Figure 1: both queries Q1 and Q2 have: (i) two abstract services; and (ii) two single measures. The difference between than is: Q1 has no dependency between abstract services which means there is no local variable. On the other hand, Q2 has one dependency between the abstract services. All concrete services have/covers only one abstract service.

The idea behind this chart is to see how the algorithm works while changing the number of concrete services and CSDs. If the number of CSDs increases, also the number of combinations increases. Consequently more processing time and memory is required.

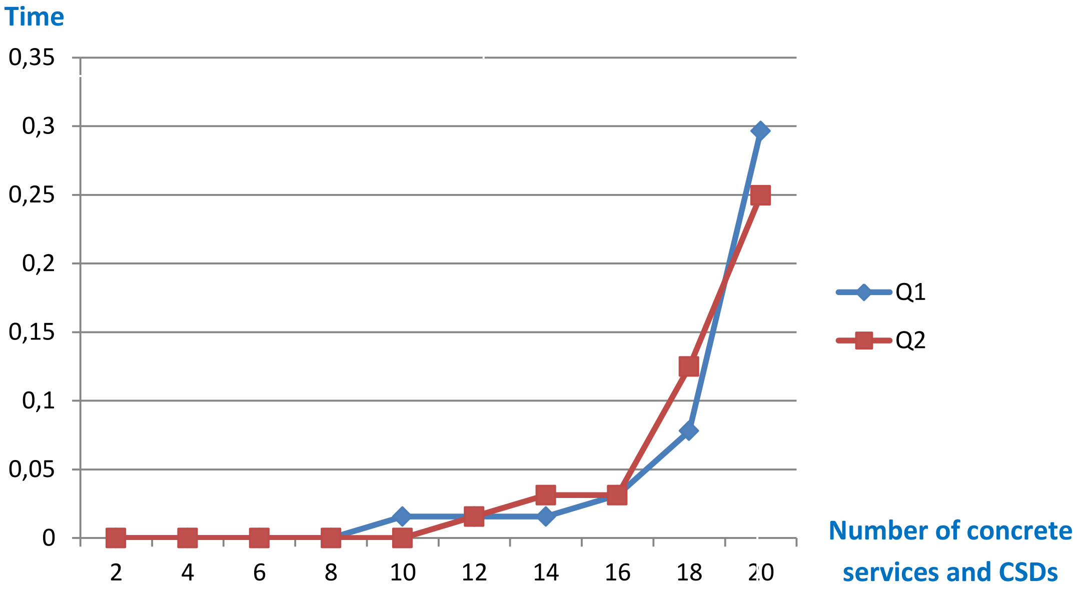


Figure 2: both queries Q1 and Q2 have two abstract services. The difference between than is: Q1 has no dependency between abstract services which means there is no local variable. On the other hand, Q2 has one dependency between the abstract services. All concrete services have/covers only one abstract service.   In this test all, all the examples have 20 concrete services and the number of single measures is being increased by 2 until 20.

The idea behind this example is to see how the algorithm works while changing the number of single measures. As we can see, there is no big variation while changing the number of measures (from 2 to 20). The algorithm processed/worked well.

