Towards Precise Metrics for Predicting Graph Query Performance

Benedek Izsó, Zoltán Szatmári, Gábor Bergmann, Ákos Horváth, István Ráth
Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
{izso,szatmari,bergmann,ahorvath,rath}@mit.bme.hu

***Abstract*—Queries are the foundations of data intensive applications. In model-driven software engineering (MDSE), model queries are core technologies of tools and transformations. As software models are rapidly increasing in size and complexity, most MDSE tools frequently exhibit scalability issues that decrease developer productivity and increase costs. As a result, choosing the right model representation and query evaluation approach is a significant challenge for tool engineers. In the current paper, we aim to provide a benchmarking framework for the systematic investigation of query evaluation performance. More specifically, we experimentally evaluate (existing and novel) query and instance model metrics to highlight which provide sufficient performance estimates for different MDSE scenarios in various model query tools. For that purpose, we also present a comparative benchmark, which is designed to differentiate model representation and graph query evaluation approaches according to their performance when using large models and complex queries.**

***Index Terms*—Performance benchmark, Model queries, Model metrics, Query metrics**

I. Introduction

Nowadays, model-driven software engineering (MDSE) plays an important role in the development processes of critical embedded systems. Advanced modeling tools provide support for a wide range of development tasks such as requirements and traceability management, system modeling, early design validation, automated code generation, model-based testing and other validation and verification tasks. With the dramatic increase in complexity that is also affecting critical embedded systems in recent years, modeling toolchains are facing scalability challenges as the size of design models constantly increases, and automated tool features become more sophisticated.

A key factor in the scalability of MDE toolchains is the performance of model representation [1], which is, in turn, determined by the characteristics of persistence, query evaluation and model manipulation operations. Traditionally, modeling tools built on state-of-the-art frameworks such as the Eclipse Modeling Framework (EMF [2]) have relied on an in-memory object model backed by an XML serialization. More recently, *model repositories* (such as CDO [3] or Morsa [4]) have emerged that aim to tackle scalability issues by making use of advances in object persistence technology. As the majority of model-based tools uses a graph-oriented data model, recent results of the NoSQL and Linked Data movement [5]–[7] are straightforward candidates for adaptation to MDE purposes.

Model queries support several essential scenarios including model validation, model transformations, model synchronization, view maintenance and model execution. As a consequence, many scalability issues can be addressed by improving query performance. This led to the development of several model indexing and query evaluation engines (such as Eclipse OCL [8], EMF Query [9], complementary approaches that translate model queries into lower level queries that can be executed on the (relational) back-end [10]). There are also several approaches (such as EMF-IncQuery [11] and the Impact Analyzer of Eclipse OCL [8]) to support the *incremental evaluation* of model queries, which reduces query response time by limiting the impact of model modifications to query result calculation.

For tool engineers, *benchmarks* may provide guidance on picking the right technology for building a new tool architecture to fulfill increasing scalability requirements. Due to their central role in data-intensive applications, the performance of persistence technologies has been evaluated by many benchmarks [12], [13] that focus on throughput and response time measurements, and investigate scalability in terms of the size of the data set and the number of transactions. In addition to these traditional scalability aspects, the semantic web community has investigated the scalability of semantic graph databases (RDF triple stores). These benchmarks rely on graph queries over a structurally richer data set and also investigate the effects of advanced semantic technologies such as inference. Up to now, the most complex benchmarking workloads have been investigated by the academic and industrial MDE tool building community in transformation tool contests [14], which feature synthetic model transformation case studies inspired by real-world applications.

Despite all these efforts, *making a well-founded technological choice* based on existing benchmarking results *remains a tough challenge*. First, MDE tools have very specific *workloads* (both in terms of model structure and transaction complexity) *that are different* in key aspects compared to traditional RDBMS and newer graph persistence benchmarks. MDE tools rely on much more complex queries and their performance is dominated by response time and re-evaluation time rather than throughput. Additionally, RDBMS and semantic technologies have key conceptual differences that require mapping layers which might have adverse and unpredictable effects on real life performance. The generalizability of bench-

mark results is further limited by the *scarcity of relevant metrics* that could be used to assess an engineering problem and predict which technology would be best suited. Existing metrics emphasize a single aspect of the problem (most typically model size), while internal metrics (used by e.g. optimizing query evaluation engines or pattern matchers inside GT tools, for estimating query evaluation difficulty) are either not documented well or not accessible in a reusable way.

