Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

Trading Accuracy for Performance in Data Processing Applications

Gala Barquero[a] Javier Troya[b] Antonio Vallecillo[a]

a. Universidad de Málaga, Spain. {gala,av}@lcc.uma.es

b. Universidad de Sevilla, Spain. jtroya@us.es

Abstract Applications that need to process large volumes of data to make informed decisions, such as stream processing systems or those dealing with social networks, usually impose strong requirements on resources like memory, processing time, disk or network latency. One approach to address this problem is to reduce the amount of data to be processed, at the cost of decreasing the confidence on the results. Estimating the errors of such *approximate* solutions becomes a critical issue. In this paper, we explore different approximation possibilities depending on how the data is organized, and what information needs to be obtained from them. We propose an approach to estimate the accuracy of these approximate solutions in the context of data processing systems, in terms of the precision and recall of the results obtained. A case study is used to validate the proposal and to evaluate the performance of different types of approximations.

Keywords Data Processing; Approximate Transformation; Accuracy.

1 Introduction

The drastic increase in the amount of information produced by existing data sources requires the efficient processing of data flows in real time, both to make informed decisions and to detect situations of interest that require instantaneous reactions. An example of the importance of being able to efficiently process large amounts of information flows is shown in the analysis carried out by the Spanish bank BBVA on the economic impact of Barcelona's 2012 *Mobile World Congress* [BBV13]. The study required the online analysis of all credit card transactions during two weeks. Another example is the need for real-time analysis of streams of information in social networks or weblogs in order to detect possible terrorist attacks [PP11, YN07].

In this context, Complex Event Processing (CEP) is a useful technology for processing of data flows that analyses streams of information represented as a sequence of simple events and obtains conclusions from them, represented as complex events [CM12, EN10, Luc02, Luc12]. Several CEP engines and Event Processing Languages (EPLs) exist, such as the Esper language [Esp19]. However, technologies

Gala Barquero, Javier Troya, Antonio Vallecillo. *Trading Accuracy for Performance in Data Processing Applications* . In Journal of Object Technology, vol. 18, no. 2, 2019, pages 9:1–24.
doi:10.5381/jot.2019.18.2.a9

like this only permit dealing with information sources in which events are not related among themselves.

In reality, information is commonly generated as a sequence of structured and interconnected data forming a graph, where we can distinguish between persistent and transient information. The former refers to data stored in the system in a persistent way (e.g., users, products or shops). Transient information refers to data that are temporarily stored (e.g., tweets, orders, bank transactions) and are discarded after some period of time—i.e., transient information *expires*. The interconnections between these data need to be processed, too, which inevitably implies a decrease in systems performance [SIRV18].

Our proposal for dealing with this increasing amount of information is based on the claim that most of the data that needs to be processed for decision making is not significantly relevant, particularly with large volumes of data. Therefore, the goal is to be able to select the relevant data subset that would still yield valid results. For this we need to answer two questions: (*a*) how to select the subset of data that is relevant for a given query; and (*b*) how to estimate the error we are making when discarding some of the input data (the one that we have considered as not relevant). Here, the goal is to find the right balance between the performance of our queries and the accuracy of their results.

This problem is of application in those systems that deal with large amounts of data, do not need extremely accurate results, and require fast response times. Recommendation systems on Facebook, Netflix or Amazon are examples of these applications. This work derives from a previous paper [TWBV14], where the concept of *Approximate Model Transformations* (AMT) was introduced to find a balance between performance and accuracy of a model transformation. Sampling techniques were used in a wireless sensor network example to show the effects of selecting certain subsets of elements. However, the study was only able to manage information composed of sequences of events, and not as graphs of highly interconnected pieces of information, some of which can be persistent whilst other are transient. We continued that work to deal with graphs in [BBTV18], defining an extension of CEP systems for managing graph-structured data. However, in that work no indication was given on how the windows (both spatial and temporal) that contained the relevant data were chosen; and no methods for estimating the (loss of) accuracy of the results were provided.

In this paper, we consider large amounts of information in the form of graph-based structures. We explore different approximation possibilities (random, temporal and spatial approximations) depending on (*i*) how the data is organized and (*ii*) what information we need to obtain from the data. Furthermore, we propose a method that permits estimating the errors produced when applying approximations, with the goal of finding the right balance between performance gain and accuracy loss when approximating data. To illustrate our proposal, we use a simplified version of the Amazon ordering service. We implement the solution using in-memory graph databases, which consume less execution time than solutions based on disk, and we choose Gremlin as a query language due to its intuitiveness and benefits over other graph query languages [HP13].

The structure of this paper is as follows. Sect. 2 presents our running example. Sect. 3 describes three different types of approximations, how to calculate the error induced by them, and their implementation with Gremlin and TinkerGraph technologies. Sect. 4 evaluates the results of the experiments. Finally, Sect. 5 relates our work to other similar proposals and Sect. 6 concludes and outlines future lines of work.

2 Running Example

As running example let us use an Amazon ordering service. It is a simplification of the real service that Amazon offers, but complex enough since it considers many of its relevant features. The metamodel of this service is depicted in Figure 1. We can see that we have *Orders* placed by *Customers*, who live in a *Geographical Area*. *Orders* are composed by different *Items*, and each *Item* corresponds to a *Product*. In addition, each *Product* belongs to a *Department*. *Customers* can be related with *Products* in three ways: (i) they can write a *Comment* on a *Product*, (ii) wish a *Product*, (iii) and *Products* can be recommended to *Customers*. Besides, *Products* can be included in an *AdCampaign*, *Orders* are destined to a *Geographical Area*, and *Geographical Areas* are connected to other areas.

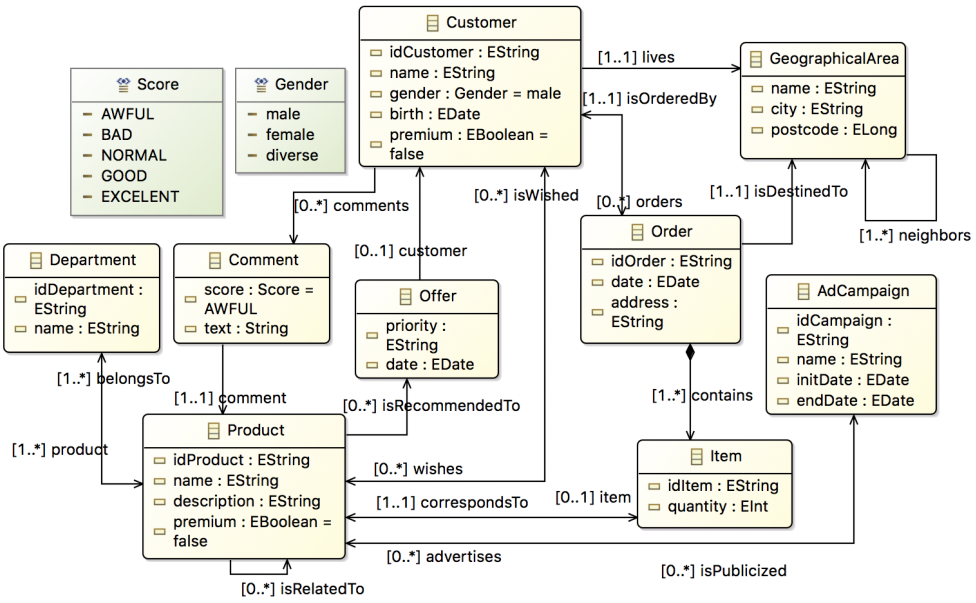


Figure 1 – The Amazon Example Metamodel.

Considering this system, we are interested in defining some queries that represent ways of processing data in this context. As a result, these queries can modify the source model or provide some other data, such as returning a specific set of products.

Q1. CreateAdCampaign: if a product has been ordered more than 1000 times during an advertising campaign period, and a relationship *isPublicized* between the product and the campaign does not exist, then the query creates the relationship.

Q2. UnpopularStock: returns all products that have been ordered by less than 3 customers during last month.

Q3. RelatedProducts: This query creates a link *isRelatedTo* if two products have been included in at least 100 common orders during the last month.

