Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Performance Analysis of Graph Queries

Author:

Zsolt Kővári

Advisors:

Gábor Szárnyas
Dr. István Ráth

2015

Contents

Kivonat

A modell központú rendszerek tervezése során kulcsfontosságú a modellen kiértékelt lekérdezések optimális teljesítménye, ennek elérése érdekében pedig a megfelelő végrehajtó eszközök megválasztása. Az elmúlt években különböző NoSQL adatbázis-kezelő rendszerek váltak népszerűvé, amelyek célja a gyors lekérdezés kiértékelés és skálázhatóság biztosítása.

A lekérdezés kiértékelésének teljesítményét nagy mértékben befolyásolja a modell topológiája és a lekérdezés komplexitása. Célunk az, hogy bizonyos, a modellt és a lekérdezést leíró metrika felhasználásával, kapcsolatot találjunk a metrikák és a teljesítmény között, úgy, hogy az adott metrikákat ismerve, a teljesítmény megjósolható legyen.

A metrikák alapján tervezési szintű döntéseket hozhatunk az optimális teljesítményre törekedve, továbbá, ez a tudás lehetőséget teremt a valósidejű lekérdezés optimalizálás területén is arra, hogy a modellt és a lekérdezést jellemző metrikák alapján döntsünk optimalizálást érintő kérdésekben.

Különböző NoSQL adatbázis-kezelő rendszereket vizsgálva, regressziós analízis segítségével olyan metrikák után kutatunk, amelyek képesek az adott rendszer futási teljesítményét jellemezni. A vizsgálandó adathalmazok előállításához, egy valós modellből kiindulva, különböző eloszlású és topológiájú gráfokat generálunk.

További célunk az, hogy eldöntsük, vajon egy tetszőleges struktúrájú adatmodellre képesek vagyunk-e a regressziós analízisünk alapján a teljesítményre becslést adni, illetve a modellhez a megfelelő, készletünkben lévő adatbázis-kezelő rendszert társítani, a legjobb teljesítményre koncentrálva.

Végezetül, a valós idejű lekérdezések optimalizálásától motiválva, választ keresünk arra, hogy egy tetszőleges adatmodellre, költséghatékony metrikaszámítással a teljesítmény előre megbecsülhető-e.

Abstract

Achieving the optimal performance of query evaluations plays an important goal in model-based systems. In the last few years, different NoSQL database systems were introduced to provide a better performance and solve the problem of scalability.

The performance of query evaluations depends on the topology of the model, and the complexity of the particular query. Our primary goal is that — by defining model and query-related quantitative metrics — to find a considerable connection between metrics and the performance, and thus, be able to predict the query evaluation time.

Based on the metrics and their effect to the performance, we are able to make decisions in design to achieve on the optimal performance. Furthermore, this knowledge can be utilized in the area of real-time query optimization engines as well.

We investigate various NoSQL database systems via regression analysis in order to find different model-related metrics that are suited to characterize their performance appropriately. Based on a real model, we generate graphs with various topologies and distributions to find metrics from different aspects.

Furthermore, we explore that whether an arbitrarily structured model's performance is predictable via our regression analysis, and also predict which database system from our scope can be associated to the model in order to achieve an optimal performance.

Finally, being motivated by the real-time optimization engines, we search answers whether by reducing the cost of metric calculations, an arbitrary model's performance is still predictable.

Introduction

Chapter 1

Background

1.1 The Trainbenchmark Framework

In the following section, we introduce Train Benchmark, a framework on which our search is based to find empirical connections between different models and the related evaluated queries in order to estimate query evaluation time. The next few sections explain the main goal of the framework and represent the most important components in the system on which we will concentrate in the later chapters. Apart from these, we also allude some obsolete parts of the framework and emphasize their disadvantages in order to make understandable the main motivation of design decisions that resulted in some considerable modifications in the system.

1.1.1 Main Concepts

The basic idea and implementation of the Train Benchmark framework was introduced by the Fault Tolerant Systems Research Group in the Budapest University of Technology and Economics. The implementation is written in Java programming language. Basically, Train Benchmark focuses on the performance of model validations in which the goal is to find those elements from a model that violate the well-formedness constraints. [1] The model validations are accomplished by different databases systems, where the validation appears as a query evaluation, and the measured performance by the system is the evaluation time of the particular query. For the sake of clarity, Figure 1 represents the main steps of the framework's workflow.

The first step is the model generation and the definition of the well-formedness constraints. The Train Benchmark framework uses artificially generated graph-based models with the same domain, it and describes different constraints on them. Since the main goal is to assess model validations, a perfectly valid model generation is not advisable, thus, some

[1] A well-formedness constraint is defined on the model and it is considered as a definition of a particular rule belonging to the elements in the model. For example, a constraint can order the cardinality of the elements, or it can define maximum bounds for attributes, etc.

fault injections are required during the generation phase, when a subset of the elements (typically 2-5%) violates the well-formedness constraints.

The second step is loading the model to the database systems. Subsequently, the generated model is already accessible from the database's repository, and stored in their own format. The next phase is the model validation itself, when a validation—defined in the particular database's query language—is evaluated and its result set includes those erroneous elements from the model that violate the well-formedness constraints. Thus, a model validation occurs and the required time of the query evaluation is measured as well. The validation time will play the most important indicator in the performance comparison among different databases. To summarize, the Train Benchmark framework uses artificially generated graph-based models to investigate the model validation times and thus, the performance of different database systems. In the following, we introduce the domain of the model and its typical characteristics, furthermore, we also define the scope of the framework, as represent the used databases.

1.1.2 The Metamodel

As we referred in 1.1, there were some components in the Train Benchmark framework that needed to reconstruct in order to obtain precise statistical analysis of model and query metric relationships. One of them was the original domain of the framework, the metamodel. However, (i) to understand the motivations of changing the domain entirely, (ii) to see the contrast between the two of them, and (iii) also recognize the possible virtues that we obtained by introducing a now domain, it is important to represent the original domain in this section.

At first, let define the basic related concepts.

Metamodel and Instance Model A metamodel is considered as an abstract model of another model, where the prior—the metamodel—defines the available elements and their cardinalities, additionally, it order rules, constraints on the elements which are used in the latter. Then, a concrete instance of a metamodel is called instance model which can use those elements that are defined in the metamodel, and it must also obey the specified rules and constraints, otherwise, it is considered as an invalid model which violates the constraints in the metamodel.

The domain, or the metamodel[2] of Train Benchmark is depicted in Figure 2, and it is related to a railway network (here comes the name of the benchmark framework). A train route is defined by a sequence of sensors. Sensors are associated with track elements which are either segments (with a specific length) or switches. A route follows certain switch positions which describe the required state of a switch belonging to the route.

[2]We use the concepts of domains and metamodels as synonyms in this paper.

Different route definitions can specify different states for a specific switch. Each route has a semaphore on its entry and exit [19].

The instance models are not related to any real life data, since the instances are generated artificially, and each cardinality of the elements originally was chosen arbitrarily. As a consequence, we cannot associate our models to real life topologies, and thus, we are not able to draw conclusions after our benchmark results in real life situations. Figure 3 shows the cardinalities of the elements, which leads to the fact that segments represent the majority of the elements.

Later, we will query the inflexibility of this metamodel, referring to some heterogeneity problems, and decide to change the domain entirely that is already appropriate for our purpose.

1.1.3 Supported Formats

The statement that the Train Benchmark framework assesses the performance of different database systems is not entirely true. Actually, the scope of the framework is extended, since we also measure various application programming interfaces (API) which are able to store the data in memory, and behave as embedded databases, furthermore, we investigate the performance of different query engines and business management tools. In order to avoid the misunderstandings and use a common expression for these systems, from now on, we reference them as tools.

The currently supported tools can be categorized into four different formats. These categorizations do not only mean conceptual differences, but these appear in the framework's architecture as well. The formats are the following:

- SQL: This category includes the Relational Database Management Systems (RDBMS) that are based on relational models. The format was given its name after the query language of these tools that is formulated in Structured Query Language (SQL).
- EMF: The EMF abbreviation denotes the Eclipse Modeling Framework that facilitates the usage of model-driven development, as it focuses on the creation and applicability of domain models [9]. Previously, in Section 1.1.2, the metamodel in Figure 1 was created with EMF.
- Graph: That tools are involved in this format which operate as graph databases and typically use the GraphML format for their primary data storage structure [18].
- RDF: The majority of the tools are related to the Resource Description Framework (RDF). Our search highly concentrates on these systems, therefore, Section 1.4 introduces the main concepts of RDF in details.

In the further sections, we only focus on the subset of Graph- and RDF-based tools.

1.1.4 Performance Comparison and Uniformity

After introducing the main concepts, the metamodel, and the different formats belonging to Train Benchmark, it is necessary to explain the performance comparison in more details.

Initially, clarify the meaning of performance in the case of Train Benchmark. The key concept on which the framework concentrates the most is the model validation, and thus, under the concept of performance is meant the required time of model validations. The measured validations are executed by various tools, and thus, a concrete validation time characterizes the performance of the particular tool. To summarize, the Train Benchmark framework assesses various tools and their query evaluations performance via model validations.

One important approach of the performance investigation is the scalability of the tools. In order to asses scalability, the benchmark uses instance models of growing sizes, and each model contains twice as many model elements as the previous one.