In this paper, we aim to address these challenges by *assessing existing metrics, along with newly proposed ones*. These metrics take instance model characteristics, static query characteristics and their combination into account. Based on our real-life experiences with tools and models, we *outline a benchmark* that uses model validation as its core scenario, thus focusing on model loading and model validation workloads. Guidelines are provided on the generation of instance models and queries, on which we evaluated the metrics and executed the benchmark using three, characteristically different graph query tools. In order to identify which metrics provide reliable performance prediction for a given workload and tool category, we *calculate the correlation with significance values* between execution times and metrics.

The rest of the paper is structured as follows. Sec. II overviews the most important concepts of modeling languages and model queries, and Sec. III discusses benchmarking and metrics related work. Sec. IV presents our analysis of existing benchmarks and proposes new metrics and benchmarks, with their evaluation presented in Sec. V. Sec. VI outlines directions for future work and concludes the paper.

II. Background

Graph based models can be used to abstract, and formally describe real world structures. Vertices of such a model can be grouped into classes (which are the *types* of these individuals), and similarly, relations can have labels to describe their types. A metamodel consists of such labels, and constraints that can restrict model combinations, or can be used for inferencing. In this section the EMF (Eclipse Modeling Framework) and RDF (Resource Description Framework) model description languages are described briefly. To process models of these languages, imperative traversal or declarative query evaluation tools can be used.

A. Model Representation Technologies and the Example Domain

1) Modeling Languages: The *Eclipse Modeling Framework (EMF)* [2] uses the *Ecore* language to describe domain-specific models and metamodels. Such models can be edited by framework provided graphical editors, which are similar to UML class diagram editors. Instance model elements are EObjects in Ecore, which have exactly one type, described by an EClass. *EReference* (or associations) connect EClasses, and *EAttribute* of EClasses point to primitive datatypes (like EString, EInt). EReferences and EAttributes can be single-valued or multi-valued and they can be ordered or unordered. EReferences may additionally imply containment, which is

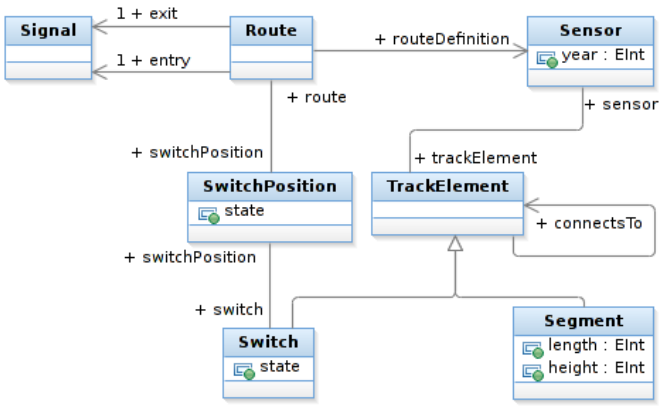


Fig. 1: Train metamodel described in EMF

mainly used during model serialization. Inheritance may be defined between classes, which means that the inherited class has all the properties its parent has, and its instances are also instances of the ancestor class, but it may further define some extra features. Note, that inheritance between relations is not supported by the language. The language supports two level metamodeling, where the metamodel is stored separately from instance models. An example metamodel is depicted in Fig. 1.

The *Resource Description Framework (RDF)* [15] is inspired by the Semantic Web, and describes graph models with {subject, predicate, object} triples. The subject and object are vertices, while the predicate is the type of the relation from subject to object. One atom of a triple statement can be an IRI (Internationalized Resource Identifier), while objects can be IRIs or XML Schema defined literals for describing values (like string or int). In contrast to EMF, RDF supports multi-level and multi-domain metamodeling. Other constraints can be expressed, like subsumption between classes or references, domain and range restrictions of relations. EMF supports inverse relations and cardinality restrictions by default, while in RDF these can be expressed only by using the OWL extension. In the measurements of the paper, for the RDF tools inferencing was not enabled and explicit metamodel was not constructed, so the metamodel of RDF documents are consist only of the types (labels) of instance model elements.