Q4. OlympicGamesTrending: considering we have a Rio de Janeiro Oympic Games *AdCampaign*, the query obtains the products that were ordered at least 100 times in Rio de Janeiro since the beginning of August 2016 until the end of the celebration of the Olympic Games. In this case, the query adds a relationship *isPublicized* between the products and the Olympic Games campaign.

Q5. RecommendsPack: if a customer has ordered *Product1* at least 5 times in

different orders in the last month and this product is related to *Product2* (through a *isRelated* connection), then an offer for *Product2* is created for the customer. Such an offer has a priority of 1—highest priority. If *Product1* is related to *Product3* indirectly through another product, then an offer for *Product3* with priority 2 is created for the customer. In this case, we say that *Product1* is related with *Product3* in two hops. Similarly, if *Product1* is related to *ProductN* in n hops, the query would create an offer with priority n. In this query, we consider offers from priority 1 to 3.

We can observe that queries create objects of type *Offer* and relationships of type *isRelatedTo* and *isPublicized*, which are not critical for the appropriate functioning of the service, so they can be approximated if the performance improvement is considerable. Also note that the queries are not static, i.e., their result will be different depending on when queries are performed. Specifically, **Q2**, **Q3** and **Q5** consider data of the *last month*. As for **Q1**, it is associated to a specific advertising campaign period. Finally, **Q4** depends on when the celebration of the Olympic Games ends.

3 Approach

Our approach is based on the fact that not all information contained in the model is needed in order to reach conclusions or make decisions. The idea therefore is to make approximations in the source model in order to improve the performance of the queries. In this section, we describe the approximations in the source model that we have explored, and describe the metrics that we propose for computing the errors of the results, expressed as the differences between the expected and the resulting query outputs. Finally, we describe the technologies we use in the implementation and show the implementation of a couple of queries with different approximations.

3.1 Reducing source data

As mentioned in the introduction, we distinguish between persistent and transient information. The former is stored persistently in the system, while the latter is discarded after some time. In our proposal, all the information will be stored in models, which at this level of abstraction are basically directed graphs with typed nodes and arcs, and whose nodes may have typed attributes. Please note we use the terminology used in models, so we use "objects" and "relationships" instead of "nodes" and "arcs", respectively.

Let us introduce the following terms for classifying the different kinds of models that we will consider in our approach:

- Source Model. It is the complete data model that serves as input for the queries. In practice it can be so large that it is impossible to be considered in full.
- Pattern Model. Queries made on this type of systems, such as the queries presented in Section 2, typically focus on a part of the model. Specifically, they (*i*) narrow down a time range and (*ii*) specify the elements[1] that participate in the query. For instance, most of the queries described above focus on the data placed during last month. As for the elements that participate in the query, we can see for instance that **Q3** considers orders and products. The Pattern Model is the subset of the Source Model that considers the elements that participate in the query, filtering by time range and element type.

[1]Throughout this paper, by *element* we refer to both the objects and the relationships of a model.

- Approximate Model. The Pattern Model can still be very large in these kinds of systems. For this reason, we want to perform approximations, so that queries can be executed faster. An Approximate Model is a subset of the Pattern Model.
- Optimal Model. Among all possible Approximate Models, by *Optimal Models* we refer to those that meet the *best* trade-off between performance and accuracy. Please note that several Approximate Models could be considered Optimal Models, and this decision is ultimately made by the user.

We may recall that, in a specific scenario, there is only one Source Model and, for each query, there is only one Pattern Model. For each Pattern Model there can be many different Approximate Models, where more than one can be considered as Optimal Models. In the following we explain the three approximation techniques that we explore in this paper.

3.1.1 Temporal Approximations

Since incoming data typically count on a timestamp and data flows can be considered infinite (think for instance of all the information stored in Facebook during its lifetime), we can build *temporal windows* filtering by the data timestamp. A temporal window will be typically determined by the query, since queries are normally focused on a specific time range. The idea is to narrow down the Source Model by selecting the subset of the model indicated by the temporal window.

3.1.2 Spatial Approximations

Information systems such as the one in our running example are typically composed of data structured as a graph. This means that objects are linked among each other through different types of relationships. In this way, we can navigate a model by starting in one object and traversing through the existing relationships. We define the concept of *hop*, which is the navigation from one object to another by the relationship that links them. For instance, in our running example and starting from one order (cf. Figure 1), we can determine in one hop the geographical area the order is destined to, by navigating the *isDestinedTo* relationship. Also, objects can be connected to other objects of the same type. For instance, from a geographical area we can reach, in one hop, all its neighboring geographical areas, through the *neighbors* relationship. Also, in two hops, we can reach all geographical areas that are neighbors of its neighbors, and so on. In this way, we can obtain *spatial windows* starting from one object and considering other objects reachable in n hops.

The concept of spatial window, which considers the idea of *spatial vicinity*, was presented as an extension of CEP windows [BBTV18]. Indeed, there are different strategies to define the vicinity graph of an element, depending on how we navigate through the graph structure, and the goal we pursue. Representative examples of algorithms for creating relevant vicinity graphs of nearby elements are used for finding related pages in the WWW [DH99, Kle99]. These algorithms use different strategies, e.g., going through the parents and children of a page, and then visiting the children and parents of those—using a backward-forward and forward-backward strategy. We could also traverse the graph moving only forward or backward, or using any other traversal strategy: in-breadth, in-depth, topological, hybrid, etc. Traversal could be done through any kind of link, or we could navigate the graph through some selected kinds of relationships.

3.1.3 Random Approximations

Approximate Models can also be obtained by applying random sampling techniques. This means that the decision on which elements of the Pattern Model will conform the Approximate Model is randomly made. For instance, we can assign a probability to each element of the Pattern Model to be included in the Approximate Model. Also, we can do approximations by element type. For example, we could determine that only 30% of the orders should be included in the Approximate Model. Many other random approximation techniques can be applied, as the authors proposed in [TWBV14]. This is also a good approach when only transient data needs to be approximated. Of course, random approximations can be combined with the other two.

3.2 Measures for accuracy

Since we are trading accuracy for performance, a very important aspect in our approach is to be able to measure both. We consider performance in terms of execution time of the queries. Regarding accuracy, we use the measures of *precision*, *recall* and *accuracy* [MRS08]. These three measures are defined by formulas that include the concepts of true positives, false positives, false negatives and true negatives. In our context, we define and calculate them as follows:

- True Positives (TPs): number of elements created or returned as the result of a query on both the Approximate Model and the Pattern Model.
- False Positives (FPs): number of elements created or returned as the result of a query on the Approximate Model but not created or returned when running it on the Pattern Model.
- False Negatives (FNs): number of elements created or returned as the result of running a query on the Pattern Model but not created or returned when running it on the Approximate Model.
- True Negatives (TNs): number of elements that are neither created nor returned as the result of running a query on both the Approximate Model and the Pattern Model. While the calculation of the other three values is straightforward, the calculation of TNs is more complex. First, we need to consider the maximum number of elements the query could create or return. For instance, in **Q2**, which returns all products ordered by less than 3 customers, if we have a total of 500 products, the total amount of products that could be returned in principle is 500. Let us name this amount as P_{re} (possibly returned elements). From this number, we need to subtract the amount of elements that are created or returned when the query is run on the Approximate Model, which is reflected in $(TP + FP)$. In summary, TNs are calculated as: $TN = P_{re} - (TP + FP)$.

Then, the three accuracy measures can be calculated as follow [MRS08]:

- Accuracy: it is the most usual performance measure. In our context, it describes the effect of FPs and FNs when running queries on the Approximate Model. It is calculated as follows: $Accuracy = (TP + TN)/(TP + TN + FN + FP)$.
- Precision: this measure is useful to determine how accurate the model is when FPs are costly. For example, in email spam detection, a FP may cause loss of important information when a non-spam email is identified as spam. It is calculated as follows: $Precision = TP/(TP + FP)$.

- Recall: this measure computes how accurate the model is when FNs are costly. As an example, a FN on illness detection may cause catastrophic consequences on the life of the patient. It is calculated as follows: $Recall = TP/(TP + FN)$.

3.3 Implementation

In this section we provide some details of the technologies we have used in our implementation and the reason for choosing them, and show the implementation of a couple of queries using different approximation techniques.