As it was already emphasized, the benchmark framework measures various tools with different supported formats which naturally indicates the diversity of the instance models. In order to prevent dissimilarities among instance models, the Train Benchmark framework constructs abstract models during the generation phase, independently on formats, and then these models are persisted to the individual acceptable formats belonging to the tools. As a result, the model generation does not depend on the specific formats, and thus, a uniform model is mapped and used by the various implementations. The most important consequence from this fact is that the performance comparison becomes possible among the tools, since they all execute the same validations on the same structured models.

Besides uniformity, it is essential to guarantee reproducibility of the instance models. In order to achieve this, we use pseudo-random[3] generators which makes the model precisely reproducible.

1.1.5 The Foundations of the Framework: Tools

In terms of the implemented tools in the benchmark framework, it is already explained that they are categorized into four different formats. These tools are listed in Table 4.1. Unfortunately, we cannot afford in our research to investigate and analyze all of the currently integrated tools from the framework, hence, we chose a subset of the tools, and typically focus on the RDF-based systems. The analyzed tools on which we concentrate are marked in bold in the table.

[3]It is an algorithm for generating a sequence of random numbers, but actually it can be completely determined by a relatively small set of initial values, called the pseudorandom generator's seed.

1.1.6 The Framework's Architecture

It is important to emphasize that our goal in this paper is not entirely related to the main motivation and aim that belong to the Train Benchmark framework. We are not concerned with model validations (or transformations[4]) , instead, we pay more attention to the first query evaluations and the precise characterization of the model. The latter option is entirely missing from the framework, since we cannot calculate metrics connected to the instance models. However, the embedded model validation components—and the integrated tools themselves—are reusable for our purpose.

Figure 4 depicts the main components are found in the benchmark framework. Based on this, it is clarified which components can be reused, and which needs to be extended or reimplemented completely for our purpose. The first considerable alteration is the Generator unit, which is responsible for the graph-based model generation on an abstract layer, meaning that the component operates independently on the existing formats. Which components are responsible for parsing the model to the specific format, those are the Format-Generator units. Only the RDF-based component needs to be extended. The generation related modifications are described later in details.

As far as the Validation unit is concerned, it can be reused, independently from the fact that we do not consider model validations in our search, however, the query execution units can be utilized.

The Publisher component provides the measurement results and serializes them to the disk in JSON formats.

In summary, the framework's architecture provides an appropriate base to investigate the relationship among the precise characterization of a model, the composition of a particular query, and the evaluation itself.

1.2 Basic Statistics

In this section we present the fundamental concepts of the field of statistics. It is important to emphasize that we do not have the intention to cover most of the theorems in statistics, on the contrary, this section only introduces briefly the important concepts on which our search is based. The majority of the definitions can be found in [25].

1.2.1 Study and Experiment

In a **designed experiment** the engineer makes purposeful changes in the controllable variables of the system or process, observes the resulting system output data, and then

[4]Some instance model transformations are also occurred in the workflow of Train Benchmark after each validation. The goal is to investigate how the implemented tools perform after certain model modifications so that they execute the same validations and transformations again, iteratively. Since currently this area is out of our scope, we do not describe it in details.

makes an inference or decision about which variables are responsible for the observed changes in output performance.

In an **observational study**, the engineer observes the process, disturbing it as little as possible, and records the quantities of interest.

The main essential difference between them is that randomized experiments allow for strong claims about causality[5].

1.2.2 Population and Sample

A **population** is a large set of objects of a similar nature which is of interest as a whole. On the other hand, a **sample** is a subset of objects is drawn from a population [14]. Another approach to define the difference between the concepts is that a population includes all of the elements from a set of data, however, a sample consists of one or more observations from the population [15].

For example, the population can be human beings, and the sample is a finite subset of humankind. To be more constructive, in our case the population contains every possible measurement, and a sample includes a several of them, restricted for a particular tool's measurement results. A population can also be imaginary.

The sample size is the number of observations in a sample and commonly denoted by n.

In the following chapters, we will refer to samples, as we build different ones and analyze them, respectively.

1.2.3 Random Variables and Distributions

Random Variables

In statistics, a variable is as an attribute belonging to an entity, and its value can vary from one entity to another.

When a variable exhibits variability in measurements—as some deviation can be observed between the variable's values—it is considered as a random variable. Assume that X represents a measurement, then the random variable's formula is

$$X = \mu + \epsilon \tag{1.1}$$

where μ represents a constant and ϵ is a random disturbance.

[5]Causation indicates that one event is the result of the occurrence of another event, which means, there is a causal relationship between the two of them.

Probability Distributions

The probability distribution of a random variable X is a description of the probabilities associated with the possible values of X. In other words, a probability distribution links each possible value that a random variable can assume with its probability of occurrence.

One differentiates the discrete and continuous distributions, when a random variable is a discrete variable, its probability distribution is called a discrete probability distribution, similarly to the continuous one.[6]

As we will introduce in details later, the base of our model generation is strongly connected to different discrete probability distributions. The ones that will be taken into account are introduced briefly in the following paragraphs [10].

Uniform Distribution A random variable X has a discrete uniform distribution if each of the n values in its range, say, x_1, x_2, ..., x_n, has equal probability. Then,

$$f(x_i) = \frac{1}{n} \tag{1.2}$$

where $f(x_i)$ denotes the probability distribution function.

Pareto Distribution If X is a random variable with a Pareto distribution[7], then the probability that X is greater than some number σ, is given by

$$\overline{F}(x) = Pr(X > \sigma) = \left(\frac{x}{\sigma}\right)^{-\alpha} \tag{1.3}$$

where σ is the scale, and α represents the shape parameter($x > \sigma$, $\sigma > 0$).

A similar approach is the power-law distribution which describes the probability of the random variable X that—on the contrary of the Pareto distribution—is not higher, but equals to x. This leads to

$$f(x) = c \cdot x^{-\gamma} \tag{1.4}$$

where c is a constant and γ is called as the *exponent* or *scale factor*. Later, we will increasingly pay attention to the power-law distributions.

Poisson Distribution The Poisson distribution is characterized by the following elementary probabilities:

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{1.5}$$

[6] From now on, we only use discrete probability distributions, and in every context, we refer to discrete automatically.

[7] More types of Pareto distributions are distinguished, the commonly known Type I distribution is introduced.

where $\lambda > 0$ is the shape parameter and $k \in \mathbb{N}$.

1.2.4 Related Measurements

It is essential to introduce some measurements related to random variables and their probability distributions on which we will refer later.

Mean

The *mean* or *expected value* of a discrete random variable X, denoted as μ or $E(X)$, is

$$\mu = E(X) = \sum_x xf(x) \tag{1.6}$$

where x represents the values of the random variable X, and $f(x)$ is the probability distribution function. A mean is a measure of the center of the probability distribution.

Variance

The *variance* of X, denoted as σ^2 or $V(X)$, equals to the following formula:

$$\sigma^2 = V(X) = E(X-\mu)^2 = \sum_x (x-\mu)^2 f(x) \tag{1.7}$$

The variance is a measure of the dispersion, or variability in the distribution, as it represents the average of the squared differences from the mean. For example, a variance of zero indicates that all the values are identical.

Standard Deviation

The *standard deviation* of the random variable X is $\sigma = \sqrt{\sigma^2}$, thus, it is the square root of the variance.

1.2.5 Covariance and Correlation

Covariance

Covariance is a measure of linear relationship between the random variables. The covariance value between two random variables X and Y can be expressed by

$$cov(X,Y) = E\big[(X-\mu_X)(Y-\mu_Y)\big] \tag{1.8}$$

where if $cov(X,Y) > 0$, then Y tends to increase as X increases, and if $cov(X,Y) < 0$, then Y tends to decrease as X increases [6]. A few examples are depicted in Figure 1 in

different scenarios where—in the terms of Figure (a) and (b)—a covariance is observable between X and Y, however in Figure (c) and (d), the covariance equals to zero.

Correlation

Similarly to covariance, a *correlation* describes the strength of the relationship between variables. A type of correlation, the Pearson product-moment correlation coefficient is formulated as

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{1.9}$$

where $-1 \leq \rho_{XY} \leq 1$, and 1 indicates a total positive linear relationship between X and Y, -1 means negative linearity, finally, 0 is interpreted as a correlation does not exist between the variables.

1.2.6 Regression Analysis

Regression analysis is a statistical technique for exploring the relationship between two or more variables. A regression model can be considered as an equation that relates a random variable Y to a function of a variable x, and a constant β. In formally, a regression model is defined as

$$Y = \beta_0 + \beta_1 x + \epsilon \tag{1.10}$$

where Y is the dependent or response variable, x is called as an independent variable or predictor, and β_0, β_1 are the regression coefficients, the intercept and the slope, respectively. Finally, ϵ is a random error. More precisely, Equation 1.10 is called a *linear regression model*, since it uses one independent variable to predict the outcome of Y.

A multiple linear regression model considers k independent variables, and the equation is extended as the following

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon \tag{1.11}$$

1.3 Graph Theory

In the following sections we introduce the most important graph metrics and network topologies on which we concentrate in our work. We assume that the reader is already familiar with the basics concepts of graph theory, including directed graph, connectivity, complete graph, adjacent nodes and degrees of vertices.