2) Example from the Railway Domain: The method described in the paper is domain independent, however for better understanding, an example from the railway domain is used throughout the paper which is depicted on Fig. 1. A train *route* can be defined by a set of *sensors*. Sensors are associated with *track elements*, which can be a track *segment* or a *switch*. A route can have associated *switch positions*, which describe the required state of a switch belonging to the route. Different route definitions can specify different states for a specific switch. Segments have lengths and heights, while sensors have production year integer properties.

3) Model Storage Backends: For EMF, the Eclipse provided in-memory backend was used. It can de-serialize models from an XML-based syntax, and perform basic inference (subsumption, inverse relations), as well as some basic validations

(like cardinality check). The Java based API provides model manipulation and traversal functions.

Sesame [7] is a standard API for handling RDF documents with implementation bundled into the distribution. RDF models can be read from multiple file formats (including RDF/XML) into various repositories, like to the in-memory MemoryStore (that we used), or to the disk based B-Tree indexed NativeStore. For reasoning and consistency checking a forward chaining inferrer can be used which implements inference rules described in the RDF standard [15].

B. Model Query Technologies and Languages

In order to process a model, a well defined part of it should be selected. As a basic solution, a Java based *imperative program* can traverse the model, and gather element tuples into a result set. Such a code usually collects elements of some types, and iterates over some of their references. This is similar to local search based approaches, however careful Java coding is needed, usually contains no special optimizations, or query plan construction, and highly depends on the actual structure (i.e.: inverse relations).

Graph patterns (GP) are used to refer to subgraphs of an instance model in a declarative way. A basic graph pattern is built up from structural constraints, prescribing the type and connection between instances (denoted by variables). The absence of an instance model combination can be described using *negative application condition*, while basic attribute checks filter results based on the values of basic data objects. Two GP evaluation engines are EMF-INCQUERY and Sesame. Patterns of EMF-INCQUERY can be evaluated on EMF models incrementally, and the language supports extensions, like pattern calls, recursive patterns and match counting. Sesame is a representative of many tools that evaluate queries over RDF that are formulated as SPARQL [16] graph patterns, which support aggregate function (like *AVG*, *SUM*) and query aggregates (like *GROUP BY* and *HAVING*).

III. RELATED WORK

Benchmarks have been proposed earlier, mainly to track improvements of a query engine, or to compare tool performance for a given use case. However, most benchmarks are only useful for predicting (relative) performance of tools depending on model size; there is rarely enough data to consider other signals such as query or model structure. Despite the abundance of benchmarks, it is still difficult to choose the best tool for a given purpose, due to the absence of common metrics.

A. RDF Benchmarks

SP²Bench [13] is a SPARQL benchmark that measures only query throughput. The goal of this benchmark is to measure query evaluating performance of different tools for a single set of SPARQL queries that contain most language elements. The artificially generated data is based on the real world DBLP bibliography; this way instance models of different sizes reflect the structure and complexity of the original real world dataset. However, other model element distributions or queries were not considered, and the complexity of queries were not analyzed.

The *Berlin SPARQL Benchmark (BSBM)* [12] measures SPARQL query evaluation throughput for an e-commerce case study modeled in RDF. The benchmark uses a single dataset, but recognizes several use cases with their own query mix. The dataset scales in model size (10M-150B), but does not vary in structure.

SPLODGE [17] is an approach, where SPARQL queries were generated systematically, based on metrics for a predefined dataset. The method supports distributed SPARQL queries (via the *SERVICE* keyword), however the implementation scaled only up to three steps of navigation, due to the resource consumption of the generator. The paper did not mention instance model complexity, and only the adequacy of the approach was demonstrated with the RDF3X engine, the effect of queries with different metrics combinations to different engines was not tested.

B. Model Transformation and Graph Transformation Benchmarks

There are numerous graph transformation benchmarks that do not focus specifically on query performance. However [18] aims to design and evaluate graph transformation benchmark cases corresponding to three usage patterns for the purpose of measuring the performance of incremental approaches on different model sizes and workloads. These scenarios are conceptual continuations of the comprehensive graph transformation benchmark library proposed earlier in [19] (described more extensively in [20]), which gave an overview on typical application scenarios of graph transformation together with their characteristic features. [21] suggested some improvements to the benchmarks described in [19] and reported measurement results for many graph transformation tools.