3.3.1 Gremlin

Gremlin [Apa19] is a graph traversal machine and a graph traversal language distributed by Apache TinkerPop that permits expressing complex traversal queries and manipulating graph databases. Graph databases are databases that use specific structures to store data in terms of edges and nodes. As a traditional database, they allow to access elements and apply queries on them. This feature is made possible by using graph query languages, whose main advantage over traditional query languages is that they allow access to graphs without using multiple join methods.

Gremlin traversal machine permits to coordinate multi-machine executions and allows the language to be used for Online Transactional Process (OLTP) and Online Analytics Process (OLAP) graph databases.

An important benefit of using Gremlin is the expressive power and usability of its syntax. It permits to express queries over graphs more elegantly than other languages like Java. Thus, while Java needs a heavier logic to write a query for a graph, Gremlin does it in a few lines. Gremlin can also be embedded within various programming languages like Java, Python or Scala.

The Gremlin language outperforms other graph query languages such as Cypher [Neo19], a declarative language to work with graphs developed by Neo4j. For example, some studies [HP13] show that Gremlin outperforms Cypher on queries that imply vicinity as Gremlin is designed to express complex traversal queries. These kinds of queries are easier to write in Gremlin than in Cypher, even though Cypher is usually easier to write. Besides, one of our highest priorities is to implement and to study vicinity queries. Another reason to use Gremlin instead of Cypher is the way Gremlin can be used with different graph databases, including in-memory graph databases. Cypher is used with Neo4j which does not have an available in-memory version. For all these reasons we chose Gremlin as graph query language in our research.

3.3.2 TinkerGraph

TinkerGraph is a light in-memory graph database developed by TinkerPop and designed to run in a single machine. Although TinkerGraph is an in-memory implementation it has the option to persist on disk. It is developed, like Gremlin, to enable OLTP and OLAP functionality [Tin19]. Some examples of uses for TinkerGraph are analysis of in-memory graphs or analysis of subgraphs for large graphs that can not be stored in memory, writing unit tests, transformating graphs, creating nodes and edges and adding properties for elements.

When storing graphs in databases, a major issue is the performance bottlenecks when managing the data. This is why we have chosen TinkerGraph to store our graph structures, since it is a lightweight database and fully compatible with Gremlin.

```
1  // Select Product elements
2  graph.traversal().V().hasLabel("Product").as("product1")
3  // Select Order elements that contains the products inside a temporal window
4  .in("contains").where(__.values("date").is(P.inside(initTime, endTime)))
5  // Filter orders by probability with coin step (Random approximation)
6  .coin(prob).as("order1")
7  // Select products in the same order
8  .out("contains").as("product2").where(P.neq("product1"))
9  //Check there is not a previous relationship "isRelatedTo" between products
10 .not(__.select("product1").outE("isRelatedTo").inV().where(P.eq("product2")))
11 //Count the number of matches between products and filter when they are at least 100
12 .select("product1","product2").groupCount().unfold().where(__.select(values).is(P.gte(100)))
13 // Add new elements to the graph
14 .select(keys).addE("isRelatedTo").from("product1").to("product2").iterate();
```

Listing 1 – Q3 for Random and Temporal approximation.

```
1  // Select Olympic Games campaign
2  graph.traversal().V().hasLabel("AdCampaign").has("name", P.eq("Olympic Games")).as("campaign")
3  // Take property "endDate"
4  .values("endDate").as("end")
5  // Select Geographical Area with postal code 24495L
6  .V().hasLabel("GeographicalArea").has("postcode", P.eq(24495L))
7  // Traverse the graph through relationship "neighbors" with vicinity
8  .repeat(__.out("neighbors")).times(hops).emit().as("area").dedup("area").select("area")
9  //Select orders destined to the area and ordered before "endDate" property
10 .in("isDestinedTo").filter(__.values("date").where(P.lte("end")))
11 // Select products contained by the orders
12 .out("contains").as("product")
13 //Check there is not a previous relationship "isPublicized" between products and campaign
14 .not(__.select("product").outE("isPublicized").inV().where(P.eq("campaign")))
15 //Count the number of matches between products and campaign and filter when they are at least 100
16 .select("campaign","product").groupCount().unfold().where(__.select(values).is(P.gte(100)))
17 // Add new elements to the graph
18 .select(keys).addE("isPublicized").from("product").to("campaign").dedup().iterate();
```

Listing 2 – Q4 with Spatial approximation.

3.3.3 Example queries

Our data models have been stored in different files to be loaded as an in-memory TinkerGraph structure, and the queries have been implemented using the Gremlin language. As explained above, the Pattern Model is obtained from the Source Model by applying the query filters. Additionally, different approximation methods are used for obtaining the Approximate Models from the Pattern Model, as shown next.

To illustrate how random and temporal approximations are integrated into Gremlin queries, Listing 1 shows the code corresponding to **Q3**. A random approximation has been implemented applying the *coin step* offered by Gremlin [Tin19]. This function allows to run the query on the Approximate Model in which orders are considered depending on a probability. We can see in line 6 how the coin step is applied so that orders are processed depending on the probability value set with the parameter *prob*.

Line 4 shows a temporal window inside the *where* clause. In this case, the temporal approximation is applied narrowing down the parameters *initTime* and *endTime*, which correspond to the initial and ending time of the temporal window, respectively.

In order to illustrate how spatial approximations can be defined in Gremlin, Listing 2 shows the implementation of **Q4**. In this query, a spatial window for geographical areas is shown from lines 6 to 8. The window starts from the area with postal code 24495, which is located in the centre of Rio de Janeiro in our models, and the *repeat-times*

block indicates the number of hops to consider from this area in order to cover the complete area of Rio de Janeiro. In this case, the spatial approximation is indicated by the parameter *hops* in the *times* clause.

We have executed the queries several times with different approximations and different Source Models, as explained next in Section 4. For implementing different approximations, we have simply modified the aforementioned parameters. The implementation of the remaining queries and the results of all experiments can be accessed on our Git repository [BTV19].

4 Evaluation

In this section we are interested in answering the following research questions (RQs):

- **RQ1 - How is performance improved when considering Approximate Models?** Running queries on Approximate Models is faster than running them on the Pattern Model. However, we want to check how much performance is gained depending on the sizes of the Pattern and Approximate Models, the type of approximation applied, and the distribution of the source data.
- **RQ2 - Are the 3 accuracy measures enough for identifying the Optimal Model?** Since we want to improve the performance of queries without compromising their accuracy, we want to discover whether the three measures presented in Section 3.2 are appropriate for measuring such accuracy.
- **RQ3 - Which approximation method provides the best trade-off between accuracy and performance?** Since there are different ways of approximating the Pattern Model, we want to find out which one is better, in terms of trade-off between performance and accuracy, depending on the source data.

4.1 Experimental Setup

4.1.1 Source Models

In all the models that we handle, data can be distributed in many forms. Considering the concept of time described in Section 3.1.1, in our running example the data can be concentrated in certain periods of time. For example, people are more likely to order products in their spare time, so evenings will normally concentrate more data than mornings. In this case, we say that data is *focused* on evenings. As for the concept of vicinity introduced in Section 3.1.2, more orders are likely to be made in Europe than in Africa, so data will be more concentrated along geographical areas in Europe. In this case, we say that data is *focused* in Europe. Now, if we consider only the data in the evenings or the data in Europe, we may have a more uniform distribution.

For experimentation purposes, the source models we have created for our running example contain information of the orders of Amazon Brazil in August 2016, and we suppose we are executing the queries in September 2016—i.e., *last month* in the queries (cf. Section 2) refers to August 2016. Furthermore, we have grouped source models in two batches, as we can see in Table 1. In batch A, data is uniformly distributed along the month, while in batch B data is mainly *focused* on the first week. In the table, the name of the model is assigned according to the number of *Customers* (cf. Figure 1). In this way, the smallest models contain 31K customers and around 255K objects

Distribution Batch	Name	Nodes	Edges
A	31K	286804	2399746
	62K	424,368	4,113,948
	125K	699,517	7,547,815
	250K	1,251,025	14,431,225
B	31K	287,731	2,477,232
	62K	425,836	4,201,686
	125K	699,945	7,635,425
	250K	1,252,316	14,543,380

Table 1 – Summary of the models used in the experiments.

linked among them by around 2.4M relationships. The largest models contain 250K customers and around 1M objects linked among them by around 14.5M relationships.

