Embedding	April 13 - April 19
B3	Shots	April 20 - April 26
B4	Annealing	April 27 - May 3

Actual order of scenarios

Scenarios	Parameters	Week
A1	**N**	**March 23 - March 29**
B1	**Chain Strength**	**March 30 - April 5**
A2	**B**	**April 6 - April 12**
B3	**Shots**	**April 13 - April 19**
B2	Embedding	April 13 - April 19
B4	Annealing	April 20 - April 26
A3	Datasets	April 27 - May 3

Sidenotes to research about

- Find what is the maximum `N` value that is supported by dwave

Scenario A1 - N

We started by experimenting several values of `N`, in order to find the maximum possible value of `N` that could be solved in a reasonable time by the classical solver.

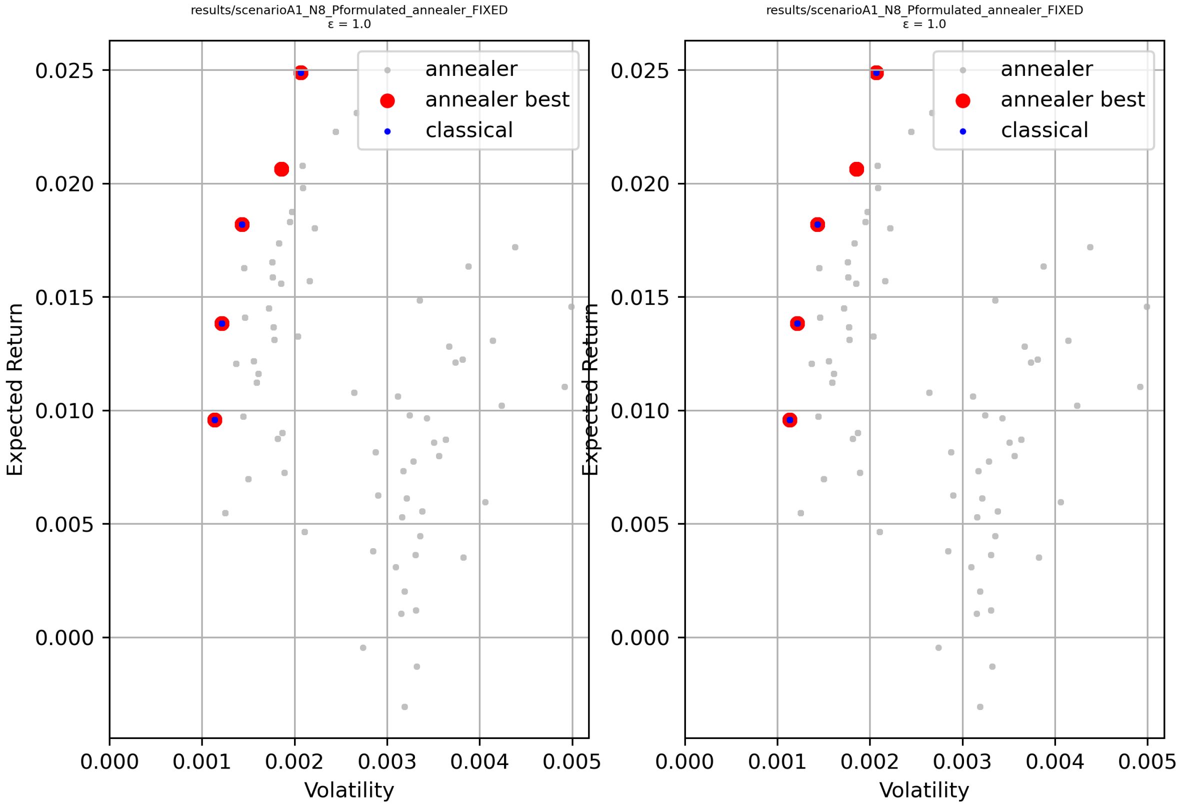
The `N` values are: 8, 16, 32, and 64. P was calculated as `P = -q * min_sigma + max_mu`

For this scenario, we used the "diversified" dataset and 1000 shots per execution. The `q_values` are listed in the following table:

N	q values	Epsilon Indicator

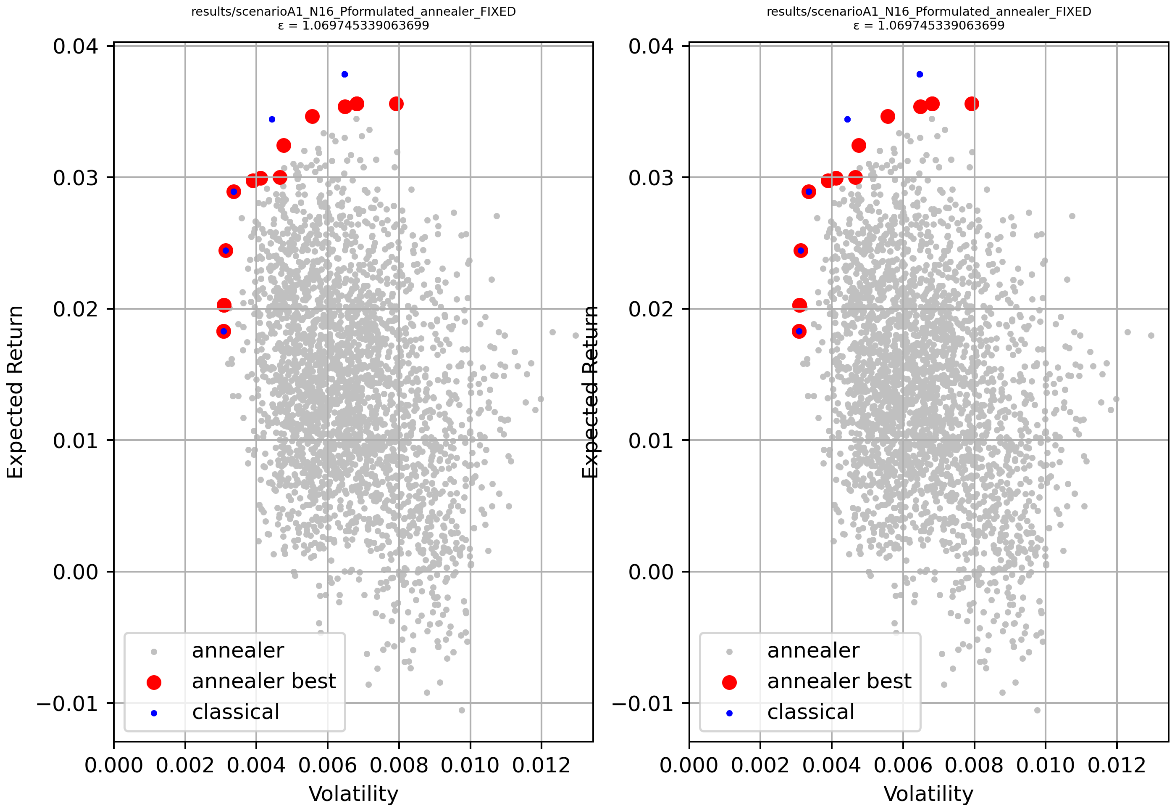
N	q values	Epsilon Indicator
8	0, 11, 20, 54	1.0
16	0, 2, 6, 100, 500	1.070
32	0, 0.4, 0.9, 2, 3, 9, 100	1.967
64	0, 0.2, 0.4, 0.6, 1.1, 1.3, 1.5, 2, 5, 6, 7, 8, 10, 100, 500	2.022