A similar approach to graph transformation benchmarking was used for the AGTIVE Tool Contest [22], including a simulation problem for the Ludo table game. Later, the GraBaTs tool contest [23] introduced an AntWorld case study [24], and the community continued to hold tool contests in the TTC [14] series.

As model validation is an important use case of incremental model queries, several model query and/or validation tools have been measured in incremental constraint validation benchmarks [11], [25], [26].

IV. BENCHMARKING AND METRICS

A. Overview

The aim of our metrics and benchmarking scenarios is to give a precise mechanism for identifying key-factors in selecting between different query evaluation technologies.

The presented metrics for our evaluation (see in Sec. IV-B) were constructed based on a set of already and widely used metrics, extended with two more specific ones that aims to give a gross upper bound on the cost of query evaluation.

For the benchmark scenario we opted for a simple execution schema (see in Figure 2) that represents a batch-validation scenario, where the underlying model is checked against a set of predefined queries in batch execution, while the execution time and memory consumption was measured. Using this scenario, the scalability and sensitivity of model queries and tools can be evaluated according to the defined metrics.

1) Read Phase: In the first phase the previously generated instance model is loaded from hard drive to memory. This includes parsing of the input, as well as initializing data structures (cache)) of the tool. The latter can consume minimal time for a tool that performs only local search, but for incremental tools indexes or in-memory caches are initialized.

2) Check Phase: In the second, check phase the instance model is queried. This can be as simple as reading the results from cache, or the model can be traversed based on some index. Theoretically cache or index building can be deferred from the read phase to the check phase, but it depends on the actual tool implementation. To the end of this phase, results must be available in a list.

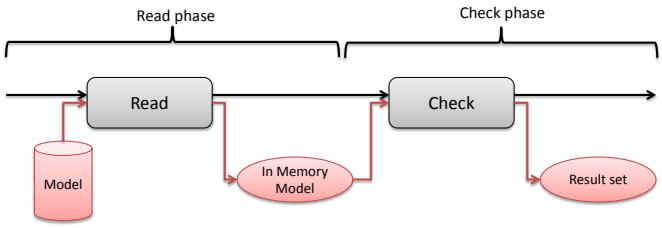


Fig. 2: Benchmark Scenario Overview

This benchmark was evaluated on three characteristically different query technologies:

1) An imperative *local search-based* approach implemented completely in Java, operating on EMF models
2) A declarative, *incremental* approach based on the concepts of Rete nets as provided by the EMF-INCQUERY framework, operating on EMF models.
3) A declarative, *black-box* execution engine as implemented in the Sesame framework based on the SPARQL query specification language, operating on RDF models.

Finally, we perform statistical correlation analysis against the data set generated from the execution of our benchmarks and the evaluation of our metrics.

B. Metrics

Our investigation relies on a selection of metrics that quantitatively describe queries tasks, independently of the actual strategy or technological solution that provides the query results. Broadly speaking, such a querying task consist of (a) an instance model, (b) a query specification that defines what results should be yielded, (c) a runtime context in which the queries are evaluated, such as the frequency of individual query evaluations and model manipulation inbetween.

The metrics discussed in the following, tipify one or more of these factors generally, without characterizing a unique property of a specific graph/query description language or environment. Most of these metrics have previously been defined by other sources, while others are newly proposed in this paper.

1) Metrics for Instance Model Only: Clearly, properties of the instance model may have a direct effect on query performance, e.g. querying larger models may consume more resources.

A first model metric is model size, which can be defined either as the number of objects (metric `countNodes`), the number of references (edges) between objects (`countEdges`), the number of attribute value assignments (not used in the paper); or some combination of these three, such as their sum (`countTriples`), which is basically the total number of model elements / RDF triples. This is complemented by the number of different classes the objects in the model belong to (`countTypes`), and the instance count distribution of the classes. Additional important model metrics characterize the distribution of the out-degrees and in-degrees (the number of edges incident on an object), particularly the maximum and average degrees (`maxInDegree`, `maxOutDegree`, `avgInDegree` and `avgOutDegree`).

The metrics discussed above have been defined e.g. in [27], along with other metrics such as the relative frequencies of the edge label sequences of directed paths of length 2 or 3.