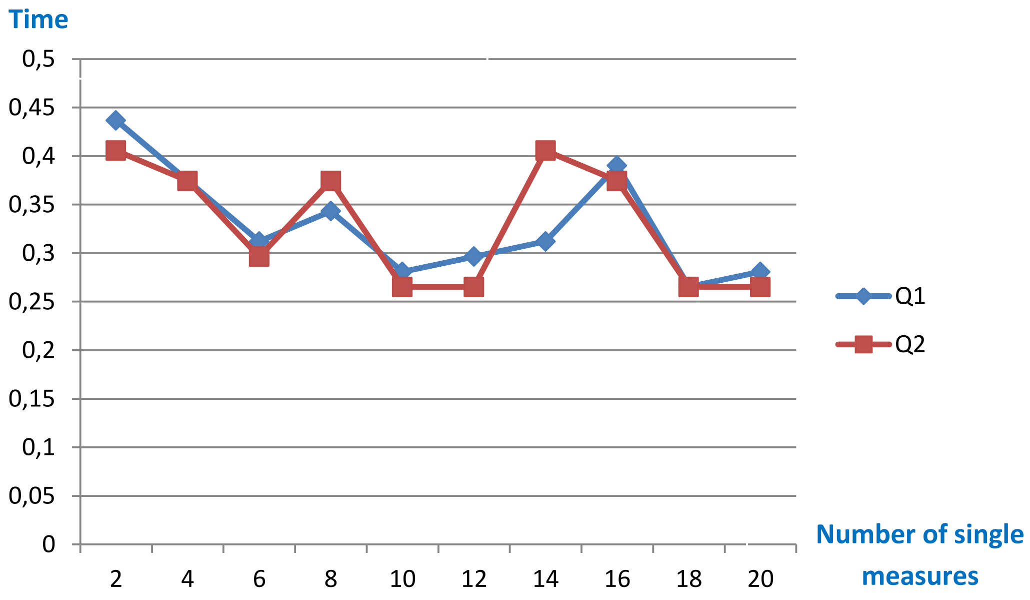


Figure 3: all queries Q1, Q2, Q3, Q4 and Q5 have six abstract services and two single measures. The difference between then is the number of dependencies between abstract services. Q1 has no dependency; Q2 has 2 dependencies; Q3 has two dependencies; Q4 has three dependencies; and Q5 has five dependencies. All concrete services have two single measures and cover all abstract services in the query.

The idea behind this chart is to see how the algorithm works while changing the amount of local variable and the number of concrete services / CSDs. We can see that the number of local variable do not change so much the time to process the query, but as in the first example, increasing the number of CSDs, the number of combination increases the processing time.

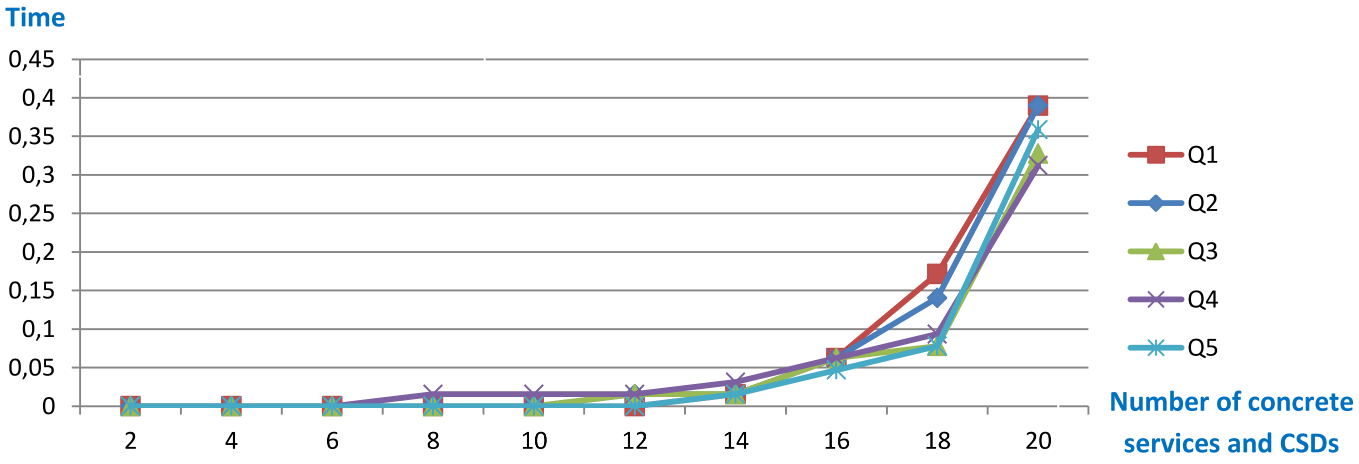


Figure 4: in this example, the query Q1 has two single measures and no dependencies between abstract services. All concrete services cover all abstract services in the query. The number of concrete services and abstract services has been increased by 5 until 30.

The idea behind this chart is to see how the algorithm works while changing the number of abstract services, concrete services and CSDs. As we can see, and also based in the other examples, the number of abstract services and concrete services do not change so much the execution time, but the number of CSD increases it too much.