Being these models of different and considerable sizes and counting on different data distributions, we want to evaluate how approximating such models improves performance, and how much this compromises confidence in terms of precision, recall and accuracy values when executing the different queries.

4.1.2 Queries and approximations

In the following we will consider the five queries presented in Section 2, and the three types of approximations defined in Section 3.1: time, spatial and random.

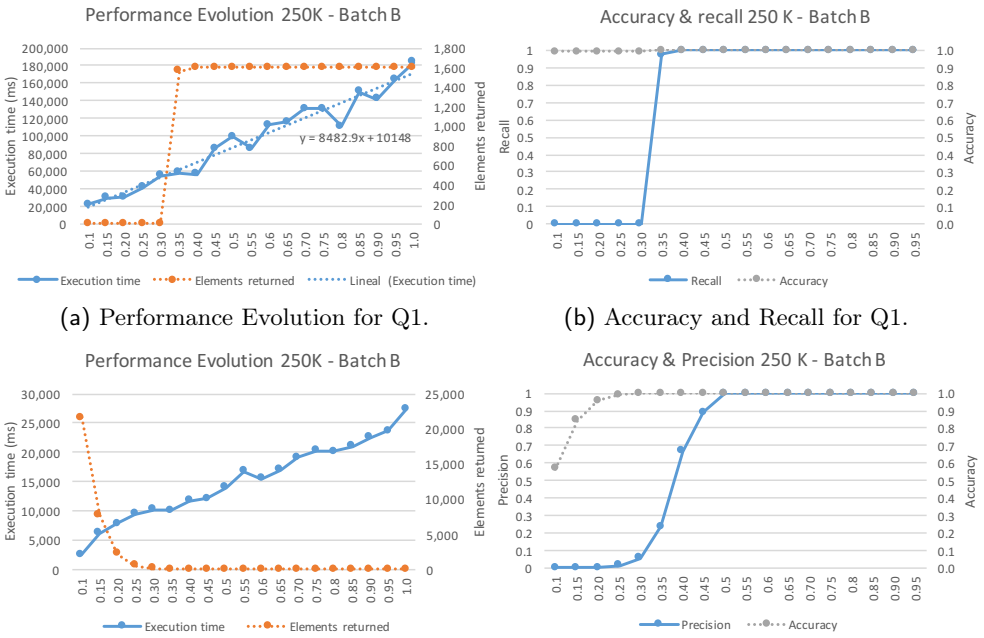
4.1.3 Experiments and data collected

Figures 2 to 6 show the execution times and accuracy obtained for each query, using different model sizes. The information displayed in each chart is the following:

- Data approximation. The type of approximation that is used in each experiment is displayed in the X axis. For instance, in the charts of Figure 2 a random approximation is applied. Thus, the X axis shows how much of the Pattern Model is being considered (i.e., it indicates the probability of each element of the Pattern Model to be included in the Approximate Model). The chart in Figure 3c shows a temporal approximation, so the X axis indicates the elements of the Pattern Model that are considered for the Approximate Model according to elements timestamps (cf. Section 3.1.1). Finally, the chart in Figure 5c displays a spatial approximation, where the X axis indicates the number of hops (cf. Section 3.1.2) taken from an initial object.

- Execution time. Whenever the execution time of the query is displayed in a chart, its values are shown on the left-hand-side Y axis, such as in the chart of Figure 2a. The performance evolution depending on the Approximate Model used is displayed with a continuous blue line.

- Number of elements returned by the query. Depending on the query, they can create relationships (such as **Q1**, **Q3** and **Q4**), objects and relationships (**Q5**) or return objects (**Q2**). The quantity of elements (objects and relationships) that are either returned or created by the queries is shown on the right-hand-side Y axis of the charts. The evolution of this quantity of elements depending on the Approximate Model used is displayed with a dashed orange line, such as in the chart of Figure 2a.



(a) Performance Evolution for Q1.

(b) Accuracy and Recall for Q1.

(c) Performance Evolution for Q2.

(d) Accuracy and Precision for Q2.

Figure 2 – Accuracy, Precision and Recall with Random Approximations.

- Precision and Recall. When they appear in a chart, their values are shown in the left-hand-side Y axis (chart in figures 2b and 2d for instance). Precision and recall evolutions are shown with a continuous blue line.
- Accuracy. When it appears in a chart, its values are shown on the right-hand-side Y axis (such as in the chart in Figure 2b). Its evolution is displayed with a dashed gray line in the chart.

4.1.4 Pattern Model vs Approximate Model vs Optimal Model

In the charts, as we move along the X axis, we see the results for different Approximate Models. For instance, let us focus in the chart of Figure 2a. When we see 0.45 in the X axis, it means that we are running **Q1** with an Approximate Model that includes 45% of the elements in the Pattern Model. Therefore, the right-most value in the X axis (1.0 in this chart) shows the result of the query when considering the whole Pattern Model. Regarding the Optimal Models, they are the Approximate Models such that, when running the query on them, obtain the *best* result. This means the best balance between performance and accuracy. Please note that the focus of our approach is not to automatically identify the Optimal Models, but to provide enough data (i.e., elements generated or retrieved by the query, and performance and accuracy values) for deciding what the Optimal Models should be depending on the user's needs and for knowing when they have been obtained. Automatically obtaining the Optimal Models is therefore out of the scope of this paper and is part of our future work.

4.1.5 Execution environment

All experiments have been run on a MacBook running operating system macOS Sierra version 10.13.2 64-bit, with 16GB of RAM memory, and an Intel Core i7-6700HQ processor with 8 cores of 2.6 GHz. We used TinkerGraph-Gremlin version 3.3.4 [Tin19] in our implementation, Java version 1.8.0_144 and Gremlin-java version 2.6.0.

4.2 Results

Figures 2 to 6 display the results of the experiments we have carried out. First of all, let us focus on the performance figures, shown in the charts of Figures 2a, 2c, 3a, 3c, 4a, 4c, 5a and 5c. In all of them, the smaller the Approximate Model considered, the faster the execution time. This means that the time taken by the Gremlin engine to filter the data that compose our conceptual Approximate Models pays off, since the engine runs the queries faster with smaller models.

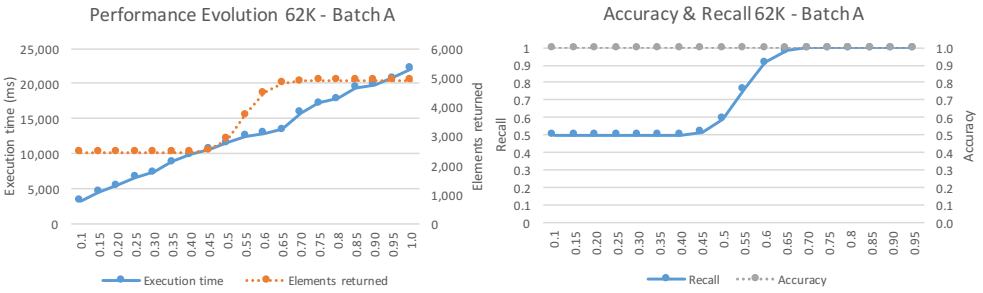
Now, let us take the type of approximation into consideration. In random approximations, displayed in Figures 2a, 2c, 3a, 4a and 5a, the execution time increases linearly as the size of the Approximate Model grows. This happens in all cases, no matter how data is distributed in the source models. For instance, Figures 3a and 4a show the performance evolution for **Q3** applying a random approximation on the source model $62K$. Figure 3a shows the result with model $62K - BatchA$ and Figure 4a with model $62K - BatchB$. We can see that there is not much difference in the execution times.

Regarding temporal approximations, shown in Figures 3c and 4c, we can see that execution times do not present much variation depending on how source data is distributed either, and that execution time also grows linearly. In a spatial approximation, such as the one shown in Figure 6a for **Q5**, we can see that execution time grows faster than linearly. This is reasonable, since a linear increase in the number of hops does not mean a linear increase of the data considered for the Approximate Model, but an exponential one. For instance, in **Q5**, hops are taken according to the *isRelatedTo* relationship among products. Since one product can be linked through this relationship with many others, doing one hop may imply considering many more elements; and even more if we take 2 hops, since the growth is exponential.