2) Metrics for Query Specification Only: The query specification is a decisive factor of performance as well, as complex queries may be costly to evaluate. Such query metrics can be defined separately, in several query formalisms. Due to the close analogies between graph patterns and SPARQL queries, we can consider these metrics applying to graph pattern-like queries in general. This allows us to formulate and calculate metrics expressed on the graph patterns interpreted by EMF-INCQUERY, and characterize the complexity of the equivalent SPARQL query with the same metric value.

As superficial metrics for graph pattern complexity, we propose the number of variables (`numVariables`) and the number of parameter variables (`numParameters`); the number of pattern edge constraints (`numEdgeConstraints`) and the number of attribute check constraints (`numAttrChecks`); finally the maximum depth of nesting NACs (`nestedNacDepth`). Some of these are similar/equivalent to SPARQL query metrics defined in [17]. Other metrics proposed by [17] are mostly aimed at measuring special properties relevant to certain implementation strategies (see also paragraph IV-B5b).

3) Metrics for Combination of Query and Instance Model: The following two metrics (defined previously in literature) characterize the query and the instance model together.

The most trivial such metric is the cardinality of the query results (metric `countMatches`); intuitively, a query with a larger result set typically takes longer to evaluate on the same model, while a single query is typically more expensive to evaluate on models where it has more matches. The metric `selectivity` is proposed by [17], is the ratio of the number of results to the number of model elements (i.e. $countMatches/countTriples$).

4) New Metrics for Assessing Query Evaluation Difficulty: We propose two more metrics that take query and instance model characteristics into account. Our aim is to provide a gross upper bound on the cost of query evaluation. We consider all *enumerable* constraints in the query, for which it is possible to enumerate all tuples of variables satisfying it; thus edge constraints and pattern composition are enumerable, while NACs and attribute checks in general are not. At any given state of evaluation, a hypothetical search-based query engine has either already identified a single occurrence of an enumerable constraint c (e.g. a single instance of an edge type for the corresponding edge constraint), or not; there are therefore $|c|+1$ possible cases for c, where $|c|$ is the number of different ways that c can be satisfied in the model. This gives $\prod_c 1+|c|$ as the overestimate of the search space of the query evaluator. To make this astronomical figure manageable, we propose the absolute difficulty metric (`absDifficulty`) as the logarithm of the search space size, i.e. $ln \prod_c (1+|c|) = \sum_c ln(1+|c|)$.

The result size is a lower bound of query evaluation cost, since query evaluation takes at least as much time or memory as the number of results. It is therefore expected that queries with a high number of matches also score high on the absolute difficulty metric. To compensate for this, the relative difficulty metric (`relDifficulty`) is defined as $ln \frac{\prod_c (1+|c|)}{1+countMatches} = \sum_c ln(1+|c|) - ln(1+countMatches)$, expressing the logarithm of the "challenge" the query poses - this is how much worse a query engine can do than the lower bound. If the relative metric is a low figure, than the cost of query evaluation will not be much worse than the optimum, regardless of the query evaluation strategy. It can be easily shown that if a part of a graph pattern is extracted as a helper pattern that is used via pattern composition, then the sum of the relative difficulties of the two resulting patterns will be the same as the relative difficulty of the original pattern. This suggests that this metric should be treated as additive over dependent queries, and also that it is worth extracting common parts of multiple patterns into reusable helper patterns.

5) Other Factors:

a) Metrics Involving a Runtime Context: Compared to the instance model or the query specification, the role of the runtime context is less evident. Nevertheless, the performance of query strategies that rely on caching or indexing (such as the incremental evaluator of EMF-INCQUERY) may depend on such factors as well. If there are very frequent queries on a rarely changing model, caching the results pays off greatly. The more model modification is carried out between each query invocation, the more diminished these advantages become. In the extreme case, if queries are only ever evaluated once, then strategies that construct and incrementally maintain caches are obviously not worth applying.

Comparing a range of such workloads is beyond the scope of the current paper. Instead, we distilled performance measurement down to three indicators (read / check times and memory, see Sec. IV-A), so that the trade-offs of incremental strategies can be observed.

b) Metrics for Query Evaluation Strategies: In this paper, metrics are used for comparing query tasks, regardless of the evaluation strategy or technology applied as a solution. Therefore metrics that are specific to query evaluation strategies, especially metrics aimed at comparing evaluation plans, are not useful for the purposes of the current investigation.