Epsilon Indicator - scenario1Y2021M04D18h23m07s48

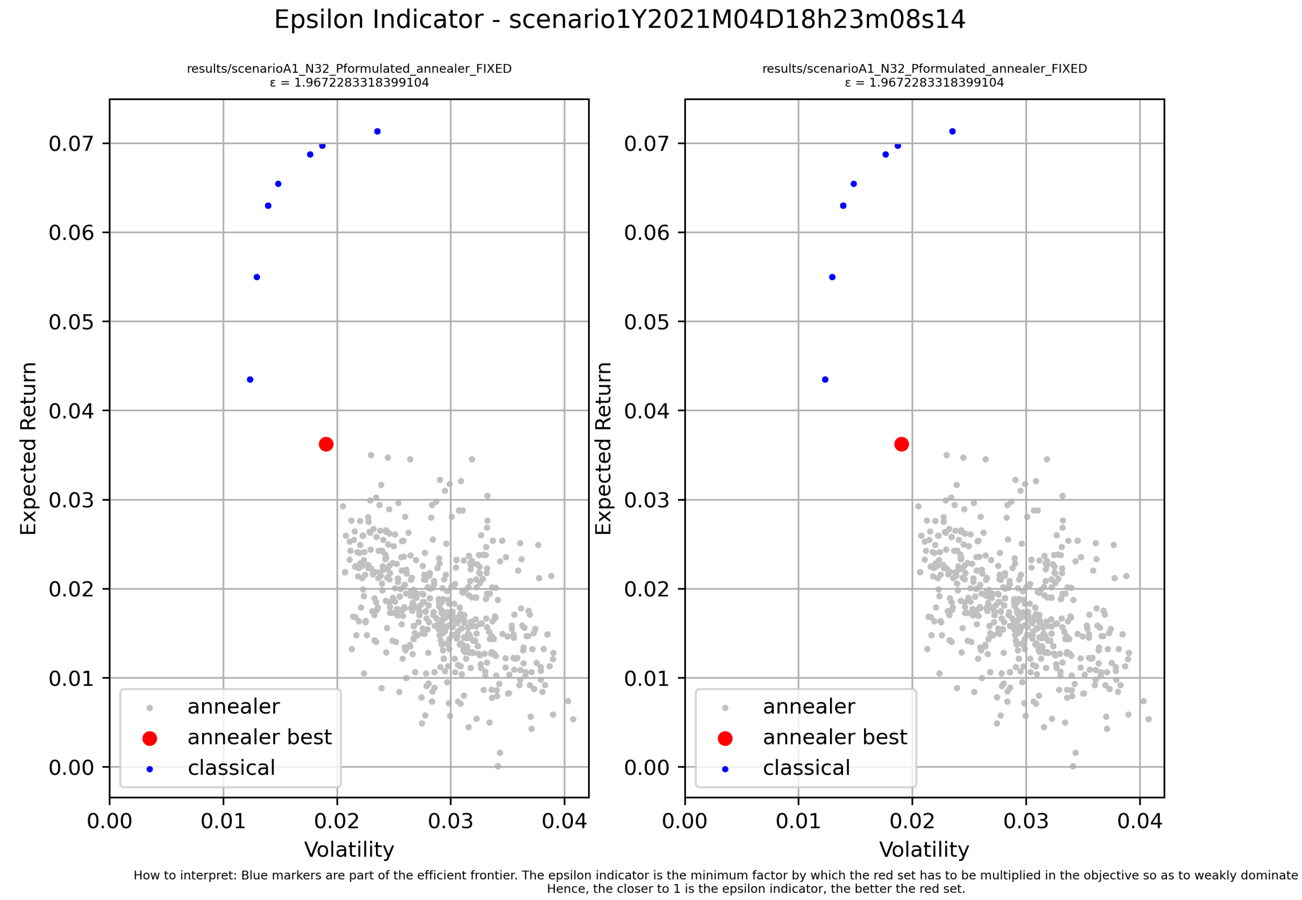


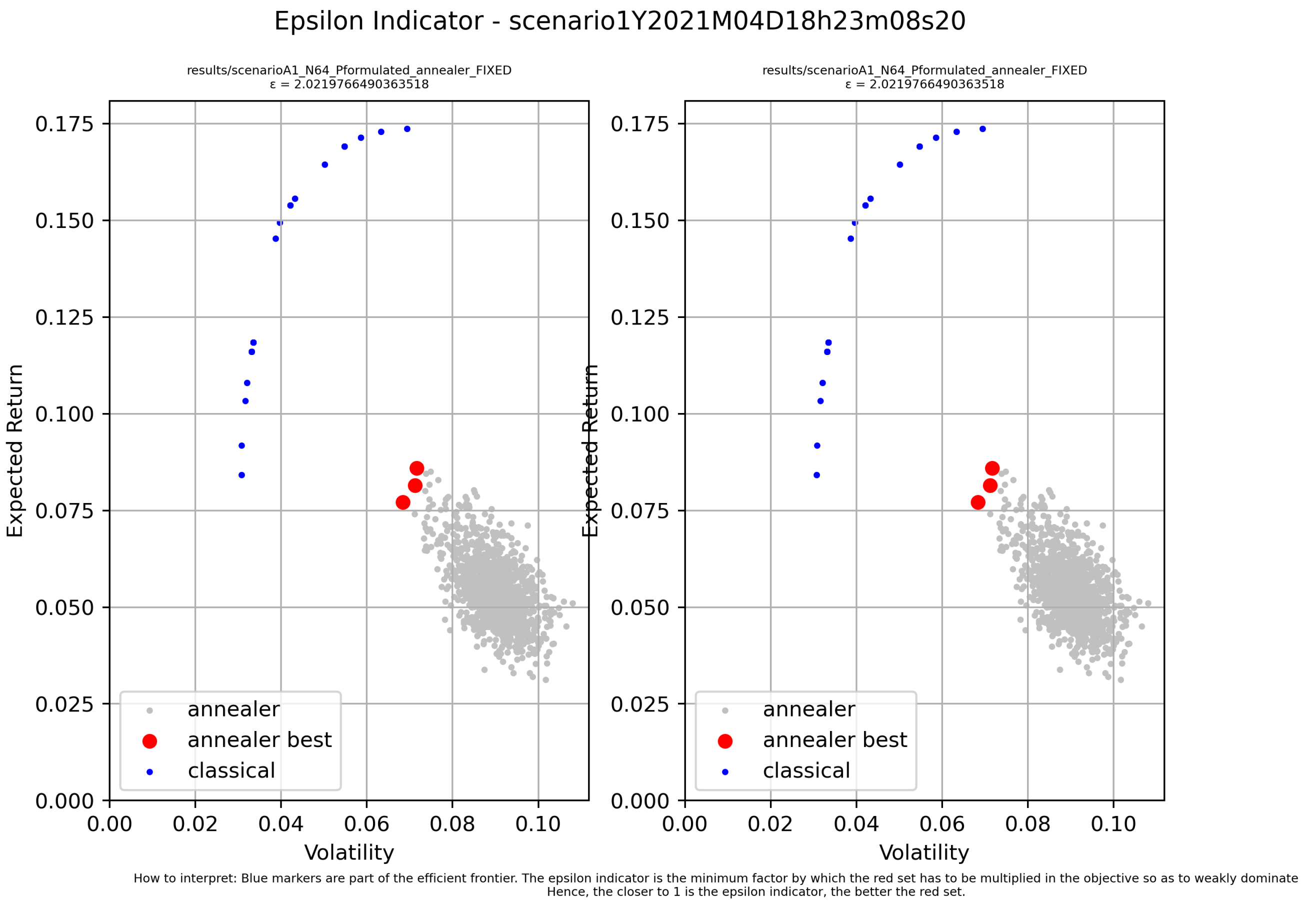
How to interpret: Blue markers are part of the efficient frontier. The epsilon indicator is the minimum factor by which the red set has to be multiplied in the objective so as to weakly dominate
Hence, the closer to 1 is the epsilon indicator, the better the red set.