With all this data, we can answer RQ1:

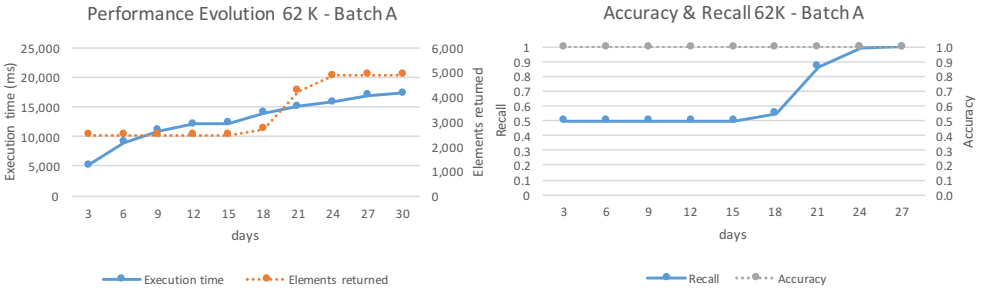
> **RQ1:** Performance improvement is noticeable when the amount of data considered for the Approximate Model is smaller than the amount of data in the Pattern Model. In fact, the execution time is directly proportional to the approximate model size considered. This means that the time taken by the engine to filter data for obtaining the Approximate Model is not significant and it pays off.

Let us now consider **Q1** and **Q2**. **Q1** creates an *isPublicized* relationship when a product appears at least 1000 times during an advertising campaign period. For this query, as the volume of data in Approximate Models decreases, some elements that should be returned are not, which means we get some FNs and no FP. Therefore, the precision value is 1 and the deviation from the query result with the Pattern Model can only be represented by the recall value. In Figure 2a we present the performance evolution for this query when run on model $250K - BatchB$ applying a random approximation. In this figure, probabilities go from 0.1 to 1 with increments of 0.1. As previously mentioned, execution time has a lineal increase. Regarding the elements



(a) Performance Evolution for Q3 with Random Approximation.

(b) Accuracy and Recall for Q3 with Random Approximation.



(c) Performance Evolution for Q3 with Temporal Approximation.

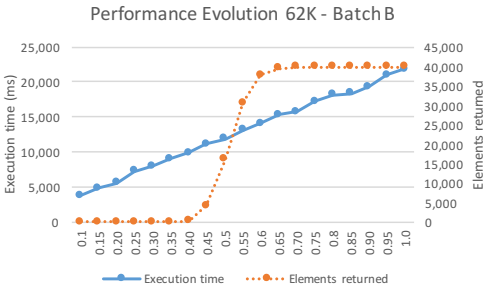
(d) Accuracy and Recall for Q3 with Temporal Approximation.

Figure 3 – Comparison between Temporal and Random Approximations with uniformly distributed data.

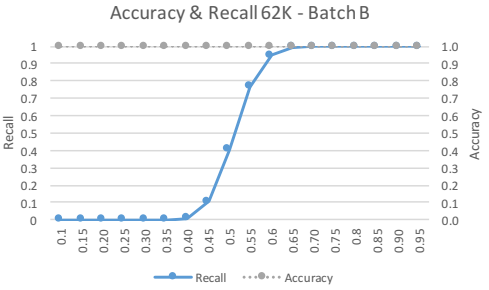
created by the query, their number grows as the probability value of the horizontal axis increases, until the line eventually stabilizes and, at this point, we could consider we have reached the amount of data considered for an Optimal Model. In order to double-check this, results for accuracy and recall values are shown in Figure 2b. The value of accuracy is not significant, since it is always 1. As for recall, it reaches 1 precisely when the number of elements returned gets stable.

Consider now query **Q2**, in which a product is returned if it has been ordered less than three times in a month. In this case, as the volume of data in the Approximate Model increases, the number of elements returned decreases. This means that we will have no FNs and therefore a recall value of 1, and therefore the precision value is the most significant for calculating accuracy. Performance evolution when running the query on model $250K - BatchB$ is depicted in Figure 2c, while Figure 2d shows the accuracy and precision values for this query. For this experiment we have used a random approximation. If we expect a precision of 1 for considering an amount of data that conforms the Optimal Model, then we can see that we get it when we approximate half of the data of the Pattern Model.

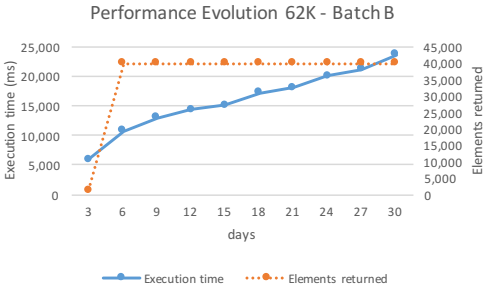
Note that in most cases accuracy values are very close to 1. This is due to the influence of TNs in the accuracy equation. Consequently, the accuracy value is not descriptive enough to represent the deviation of running the query on Approximate Model versus running it on the Pattern Model. Therefore, we can come to a response for RQ2:



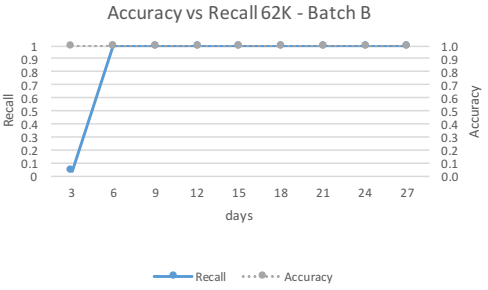
(a) Performance Evolution for Q3 with Random Approximation.



(b) Accuracy and Recall for Q3 with Random Approximation.



(c) Performance Evolution for Q3 with Temporal Approximation.



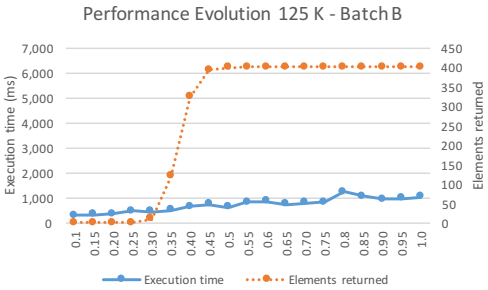
(d) Accuracy and Recall for Q3 with Temporal Approximation.

Figure 4 – Comparison between Temporal and Random Approximation with temporal focus on the data.

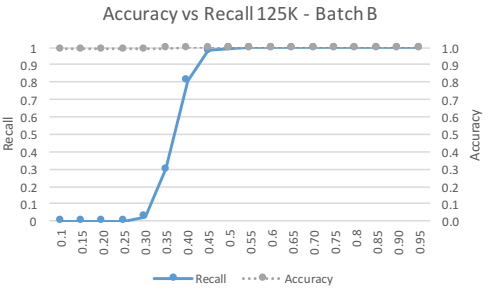
RQ2: Accuracy is not well suited as a measure for determining the amount of data to be considered in the Optimal Model. In contrast, precision and recall are valid measures when we get FPs and FNs, respectively.

Since we have different approximation possibilities for obtaining the data of Approximate Models, we want to find out which one is more convenient depending on the situation. First of all, temporal and spatial approximations only make sense when the query filters according to time, or to some spatial concept, respectively. Let us focus first on **Q3**. We have used random and temporal approximations for running it and have used the two types of source models presented in Section 4.1.1, this is, those where data is uniformly distributed along the month (Batch A) and those where there is a temporal focus on the first week (Batch B). In the random approximation, Approximate Models start considering 10% of the model, with increments of 5%, until we consider the Pattern Model (same as in the charts discussed before). In the temporal approximation, the first Approximate Model considers the first 3 days in the month, the second one considers 6 days, the third one 9 days, and so on.

