Evaluating the Scalability of MayBMS, a Probabilistic Database Tool

Kevin Booijink
University of Twente
P.O. Box 217, 7500AE
The Netherlands
k.g.booijink@student.utwente.nl

ABSTRACT

This paper proposes to create a benchmark tool for measuring and comparing the scalability of probabilistic data tools. The benchmark includes a data generator, and can be used to measure the execution time of several queries. The validity of the benchmark will be tested by using it on the MayBMS probabilistic data tool. Firstly, some background is given on the subject of probabilistic data. Then, the state of the art will be explained through related work, and a short introduction on probabilistic data tools is given. After that, the methodology will be explained in detail, results will be displayed, and a conclusion will be drawn from those results. The research in general is discussed, and finally, potential future work on the subject of probabilistic data is proposed.

Keywords

Probabilistic data, uncertain data, benchmark, evaluation, data generation, scalability, database tools

1. INTRODUCTION

Nowadays, most data is stored in large, neatly organized databases. For a lot of projects, it could be very useful to combine (integrate) multiple data sources together. This means there is more data to use, and thus the results are generally more reliable. Unfortunately, it could be the case that 2 (or more) data sources disagree about the value of a certain attribute. In cases like this, Probabilistic Data Integration (PDI) [5] can be used so that all available data can still be used.

Probabilistic data is data of which the value is uncertain. For example, the value could be 32 with a probability of 45%, or 36 with a probability of 55%. The idea behind this is that if numerous sources disagree about a value, this data can still be integrated, and the disputed value is stored as an uncertain value.

A few research prototypes for probabilistic database technology exist, such as MayBMS [3] and Trio [8], as well as probabilistic logic tools such as ProbLog [2] and JudgeD [6], which may also store and query probabilistic data. Unfortunately, due to time constraints and issues in getting the other tools to work properly, this paper focuses purely on the MayBMS tool.

31st Twente Student Conference on IT July 5th, 2019, Enschede, The Netherlands.

2. PROBLEM STATEMENT

As data gets bigger in size, or the data gets more complex, it might take longer for queries to be executed. There is more data to be considered, after all. The extent of this change in execution time is largely dependent on the internal structure of the tool in question. Increasing the amount of conditions a query needs to fulfill might also increase the execution time. In this paper, the scalability of a database tool will thus be referred to as 'the rate of change in execution time of queries'.

Currently, there is no standard for evaluating and comparing probabilistic database tools for how quickly they execute queries on data of increasing sizes. for their scalability on data integration tasks. This research will attempt to create such a standard, with the following research questions:

1. How can the scalability of probabilistic data tools be measured?
   (a) What variables contribute to scalability?

In order to answer these questions, this research provides a benchmark for probabilistic data tools. The benchmark will contain a data generator, capable of generating data integration results of varying sizes. This data can then be queried, and the execution times of these queries can be measured. To validate the benchmark, the scalability of probabilistic data tools can be compared and evaluated, by measuring the execution time of queries multiple times on data of varying sizes, and analyzing the results.

3. RELATED WORK

In Van Keulen[5], the process of PDI is divided in two phases: (i) a quick, partial integration where certain data quality problems are not solved, but instead represented as uncertain data in probabilistic databases, and (ii) continuous improvement by using the data (querying the database, resulting in possible or approximate answers) and gathering further evidence to improve data quality. It explains the formal semantics of probabilistic databases are based on *possible worlds*. In a direct quote from Van Keulen [5]: 'Assuming a single table, let I be a set of tuples (records) representing that table. A probabilistic database is a discrete probability space $PDB = (W, P)$, where $W = I_1, I_2, ..., I_n$ is a set of possible instances, called possible worlds, and $P : W[0, 1]$ is such that $\sum_{j=1..n} P(I_j) = 1$.'

In reality, stating all possible worlds is impossible, and many different representation formalisms have been proposed. (For a thorough overview, see Panse[4], Chapter 3)

Van Keulen later [7] expands on this by introducing 'a generic probabilistic approach to combining grouping data in which an evolving view on integration can be iteratively refined.' It introduces three different integration views for possible worlds where grouping is involved, and gives examples for each: SRC, where each data source is a possible world; COMP, where a possible world is a combination of independent components; and COLL, where a possible world is a collision-free combination of groups. It performs experiments regarding Mean Query Times, World Set Descriptor size, and Number of World Set Descriptors and Random Variable assignments, involving these different integration views. The experiments in this research will be loosely based on the timing methods used in those experiments, but the integration views will not be used.