Epsilon Indicator - scenario1Y2021M04D18h23m08s05



How to interpret: Blue markers are part of the efficient frontier. The epsilon indicator is the minimum factor by which the red set has to be multiplied in the objective so as to weakly dominate
Hence, the closer to 1 is the epsilon indicator, the better the red set.





Key Takeaways:

As expected, the epsilon indicator increases with the `N` value. However, during those executions, dwave's problem inspector warned that the chains were too weak, and that, in the case of `N=64`, all samples had broken chains. Based on this warning, we decided to immediately execute scenario B1, changing the original order of scenarios.

Scenario B1 - Chain Strength

Looking at the fraction of chain breaks in Scenario A1, we know that on average each sample had almost a third (`0.31`) of its chains broken when `N=32`. This fraction increases to over half (`0.54`) when `N=64`! Those values are very high and are another clue that the chain strength needs to be adjusted, especially for those values of `N`.

A good starting value for the chain strength is the maximum absolute value (`maxAbs`) of the QUBO matrix. However, this is not always the most optimal value. We need to test several values based on this initial value. By testing those values, we can find a value near the sweet spot between the probability that the chains are intact and the probability of finding optimal values. Refer to: https://www.dwavesys.com/sites/default/files/2_Wed_Am_PerfTips.pdf

We have three tables, one for the epsilon indicator, one for the fractions of valid solutions, and one for the average fractions of chain breaks.

Starting with the average fractions of chain breaks (Lower is better):

Chain strength	N8	N16	N32	N64
default value	0.00081	0.01153	0.31350	0.54426
0.125 * maxAbs	0.00397	0.02741	0.31014	0.38301
0.250 * maxAbs	0.00034	0.00106	0.00170	0.00683
0.375 * maxAbs	0.00006	0.00032	0.00111	0.00453
0.500 * maxAbs	0.00006	0.00026	0.00149	0.00475
0.625 * maxAbs	0.00006	0.00031	0.00112	0.00453
0.750 * maxAbs	0.00006	0.00029	0.00130	0.00454
0.875 * maxAbs	0.00006	**0.00017**	0.00102	0.00461
1.000 * maxAbs	0.00003	0.00034	**0.00100**	0.00439
1.125 * maxAbs	**0.00000**	0.00030	0.00119	**0.00401**
1.250 * maxAbs	**0.00000**	0.00042	0.00125	0.00419
1.375 * maxAbs	0.00006	0.00028	0.00108	0.00424
1.500 * maxAbs	0.00009	0.00025	0.00201	0.00430

Next, we obtained the following fractions of valid solutions (Higher is better):

Chain strength	N8	N16	N32	N64
default value	0.877	**0.688**	0.121	0.094

Chain strength	N8	N16	N32	N64
0.125 * maxAbs	0.001	0.002	0.076	0.205
0.250 * maxAbs	**0.934**	0.622	**0.395**	**0.243**
0.375 * maxAbs	0.848	0.543	0.325	0.220
0.500 * maxAbs	0.781	0.485	0.299	0.186
0.625 * maxAbs	0.703	0.444	0.261	0.172
0.750 * maxAbs	0.665	0.388	0.252	0.170
0.875 * maxAbs	0.630	0.406	0.242	0.163
1.000 * maxAbs	0.598	0.366	0.235	0.151
1.125 * maxAbs	0.594	0.370	0.219	0.148
1.250 * maxAbs	0.556	0.342	0.223	0.129
1.375 * maxAbs	0.540	0.330	0.212	0.136
1.500 * maxAbs	0.512	0.310	0.198	0.138

Finally, we obtained the following epsilon indicators (Lower is better):

Chain strength	N8	N16	N32	N64
default value	1,000	1,070	1,967	2,022
0,125 * maxAbs	1,368	18,844	1,767	1,977
0,250 * maxAbs	1,000	1,075	1,178	1,474
0,375 * maxAbs	1,000	1,057	1,203	1,580
0,500 * maxAbs	1,000	1,099	1,331	1,500
0,625 * maxAbs	1,000	1,098	1,269	1,410
0,750 * maxAbs	1,000	1,120	1,429	1,523
0,875 * maxAbs	1,000	1,123	1,430	1,587
1,000 * maxAbs	1,000	1,119	1,250	1,526
1,125 * maxAbs	1,000	1,099	1,142	1,539
1,250 * maxAbs	1,000	1,092	1,355	1,610
1,375 * maxAbs	1,000	1,110	1,352	1,465
1,500 * maxAbs	1,000	1,101	1,345	1,423

To validate such results, this scenario has been repeated for `N=16`, `N=32`, and `N=64`.