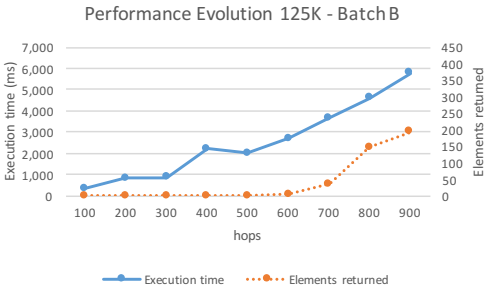
Figures 3 and 4 display execution results for models $62K - BatchA$ and $62K - BatchB$, respectively. Let us focus first on Figure 3. Figures 3b and 3d display the recall (and accuracy, but, as we said before, we do not take this one into account) with random and temporal approximations, respectively. We can see that recall reaches the value 1 first in the random approximation, specifically, when 65% of the Pattern Model is considered in the approximation. In the temporal approximation, recall does



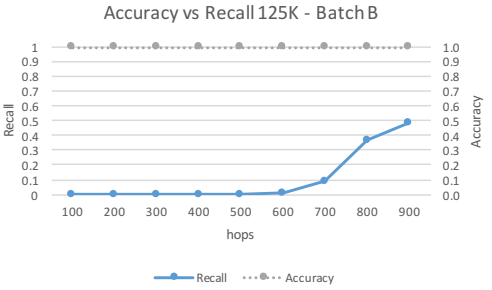
(a) Performance Evolution for Q4 with Random Approximation.



(b) Accuracy and Recall for Q4 with Random Approximation.



(c) Performance Evolution for Q4 with Spatial Approximation.



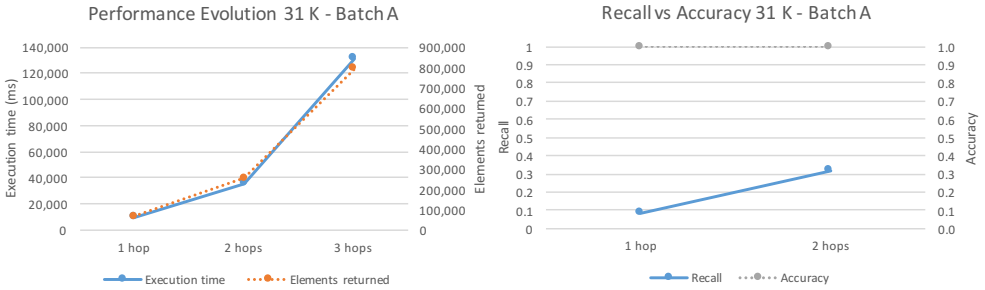
(d) Accuracy and Recall for Q4 with Spatial Approximation.

Figure 5 – Comparison between Spatial and Random Approximations.

not reach the value 1 until the approximation contains 80% of elements of the Pattern Model. Furthermore, in Figures 3a and 3c, we see that the execution time is lower in the random approximation when the number of elements returned stabilizes with respect to when the number of elements stabilizes in the temporal approximation.

Let us now consider Figure 4, which displays the results with model $62K - BatchB$. Recall that data in this model has a temporal focus in the first week. Figures 4c and 4d reflect this. Indeed, we can see that recall reaches 1 when the approximation considers the first 7 days of the month, and we also see that the number of elements returned by the query stabilizes in this time. In the random approximation (Figures 4a and 4b), however, it occurs like in the example before: query result and recall stabilize when the approximation considers 65% of the model. In fact, we see that random approximations always behave the same, no matter how data is distributed in the source models. In temporal approximations, the accuracy of the query result depends on the temporal distribution of the data in the model.

For comparing random and spatial approximations, we use **Q4** (Figure 5). Spatial approximation is made by considering geographical areas. In particular, we start from postal code 24495 (cf. Listing 2), which is located in the centre of Rio de Janeiro. In every increment in the approximation, we consider those areas within the next 100 hops (cf. Section 3.1.2). Figure 5 shows the results with both approximations for model $125K - BatchB$. Note that when running experiments applying spatial approximation, results do not stabilize. In fact, we discovered that Gremlin can only process a limited number of hops. In our case, we could not perform more than 900 hops, so the Approximate Model does not cover the complete area of Rio de Janeiro.



(a) Performance Evolution for Q5 with Spatial Approximation

(b) Accuracy and Recall for Q5 with Spatial Approximation

Figure 6 – Spatial Approximations with several sources

Apart from this limitation, we can see in Figures 5a and 5c that the execution time when considering spatial approximation is higher. This is because it is more expensive to traverse the model by applying hops (the model is traversed through objects and relationships among them) instead of filtering by attributes. We see that, in this case, random approximation is more efficient and accurate. However, there might be cases where it is not possible to filter by property, such as in some social networks analysis, where some conclusions can only be taken by traversing the model, so the use of a spatial window is essential.

Let us now consider **Q5** for further studying the performance evolution of spatial approximations. As we see in Figure 6a, execution time seems to increase exponentially as the number of hops grows. Indeed, as we see in Figure 6b, recall grows slowly in comparison with the execution time.

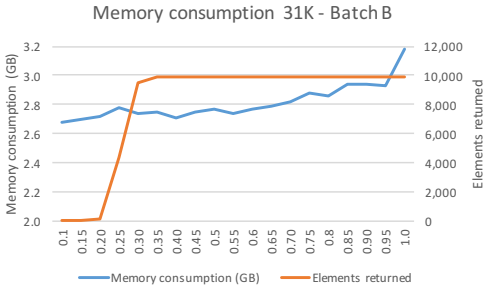
> **RQ3**: There is no approximation method that always provides the best trade-off between performance and accuracy. While random approximations typically behave the same no matter how data is distributed in source models, temporal approximations behave differently depending on how source data is temporally distributed. As for spatial approximation, approximating using hops is expensive in terms of runtime, but there might be systems or situations where it is the only possible approximation.

4.3 Discussion

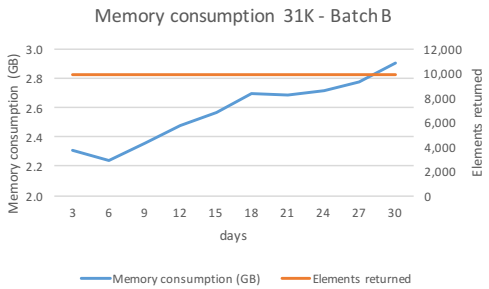
First of all, the concepts defined in Section 3.1 serve for explaining our approach from a conceptual point of view. However, we do not try to obtain first a Pattern Model and then Approximate Models. In our implementation, these tasks are implicitly done by the query, as we have shown in Listings 1 and 2.

These Approximate Models offer the *best* balance between performance and accuracy. Since this balance can be subjective, this decision must be ultimately taken by the user. Having said that, we would like to explore the use of search-based algorithms for determining the Pareto front of Optimal Models, according to a set of optimization criteria (such as decreasing execution time and improving precision and recall).

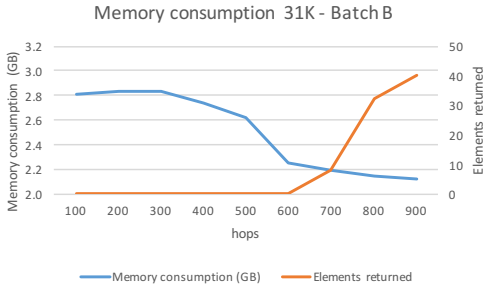
Although performance is usually defined in terms of execution time and memory consumption, this paper is focused on the first feature, since we consider it is the main



(a) Memory consumption evolution for Q3 with Random Approximation



(b) Memory consumption evolution for Q3 with Temporal Approximation.



(c) Memory consumption evolution for Q4 with Spatial Approximation.

Figure 7 – Memory consumption for Q3 and Q4.

concern in data processing applications. However, in order to have a first estimation, we have measured memory consumption in model 31K of batch B for **Q3** with Temporal and Random approximations, and **Q4** with Spatial approximation. Results of the experiments are shown in Figure 7. As we can observe in Figures 7a and 7b, memory consumption grows as the Approximate Model increases its size for Temporal and Random approximations. However, in Figure 7c, memory consumption decreases as more hops are considered in the Spatial approximation. This may be due to the fact that some parts of the model are discarded as we do more hops. More experiments regarding memory consumption will be performed as part of our future work.

The conclusions presented in the previous section can be summarized as follows:

- Random approximations are the best option when a query does not contain temporal or spatial filtering.
- Results of applying random approximations are similar no matter how the source data is distributed.
- If a query contains a temporal filter and the data is distributed with a temporal *focus*, then it is convenient to use a temporal approximation centered on the focus.
- If a query contains a temporal filter but the source data is uniformly distributed, then random approximations seem to perform best.
- Spatial approximations by means of hops are very expensive in terms of runtime. They are only recommended when there is no other option.