4. PROBABILISTIC DATABASE TOOLS

At the start of this research, four different probabilistic data tools were considered for possible evaluation. These are MayBMS [3], Trio [8], ProbLog [2] and JudgeD[6]. Unfortunately, due to time constraints and installation issues, it was decided that this research is focused purely on the MayBMS tool, and future work can be done for the other tools. In this section, a short introduction on each of these tools is provided.

MayBMS is a 'complete probabilistic database management system that leverages robust relational database technology'. It is an extension of the Postgres server backend. Therefore, it also has full support of all features of the PostgreSQL 8.3.3, while stating it has essentially no performance loss on PostgreSQL 8.3.3 functionality.

Trio is also a probabilistic database management system, and is based on an extended relational model called ULDBs[1]. It also supports a SQL-based query langauage called *TriQL*.

ProbLog is a tool that combines logic programming with uncertainty. It is a Python package, and its knowledge base can be represented as Prolog/Datalog facts, CSV-files, SQLite database tables, and through functions implemented in the host environment.

JudgeD is a probabilistic datalog. A JudgeD program 'defines a distribution over a set of traditional datalog programs by attaching logical sentences to clauses to implicitly specify traditional data programs'. It is implemented as a proof-of-concept in python, and the implementation allows connection to external data sources.

Unfortunately, some of these tools were rather difficult, if not impossible, to install because of their age, and because of these technical difficulties and time constraints, it was decided to put the main focus on MayBMS, which did manage to install correctly and works properly.

5. METHODOLOGY

The methodology for this project is split into two parts: building the data generator, and evaluating the available database tools using the generated data. Both of these sections will be explained in further detail below.

This program is developed in the Python programming language, as it is a versatile language and thus useful for communicating/integrating with the various other tools that are used. Most of the suggested probabilistic data tools are also developed in Python.

5.1 Data Generator

In order to properly evaluate the scalability of probabilistic database tools, a lot of data is needed. For this research, a data generator will be made, so that experiments can be run again with different (randomly) generated data. This decreases the influence of the data itself in the query tests, and thus increases the validity of our experiments, by making sure the only independent variable is the probabilistic database tool of choice.

The data model this benchmark uses resembles the data set of [7]. It consists of a number of elements, and a number of groups these elements are divided into. For the sake of this research, it does not matter by what criteria these elements are divided. In this case, the uncertainty comes from the integration of multiple data sources. Some of these data sources may agree on the group an element belongs to, while others have that element listed under another group. It may even be possible that the element is not present in that data source at all.

As this research is focused on the scalability of probabilistic data tools, the data generator will need to be able to generate data of varying sizes. In this version of the program, this is done by input prompts. When the program is started, the user is prompted to input the total number of elements and the number of (possible) data sources. The number of possible groups has been set to 15, as this value only influences the amount of database entries, just as the total number of elements. By having only one of these be alterable, it is easier to control the (approximate) amount of database entries that are created.

For each element, there is a set probability, changeable in the source code, for each source to contain information on that element. This results in a list of 'sources' that contain this element. For each of these sources, a random group is selected for the element to be linked to. This selection is random to prevent bias, and the way the data is assigned is not in the scope of this research. The combination of Element, Group, and Source is added to a list, and this continues for every element. This list contains all the data the probabilistic data tools will need, albeit not in a proper structure yet.

5.2 Database Tool Evaluation

In order to measure the scalability, it was decided to focus the research on variables that could influence the execution time. These variables are as follows:

- Query Complexity
- Database size

Both of these were chosen because in theory they have the highest chance of influencing execution times. Increasing the size of the database makes it more difficult for the system to collect the correct values, as there are simply more entries to sort through. Increasing the complexity of the queries gives more conditions for the system to consider, possibly increasing the execution time of queries as well.

Due to all data tools working in (slightly) different ways, it is unfortunately not possible to use the exact same evaluation system for every program. Fortunately, a lot of communication between Python and the various tools can be done using the Python subprocess module, which allows users to control third party programs by parsing strings as inputs. The timing of the queries is done by measuring the execution times of these subprocess statements.