1.3.1 Metrics

Degree Distribution

The fluctuation among the degrees of the nodes is characterized by a distribution function $P(k)$ which shows the probability that a randomly selected node's degree is equal to k. $P(k)$ is called as the degree distribution.

Clustering Coefficient

The C_n clustering coefficient of an n node is equal to the proportion of connections among those vertices that are adjacent to i divided by the maximum number of connections that can be drawn between them. Formally written, in undirected graphs $C_n = \frac{2 \cdot e_n}{k_n \cdot (k_n - 1)}$, where k_n is the number of neighbors of n and e_n is the number of connected pairs between all neighbors of n ([21]). As a result, the clustering coefficient is always quantified between 0 and 1.

An example is showed in Figure 1.1. In this case, the clustering coefficient of A is $C_A = \frac{2}{6}$, since it has three neighbors, so $k_A = 3$, and only one connection occurs among its adjacent nodes—between B and C—indicates that $e_A = 1$.

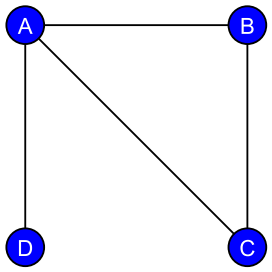

Figure 1.1. An example graph for illustrating the calculation of clustering coefficient.

Betweenness Centrality

The betweenness centrality of an n node is quantified by the number of shortest paths that include n as an intermediate node, divided by the total number of shortest paths. Consequently, this metric is mapped between 0 and 1.

1.3.2 Network Topologies

Random Graph

Small-World Model of Watts-Strogatz

Scale-Free Model of Barabási and Albert

Hierarchical Network

1.4 Resource Description Framework: RDF

Chapter 2

Related Work

In the following sections, we introduce four benchmark frameworks that are designed to investigate the performance of RDF databases. In the end of the chapter 2.5, we summarize the frameworks and compare them to our work which is the introduction of a new approach to the existing benchmark frameworks.

2.1 Berlin SPARQL Benchmark

The Berlin SPARQL Benchmark (BSBM) is settled in an e-commerce use case in which a set of products is offered by different vendors and consumers who posted reviews about the products [24]. BSBM measures the SPARQL query performance of RDF-based tools via realistic workloads of use case motivated queries. The benchmark defines three different use cases and a suite of benchmark queries—a *mix* of queries—in each of them [4], simulating the search and navigation pattern of a consumer looking for a product. The queries include read and update operations as well.

BSBM generates artificial models in different sizes, by using normal distributions among the elements. For example, the number of reviews and different properties per products are distributed following a normal distribution.

The benchmark defines particular performance metrics that relate to the query execution times from different perspectives. The most important metrics are the following:

- *Queries per Second*: It is equal to the number of queries executed within a second.
- *Query Mixes per Hour*: Denotes the number of mixed queries with different parameters that evaluated within an hour.
- *Overall Runtime*: The overall time that a certain amount of query mix require.

2.2 DBpedia SPARQL Benchmark

The DBpedia SPARQL Benchmark (DBPSB) proposes a benchmark framework for RDF databases based on the DBpedia [7] knowledge base, and it measures the performance of real queries that were issued against existing RDF data [31]. DBPSB generates models in various sizes trying to obtain a similar characteristic belonging to the original DBpedia dataset.
Similarly to BSBM, DBPSB also defines metrics to investigate the performance from different aspects, such as the *Query Mixes per Hour* and *Queries per Second*.

2.3 SP^{2}Bench

The SPARQL Performance Benchmark (SP^{2}Bench) [35] is based on the *Digital Bibliography and Library Project* (commonly known as *dblp*) which provides an open bibliographic information on major computer science publications [8]. The benchmark queries are not explicitly evaluated over the *dblp* dataset, since SP^{2}Bench uses arbitrarily large, artificially generated models for the measurements that are created to follow the characteristics of the original *dblp* dataset, such as the power-law distribution.

SP^{2}Bench is designed to test the most common SPARQL constructs, operator constellations and a broad range of RDF data access patterns. Instead of defining a sequence of use case motivated queries, the framework proposes various systematic queries that cover specific RDF data management approaches.

Similarly to BSBM, SP^{2}Bench also measures additional performance related metrics besides the evaluation time, such as the disk storage cost, memory consumption, data loading time and success rates, hence, every metric captures different aspects from the evaluations.

2.4 Train Benchmark Framework

2.4.1 Overview

The basic idea and elaboration of the Train Benchmark framework was introduced by the Fault Tolerant Systems Research Group in the Budapest University of Technology and Economics [29]. The benchmark investigates the performance of model validations via different graph queries by measuring the evaluation times of different RDF, SQL and graph databases.

An overview of the Train Benchmark framework is depicted in Figure 2.1, illustrating the framework how connects the model validations to the database systems. In the 1. step, the framework generates a graph-based model derived from a particular domain. After defining different constraints (rules) on the domain—such as the presence of an edge between two types of nodes—the benchmark injects erroneous elements into the model (step

2.), which elements are considered as violations of the constraints. During the 3. step, the framework loads the invalid model to the particular database system.
Every constraint has a corresponding validation pattern that is the negation of the constraint and it includes a pattern of elements that violates the certain constraint (4.). Every validation pattern is defined in the own query language of the particular database (5). Step 6. denotes the query evaluation that represents the model validation. The result set of the evaluation contains the invalid elements which violated the constraint. The performance indicator of the databases is the required time of model validation, so the query evaluation time.
Besides the validations, the framework also performs model transformations to alter the amount of invalid elements in the model. Every transformation is followed by another validation.

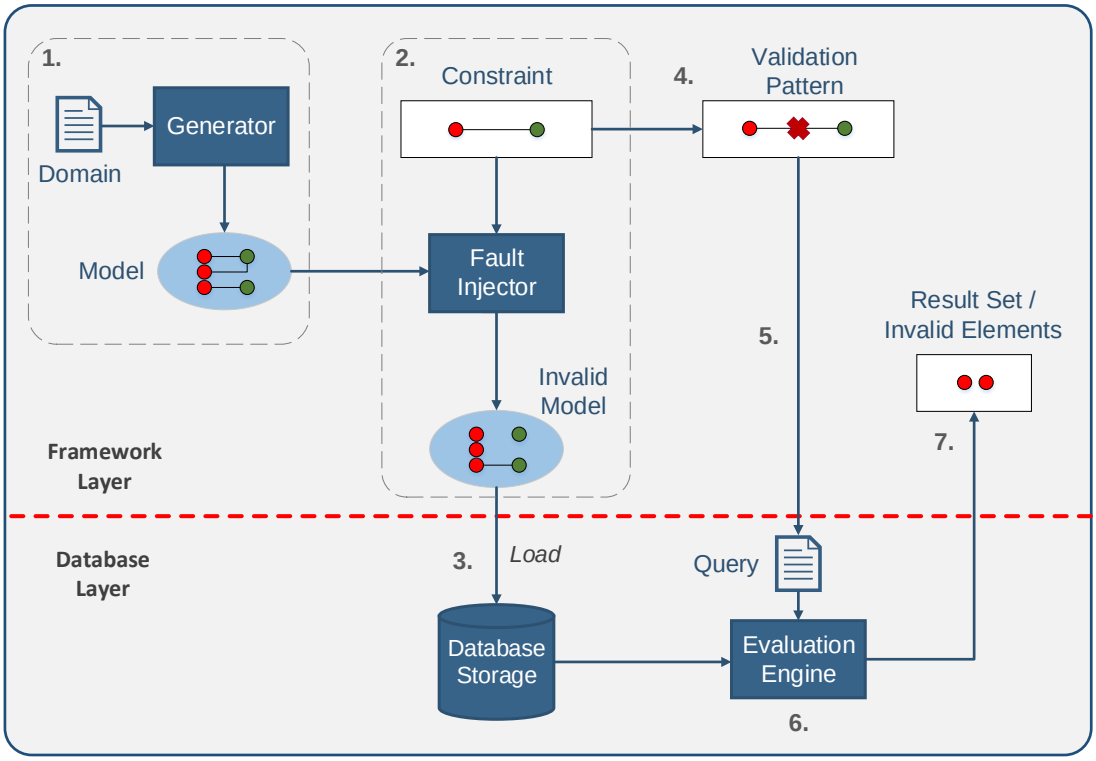

Figure 2.1. An overview of Train Benchmark.

2.4.2 Model Generation

The Train Benchmark framework uses artificially generated graph-based models for the measurements over a *Railway* domain, illustrated in Figure 2.2. In the models, a train route is defined by a sequence of sensors, and the sensors are associated with track elements which are either segments (with a specific length) or switches. A route follows certain switch positions which describe the required state of a switch belonging to the route. Each route has a semaphore on its entry and exit [19].