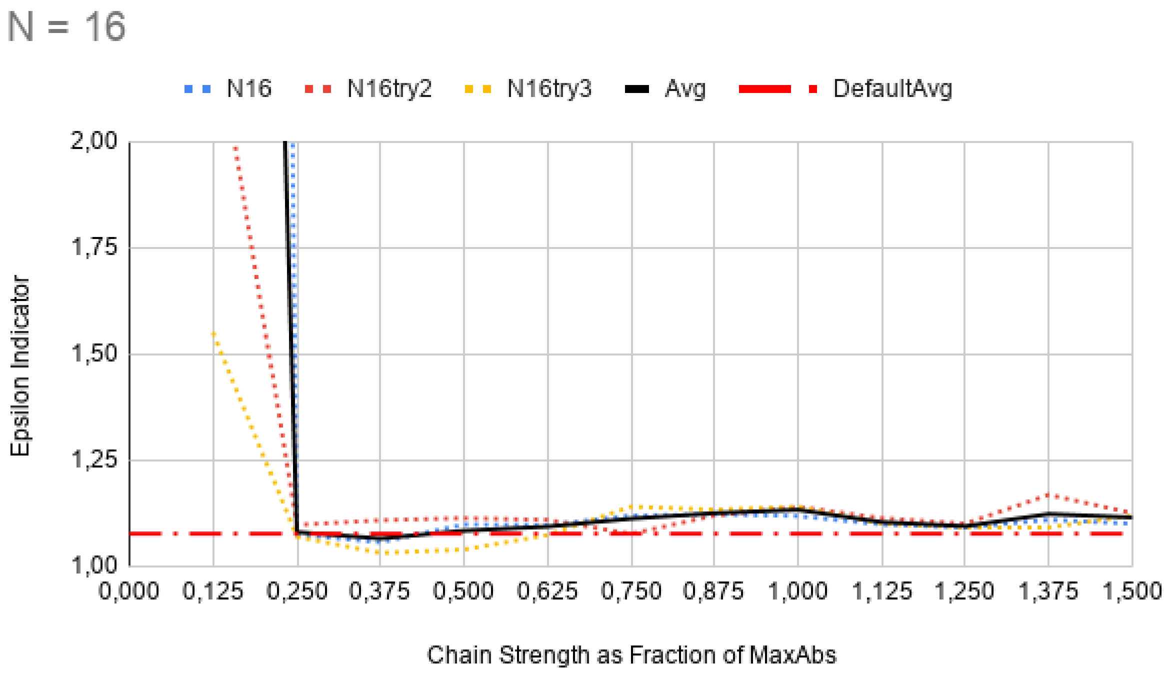
Chain strength	N16	N32	N64

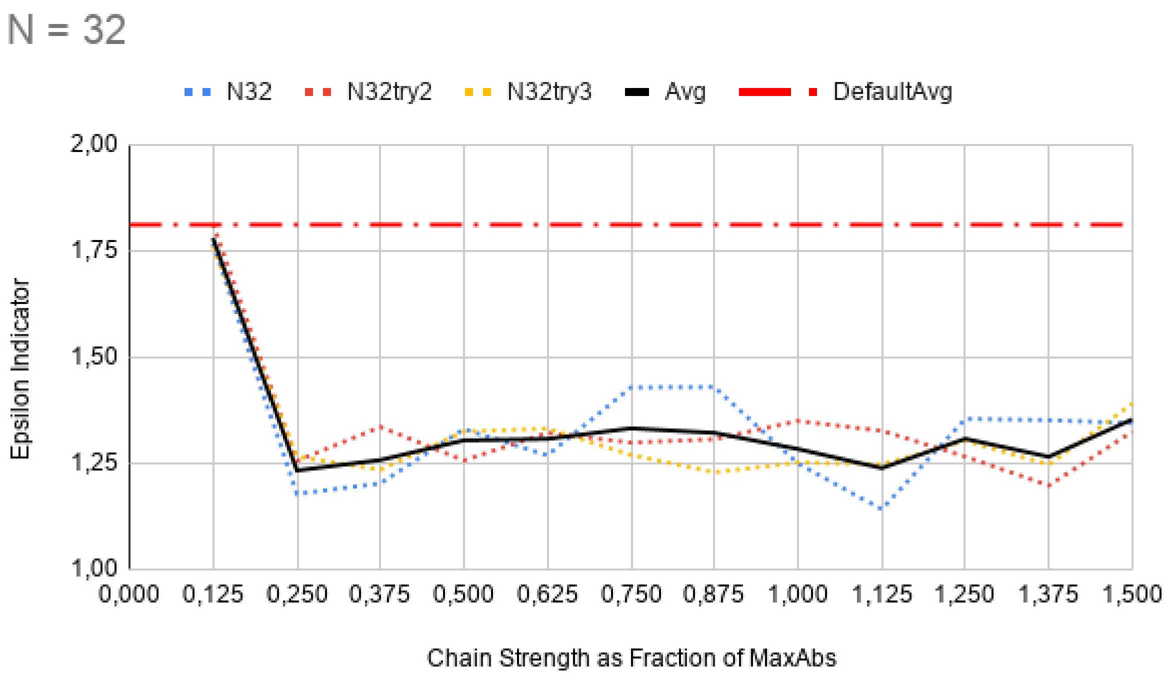
Chain strength	N16	N32	N64
default value	1,092	1,739	1,981
0,125 * maxAbs	2,314	1,813	2,185
0,250 * maxAbs	1,098	1,256	1,550
0,375 * maxAbs	1,109	1,336	1,492
0,500 * maxAbs	1,114	1,257	1,502
0,625 * maxAbs	1,110	1,322	1,503
0,750 * maxAbs	1,077	1,299	1,516
0,875 * maxAbs	1,120	1,307	1,489
1,000 * maxAbs	1,141	1,350	1,485
1,125 * maxAbs	1,114	1,327	1,430
1,250 * maxAbs	1,101	1,266	1,549
1,375 * maxAbs	1,169	1,198	1,508
1,500 * maxAbs	1,126	1,325	1,597

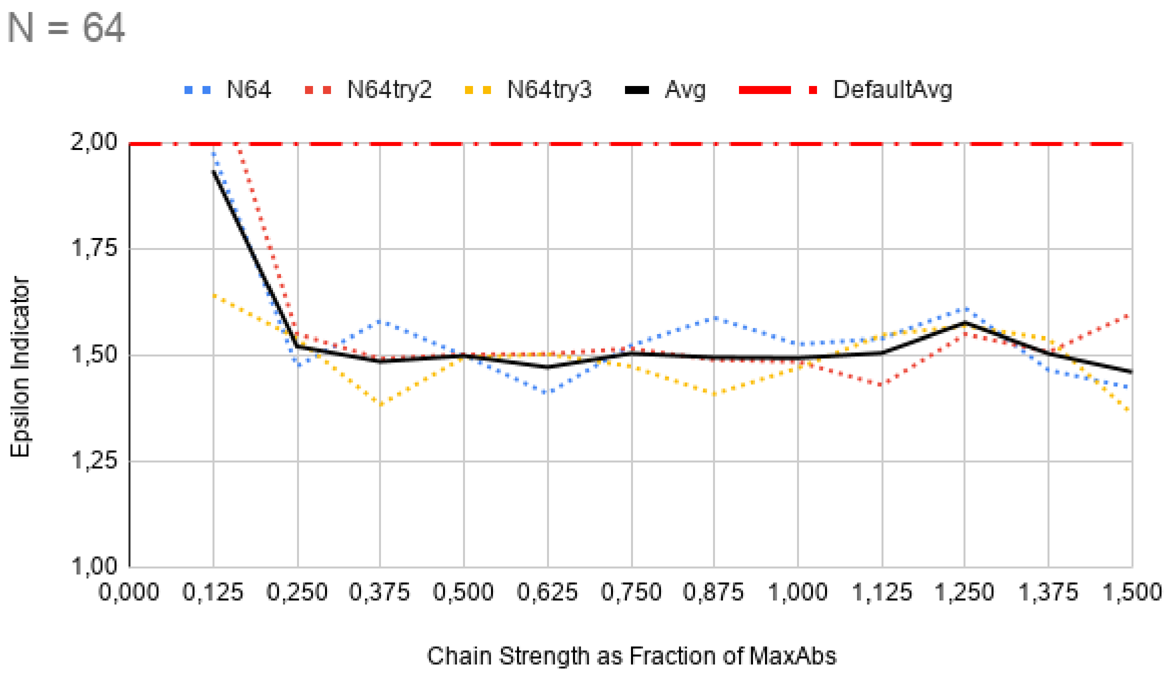
And one more time:

Chain strength	N16	N32	N64
default value	1,070	1,728	1,988
0,125 * maxAbs	1,551	1,760	1,906
0,250 * maxAbs	1,070	1,266	1,462
0,375 * maxAbs	1,032	1,235	1,583
0,500 * maxAbs	1,040	1,325	1,514
0,625 * maxAbs	1,074	1,332	1,551
0,750 * maxAbs	1,141	1,270	1,458
0,875 * maxAbs	1,134	1,229	1,515
1,000 * maxAbs	1,141	1,252	1,536
1,125 * maxAbs	1,101	1,248	1,547
1,250 * maxAbs	1,092	1,303	1,523
1,375 * maxAbs	1,092	1,247	1,560
1,500 * maxAbs	1,120	1,391	1,519
5,000 * maxAbs	1.177	1,297	1,627

The results are summarized in the following charts.