4.4 Threats to Validity

In the following we describe the four types of threats that can affect the validity of our study, according to Wohlin et al. [WRH+12].

4.4.1 Construct validity threats

They are concerned with the relationship between theory and what is observed. Performance is typically measured in terms of memory consumption and execution time. In this paper, we have only used execution time for performance comparison between different types of approximation techniques, with different queries and data distribution. Therefore, a possible construct validity threat is the use of only execution time for performance measurement. However, since one of the critical points of many data processing applications is to respond to the situations described by the queries as quickly as possible, we believe that this measure is descriptive enough for this work, although we plan to also consider memory consumption in future work.

4.4.2 Conclusion validity threats

Threats to the conclusion validity are concerned with the issues that affect the ability to draw correct conclusions from the data obtained from the experiments. In our experiments, the transitory load of the machine where the experiments were executed can influence the execution time results. Besides, the elements returned for Random approximation experiments may have small variations depending on the input data. To mitigate both threats, we have run all experiments 6 times on the same machine and taken the average execution time and elements returned by the 3 last runs.

4.4.3 Internal validity threats

These threats are related to those factors that might affect the results of our evaluation. In our experiments, we have considered five queries. Although they present a certain degree of variability, having considered more queries could have yielded different results. Furthermore, for classifying such queries, we have considered the type of window according to the definition of the query, and conclusions have been drawn according to such type of window. However, other features could have been considered in the classification, such as number and type of filters, action resulting from the query (deletion, insertion, update...) or traversal path. These concepts might also have an influence on the type of approximation recommended for each query in order to obtain the best trade-off between accuracy and performance. Finally, the temporal or spatial *focuses* present in the models influence the number of elements returned by Temporal and Spatial approximations. We have considered models with different types of focus. However, if we had consider a higher variability of these focuses, we could have obtained different results.

We will aim for the mitigation of these threats by considering the different dimensions mentioned for classifying the queries, as well as by considering more models with higher variability.

4.4.4 External validity threats

These threats have to do with the extent to which it is possible to generalize the findings of the experiments. The first threat is that the results of our experiments have been obtained with one case study, which externally threatens the generalizability of our results. To mitigate this threat, we have tried to select five queries that consider

different situations with different approximations. Furthermore, the case study has been selected from a real situation. In any case, we plan to consider more case studies for future experiments. Second, we have used Gremlin and TinkerGraph as the technologies for implementing our approach, since they are state-of-the-art technologies for database querying and in-memory data storage, respectively. In any case, we do believe the same conclusions would have been drawn with similar technologies, such as Cypher, but we plan to perform further experiments as future work. Finally, the results of the experiments depend on the size and distribution of the data. To mitigate this threat, all experiments have been run with different data sizes and distributions.

5 Related Work

Our approach derives from two previous works. First, [TWBV14] introduces the concept of Approximate Model Transformation (AMT), and the error from applying AMTs is calculated with statistical formulas. However, data are not related to each other, but they are just a stream of simple events. Then, [BBTV18] provides a prototypical solution based on CEP to process graph-structured information. The system was implemented with Spark and the *GraphX* tool for graph-parallel computation. The concept of *spatial window* was introduced using vicinity graphs. This paper complements these works by exploring different strategies for selecting the subset of data to perform the approximations with graph-structured information, and by providing mechanisms to estimate the accuracy of the different options.

Some studies have applied approximations similar to the ones we propose in this paper. For instance, some works have used spatial windows when dealing with streaming models [CdL13] and very large or even infinite models [CTB12], where a *sliding window* limits the data to be processed at any given moment in time. Similarly, the work [JVEV18] uses a grid to represent a forest-fire spreading example, where the spreading is represented with vicinity links between the cells. Although our approach is in the context of these works, we specially focus on providing the necessary mechanisms for obtaining the accuracy of the results when applying different types of approximations. Regarding random approximations, other authors use sampling-based approaches to improve the execution time in large models [BHK18]. However, their proposal is limited to models with up to 20,853 elements, clearly insufficient for the kinds of models our proposal targets, with more than 2 millions of elements.

There are some works that apply crowd-sourcing techniques [TKFS13, TKFS16] and develop statistical tools to find a trade-off between cost/time and completeness of results. However, while the query result is constructed incrementally (the query is performed on an initial small dataset and the result is refined as more source data arrives to the system), in our proposal an initial large dataset is approximated at once. There are more works that deal with *incremental* queries and transformations [SIR$^+$, UBH$^+$15, BHR$^+$10, JE04, JT10, RK12], where the input model changes with time. However, they do not compute the error produced when not considering the complete model for the transformation, what is of key importance in our approach. There are some other works peripheral to ours. For instance, in [KS03] the authors select only a subset of the information in order to manage information overload, so they mainly aim for system survival, but sacrificing system reliability.

Some studies have proposed to apply precomputation in the context of Approximate Query Processing (AQP) in database systems [CDK17, LL18]. This consists of storing a summary of the source data with interesting information for the query

(offline precomputation step) and then using this summary to approximate the source information and performing the query. However, there are not many solutions that use these techniques with complex data structures such as graph-based structures [CDK17, LL18]. In our case, performing precomputation with large models would be too costly, so we aim to perform the queries online, with no offline precomputation step.

Finally, technologies such as the distributed streaming platform of Apache Kafka [Kaf19] or the streaming extension of Apache Spark [Spa19] are aimed for processing information flows. They can generate streams of data as well as handle them. Our work does not pretend to replace these technologies, but it can rather complement them. In fact, our main goal is to obtain the best trade-off between execution time and information loss when processing large amounts of graph-structured information. An approach for reducing information to improve performance in complex stream processing scenarios has already been developed for Streaming Transformations for XML (STX) [CBN+19]. It consists of performing transformations to XML documents, which are provided as sequences of XML events, accessing just a part of the entire document in order to run the query. The part to be processed is selected with precomputing tasks. This work differs from our approach in two aspects: it works with XML-tree-like structures instead of graph structures and makes use of preprocessing tasks to perform the approximation of the source data.

6 Conclusions and Future Work

In this paper we have explored different possibilities for approximating the queries performed on models containing large amounts of data, and have proposed a method for measuring the accuracy of the approximations, so that the user can find the right balance between accuracy loss and performance gain. We have also analyzed how the distribution of the input data affects these approximations.

Our experiments conclude that performance can indeed be improved. In fact, an optimal accuracy value can be acquired when considering only part of the source model. The results obtained do not pretend to be final nor conclusive. Indeed, we plan to conduct further experiments. To begin with, we are interested in investigating how the presence of more than one data focus in different time intervals can affect the approximations. Similarly, we want to investigate spatial data focuses located in different points of the model and their effects in the approximations.

Regarding the traversal algorithm for spatial approximations, Gremlin applies a depth-first approach by default. It is interesting to study how applying other algorithms affects accuracy of the approximations as well as execution time, so it is something we plan to explore in future work. In this line, we also plan to consider what benefits model indexing technologies can bring.

We also plan to study the evolution of performance in approximations that imply both FPs and FNs in the same query, and how to obtain the amount of data for the Optimal Models. In this case, both precision and recall will be evaluated. Of course, more case studies are needed to better assess our proposal.

We also plan to address a new problem: how to automatically determine the Optimal Models. We envision the application of search-based algorithms for determining the Pareto front of Optimal Models, according to a set of optimization criteria, such as decreasing execution time and memory consumption, and improving precision and recall. Finally, we plan to do a more exhaustive evaluation of memory consumption with many more experiments and study how the different approximation types can

affect this feature.

Verifiability

For the sake of verifiability, our implementation prototype as well as all artifacts of the experiments and many more charts resulting from running all queries are available on our project's website, located on Github [BTV19].

References

[Apa19] Apache TinkerPop. The Gremlin Graph Traversal Machine and Language, (accessed March 2019). https://tinkerpop.apache.org/gremlin.html.

[BBTV18] Gala Barquero, Loli Burgueño, Javier Troya, and Antonio Vallecillo. Extending complex event processing to graph-structured information. In *Proc. of MODELS'18*, pages 166–175. ACM, 2018. doi:10.1145/3239372.3239402.

