In the last days, I have worked in three tasks: (1) correcting the formalization of the Rhone algorithm; (2) creating the pseudocodes for the entire algorithm; and (3) working on the reviewers’ comments.

**Correcting the formalization of the Rhone algorithm**: I did the corrections you proposed for the formalization, and updated the text to new changes. The part regarding the aggregation function was also added, but I still need to work on this text. You can find a file formalization in the dropbox and also attached to the email.

**Creating the pseudocodes for the entire algorithm**: this part is still being developed. Once I have changed some parts of the algorithm, I have one draft handmade, but not included to the latex file. It is not in the form I think is presentable. By the end of this next week it will be added to the latex file.

**Working on the reviewers’ comments**: here is the part I spent more time. The reviewers point out our experiment as the weak part of the work. Based on that I read the paper we have as related work and other to see how they produce their experiments, the amount of services and etc. I realized that the authors produced the experiments in order to show exactly what they want in a well-controlled way (for me it is like manipulating the results), and also some of them do not describe well the details of the experiments, for example the type of the queries and the views used.

To be able to run experiments including more services I made some improvements in the algorithm concerning the method to combine the possibilities of CSDs. I got some good results.

Below you can see the results for queries in which the concrete services involved can fully answer it. In such cases I can run examples with more than 128000 concrete services because, now, the algorithm produce less combinations and also avoid producing unnecessary ones.

Testcase: 0 Number of CSDs: 1000 Number of rewritings: 1000 --Done in: 0.0468003 seconds

Testcase: 1 Number of CSDs: 2000 Number of rewritings: 2000 --Done in: 0.0468003 seconds

Testcase: 2 Number of CSDs: 4000 Number of rewritings: 4000 --Done in: 0.0936006 seconds

Testcase: 3 Number of CSDs: 8000 Number of rewritings: 8000 --Done in: 0.1092007 seconds

Testcase: 4 Number of CSDs: 16000 Number of rewritings: 16000 --Done in: 0.156001 seconds

Testcase: 5 Number of CSDs: 32000 Number of rewritings: 32000 --Done in: 0.3276021 seconds

Testcase: 6 Number of CSDs: 64000 Number of rewritings: 64000 --Done in: 0.5616036 seconds

Testcase: 7 Number of CSDs: 128000 Number of rewritings: 128000 --Done in: 1.2012077 seconds

In this other example, I was able to run around 100 concrete services. However this example is different. The concrete services do not cover the entire query, they cover just one abstract service. The query has six abstract services. In the experiment I produced the concrete services varies from 24 to 96 where for each abstract service in the query I have the same number of concrete services that can cover it. This fact increases the number of rewritings and also the time cost for producing all combinations.

Testcase: 0 Number of CSDs: 24 Number of rewritings: 4096 --Done in: 0.0936006 seconds

Testcase: 1 Number of CSDs: 36 Number of rewritings: 46656 --Done in: 0.2028013 seconds

Testcase: 2 Number of CSDs: 48 Number of rewritings: 262144 --Done in: 0.8580055 seconds

Testcase: 3 Number of CSDs: 60 Number of rewritings: 1000000 --Done in: 2.8392182 seconds

Testcase: 4 Number of CSDs: 72 Number of rewritings: 2985984 --Done in: 7.9092507 seconds

Testcase: 5 Number of CSDs: 84 Number of rewritings: 7529536 --Done in: 19.7653267 seconds

Testcase: 6 Number of CSDs: 96 Number of rewritings: 16777216 --Done in: 43.8206809 seconds

Now I am working on other experiments to see how the algorithm works and build new charts, but I have no idea of how many services are an amount good enough for the evaluation.

Function selectServices(Q, S)

For each Si in S do

If !Si satisfies the quality measures in Q do

Continue

End do

B <- true

For each Aj in Si

If !Q contains Ai do

B <- false

break

End if

End for

If b = true do

Sc <- Sc U {Si}

End do

End for

Return Sc

End function

Function createCSDs (Q, Sc)

For each Si in Sc

If there are mappings h and phi for Si to Q do

For each Ai in Si

G = G U {Ai}

End for

For each Mi in Si

P = P U {Mi}

End for

CSD = <S, h, phi, G, P>

Lcsd = Lcsd U {CSD}

End do

End for

Return Lcsd

End function

Function CSDorganizer (Q, Lcsd)

If Q contains composed measures do

Organize the CSDs based on this measure

Update Lcsd

End do

Return Lcsd

End function

Function produceCombinations (Q, Lcsd)

For each Ai in Q

For each CSD in Lcsd

If CSD contains Ai do

Group <- Group U Ai

CSD <- flag OK

End do

Lgroups <- Lgroups U Group

End for

End for

Lcomb <- produce combinations of Lgroups

Return Lcomb

End function