Key Takeaways:

Looking at the results, we notice that the impact of any change to the chain strength is higher for higher values of `N`.

It also becomes clear that, especially for higher values of `N`, the default chain strength is far from being the best value. It seems that for higher values of `N`, the farther is the default chain strength value from the best value.

Another thing that also becomes clear is that the fractions of chain breaks and valid solutions are not directly synonymous with the quality of the solutions.

For the case of `N=8`, every try gave a perfect score of `1.000`.

For the case `N=16`, the epsilon values are so similar that they fall under the margin of variation. Thus we cannot place conclusions based on these results. (Note: in this case, the default strength is always the best!)

There is an exception for both cases of `N=8` and `N=16`. When `chain_strength = 0.125 * maxAbs` there is a high fraction of chain breaks and almost no samples are valid solutions. Thus, for this value of chain strength, the results are very bad.

This behavior is also noticeable for `N=32` and `N=64`, that present a relatively high epsilon indicator with this chain strength.

It seems that, after this very weak chain strength, the following values of chain strength rapidly attain the lowest epsilon indicators registered, with a very slow climb afterwards.

In the end, the results suggest that it is okay to choose any value that is part of the slow climb. However, from theory, we know that we should avoid any value over `1.000 * maxAbs`, since it scales down the problem.

Therefore, for all `N` values, a safe range seems to be between `0.250 * maxAbs` and `1.000 * maxAbs`.

Based on those findings, the case `N=8` will not be tested in the remaining scenarios, since the annealer already achieved optimality.

Scenario A2 - B

For this scenario, we will be looking at how different budgets affect the performance of the annealer. Therefore, different fractions of `B` are going to be tested: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9.

I will be using `chain_strength = 1.000 * maxAbs`, for the reasons explained in the previous scenario. **Reminder: the fraction used in previous scenarios was `B=0.5`!**

Obviously, for each value of B, we first need to solve it classically. Then, from the results, we get the sequences of `q_values` to be used in the annealer.

N	q values	Budget fraction
16	0, 20, 500	0.1 (1)
32	0, 7, 20, 40	0.1 (3)
64	0, 0.6, 2, 4, 6, 8, 20, 40, 80, 500	0.1 (6)
16	0, 8, 10, 40	0.2 (3)
32	0, 5, 8, 20, 30, 80	0.2 (6)
64	0, 0.3, 0.8, 2, 4, 5, 7, 9, 20, 30, 500	0.2 (12)
16	0, 2, 6, 20, 60	0.3 (4)
32	0, 3, 4, 10, 20, 50	0.3 (9)
64	0, 0.2, 2, 3, 4, 5, 7, 9, 20, 30, 100	0.3 (19)
16	0, 2, 5, 10, 30	0.4 (6)
32	0, 0.2, 0.9, 2, 4, 20, 30, 70, 500	0.4 (12)
64	0, 0.3, 0.6, 1, 2, 3, 4, 6, 8, 20, 30, 90	0.4 (25)
16	0, 2, 6, 100, 500	0.5 (8)
32	0, 0.4, 0.9, 2, 3, 9, 100	0.5 (16)
64	0, 0.2, 0.4, 0.6, 1.1, 1.3, 1.5, 2, 5, 6, 7, 8, 10, 100, 500	0.5 (32)
16	0, 0.1, 0.8, 3, 20, 30	0.6 (9)
32	0, 0.1, 0.5, 1, 2, 3, 7, 8, 20, 30	0.6 (19)
64	0, 0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 2, 3, 7, 9, 20	0.6 (38)
16	0, 0.7, 20	0.7 (11)
32	0, 0.4, 2	0.7 (22)
64	0, 0.1, 0.2, 0.3, 0.7, 1, 2, 3, 4, 6, 20	0.7 (44)

N	q values	Budget fraction
16	0, 4	0.8 (12)
32	0, 0.8, 7, 9	0.8 (25)
64	0, 0.1, 0.2, 0.4, 0.5, 0.6, 1, 2, 3, 6, 20	0.8 (51)
16	0, 50	0.9 (14)
32	0, 0.8, 3	0.9 (28)
64	0, 0.6, 1, 2, 5, 500	0.9 (57)

Question: Is it bad to have different number of samples between cases?

With those results, we obtained the following epsilon indicators:

Budget fraction	N16 (AvgChainBreak)	N32 (AvgChainBreak)	N64 (AvgChainBreak)
0,1	1,000 (0,00406)	1,142 (0,00979)	1,846 (0,01572)
0,2	1,000 (0,00048)	1,334 (0,00151)	2,929 (0,00493)
0,3	1,026 (0,00044)	1,373 (0,00152)	1,750 (0,00473)
0,4	1,113 (0,00024)	1,281 (0,00134)	1,640 (0,00452)
0,5	1,075 (0,00031)	1,311 (0,00127)	1,513 (0,00484)
0,6	1,146 (0,00033)	1,293 (0,00141)	1,441 (0,00470)
0,7	1,162 (0,00038)	1,421 (0,00108)	1,907 (0,00464)
0,8	1,005 (0,00031)	1,408 (0,00144)	inf (0,00462)
0,9	1,103 (0,00034)	inf (0,00129)	inf (0,00447)

Gráficos com os resultados deste cenário estão no cenário seguinte

Key Takeaways:

The first thing I notice is that there is a high chain break fraction when the budget is `B=0.1`. Afterwards, it attains a consistently low fraction, with small variation.

For `N=32` and `N=64`, as expected from theory, the epsilon indicator is lower when the budget is or is close to `B=0.5`, since this value has the highest number of admissible solutions.

Behavior for all `N` values is hard to grasp. Nonetheless, budget fraction is a parameter that is particular to each practitioner.

Scenario B3 - Shots

The previous scenario, A2, made us wonder about the number of samples. That is, there is a possibility that the cases where B is farthest from `B=0.5` have worse performance because of having less values of `q` and thus less samples taken.

Therefore, we pose a question: Is it better to increase the number of shots per value of `q` or to add more values of `q` to be executed?

Since the results so far seem to have a good coverage of the efficient frontier, but still far from it, we believe that the issue is related to the number of samples per value of `q`. Hence, we are going to repeat the previous scenario with a new methodology to define the number of samples per value of `q`. This methodology is called `Allocated`.

Initially, each value of `q` had 1000 shots, i.e., 1000 samples taken. This time, each case will have a total allocated number of shots for every value of `q`. For example, if we have a case with three values of `q` and another case with five values of `q`, then, with a total allocation of 5000 shots per case, then the first case will have 1666 shots per value, while the second case will have 1000 shots per value.