[BBV13] The impact of the Mobile World Congress in a dynamic visualization by BBVA and CartoDB, 2013. https://www.bbva.com/en/impact-mobile-world-congress-dynamic-visualization-bbva-cartodb/. (accessed March 2019).

[BHK18] Julien Bernard, Pierre-Cyrille Héam, and Olga Kouchnarenko. An approximation-based approach for the random exploration of large models. In *Proc. of TAP'18*, pages 27–43, 2018. doi:10.1007/978-3-319-92994-1_2.

[BHR+10] Gábor Bergmann, Ákos Horváth, István Ráth, Dániel Varró, András Balogh, Zoltán Balogh, and András Ökrös. Incremental evaluation of model queries over EMF models. In *Proc. of MODELS'10*, pages 76–90, 2010. doi:10.1007/978-3-642-16145-2_6.

[BTV19] Gala Barquero, Javier Troya, and Antonio Vallecillo. Approximate Transformation git repository, 2019. https://github.com/atenearesearchgroup/approximateTransformation.git. (accessed March 2019).

[CBN+19] Petr Cimprich, Oliver Becker, Christian Nentwich, Honza Jiroušek, Manos Batsis, Paul Brown, and Michael Kay. Streaming Transformations for XML (STX). Working Draft, (accessed May 2019). http://stx.sourceforge.net/documents/.

[CDK17] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate query processing: No silver bullet. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 511–519. ACM, 2017. doi:10.1145/3035918.3056097.

[CdL13] Jesús Sánchez Cuadrado and Juan de Lara. Streaming Model Transformations: Scenarios, Challenges and Initial Solutions. In *Proc. of*

ICMT'13, volume 7909 of *LNCS*, pages 1–16. Springer, 2013. doi:10.1007/978-3-642-38883-5_1.

[CM12] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012. doi:10.1145/2187671.2187677.

[CTB12] Benoit Combemale, Xavier Thirioux, and Benoit Baudry. Formally Defining and Iterating Infinite Models. In *Proc. of MODELS'12*, volume 7590 of *LNCS*, pages 119–133. Springer, 2012. doi:10.1007/978-3-642-33666-9_9.

[DH99] Jeffrey Dean and Monika R. Henzinger. Finding related pages in the world wide web. *Comput. Netw.*, 31(11-16):1467–1479, May 1999. doi:10.1016/S1389-1286(99)00022-5.

[EN10] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications, 2010.

[Esp19] Esper. Esper Tech, (accessed March 2019). http://www.espertech.com/esper/.

[HP13] Florian Holzschuher and Prof. Dr. René Peinl. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Joint EDBT/ICDT 2013 Workshop GraphQ (EDBT/ICDT '13)*, pages 195–204, 2013. doi:10.1145/2457317.2457351.

[JE04] Sven Johann and Alexander Egyed. Instant and incremental transformation of models. In *Proc. of ASE'04*, pages 362–365, 2004. doi:10.1109/ASE.2004.10047.

[JT10] Frédéric Jouault and Massimo Tisi. Towards incremental execution of ATL transformations. In *Proc. of ICMT'10*, pages 123–137, 2010. doi:10.1007/978-3-642-13688-7_9.

[JVEV18] Maris Jukss, Clark Verbrugge, Maged Elaasar, and Hans Vangheluwe. Scope in model transformations. *Software and System Modeling*, 17(4):1227–1252, 2018. doi:10.1007/s10270-016-0555-8.

[Kaf19] Apache Kafka. Apache Kafka. A distributed streaming platform, (accessed May 2019). https://kafka.apache.org/intro.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999. doi:10.1145/324133.324140.

[KS03] John C. Knight and Elisabeth A. Strunk. Achieving critical system survivability through software architectures. In Rogério de Lemos, Cristina Gacek, and Alexander B. Romanovsky, editors, *Architecting Dependable Systems II - [the book is a result of the ICSE 2003 Workshop on Software Architectures for Dependable Systems]*, volume 3069 of *Lecture Notes in Computer Science*, pages 51–78. Springer, 2003. doi:10.1007/978-3-540-25939-8_3.

[LL18] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Science and Engineering*, 3(4):379–397, 2018. doi:10.1007/s41019-018-0074-4.

[Luc02] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.

[Luc12] David C. Luckham. *Event Processing for Business: Organizing the Real-Time Enterprise*. Wiley, 2012.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. doi:10.1007/s10791-009-9115-y.

[Neo19] Neo4j. Cypher Query Language, (accessed March 2019). https://neo4j.com/developer/cypher-query-language/.

[PP11] Arie Perliger and Ami Pedahzur. Social network analysis in the study of terrorism and political violence. *PS: Political Science amp; Politics*, 44(1):45–50, 2011. doi:10.1017/S1049096510001848.

[RK12] Ali Razavi and Kostas Kontogiannis. Partial evaluation of model transformations. In *Proc. of ICSE'12*, pages 562–572, 2012. doi:10.1109/ICSE.2012.6227160.

[SIR+] Gábor Szárnyas, Benedek Izsó, István Ráth, Dénes Harmath, Gábor Bergmann, and Dániel Varró. incquery-d.

[SIRV18] Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. The train benchmark: cross-technology performance evaluation of continuous model queries. *Software & Systems Modeling*, 17(4):1365–1393, Oct 2018. doi:10.1007/s10270-016-0571-8.

[Spa19] Apache Spark. Spark Streaming Programming, (accessed May 2019). https://spark.apache.org/docs/latest/streaming-programming-guide.html.

[Tin19] TinkerPop. Apache TinkerGraph, (accessed March 2019). http://tinkerpop.apache.org/docs/current/reference/#tinkergraph-gremlin.

[TKFS13] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Crowdsourced enumeration queries. In *Proc. of ICDE'13*, pages 673–684, 2013. doi:10.1109/ICDE.2013.6544865.

[TKFS16] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Answering enumeration queries with the crowd. *Commun. ACM*, 59(1):118–127, 2016. doi:10.1145/2845644.

[TWBV14] Javier Troya, Manuel Wimmer, Loli Burgueño, and Antonio Vallecillo. Towards approximate model transformations. In *Proc. of the Workshop on Analysis of Model Transformations (AMT@MoDELS'14)*, pages 44–53. CEUR-WS, 2014.

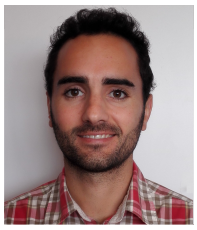
[UBH+15] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. EMF-IncQuery: An integrated development environment for live model queries. *Sci. Comput. Program.*, 98:80–99, 2015. doi:10.1016/j.scico.2014.01.004.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. *Experimentation in Software Engineering*. Springer, 2012.

[YN07] Christopher C. Yang and Tobun D. Ng. Terrorism and crime related weblog social network: Link, content analysis and information visualization. In *2007 IEEE Intelligence and Security Informatics*, pages 55–58, May 2007. doi:10.1109/ISI.2007.379533.

About the authors

Gala Barquero is PhD student at the University of Málaga, Spain. She worked as Java developer at Viewnext company from the IBM group before that time (2015-2017). She graduated as Telecommunications Engineer at University of Jaén, Spain (2015). Her current research interests include Model-based Software Engineering, Real-time Analytics and Software Quality. Contact her at gala@lcc.uma.es.



Javier Troya is Assistant Professor of Software Engineering at the University of Seville, Spain. Before, we was a post-doctoral researcher in the TU Wien, Austria (2013-2015), and obtained his International PhD with honors from the University of Málaga, Spain (2013). His current research interests include Model-based Software Engineering, Software Testing and Software Quality. Contact him at jtroya@us.es, or visit http://www.lsi.us.es/~jtroya/.

Antonio Vallecillo is Professor of Software Engineering at the University of Málaga, Spain, where he leads the Atenea Research Group. His current research interests include Model-based Software Engineering, Open Distributed Processing, and Software Quality. More information about his publications, research projects and activities can be found at http://www.lcc.uma.es/~av. He can be contacted at av@lcc.uma.es.

Acknowledgments This work has been partially supported by the European Commission (FEDER) and Spanish Government under research projects TIN2014-52034-R, TIN2015-70560-R, RTI2018-101204-B-C21, and PGC2018-094905-B-I00.