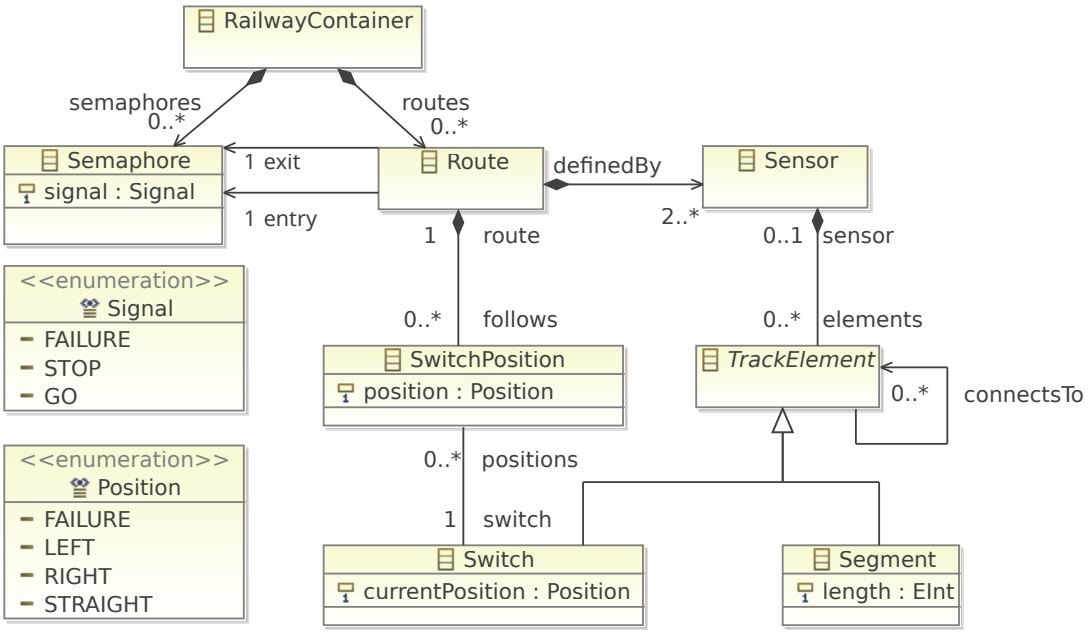

Figure 2.2. The Railway domain of Train Benchmark.

The generated models over the Railway domain are not related to a real-life model, which leads to the fact that the cardinalities of the elements and their distributions do not follow a real model's characteristic, therefore, the measurement results of different databases cannot be claimed to be representative in a real-life use case.

2.5 Conclusion

The represented benchmark frameworks propose different comprehensive use cases to assess the performance of graph queries typically over RDF data. BSBM and DBpedia use representative queries for the measurements that demonstrate real-life use cases, and the SP^{2}Bench framework concentrates on the creation of various systematic queries that investigate the specific RDF data management approaches. The Train Benchmark framework aims a different aspect of performance comparison by concentrating on model validations via graph queries. As a conclusion, these frameworks guarantee a comprehensive performance evaluation of a workload—the ensemble of model, query and tool—by emphasizing the impact of *queries* to the performance.

However, in case of an arbitrary workload, the *model* also represents a dominating factor to the performance. BSBM, DBpedia or the SP^{2}Bench framework do not consider model modifications in their workloads, and thus, they do not investigate the impact of various model characteristics to the performance, they only generate the same structural model in different sizes to measure scalability. Even though Train Benchmark considers model transformations, yet, it modifies only a subtle set of the elements during the measurements, and the generated models are still considered as static models. As a conclusion, all of the

introduced frameworks concentrate on the generation of one static model without altering its internal structure and analyzing their effect to the performance.

In order to introduce a new approach to the field of graph-based benchmark frameworks, we focus on elaborating a framework that proposes various models with different characteristics and investigates the relationships between the performance and the internal structure of the model.

The main features of the introduced frameworks and our new approach are summarized in Table 2.1.

Feature	BSBM	Dbpedia	SP^{2}Bench	Train Benchmark	Our Approach
Based on a real-life model		•	•		•
Realistic domain-based queries	•	•			
Systematic queries			•		•
Model transformations				•	
Dynamic models					•
Measurement of scalability	•	•	•	•	•
Performance metrics	•	•	•		
Model metrics					•
Query metrics					•
Extended scope besides RDF				•	

Table 2.1. A comparison between the existing benchmark frameworks and our approach.

Basically, we concentrate on the generation of dynamic models with different internal connectivities based on a real-life data, and we assess the evaluation time of systematic queries that test specific features of the models. In this part of our research, we highly focus on model and performance correlations.

Chapter 3

Design

3.1 Overview of the Approach

Our concept includes the following systems illustrated in Figure 3.1. We rely on two existing frameworks in our work, the Train Benchmark and MONDO-SAM frameworks, furthermore, we elaborate MONDO-MAP, the *MONDO-Metrics-Based Analysis of Performance* framework and the Homogeneous Graphs Benchmark.

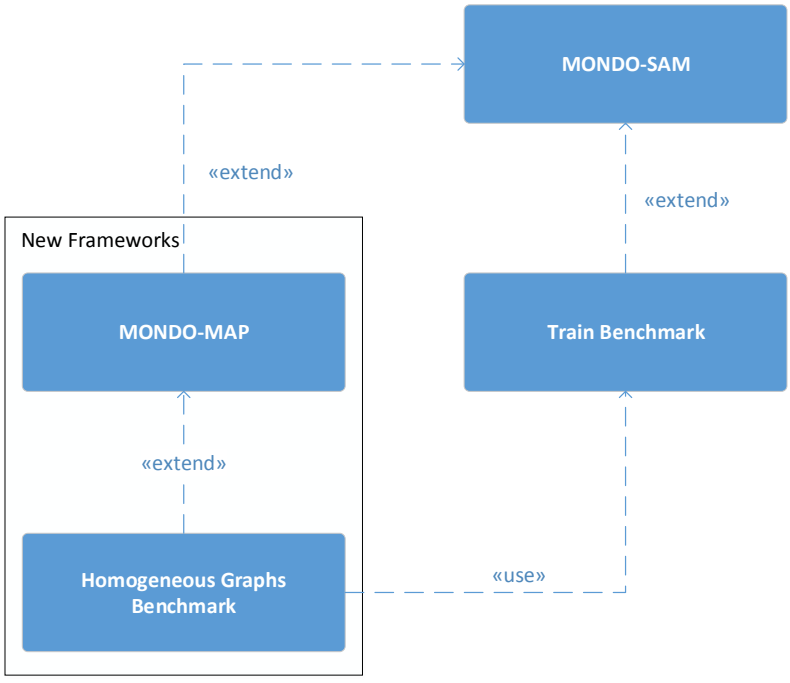

Figure 3.1. An overview of the frameworks in our approach.

MONDO-SAM The MONDO-SAM framework was created under the project MONDO [12] with the motivation to provide a common benchmark framework in Model

Driven Engineering (MDE) for benchmark developers [28]. MONDO-SAM can be considered as an abstract layer that proposes an evaluation engine to execute arbitrary workflows independently on the current workload, furthermore, MONDO-SAM also guarantees the serialization and data visualization of the benchmark results.

Train Benchmark The Train Benchmark framework is based on the evaluation engine provided by MONDO-SAM, and it proposes a benchmark for measuring continuous model validations and transformations, introduced in Section 2.4.

MONDO-MAP MONDO-MAP extends MONDO-SAM with further features, as it provides a framework for analyzing model and performance relationships in the light of an arbitrary workload by characterizing the performance quantitatively with model related metrics. Similarly to MONDO-SAM, MONDO-MAP is an abstract framework and can be extended by an arbitrary benchmark case.

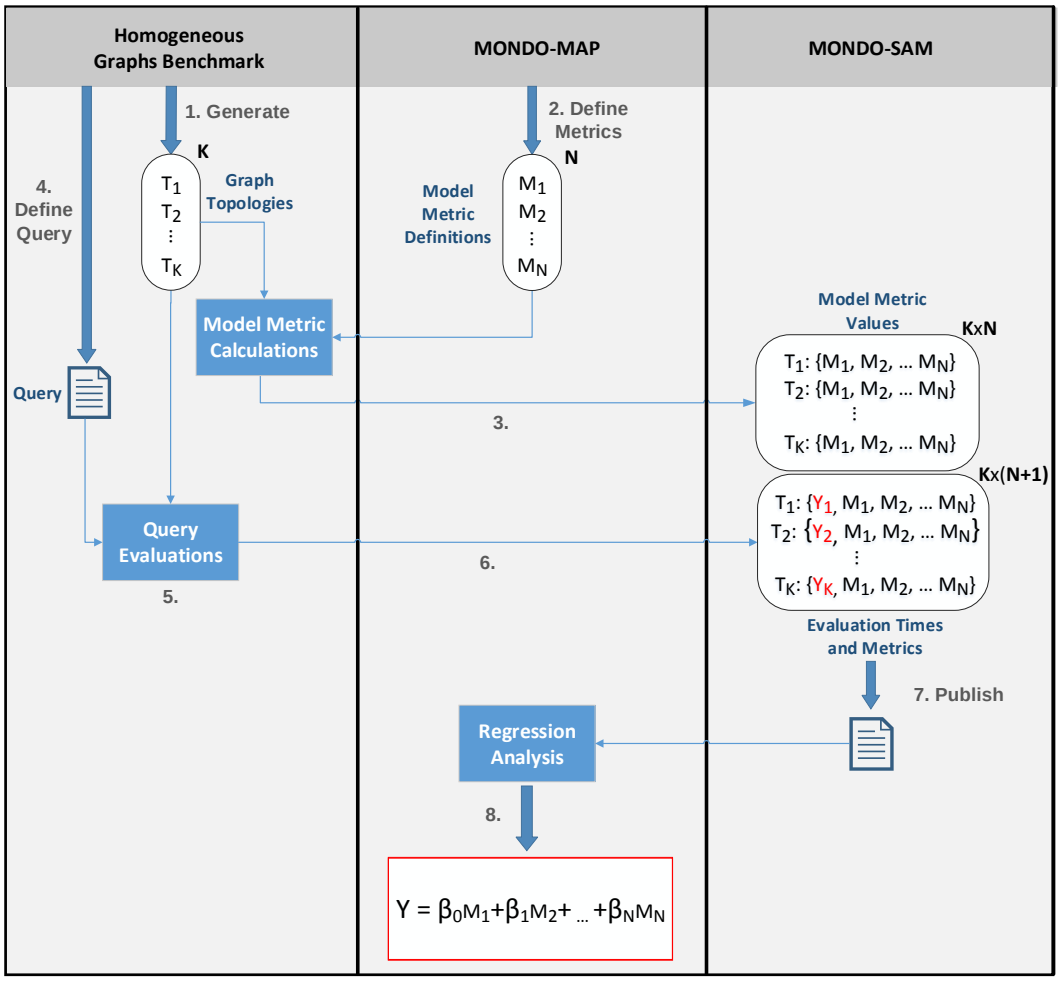

Figure 3.2. The main concept of the metric-based performance analysis.