Nevertheless, we briefly mention a few such metrics found in literature. The SST weight [28] is a cost estimate of search-based query evaluation plans for graph patterns, which can be used by a query optimizer to select the best alternative. Analogously, a cost estimate of OWL/SPARQL query plans is presented in [29]. For optimized parallelization of graph pattern matching on large distributed graphs, query variables (graph pattern nodes) are ranked by [30] according to a so-called f-value. Join operations in SPARQL queries are classified as star-shaped and path-shaped and counted in [17].

C. Benchmark Models

Models are characterized by their size and structure. Here size refers to the cardinalities of node and edge types. The fact that increasing model size tends to increase the cost of queries is intuitively self evident (and also empirically confirmed e.g. by our previous experiments [11], [18]). Handling large models in real life is a great chellenge, but model structure (that determines which nodes are connected to each other by which edges) must also be take into account, which here means edge distribution of nodes. Different edge distributions also present in real-world networks: the internet or protein-protein interaction networks show *scale-free* characteristics [31], while in other areas self-healing algorithms for *binomial computer networks* are studied [32]. Average degree can impact performance greatly, which is a typical property of different model kinds. For example, software models have usually nodes with *low degree*, while social models are usually *dense graphs*.

We have conducted the experiments on synthetic models, generted automatically with our model builder, belonging to three *model families* (without comparing them directly to real world ones, leaving it as a future work). All generated models within a family have the same approximate size; though there is some random variation due to the generation process (see later), with low standard deviation (e.g. measured as 0.3% in family A). Family A models are relatively dense graphs (~26 edges/nodes, i.e. metric `avgOutDegree`) with 1.8 million total model elements (metric `numTriples`); family B models are equally dense, but are scaled back to only 113 thousand elements; finally family C models have almost 1.3 million model elements that form a relatively sparse graph ($8.4-8.7$ edges/nodes).

Each model of a family has the same (expected) number of instances for any given type. However, these models of the same size still differ in their internal *structure*. Given the cardinalities of each type, our generator first created the instance sets of node types, along with generating attribute values according to an associated distribution. Then for each edge type, the generator created edge instances (with the given expected cardinality) between instances of the source

type and instances of the target type. The structure of the graph is induced by the method of choosing which source node and which target node to connect. We have applied the following four methods, each taking as input the set S of source node candidates, the set T of target node candidates, and the expected cardinality e of the edge type.

Binomial case. Inspired by the well-known Erdős-Rényi model of random graphs [33], the first approach is to take each pair of source and target nodes, and draw an edge between them with a given probability p. This makes the expected cardinality of edges $e = p \times |S| \times |T|$, thus p is chosen as $\frac{e}{|S| \times |T|}$. The degrees of nodes will be binomially distributed, e.g. out-degrees with parameters $|T|$ and p.

Hypergeometric case. While the previous solution ended up with a random number of edges (with expected value e), this slightly different approach will generate exactly e edges, by taking each pair of source and target nodes, and randomly selecting e from them into the graph. The degrees will be hypergeometrically distributed, e.g. out-degrees with parameters $|S| \times |T|$, $|T|$ and e.

Regular case. In software engineering models, one often finds for a given edge type that out-degrees of all nodes of the source type are roughly equal, and the same is true for in-degrees. This motivated a method that tries to uniformly (but randomly) divide e edges between the source nodes, so that the difference between any two out-degrees is at most 1; while also dividing the same edges between the target nodes, with a similar restriction on in-degrees.

Scale-free case. It has been observed in many different disciplines that degree distributions of certain large graphs follow a power law, especially growing / evolving graphs with the *preferential attachment* property (a new edge is more likely to connect to a node which already has a higher degree). We have used a variant of the preferential attachment bipartite graph generator algorithm of [34] to generate the connections from source nodes to target nodes.

The four generation methods induced significantly different degree distributions. This and other differences are shown in Table I.

One-to-many relationships were treated in a special way to meet the multiplicity restriction. In particular, a single top-level container element (not depicted in the metamodel figure, neither involved in any queries) was used to contain all elements; it therefore has an outgoing containment edge for every other object, thereby "polluting" the `maxOutDegree` metric.

D. Benchmark Queries

Based on the previously defined metrics for query specification and based on the metamodel, six series of model query specifications were systematically constructed. Each query series includes four to seven model queries that aim to be different in only one of the defined model query metrics. Executing a query series on the same model and tool results in a data series that shows how the tool is scalable according to the represented model metric.