Based on this methodology, we will start with a total allocation of 15000 shots, such that each of the 15 values of `q` from case `B=0.5` have 1000 shots.

N	q values	Budget fraction	Shots per value of q
16	0, 20, 500	0.1 (1)	5000
32	0, 7, 20, 40	0.1 (3)	3750
64	0, 0.6, 2, 4, 6, 8, 20, 40, 80, 500	0.1 (6)	1500
16	0, 8, 10, 40	0.2 (3)	3750
32	0, 5, 8, 20, 30, 80	0.2 (6)	2500
64	0, 0.3, 0.8, 2, 4, 5, 7, 9, 20, 30, 500	0.2 (12)	1363
16	0, 2, 6, 20, 60	0.3 (4)	3000
32	0, 3, 4, 10, 20, 50	0.3 (9)	2500
64	0, 0.2, 2, 3, 4, 5, 7, 9, 20, 30, 100	0.3 (19)	1363
16	0, 2, 5, 10, 30	0.4 (6)	3000
32	0, 0.2, 0.9, 2, 4, 20, 30, 70, 500	0.4 (12)	1666
64	0, 0.3, 0.6, 1, 2, 3, 4, 6, 8, 20, 30, 90	0.4 (25)	1250
16	0, 2, 6, 100, 500	0.5 (8)	3000
32	0, 0.4, 0.9, 2, 3, 9, 100	0.5 (16)	2142
64	0, 0.2, 0.4, 0.6, 1.1, 1.3, 1.5, 2, 5, 6, 7, 8, 10, 100, 500	0.5 (32)	1000
16	0, 0.1, 0.8, 3, 20, 30	0.6 (9)	2500
32	0, 0.1, 0.5, 1, 2, 3, 7, 8, 20, 30	0.6 (19)	1500
64	0, 0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 2, 3, 7, 9, 20	0.6 (38)	1250
16	0, 0.7, 20	0.7 (11)	5000
32	0, 0.4, 2	0.7 (22)	5000

N	q values	Budget fraction	Shots per value of q
64	0, 0.1, 0.2, 0.3, 0.7, 1, 2, 3, 4, 6, 20	0.7 (44)	1363
16	0, 4	0.8 (12)	7500
32	0, 0.8, 7, 9	0.8 (25)	3750
64	0, 0.1, 0.2, 0.4, 0.5, 0.6, 1, 2, 3, 6, 20	0.8 (51)	1363
16	0, 50	0.9 (14)	7500
32	0, 0.8, 3	0.9 (28)	5000
64	0, 0.6, 1, 2, 5, 500	0.9 (57)	2500

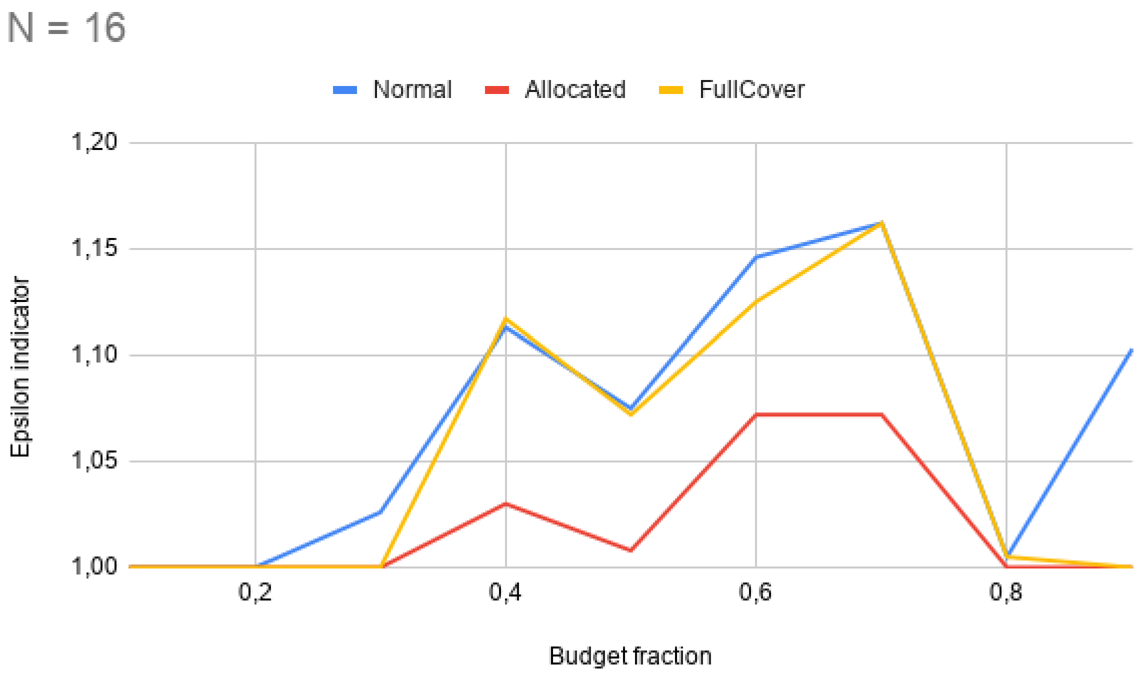
We obtained the following epsilon indicators:

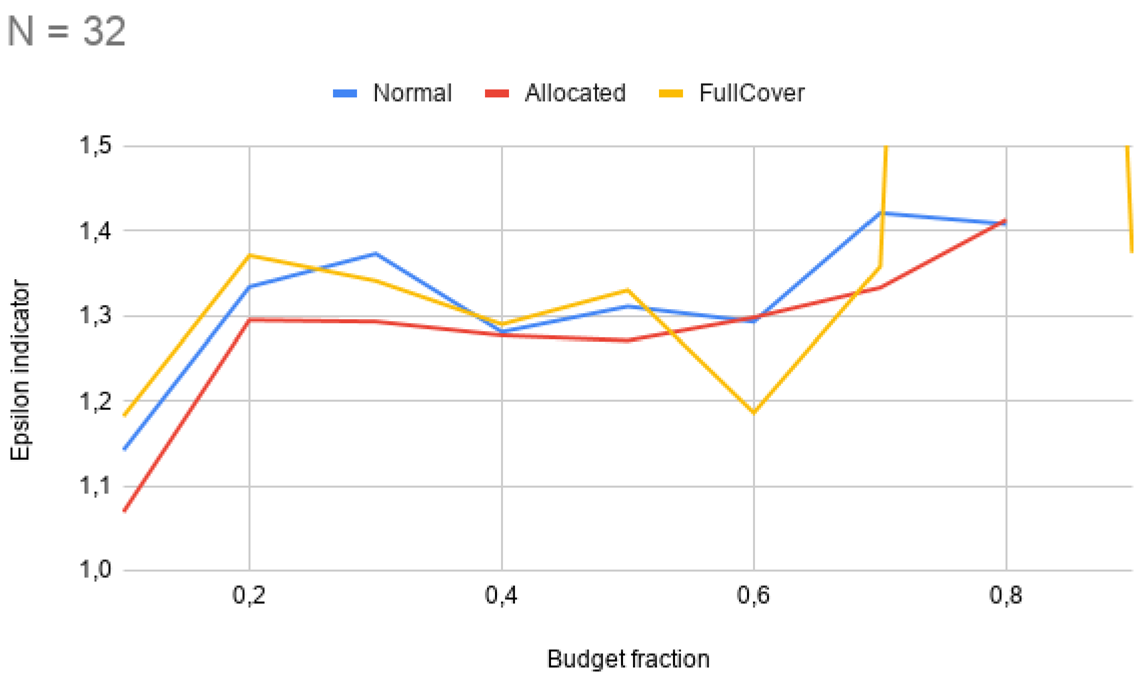
Budget fraction	N16	N32	N64
0,1	1,000	1,069	1,667
0,2	1,000	1,295	2,434
0,3	1,000	1,293	1,651
0,4	1,030	1,277	1,632
0,5	1,008	1,271	1,507
0,6	1,072	1,298	1,521
0,7	1,072	1,333	1,926
0,8	1,000	1,413	inf
0,9	1,000	inf	inf