Homogeneous Graphs Benchmark The Homogeneous Graphs Benchmark (HG Benchmark) extends the MONDO-MAP framework with a benchmark case for RDF databases. It generates homogeneous graphs —well-known network topologies—and it investigates the relationships between the model metrics and performance of query evaluations with respect to an arbitrary query and database. The goal is to generate artificial graphs with various characteristics and obtain a fluctuation in their descriptive metrics, and thus showing a quantitative connection between the model metrics and performance. As Figure 3.1 suggests, in the HG Benchmark we use a part of the components of Train Benchmark that are adequate for our purpose as well.

Figure 3.2 illustrates the main concept of our work. First, the HG Benchmark generates different K number of graph topologies. Using the graph metric definitions from MONDO-MAP (2.), we calculate the descriptive metrics for every topology in step 3. As it is showed, every topology contains an own values for every N number of metrics, and they are stored in MONDO-SAM. In the next two steps (4-5.) we define a query and evaluates it on every topology in the particular RDF database that we intend to measure. The measurement result is the query evaluation time (Y) that represents another variable belonging to the topologies besides their metrics (6.). As it was mentioned before, the MONDO-SAM framework is responsible for publishing the benchmark results(7.). Finally, in MONDO-MAP we analyze the results by creating regression models to estimate the influence of metrics to the performance.

3.2 Models and Metrics

3.2.1 Real-Life Networks

In the discipline of graph theory, the internal structures of real-life networks are comprehensively investigated. The main approach in the analysis of these networks is to explore the degree distributions and study the specific metrics—typically the clustering coefficients, average degrees, average shortest paths—that are suited to characterize the graphs appropriately. Based on the degree distributions and metrics, one can draw conclusion how a particular real-life network shows a similar characteristic to the well-known topologies such as the random graph, scale-free model, small-world model of Watts-Strogatz, or hierarchical network.

For example, the network of world-wide-web is studied in [33] and [27] as well, and the authors observe that the degree distribution of the *www* follows a power-law distribution with a heavy tail[1], which indicates the presence of web pages with significantly higher degrees than the average degree. Since the probability of occurrences of these web pages is considerably low, the connectivity of the world-wide-web can be represented by the scale-free model of Barabási and Albert.

[1]The concept of heavy tail means that there is a larger probability of receiving significantly higher values, than it is normally expected [11].

More examples can be found in the study of Barabási and Albert [22], as they review the advances of different publications and investigate the characteristics of different real-life networks (Table). Empirical results prove that the *movie actor collaboration network*, *cellular networks*, *phone call* and *citation* networks also follow power-law distributions. Many of the studied networks can be considered as scale-free models, however, a part of these graphs—for example the network of movie actors—also show small-world properties and high clustering coefficient in their connectivity similarly to the Watts-Strogatz or hierarchical topology.

As far as the Watts-Strogatz model—and the random graph[2]—are concerned, their specific degree distribution—Poisson—rarely appears in the real-life networks, as it is emphasized in [32]. Actually, none of the artificial topologies can be identified perfectly to real-life models, however, the ensemble of the specific attributes of these well-known topologies are frequently appear in them. As a conclusion, we conjecture that the characteristics belonging to the well-known topologies are the foundations of our solution, namely, finding those representative metrics of graphs that are suited to characterize not only the model appropriately, but the relationship between model and performance as well.

3.2.2 Network Topologies and Representative Metrics

Barabási and Albert inspect the natures of the well-known graph topologies in [22], such as the random graph, scale-free and the Watts-Strogatz model. As a main result, they observe that there are significant differences among the topologies regarding specific graph metrics. Based on their study and the research of hierarchical graphs [34], the following metric deviations are assumed between the four topologies, illustrated in Table 3.1. The random graph is considered as a reference point, and every value is compared to its metrics by assuming that the networks are in the same size. As Table 3.1 demonstrates, each

Metric	Random	Hierarchical	Scale-free	Watts-Strogatz
Max Degree	•	•••	•••	•
Clustering Coefficient	•	•••	•	••
Avg Shortest Path Length	•	••	•	••

Table 3.1. Graph topologies and their descriptive metrics

topology can be characterized by different metric values, which leads to the assumption that if the diversity between the topologies may cause fluctuations in the performance of a particular query evaluation, then these metrics are adequate to characterize the model and performance relationships quantitatively.

However, the values of metrics in Table 3.1 are misleading due to the reason that the metrics of the Watt-Strogatz model highly depend on the initialization of the network, namely, the value of p probability that is used in the generation.

[2]The topology of Watts-Strogatz is actually considered as a bridge between a lattice and a random graph based to its p probability value. Thus, according to p, the WS model shows uniform or Poisson distribution.

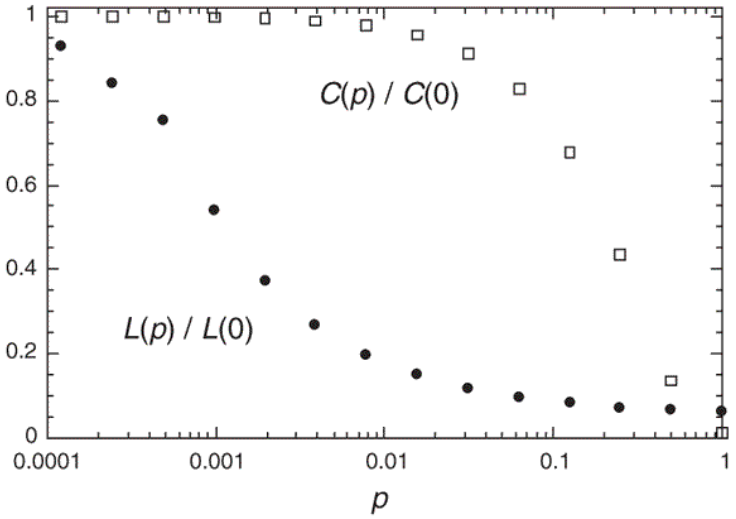

Figure 3.3. Characteristic path length $L(p)$ and clustering coefficient $C(p)$ of Watts-Strogatz model

By modifying p, the Watts-Strogatz model represents a bridge between a lattice and a random graph. As Figure 3.3 illustrates[3], the clustering coefficient ($C(p)$) and average shortest path ($L(p)$) metrics are changed with respect to p scaling. The values are normalized by $L(0)$ and $C(0)$ that represent the clustering coefficient and average shortest path metrics for a lattice graph. As a conclusion, the Watts-Strogatz model shows significant deviations in these two metrics in the light of the p probability value.

Besides the models in Table 3.1, the metrics also require modifications. The problem is that the maximum degree metric alone does not include a comprehensive information about the internal structure of the network, since it does not emphasize the role of a node with maximum degree in the connectivity of the graph. Hence, we introduce another metric—the *betweenness centrality*—which characterizes adequately the importance of a higher degree, since the higher value of betweenness centrality belongs to a vertex, the more shortest paths include that node, symbolizing that node represents the center in the graph.

After these minor modifications, the topologies and the related metrics are showed in the extended Table 3.2. The WS-x abbreviations symbolize the Watts-Strogatz models indicating that the p value is equal to x. The values of the betweenness centrality are determined by our initial conjecture considering that the center node in a hierarchical network and the *hubs* in a scale-free model may occur more times in the shortest paths due to the fact that they have higher degrees. The main conclusion is that we can achieve a higher deviation among the metric values by using these topologies, and thus, in the following we concentrate on these networks.

[3]The original figure can be found in [36].

Metric	Random	Hierarchical	Scale-free	WS-0.1	WS-0.01	WS-0.001
Max Degree	•	•••	•••	•	•	•
Clustering Coefficient	•	•••	•	••	•••	•••
Avg Shortest Path Length	•	••	•	•	••	•••
Betweenness Centrality	•	•••	••	•	•	•

Table 3.2. Graph topologies and their descriptive metrics with extensions

3.3 Metric and Performance Comparison

Showing an appropriate performance and metric relationship is an essential part of our approach. The first notion is the search of correlations between the metrics and performance.