Since MayBMS is an extension of the Postgres back-end, this research is only interested in the additions it has made. In order to translate a regular database into a probabilistic database in MayBMS, one needs to add the probability of the occurrence as a column. The table can then be

converted for probabilistic use using a 'repair-key' statement. The user can then use this newly created table for confidence queries, which are used to calculate probabilities. Thus, the only queries this research is interested in is these confidence queries, with the most common usage being P(*Element x,Group y*), indicating the probability that Element x belongs to Group y, due to the simplicity of the data model that is used. The complexity of the queries is increased by asking for multiple conditions. (For example, querying if Element x belongs to Group y AND if Element a belongs to Group b.)

6. RESULTS

The execution time of one query is very small (in scale of nanoseconds) and timing this small a time frame using Python is tricky. Therefore, each query was executed 1000 times, and the total time for this execution was measured, in order to gain more accurate results.

This research was focused on the database size and the complexity of queries. For each batch of generated data, 10 randomly selected queries were timed for each level of complexity. It was decided to do this for up to 20 levels of complexity, which should be enough to notice any effect the complexity might have on the execution time. This means that for each batch of data, 200 timing results were generated.

Due to the nature of the data generator, the data size was not fixed, but heavily influenceable by the amount of elements in the data. However, when reaching large sizes of data (1.000.000+), MayBMS started reporting memory issues, so the decision was made to cap the data size to 1.000.000 entries.

As CPU time was being measured, it was possible for other processes to execute throughout the timing, making some execution times longer than they actually were. Because of this, strong outliers (values of 0.04 seconds or more) were removed from the timing results.

In Figure 1, the average execution time is plotted against query complexity. This graph mostly resembles a linear progression, indicating that query complexity most likely influences execution times on a linear scale.

In Figure 2, the average execution time is plotted against the data size. There is no identifiable pattern here, other than that the execution time seems to slightly increase for higher data sizes.

For a tabular representation of the results, grouped by data size and complexity, see Appendix A.

All timings were done on a Intel i5-7200U 2.50 GHz CPU.

7. CONCLUSION

As can be seen in Figure 1, the average execution time seems to have a linear increase when the complexity is increased, starting at 0.005 seconds at a complexity of 1, to 0.022 seconds at a complexity of 20. The progression is not perfectly linear, and this is likely due to the randomness for measuring timing. Other processes might have interfered in the timing, or the CPU was running at a slightly different frequency. The average increase per complexity level is $9.339 * 10^{-4}$ seconds (over 1000 queries), calculated by measuring the increase/decrease between each step, and calculating the average of those numbers.

When looking at the data size chart in Figure 2, there is no identifiable pattern at all. The average execution time appears to peak at 250.000 database entries, making a large jump. However, this can be just as easily accredited to the random interference from other processes mentioned before. It does appear, however, that the average execution time is slightly higher for larger databases, averaging roughly 0.015 seconds on data over 200.000 entries, where lower data sizes sit at around 0.01 seconds. A proper explanation for this has not been found.

There is no visualization for the combination of both variables. However, the results indicate that neither variable has influence on each other. For example, there was no significant increase in the execution times of queries with low complexity and low data size as opposed to queries with high complexity and high data size, when compared to the individual results. This indicates both variables are independent from each other.

To conclude, the complexity of a query seems to affect its execution time, by a scale of $9 * 10^{-7}$ seconds per level of complexity. Because of the irregularity of the results, it cannot be determined if the data size has any influence on the execution time, although smaller data sizes seem to be slightly faster. The two variables appear to be completely independent from each other.

8. DISCUSSION

This paper serves as a basis for probabilistic data benchmarking. It provides methods for evaluating the scalability of probabilistic data systems, and applies these methods to the MayBMS system. Unfortunately, there was not enough time to extend this to other probabilistic data tools as well. However, it should be possible for these methods to be applied on other tools as well, and the results of this research can be used for comparison. The paper provides evidence that the complexity of a query has influence on its execution time (at least, in the MayBMS tool), but it is uncertain whether the size of the database also has influence on the execution time. Since the data and the queries used were randomized, the complexity and the size of the data are the only variables that could have influenced the execution time. However, it could be possible that the results that were found are not entirely correct.