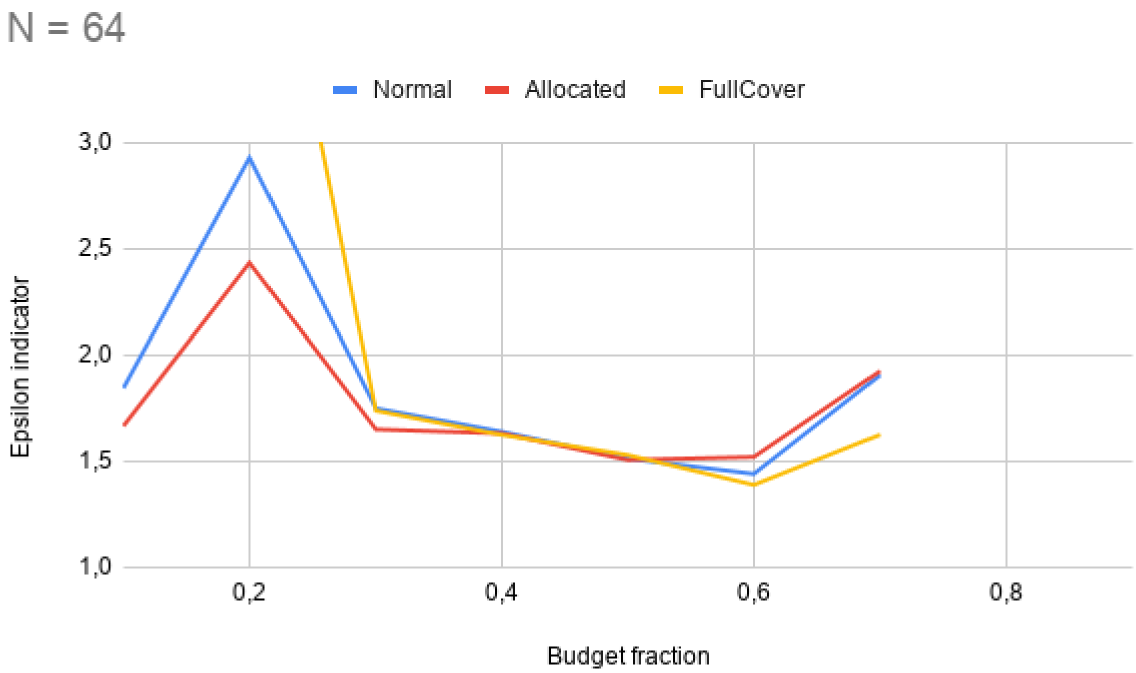
However, we need to take into account that in real case scenarios, we won't be able to have these carefully chosen values of q. In fact, they were discovered because it was feasible to classically solve these scenarios! For this reason, we introduce another methodology, called `FullCoverage`. This methodology will execute the same values of q for every scenario. The list of values of q is based on guesswork and gained experience with the given scenarios: `0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 1000`. As with `Allocated` methodology, this list is allocated to a total of 15000 samples (500 per value of q).

Budget fraction	N16	N32	N64
0,1	1,000	1,182	5,000
0,2	1,000	1,371	4,625
0,3	1,000	1,341	1,739
0,4	1,117	1,290	1,625
0,5	1,072	1,330	1,530

Budget fraction	N16	N32	N64
0,6	1,125	1,186	1,389
0,7	1,162	1,358	1,626
0,8	1,005	4,442	inf
0,9	1,000	1,374	inf







Key Takeaways:

Compared to the previous methodology, called `Simple`, the `Allocated` methodology brings improvements in almost every case. This is expected, since all the cases had their number of samples increased, minus the case `N=64 B=0.5`, which keeps the same number of samples (and also has the same performance in both methodologies).

When looking at the more "realistic" `FullCover` methodology, the results are not the best, but don't fall shortly compared to `Allocated`.

For the next scenarios, we are going to use the `Allocated` methodology, as well as `B=0.5`.

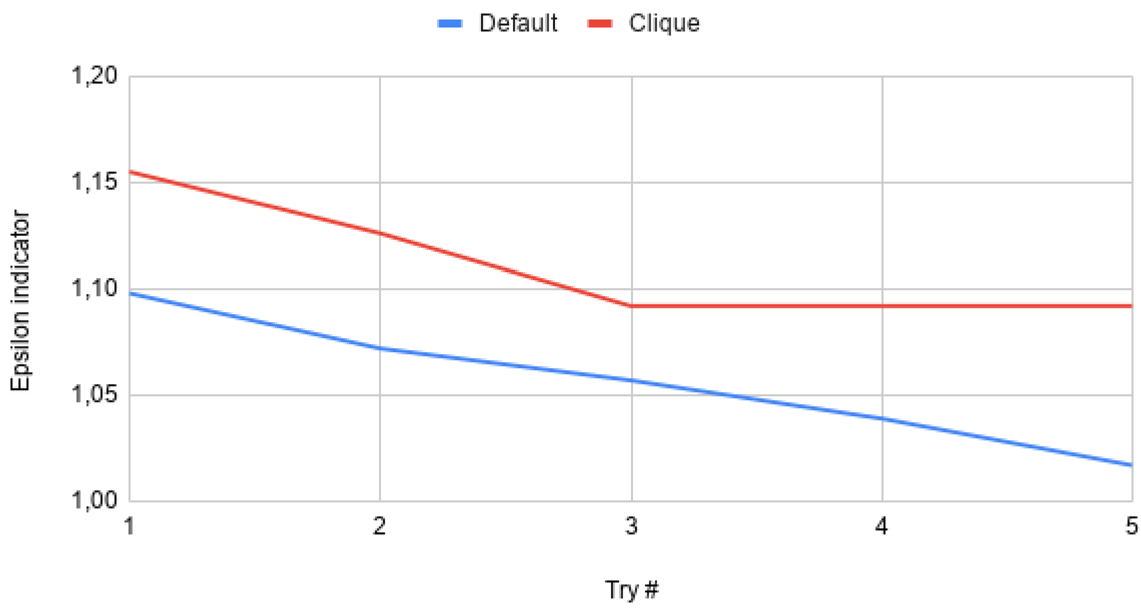
Scenario B2 - Embedding

So far, we used the `default` embedding. D-Wave offers another embedding option, called `clique` embedding, which is going to be compared.