1) `Locals` Query Series for the `numVariables` Metric: Five queries were defined, where each one includes the same number of edge constraints, but the number of local variables increases.

It can be realized using the `Segment` type and `connectsTo` reference, so it means, that only one node type and reference type is used in these patterns and the focus is on the structure of the patterns. The simple graph based visualisation of these pattern structures is shown in Fig. 3, where the node drawn with empty circle represents the single pattern parameter.

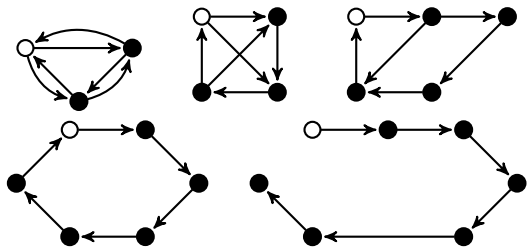


Fig. 3: `Locals` patterns

2) `Refs` Query Series for the `numEdgeConstraints` Metric: The `Refs` query series is also constructed based on the `Segment` type and `connectsTo` reference.

Here, the number of edge constraints increases along the series, but the number of local variables is constant in all of the generated four queries. The visualisation of these pattern structures is shown in Fig. 4.

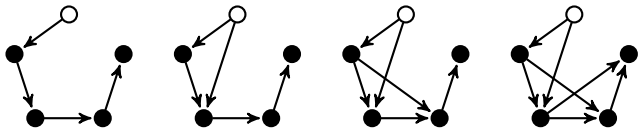


Fig. 4: `Refs` patterns

3) `Params` and `ParamCircle` Query Series for the `numParameters` Metric: Two series of queries were constructed for this metric, because of performance reasons. The `Params` query series is the more complex and some tools exceed the time limit in the benchmark. The `ParamsCircle` query series is a simplification of the `Params` series.

The goal of this constructed query series is to create patterns with the same body, but with an increasing number of parameters. The first of these queries (returning one parameter) is shown in Fig. 5, where the parameter is the blue object. Other queries use the same body, but add sen1, then sen2, then other variables to the parameter list.

4) `Checks` Query Series for the `numAttrChecks` Metric: Each `Checks` query use the same pattern body described by the pattern schema in Fig. 6, but each one is extended with an increasing number of attribute check constraints. These check constraints filter results based on the *year, height* and *length* value of the segments *seg1* and *seg2*, resulting in seven queries.

5) `Negs` Query Series for the `nestedNacDepth` Metric: `Negs` queries present increasing number of nested *neg* constraints.

TABLE I: Values of model metrics on the generated instance models

Model Family	Model structure	countNodes	countEdges	countTriples	countTypes	avgOutDegree	avgInDegree	maxOutDegree	maxInDegree
A	Regular	63289	1646386	1811752	7	26.01	26.01	63288	44
A	Binomial	63289	1649179	1814545	7	26.06	26.06	63288	69
A	HyperGeo	63289	1646386	1811752	7	26.01	26.01	63288	74
A	Scalefree	63289	1660033	1825399	7	26.23	26.23	63288	10390
B	Regular	3954	102839	113170	7	26.01	26.01	3953	44
B	Binomial	3954	102984	113315	7	26.05	26.05	3953	64
B	HyperGeo	3954	102839	113170	7	26.01	26.01	3953	69
B	Scalefree	3954	96029	106360	7	24.29	24.29	3953	918
C	Regular	120001	1040000	1280001	7	8.67	8.67	120000	13
C	Binomial	120001	1041323	1281324	7	8.68	8.68	120000	30
C	HyperGeo	120001	1040000	1280001	7	8.67	8.67	120000	29
C	Scalefree	120001	1012858	1252859	7	8.44	8.44	120000	8929

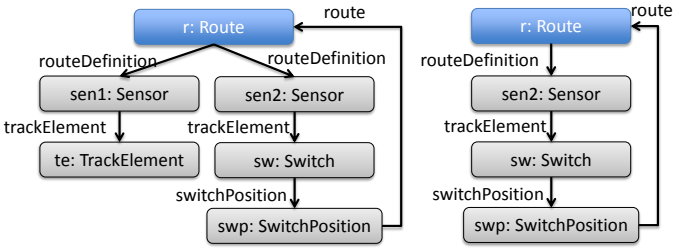


Fig. 5: `Params` and `ParamsCircle` pattern schemas (first step)