A similar problem is studied in [30] and [23], where the authors generate well-known topologies and inspect the connectivity and robustness of the networks. In their case, a network is said to be robust if its performance is not sensitive to the changes in topology. In [30], the algebraic connectivity metric is studied to search robustness and metric relationships, however, they show that the algebraic connectivity is not trivially correlated to the robustness of the network.
The authors in [23] investigate the impact of betweenness centrality, algebraic connectivity and average degree to robustness, and they also draw the conclusion that there is no unique graph metric to satisfy both connectivity and robustness objectives while keeping a reasonable complexity, since each metric captures some attributes of the graph.

Partly based on the advances of these two publications and our initial assumption of the topologies and their metrics (Table 3.2), we expect that we cannot find a correlation between one metric and the performance, hence, we conjecture that only the ensemble of more metrics is suited to find relationship.

In order to find quantitative connections, we use regression analysis to show how the various metrics impact the performance.

3.3.1 Sample Choosing

By using regression analysis on a sample, it is inevitable to regard the sample to be unbiased. In our case, a bias in a sample of graph topologies means a fluctuation in the size of the models. Obviously, one topology in the sample with larger amount of nodes can bias the connection between the models and the performance, therefore, our framework must support the generation of *uniform* models with respect to the amount of nodes and edges, even in the case of different topologies as well.[4]

[4]From now on, under the concept of uniform models we mean models with approximately equal number of nodes and edges, despite the diversity of their internal structures.

Chapter 4

Contributions

In the following sections, we present the main contributions related to the MONDO-MAP framework and the Homogeneous Graphs Benchmark.

4.1 Overall Architecture

Figure 4.1 depicts the frameworks and the main components belonging them, additionally, it also denotes that which components are reused from Train Benchmark.

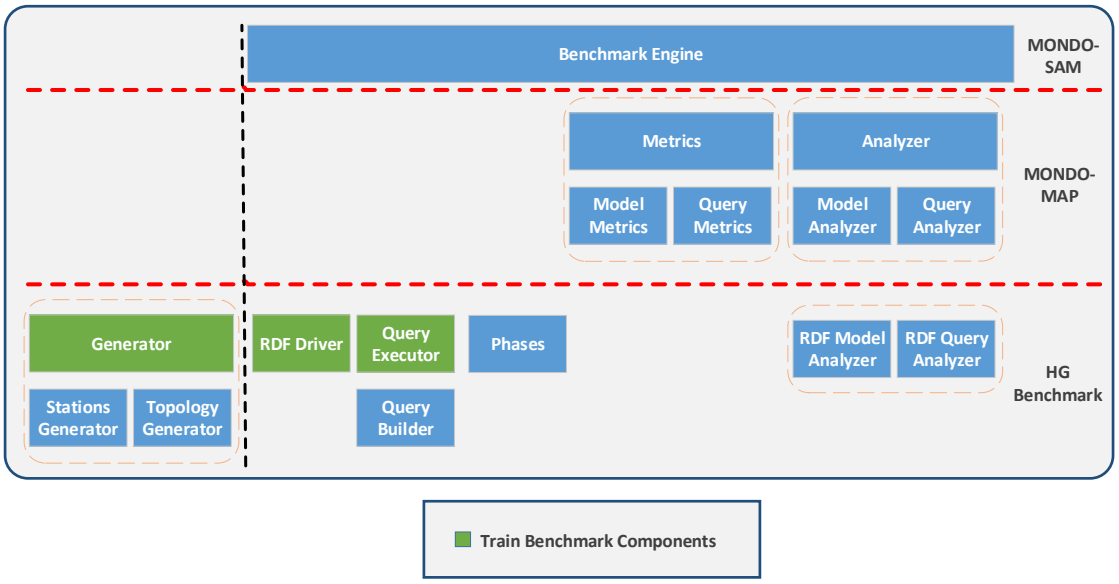

Figure 4.1. The architecture of our approach.

In the following parts, we introduce the components.

Benchmark Engine The Benchmark Engine in MONDO-SAM is responsible for evaluating an arbitrary sequence of phases consecutively. A phase is considered as the atomic execution unit in a benchmark. The engine also measures the evaluation times of phases,

hence, it is the component that measures the performance of query evaluations in our work.

Analyzer Components The Model and Query Analyzer units belong to the MONDO-MAP framework and define an interface for the metrics calculation. The Model Analyzer investigates the model related metrics, and the Query Analyzer relates to the query definitions. As it can be observed, the concrete metric calculations—RDF Model Analyzer and RDF Query Analyzer—appear in the HG Benchmark.

Metrics The definitions of metrics—such as Model Metrics and Query Metrics—can be found in MONDO-MAP. The model metrics symbolize graph metrics with applying the commonly used naming conventions from graph theory. Note that the HG Benchmark does not contain further RDF-based metrics implying that we use the graph-based naming conventions in our work.
The query metrics connect to the definitions of the queries, and they do not relate to the query performance or the result set of the evaluations.

Generators The generator components belong to the HG Benchmark. The abstract Generator unit is utilized from Train Benchmark. The Stations Generator component is responsible for transforming the real-life British Railway Stations model to RDF. Last, the Topology Generators construct different homogeneous graphs fitting to well-known topologies and transform them to RDF format.

RDF Driver The RDF Driver manages the connections between the measured RDF databases and the benchmark framework, furthermore, it also accomplishes the loading of the models.

Query Executor and Query Builder The query evaluations are achieved by the Query Executor component. The Query Builder is responsible for creating and altering the query definitions in runtime.

4.2 British Railway Stations

In order to test our regression models on an entirely different graph, we introduce a real-life model to our work. We use a model of train stations provided by the *Network Rail* company that runs, maintains, and develops Britain's rail tracks [13]. Network Rail publishes a number of different data available to developers, one of them is a *daily extracts and updates of train schedules* [20] that we intend to use.

The data contains approximately 10 000 British stations that belong to train schedules as destinations. We concentrate on the network of stations, and map it to a graph where every

vertex symbolizes a station. An edge is drawn between two nodes—so two stations—if these stations appear consecutively in a destinations path belonging to a schedule.

4.3 Uniform Model Generation

It is an essential expectation from the HG Benchmark to guarantee uniform model generation among the topologies indicating the same size of the generated models. We propose a model generation technique to generate topologies with the same amount of nodes and edges.

4.3.1 Number of Nodes

The random graph and the Watts-Strogatz model are constructed by initializing $|V| = N$ number of vertices, and then the algorithms determine which one of them become adjacent. In the scale-free model generator, the nodes are created incrementally until $|V| = N$. As a conclusion, a precise number of nodes can be obtained regarding these topologies.

The only one problem about generating topologies with a certain number of nodes is the recursive algorithm of the hierarchical graph, which algorithm has to be terminated.

The Termination of the Hierarchical Network Algorithm

Since the generation of hierarchical network is recursive instead of being incremental—as in the case of the three other topologies—it is inevitable to determine a termination from the recursion. The termination point is evident, as soon as the number of created nodes reaches the limit, the algorithm has to be stopped. However, it cannot be predicted in which phase the algorithm stops exactly. As a consequence, the possible failures have to be managed.

The possible problematic cases are demonstrated in Figure 4.2. In Case A, B, and C, the expected numbers of nodes are 20, 19, and 16, respectively. As it can be observed in these cases, this limit is always reached before the fourth cloning occurs, since 5 clusters should be created with 25 vertices at the end of this step in the recursion.

In the solution in Case A, the generator stops the clone procedure and connects the diagonals to the center. Regarding B, the termination happens during the generation of a cluster. As a solution, the last cluster becomes partial, and similarly, every diagonal is attached to the center. Case C represents that scenario when the last cluster only consists of one node. To prevent isolation, the last vertex is considered as a diagonal, and be connected to the center.