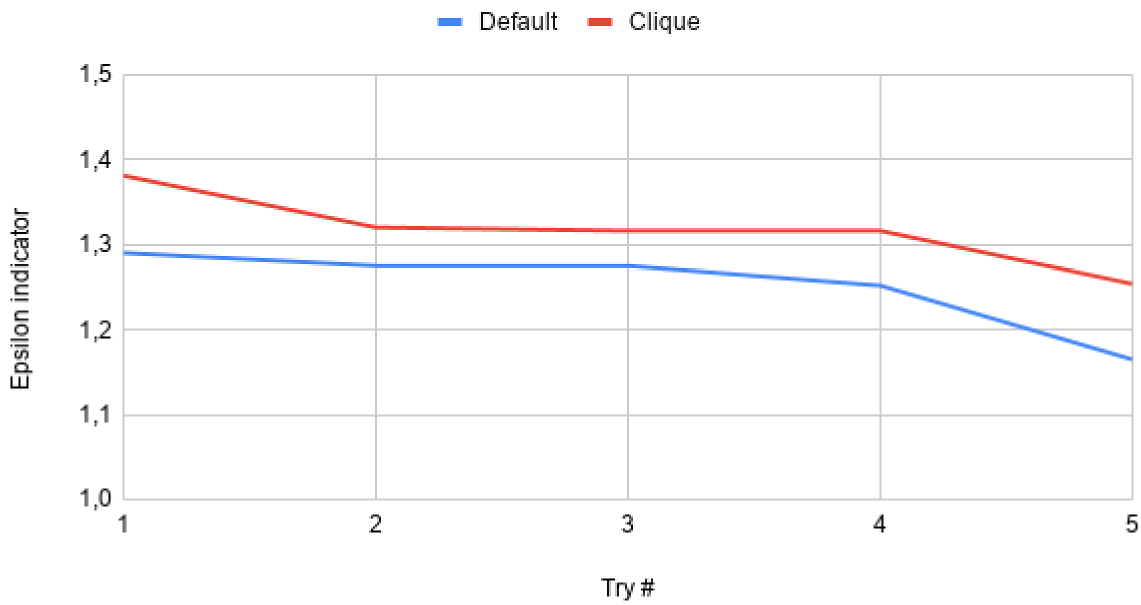
Embedding	N16	N32	N64
`default` try1	1,057	1,165	1,593
`default` try2	1,039	1,275	1,548
`default` try3	1,072	1,275	1,568
`default` try4	1,017	1,290	1,487
`default` try5	1,098	1,252	1,335
`clique` try1	1,092	1,316	1,546
`clique` try2	1,092	1,320	1,510
`clique` try3	1,155	1,381	1,428

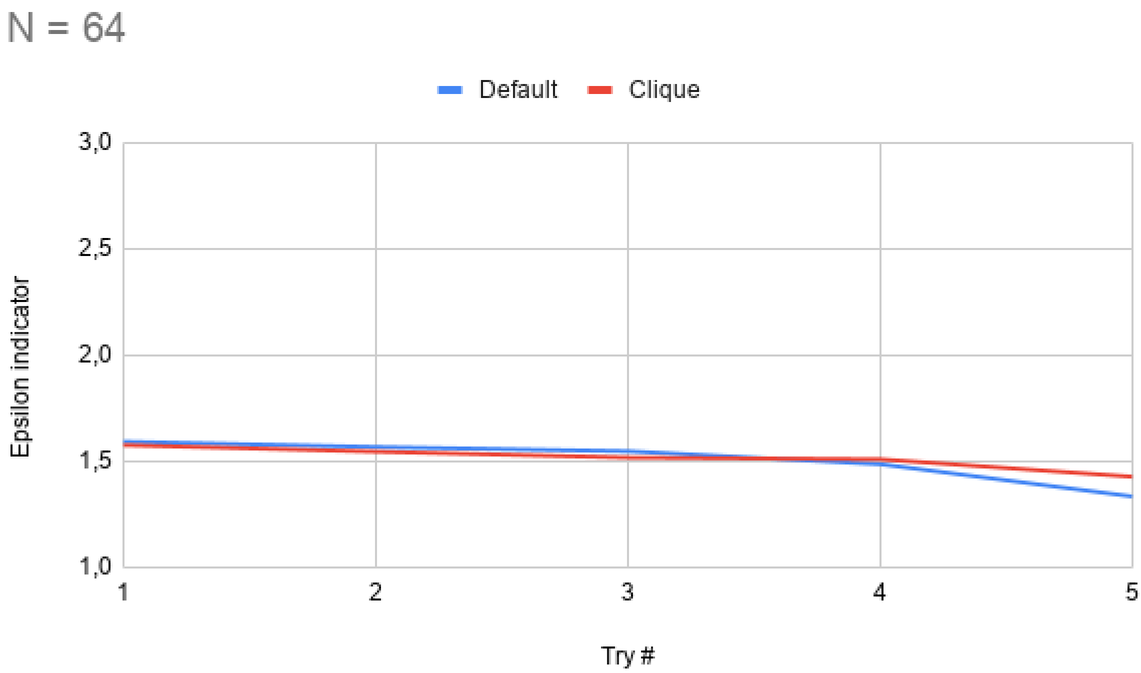
Embedding	N16	N32	N64
`clique` try4	1,126	1,316	1,577
`clique` try5	1,092	1,254	1,518

N = 16



N = 32





Scenario A3 - Dataset **UNFINISHED AND NEEDS CSV FIX!**

For this scenario, we will study the influence from the dataset. Previous scenarios used a `diversified` dataset, with assets as uncorrelated as possible. Therefore, we are going to introduce another dataset, called `strongly_correlated`, from the same source, however, with strongly correlated assets. That is, with assets from the same sub-industry.

The results are executed for sizes `N=32` and `N=64`, with parameters `chain_strength = 1.000 * maxAbs` and `B=0.5`. Since this scenario is small, the results have been repeated two more times, for a total of three tries.

N and Dataset	q values
`N32_diversified`	0, 0.4, 0.9, 2, 3, 9, 100
`N32_strongly_correlated`	0, 1, 6, 10, 70, 90
`N64_diversified`	0, 0.2, 0.4, 0.6, 1.1, 1.3, 1.5, 2, 5, 6, 7, 8, 10, 100, 500
`N64_strongly_correlated`	0, 0.1, 0.2, 0.3, 0.6, 1, 2, 3, 4, 6, 10, 20, 80

Dataset	N32 (AvgChainBreak)	N64 (AvgChainBreak)
`diversified` try1	1.433 (0.00075)	1.516 (0.00409)
`diversified` try2	1.505 (0.00092)	1.435 (0.00413)
`diversified` try3	1.500 (0.00074)	1.501 (0.00384)
`strongly_correlated` try1	1.300 (0.00093)	1.701 (0.00370)
`strongly_correlated` try2	1.352 (0.00103)	1.641 (0.00363)
`strongly_correlated` try3	1.482 (0.00076)	1.668 (0.00360)

Key Takeaways:

The dataset choice does make a significant difference in the performance of the annealer. However, it does not seem to be caused by whether it is diversified or not.