Firstly, the CPU time is measured, which includes random processes that are executed in between the queries. It has been tried to keep this influence as small as possible by having this program be the only one that is actively running. However, there is no way to (temporarily) disable the passive background processes.

Secondly, the program was not 100% stable. At times, the MayBMS system reported errors while the program was running. The results of runs where this happened were discarded, but it does indicate the system is not fully waterproof, which could also have influenced the timing.

Lastly, the timing results could be incomplete. Due to uncertainty about exactly the subprocess module operates, it could be possible that only the input is timed. The ideal timing is to have both the input and the response timed in one go. However, it is unproven if this is actually what was being timed.

9. FUTURE WORK

There is a lot of work still to be done in the area of probabilistic data tools and their evaluation. For example, this research was focused on just the MayBMS tool, because of the technical difficulties involved and the limited time for research. In the future, the methods that were used in this research could be applied for different probabilistic data tools, to see if the scalability differs per tool.

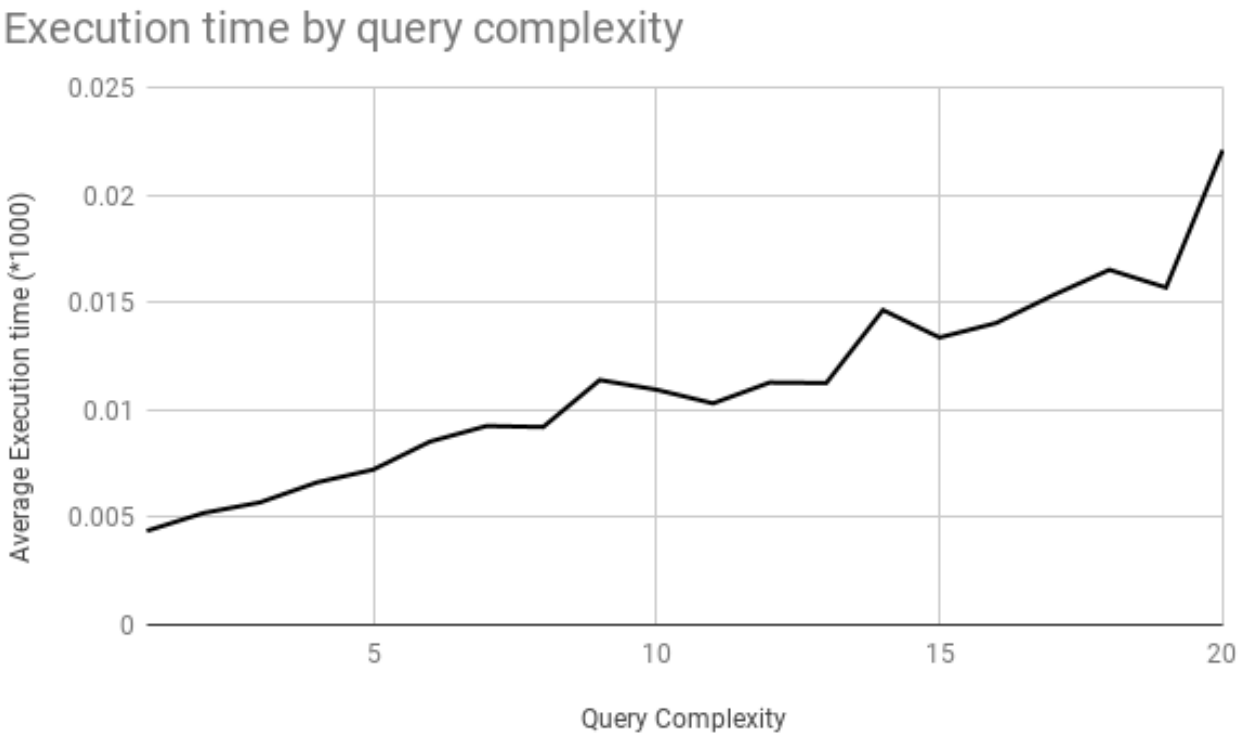

Figure 1. Execution time of queries based on complexity

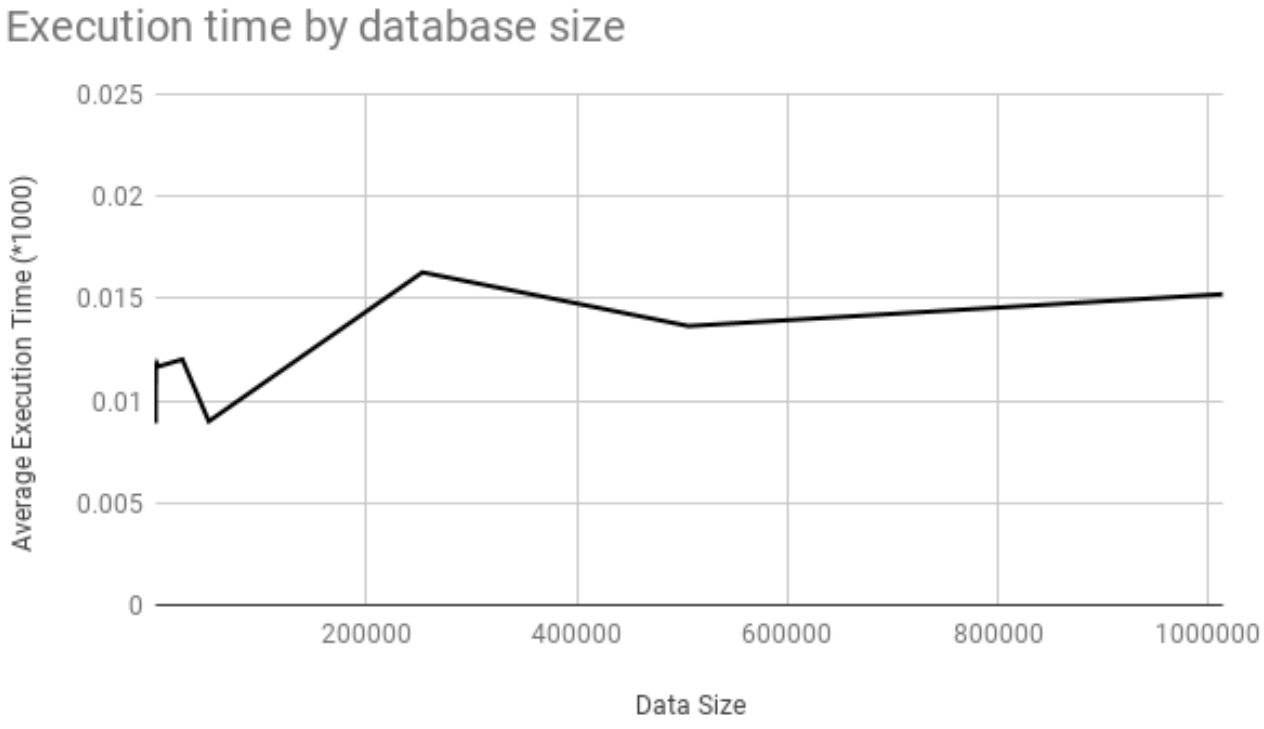

Figure 2. Execution time of queries based on data size

Furthermore, the database model used in this research was rather simple. It might be worth replicating this research with more complex database models, to see if the complexity of the database itself has influence on the execution time of queries. A more complex database also allows for more variety in queries, something that was lacking with this particular model.

What could also be worth looking into is building a standardized layout for evaluation, where only a small amount of adjustment is required in order to make it work for different programs. Since this program was focused on only one probabilistic data tool, it is unsure if this layout will also work for different tools, although the layout is simple enough that it will probably work.

Lastly, the stability of the current program can be increased. Due to the limited time frame of this research, not a lot of focus was put on stabilizing. For example, subjects like error handling, exceptions, etc. are not present in the tool, and most problem identification had to be done manually. It would be great if this could be added, in order to decrease the necessary human input for the program.

10. REFERENCES

[1] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 953–964. VLDB Endowment, 2006.

[2] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilisic prolog and its application in link discovery. In *Proceedings of the 20th International Join Conference on Artificial Intelligence (IJCAI-07)*, pages 2462–2467, 2007.

[3] C. Koch. Maybms: A system for managing large uncertain and probabilistic databases. In *Managing and Mining Uncertain Data*. Chapter 6, Springer, 2008.