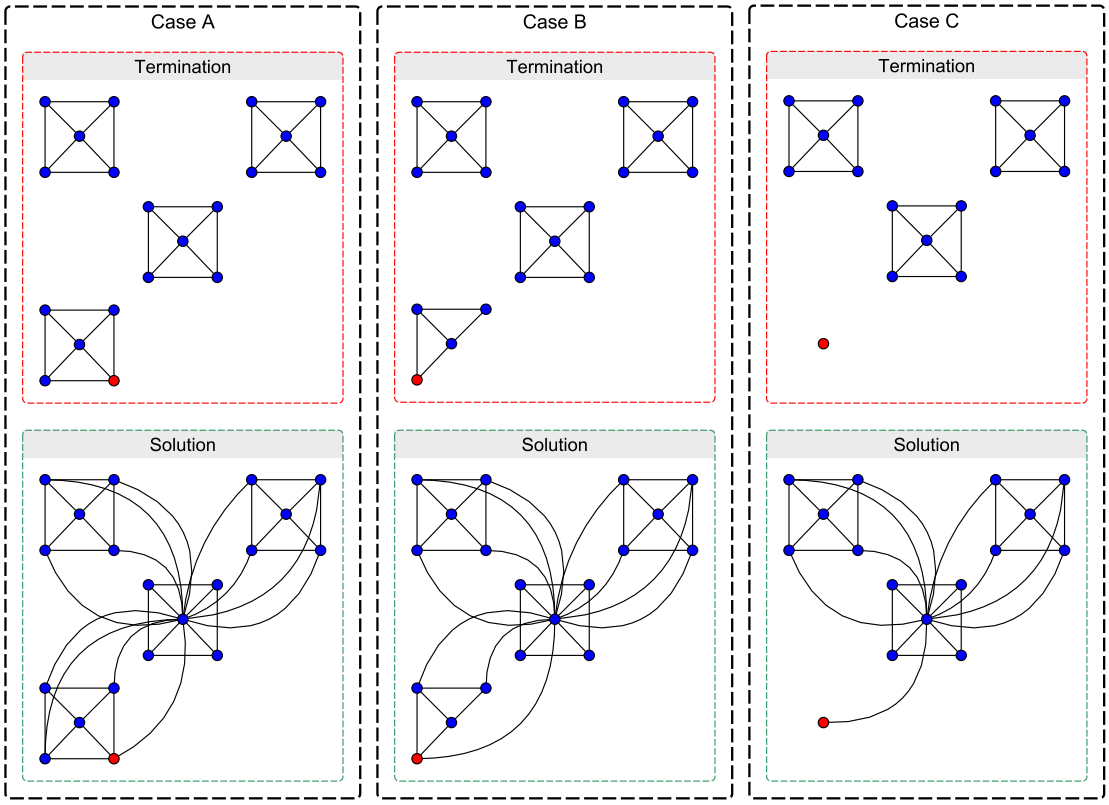

Figure 4.2. Possible termination problems in the hierarchical graph generation

4.3.2 Number of Edges

In terms of the random graph, scale-free, and WS model, their generation algorithms can be adjusted arbitrarily, meaning that an optional number of nodes or edges can be achieved. Actually, reaching a certain amount of nodes or edges in these networks are handled separately.

Unfortunately, the creation of nodes and edges in the hierarchical graph occurs together. Since the amount of edges depends on the number of nodes and iterations in the recursive algorithm, it cannot be configured arbitrarily. A solution is that we adjust the other topologies to have the same number of edges as the hierarchical model. This solution requires to calculate the exact number of edges in a hierarchical network with respect to the iteration.

Estimate the Number of Edges in Hierarchical Graphs

The literature relating to hierarchical graphs does not mention the exact number of edges or its correlation to the amount of nodes, hence, we propose a solution to estimate $|E|$ in the recursive algorithm for every iteration.

At first, let define the necessary variables hereunder:

- i: Represents the current iteration in the original hierarchical algorithm.
- c: Indicates the number of clones in every iteration.
- n: The cluster size is denoted by n, which cluster is a K_n complete graph.
- F_i: It indicates the constructed graph after the i iteration.
- $|E_{F_i}|$: The number of edges of F_i.

In the 0. iteration, the hierarchical graph consists of one K_n cluster. Formally defining, $F_0 = K_n$, and $|E_{F_0}| = |E_{K_n}| = \frac{n \cdot (n-1)}{2}$. In the 1. iteration, the algorithm clones F_0 for c times, and connects the peripheral nodes from each F_0 to the center node. It entails that

$$|E_{F_1}| = (c+1) \cdot |E_{K_n}| + c \cdot (n-1) \tag{4.1}$$

since $c+1$ number of K_n can be found in F_1, and $c \cdot (n-1)$ edges can be drawn from the c number of replicas to the center.

Note that in the first iteration K_n can be substituted with F_0, and the algorithm in the first part of the i iteration creates c number of replicas of the result of the $i-1$ iteration, namely, F_{i-1}. In the second part of the i iteration, the algorithm connects the clusters from the cloned replicas to the center node. These connections are made between the peripheral nodes in each replica and the center, which indicates that the algorithm—in the i iteration—connects $n-1$ peripheral nodes from c number of replicas of F_{i-1}, and due to F_{i-1} includes c^{i-1} number of clusters, we obtain

$$|E_{F_i}| = (c+1) \cdot |E_{F_{i-1}}| + c \cdot c^{i-1} \cdot (n-1) = (c+1) \cdot |E_{F_{i-1}}| + c^i \cdot (n-1) \tag{4.2}$$

We are not finished yet, since Equation 4.2 is equal to the number of edges of a completely finished hierarchical network. However, the generation algorithm in MONDO-MAP is possibly terminated to reach a certain number of nodes, which leads to the fact that $|E_{F_i}|$ must be scaled down by the proportion of the maximum ($|V_{F_i}|$) and the required number ($|V_H|$) of vertices. If the hierarchical graph we intend to generate is denoted by H, then

$$|E_H| = |E_{F_i}| \cdot \frac{|V_H|}{|V_{F_i}|} \tag{4.3}$$

where $\frac{|V_H|}{|V_{F_i}|} \leq 1$.

By using Equation 4.3, we can calculate the number of edges of a hierarchical graph and configure the other topologies to reach the same quantity.

Configure Random Graph

From the two known algorithms, Gilbert's $G(n, p)$ model is adapted to the framework, which implies that the exact value of p has to be determined from the number of edges in the hierarchical graph, $|E_H|$. Based on [26], the p probability can be calculated from the number of nodes and edges as follows:

$$p = \frac{|E_H|}{\binom{|V|}{2}} \tag{4.4}$$

where $|V|$ denotes the number of nodes.

Configure Watts-Strogatz Model

Regarding the Watts-Strogatz model, in the beginning of the generation algorithm, N number of consecutive nodes are connected to each other. During the algorithm, by rewiring the edges the amount of $|E|$ is not changed. As a conclusion, in order to achieve a uniform size similarly to the hierarchical graph, N has to be adjusted as $N = \frac{|E_H|}{|V|}$.

Generally, the N value in the algorithm is a constant integer. In order to configure the WS model properly, we extend the algorithm by defining an inclusive lower (N_1) and upper (N_2) bound for N, as $N \in [N_1, N_2]$, and we also assign a p probability to N that determines the likelihood of N is equal to N_2 and $1 - p$ to N_1. Derived from the equation $N = \frac{|E_H|}{|V|}$, it results in $N_1 = \left\lfloor \frac{|E_H|}{|V|} \right\rfloor$ and $N_2 = \left\lceil \frac{|E_H|}{|V|} \right\rceil$, additionally, the p probability equals to the fractional part, as $p = \left\{ \frac{|E_H|}{|V|} \right\}$. Hence, by turning N to a random variable, we can generate WS models with the same number of edges as $|E_H|$.

Configure Scale-Free Model

The scale-free topology is generated incrementally, since every step a new node is inserted to the graph with m number of new connections. To obtain $|E_H|$, the m variable has to be configured. This leads to $m = \frac{|E_H|}{|V|}$.

In the original generation algorithm, every new vertex connects to a constant number of disjunct nodes, which indicates that m is a constant integer. Similarly to the notion in the Watts-Strogatz model generation (4.3.2), this constant value is converted to a random variable based on a particular probability, derived from $|E_H|$.

4.3.3 Possible Model Configuration

In the artificially generated models the size, topology and density is optionally configurable.

The size of the model—the number of nodes—is calculated by a formula as $|V| = s \cdot 2^i$, where s is the step size constant and i is a positive integer. It implies that an arbitrary model size can be obtained among the topologies.

Besides the size, the density of the graphs is also configurable, so the number of edges in the topologies. Since the hierarchical network is considered as the reference model due to the uniform model generation, the density parameter calibrates the generation algorithm of the hierarchical graph, namely, the size of the K_n clusters with altering n.

4.4 Performance Analysis

4.4.1 Workflow

A workflow in the HG Benchmark is divided into phases that are considered as the atomic execution units. An arbitrary sequence can be created among the phases, which—during the benchmark—are executed consecutively by the evaluation engine in MONDO-SAM.

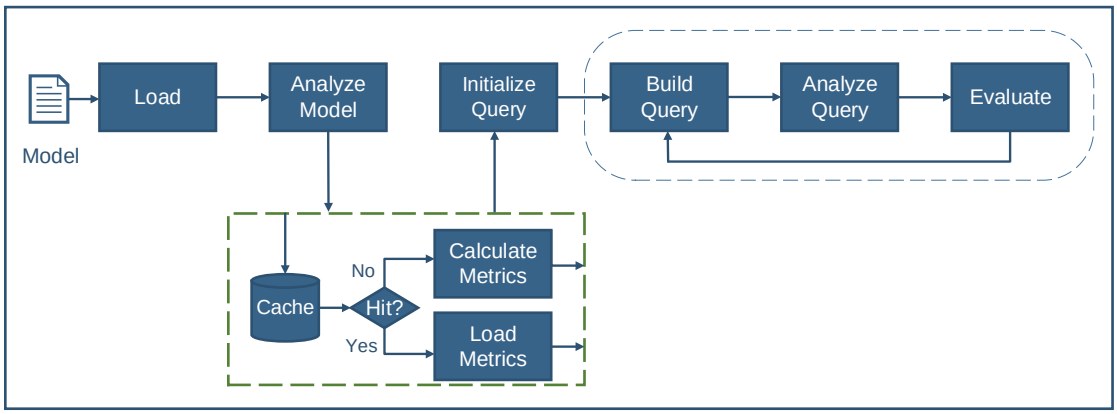

Figure 4.3. The workflow of the HG Benchmark.