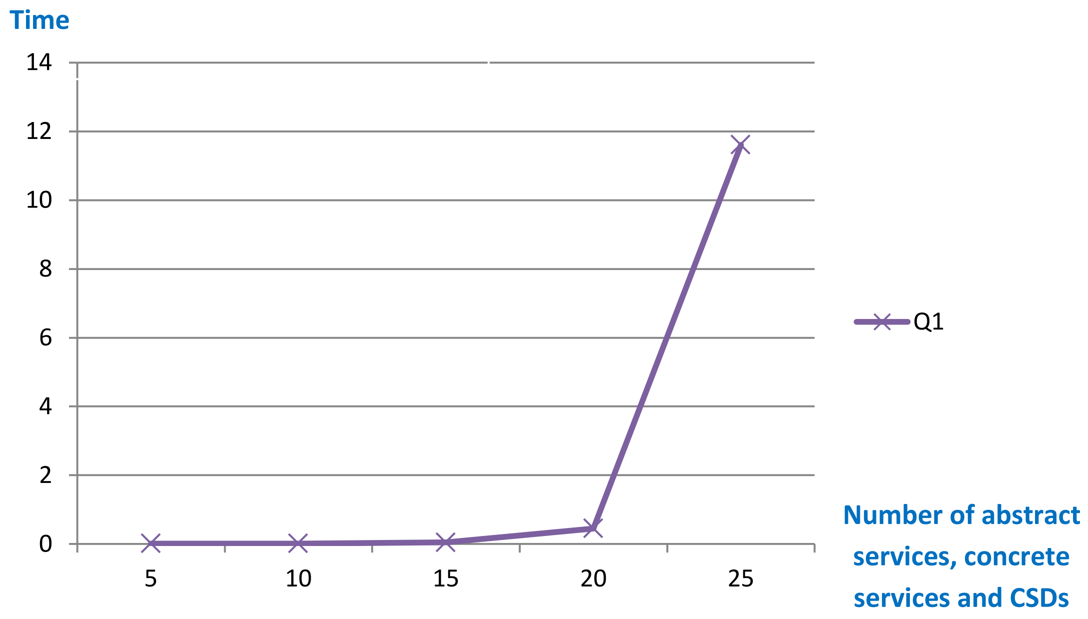
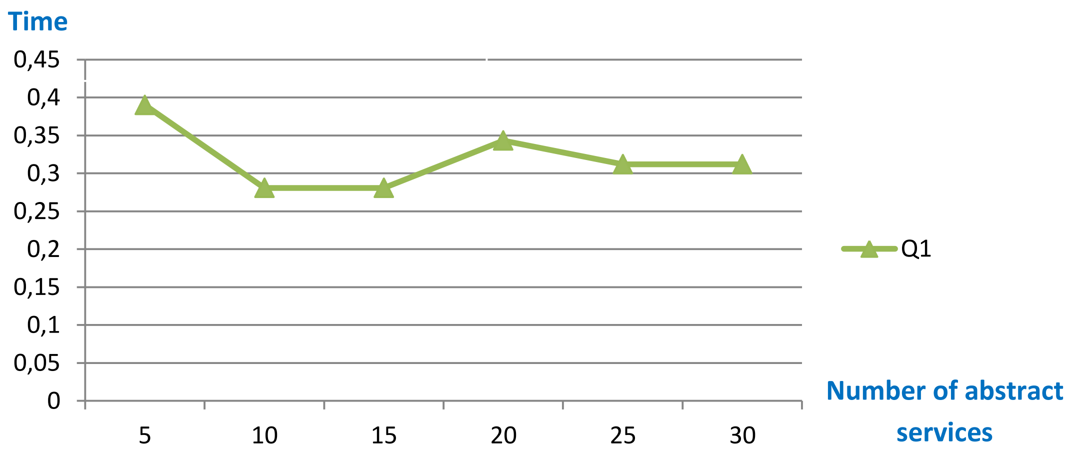


Figure 5: the query Q1 has two single measures and no dependencies between abstract services. All concrete services cover all abstract services in the query. The number of concrete services in all examples is 20. The number of abstract services has been increased by 5 until 30.

The idea behind this chart is to see how the algorithm works while changing the number of abstract services. As we can see, the number of abstract services does not produce considerable variations in the processing time.



I was not able to run examples with more than 25 CSDs. The process of combining all of them consumes too much memory, and the PC has only 8 GB.

In general the size of the query and concrete services, the number of local variables (dependencies), the number of measures, and the number of abstract services do not produce big variation in the execution time. However, the number of CSDs is an important issue. The number of CSDs increases the number of combinations of services, and this process is the one that spend more time. I have been thinking of how to solve this problem, and a ended up with the following solutions:

- Of course, our preferences can reduce the number of CSDs
- Somehow classify the services and try to process the rewritings with a small amount of them and just increase this number if necessary.
- I was thinking about ordering the services based first on the composite measures. For example, if I have a composite measure “total cost” this means that probably the user want to obtain more results using less money as possible. So the idea is to order based on the services that have the lower value of “price per call” (the single measures that compute the value of the composite measure).