[4] F. Panse. *Duplicate Detection in Probabilistic Relational Databases*. PhD thesis, University of Hamburg, 2015.

[5] M. van Keulen. Probabilistic data integration. In *Encyclopedia of Big Data Technologies*, pages 1–9, Cham, 2018. Springer International Publishing.

[6] B. Wanders, M. van Keulen, and J. Flokstra. Judged: a probabilistic datalog with dependencies. 2 2016. Workshop on Declarative Learning Based Programming, DeLBP 2016 ; Conference date: 13-02-2016 Through 13-02-2016.

[7] B. Wanders, M. van Keulen, and P. van der Vet. Uncertain groupings: Probabilistic combination of grouping data. In *Proceedings of the 26th International Conference on Database and Expert Systems Applications, DEXA 2015*, Lecture Notes in Computer Science, pages 236–250. Springer, 9 2015. 10.1007/978-3-319-22849-5_17.

[8] J. Widom. Trio: A system for data, uncertainty, and lineage. In *Managing and Mining Uncertain Data*. Springer, 2008.

APPENDIX

A. FULL TABLE OF RESULTS

Data size	Complexity	Average Execution Time
29	1	0.0034655861
29	2	0.0038105437
29	3	0.0043084592
29	4	0.0049928619
29	5	0.0049170437
29	6	0.0065853836
29	7	0.0066099641
29	8	0.0063971595
29	9	0.0088321771
29	10	0.0075910696
29	11	0.0081230429
29	12	0.0081313498
29	13	0.0088708039
29	14	0.0141820415
29	15	0.0102466697
29	16	0.0106348982
29	17	0.0115287068
29	18	0.0154982119
29	19	0.0128984183
29	20	0.0211623365
32	1	0.0035591127
32	2	0.004607843
32	3	0.0046039539
32	4	0.0052277547
32	5	0.005071965
32	6	0.0066604089
32	7	0.0061725368
32	8	0.006699111
32	9	0.0099099056
32	10	0.0077791049
32	11	0.0085694944
32	12	0.0088009515
32	13	0.0096338185
32	14	0.0136662286
32	15	0.0108143246
32	16	0.0105124109
32	17	0.01102694
32	18	0.01157598
32	19	0.0117650345
32	20	0.0214137675
48	1	0.0036354
48	2	0.004702
48	3	0.0048787
48	4	0.0057308
48	5	0.00586
48	6	0.0072855
48	7	0.0063317
48	8	0.0080763
48	9	0.0109112
48	10	0.0082679
48	11	0.0085594
48	12	0.0090198
48	13	0.0101404
48	14	0.015341
48	15	0.0115066
48	16	0.0110647

Continued on next column

Continued from previous column

Data size	Complexity	Average Execution Time
48	17	0.0118004
48	18	0.0123696
48	19	0.0122446
48	20	0.0228794
50	1	0.0039208725
50	2	0.004001184
50	3	0.0046344623
50	4	0.0052085737
50	5	0.0056512113
50	6	0.0072018596
50	7	0.0062550381
50	8	0.0067906364
50	9	0.008581237
50	10	0.0079413511
50	11	0.008557223
50	12	0.0089451113
50	13	0.0095015142
50	14	0.0142586904
50	15	0.0109192917
50	16	0.0111455758
50	17	0.0110739867
50	18	0.011548983
50	19	0.0119788962
50	20	0.0206607584
52	1	0.0043312
52	2	0.0063668
52	3	0.00623745
52	4	0.00658845
52	5	0.0089099
52	6	0.0089576
52	7	0.01263415
52	8	0.01064395
52	9	0.0116718
52	10	0.01650995
52	11	0.00519605
52	12	0.0062751
52	13	0.0066231
52	14	0.006789
52	15	0.00911435
52	16	0.00899475
52	17	0.0118031
52	18	0.0104245
52	19	0.01160585
52	20	0.0174046
259	1	0.0045915
259	2	0.0055823
259	3	0.0045401
259	4	0.0079407
259	5	0.0074695
259	6	0.0093396
259	7	0.0085772
259	8	0.0095913
259	9	0.0120462
259	10	0.0099934
259	11	0.01256
259	12	0.0107636
259	13	0.0135829
259	14	0.0173189
259	15	0.010639