The workflow of the HG Benchmark is represented in Figure 4.3. After loading the model, the framework calculates the model related metrics. Due to the fact that in the current phase of our research we do not consider model transformations, therefore, a particular model's metrics can be calculated only once in the beginning of the workflow, and more importantly, different runs of the benchmark can utilize the previously calculated metrics that belong to the same model. As it can be observed in the model analysis phase, the solution is achieved by using a cache for the calculated metrics and reusing its content if possible.

The features in the Initialize and Build Query phases are strongly correlated. These two phases entail the creation of dynamic queries. The first one provides a default query definition that can be parameterized, and the second phase assembles a complete query for the evaluation, as it injects parameters or alters the entire syntax. The latter operation implies that entirely different queries can be executed in a sequence.

Last, the evaluation phase is responsible for executing the query. The building and evaluating phases can be repeated implicating that more then one query can be evaluated in a sequence, even with different definitions.

4.4.2 Metrics Calculation

As we already emphasized, the MONDO-MAP framework proposes two types of metrics. The first is the set of model descriptive metrics and the second relates to the query syntaxes. In the following, we introduce them and their calculations in the HG Benchmark.

Model Metrics

The model-based metrics are connected to graph metrics which appear in their naming conventions as well. Since we are concentrating on RDF tools in our work, we also define the corresponding interpretations. The metrics are listed hereunder.

1. **Nodes:** the number of nodes in the graph. In RDF, this equals to the number of unique subject and object values.
2. **Edges:** the number of edges in the graph. Regarding RDF, this is equal to the number of predicates[1] in the data.
3. **Maximum Degree:** the maximum number of predicates per subjects.
4. **Average Degree:** it is determined by calculating the degree of every existing node.
5. **Average Degree Distribution:** denotes the probability that a randomly selected node's degree is equal to the average degree.
6. **Higher Degree Distribution:** the cumulative distribution of those nodes that have higher degrees than the average.
7. **Average Clustering Coefficient:** this metric implies the calculation of clustering coefficient per every node.
8. **Average Shortest Path Length:** the calculation of this metric requires the most cost, hence, the framework searches a limited number (100) of shortest paths between randomly chosen vertices and calculates the average length of them.
9. **Maximum Betweenness Centrality:** the value of this metric is determined by the shortest paths. We count the occurrences of every intermediate node in the paths—which is the betweenness centrality of the vertices—and normalize the values to the $[0, 1]$ interval by dividing them with the number of visited nodes. Since the value of betweenness centrality is individual per nodes, we use the maximum of them.

[1] With the consideration of rdf:type predicates, the number of edges metric represents the number of triples.

4.4.3 Queries

We investigate the performance of two queries in our HG Benchmark. The first one relates to the concept of shortest path, and the second connects to the notion to investigate the spread of information in the graphs.

Shortest Path Query

The SPARQL definition of the query is shown below:

```
PREFIX  base: <http://www.semanticweb.org/ontologies/2015/hgbenchmark#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  (count(*) AS ?count)
WHERE
  { ?sourceStation rdf:type base:Station .
    ?sourceStation (base:neighbor)* ?targetStation
    FILTER ( ?sourceStation = base:_ID1 )
    FILTER ( ?targetStation = base:_ID2 )
  }
```

The query uses the * operator from SPARQL Property Paths [17] in the (base:neighbor)* predicate which means an arbitrary length of path with the neighbor predicates. The query is considered as a parameterized query in which we inject two random identifiers instead the ID parameters. Finally, the query searches a path between those two randomly chosen nodes.

Information Spread Query

The query investigates the spread of information in the graph—by starting from a randomly chosen node—and it traverses the graph via the neighbor predicates to a three-hop distance. The concept of information spreading means that how fast the information can be forwarded among the nodes in the graph, or in other words, how many vertices can be reached from a certain node.
The SPARQL definition of the query is showed below. As it can observed, in every new navigation we filter the previously found nodes to prevent the traversal of the same nodes again.

```
PREFIX  base: <http://www.semanticweb.org/ontologies/2015/hgbenchmark#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  ( COUNT(*) AS ?count)
WHERE
  { ?station1 rdf:type base:Station
    FILTER ( ?station1 = base:_ID ) .
    ?station1 base:neighbor ?station2 .
    ?station2 base:neighbor ?station3
    FILTER (?station1 != ?station3) .
    ?station3 base:neighbor ?station4
    FILTER (?station1 != ?station4 && ?station2 != ?station4) .
```

```
}
```

4.4.4 Tools

We concentrate on RDF tools in our work which are listed in Table 4.1. The last two columns illustrate whether the execution of our defined queries is feasible in the particular tool or not. As the table suggests, the first query cannot be evaluated in 4store, since it does not support the usage of property paths.

Name	Programming Language	Version	Query 1	Query 2
AllegroGraph [2]	C, C++	5.0.0	•	•
Blazegraph [5]	Java	1.5.2	•	•
4store [1]	C	1.1.5		•
Jena TDB [3]	Java	2.13.0	•	•
Sesame [16]	Java	2.7.9	•	•

Table 4.1. The implemented tools in Train Benchmark

4.4.5 Regression Analysis

MARS

Chapter 5

Evaluation

(size/evaluation time plot?) sample results per tools regression -> residuals -> series mars metrics calculation

5.1 Sample

5.1.1 Framework Configuration

5.1.2 Evaluated Queries

5.2 Benchmarking Environment

5.3 How to Read the Charts

5.4 Model Analysis

5.5 Performance Analysis

5.5.1 Multiple Regression Analysis

5.5.2 MARS

5.6 Conclusions

Chapter 6

Summary

Acknowledgements

Thanks!

List of Figures

List of Tables

Bibliography

[1] 4store. http://4store.org/.

[2] AllegroGraph. http://franz.com/agraph/allegrograph/.

[3] Apache Jena TDB. https://jena.apache.org/documentation/tdb/index.html.

[4] Berlin SPARQL Benchmark (BSBM) Specification V3.1. http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/.

[5] Blazegraph. https://www.blazegraph.com/product/.

[6] Covariance. http://mathworld.wolfram.com/Covariance.html.

[7] DBpedia. http://wiki.dbpedia.org/.

[8] Digital Bibliography and Library Project. http://dblp.uni-trier.de/db/.

[9] Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/.

[10] A guide on probability distributions. http://dutangc.free.fr/pub/prob/probdistr-main.pdf.

[11] Heavy tail distribution. https://reference.wolfram.com/language/guide/HeavyTailDistributions.html.

[12] MONDO. http://www.mondo-project.org/.

[13] Network Rail. http://www.networkrail.co.uk/about-us/.

[14] Population and Sample Definition. http://www.statistics.com/glossary&term_id=812.

[15] Populations and Samples. http://stattrek.com/sampling/populations-and-samples.aspx?Tutorial=AP.

[16] Sesame. http://rdf4j.org/.

[17] SPARQL Property Paths. http://www.w3.org/TR/sparql11-property-paths/.

[18] The GraphML File Format. http://graphml.graphdrawing.org/.

[19] The TTC 2015 Train Benchmark Case for Incremental Model Validation. https://www.sharelatex.com/github/repos/FTSRG/trainbenchmark-ttc-paper/.

[20] Train Schedules Data. http://nrodwiki.rockshore.net/index.php/SCHEDULE.

[21] A. L. Barabási, Z. N. Oltvai. Understanding the cell's functional organization nature genetics. *Network Biology*, 5:101–114, 2004.

[22] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[23] Alireza Bigdeli, Ali Tizghadam, and Alberto Leon-Garcia. Comparison of network criticality, algebraic connectivity, and other graph metrics. In *Proceedings of the 1st Annual Workshop on Simplifying Complex Network for Practitioners*, SIMPLEX '09, pages 4:1–4:6, New York, NY, USA, 2009. ACM.

[24] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[25] George C. Runger Douglas C. Montgomery. *Applied Statistics and Probability for Engineers*.

[26] P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, page 20, 1960.

[27] L. A. Huberman, B. A. ; Adamic. Growth dynamics of the world-wide web. In *Nature*, volume 401, page 131, 1999.

[28] Benedek Izsó, Gábor Szárnyas, István Ráth, and Dániel Varró. MONDO-SAM: A Framework to Systematically Assess MDE Scalability. In *BigMDE 2014 2nd Workshop on Scalable Model Driven Engineering*, page 40. ACM, ACM, 2014.

[29] Benedek Izsó, Gábor Szárnyas, István Ráth, and Dániel Varró. Train benchmark technical report. https://www.sharelatex.com/github/repos/FTSRG/trainbenchmark-docs/builds/latest/output.pdf, 2014.

[30] A. Jamakovic and S. Uhlig. On the relationship between the algebraic connectivity and graph's robustness to node and link failures. In *Next Generation Internet Networks, 3rd EuroNGI Conference on*, pages 96–102, May 2007.

[31] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark: Performance assessment with real queries on real data. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pages 454–469, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64:026118, Jul 2001.

[33] A. L. Barabási R. Albert, H. Jeong. The diameter of the world wide web. In *Nature*, volume 401, pages 130–131, 1999.

[34] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Phys. Rev. E*, 67:026112, Feb 2003.

[35] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. Sp2bench: A sparql performance benchmark. In Roberto de Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Semantic Web Information Management*, pages 371–393. Springer Berlin Heidelberg, 2010.

[36] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, June 1998.