Continued on next column

Continued from previous column

Data size	Complexity	Average Execution Time
259	16	0.0157416
259	17	0.0141345
259	18	0.0160481
259	19	0.0156976
259	20	0.0271842
1229	1	0.0041375
1229	2	0.0047341
1229	3	0.0064352
1229	4	0.0068183
1229	5	0.0071763
1229	6	0.0094467
1229	7	0.0073483
1229	8	0.0087032
1229	9	0.0139239
1229	10	0.0109329
1229	11	0.0119867
1229	13	0.0125193
1229	14	0.0185021
1229	15	0.014541
1229	16	0.0127042
1229	17	0.0165207
1229	18	0.0146461
1229	19	0.0140554
1229	20	0.0259799
2573	1	0.0042478
2573	2	0.0048457
2573	3	0.0058301
2573	4	0.0072029
2573	5	0.0062792
2573	6	0.0095859
2573	7	0.0091995
2573	8	0.009687
2573	9	0.0122943
2573	10	0.0107391
2573	11	0.0105899
2573	12	0.015362
2573	13	0.0126991
2573	14	0.0170183
2573	15	0.0132298
2573	16	0.0132072
2573	17	0.0147358
2573	18	0.0162331
2573	19	0.0149208
2573	20	0.025575
25449	1	0.0046957
25449	2	0.0051455
25449	3	0.0054257
25449	4	0.006692
25449	5	0.0057239
25449	6	0.0089451
25449	7	0.0087357
25449	8	0.0107315
25449	9	0.0129706
25449	10	0.0106672
25449	11	0.0109464
25449	12	0.0104736
25449	13	0.0140068
25449	14	0.0187443
25449	15	0.0132876

Continued on next column

Continued from previous column

Data size	Complexity	Average Execution Time
25449	16	0.0154586
25449	17	0.0159229
25449	18	0.015668
25449	19	0.019346
25449	20	0.0268313
50565	1	0.0033791
50565	2	0.0039373
50565	3	0.0046275
50565	4	0.0055455
50565	5	0.0055902
50565	6	0.0066246
50565	7	0.006156
50565	8	0.0067634
50565	9	0.0092906
50565	10	0.007982
50565	11	0.0093081
50565	12	0.0095727
50565	14	0.0137609
50565	15	0.0105793
50565	16	0.0113629
50565	17	0.0114945
50565	18	0.011933
50565	19	0.0119397
50565	20	0.0202056
253423	1	0.0061134
253423	2	0.0075854
253423	3	0.0091407
253423	4	0.0089108
253423	5	0.0098249
253423	6	0.0105713
253423	7	0.0136325
253423	8	0.0126364
253423	9	0.0141915
253423	10	0.014517
253423	11	0.015392
253423	12	0.021658
253423	13	0.0158406
253423	14	0.0187465
253423	15	0.0205983
253423	16	0.0216967
253423	17	0.0249478
253423	18	0.0221377
253423	19	0.0333713
253423	20	0.0241635
506196	1	0.0049108
506196	2	0.0045898
506196	3	0.0053401
506196	4	0.0069493
506196	5	0.0102187
506196	6	0.0092784
506196	7	0.0093178
506196	8	0.0090318
506196	9	0.0104469
506196	10	0.0097809
506196	11	0.0142528
506196	12	0.0110455
506196	13	0.0120268
506196	14	0.0128965
506196	15	0.0145084

Continued on next column

Continued from previous column

Data size	Complexity	Average Execution Time
506196	16	0.026087
506196	17	0.0279514
506196	18	0.0416161
506196	19	0.0165284
506196	20	0.0163322
1013316	1	0.006012
1013316	2	0.0066978
1013316	3	0.0076635
1013316	4	0.0085192
1013316	5	0.0097272
1013316	6	0.0101377
1013316	7	0.0161742
1013316	8	0.0127238
1013316	9	0.0130299
1013316	10	0.0142476
1013316	11	0.0153028
1013316	12	0.0164683
1013316	13	0.0159388
1013316	14	0.0174093
1013316	15	0.0283916
1013316	16	0.0193554
1013316	17	0.0202652
1013316	18	0.0215618
1013316	19	0.0221473
1013316	20	0.0225507

Concluded