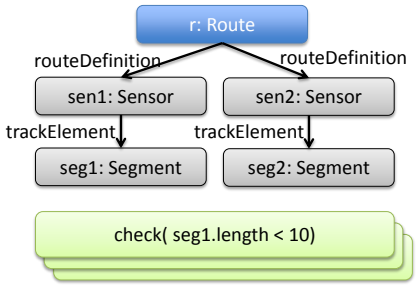


Fig. 6: `Checks` pattern schema

These queries are defined based on the following schema: at the bottom there is a pattern checking for segments with length less than ten. Next, for each query a new segment is matched, and the previous pattern is encapsulated in a negative pattern call. $i = 5$ queries are defined in the benchmark, described by the schema in Fig. 7.

After the construction of the query series, we evaluated the query only metrics on them. Table II shows the results of the evaluation: each cell contains the value or range of values that we got on each query series. This table confirms that there are query series for every metric (shown in blue), and each query series differ in one or more metrics.

E. Implementation Details

These query series are first defined in graph patterns and then formalized using a query language suited for the given tool. The presented query series were implemented using each model query technology mentioned in Sec IV-A. In Fig. 8 the sample implementation of the `locals_3` query is depicted.

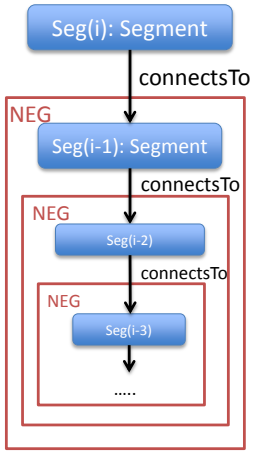


Fig. 7: `Negs` pattern schema

In the IncQuery Pattern Language [35] (Fig. 8a), object constraints and reference constraints are used to describe the structure, and individuals matching the `Seg1` variable are returned.

Using the SPARQL [16] notation (Fig. 8b), triples describe the same structural constraint, and the semantically equivalent query returns distinct matches of the `xSeg1` variable.

The query function was also coded in Java (illustrated in Fig. 8c). The model is traversed by embedded iterations and for every `Segment` the connections are checked. The implementation does not contain any search plan specific optimization (i.e. the embedding order of `for` cycles is ad-hoc), but it cuts unnecessary search branches at the earliest possibility. This coding style represents an experienced programmer, who writes good quality source code. This way, such Java implementation could be used as a baseline in the future, to compare multiple tools qualitatively.

V. Experimental Evaluation

A. Measurement Setup

For the implementation details, source codes and raw results, see the benchmark website[1]. In this section we describe the runtime environment, and highlight some design decisions.

[1] http://incquery.net/publications/benchmarkmetrics

TABLE II: Query-only metrics

Query series	numParameters	numVariables	numEdgeConstraints	numAttrChecks	nestedNacDepth
Param	1 - 5	8	8	0	0
ParamCircle	1 - 5	6	6	0	0
Locals	1	3 - 7	6	0	0
Refs	1	5	4 - 7	0	0
Checks	1	5 - 11	4 - 10	0 - 6	0
Neg	2 - 6	3 - 11	1 - 5	1	0 - 10

```
pattern locals_3(Seg1) =
{
  Segment(Seg1);
  Segment(Seg2);
  Segment(Seg3);
  Segment.connectsTo(Seg1,Seg2);
  Segment.connectsTo(Seg2,Seg3);
  Segment.connectsTo(Seg3,Seg1);
  Segment.connectsTo(Seg2,Seg1);
  Segment.connectsTo(Seg3,Seg2);
  Segment.connectsTo(Seg1,Seg3);
}
```

(a) EMF-IncQuery graph pattern

```
SELECT distinct ?xSeg1
WHERE
{
  ?xSeg1 rdf:type base:Segment .
  ?xSeg2 rdf:type base:Segment .
  ?xSeg3 rdf:type base:Segment .
  ?xSeg1 base:connectsTo ?xSeg2 .
  ?xSeg2 base:connectsTo ?xSeg3 .
  ?xSeg3 base:connectsTo ?xSeg1 .
  ?xSeg2 base:connectsTo ?xSeg1 .
  ?xSeg3 base:connectsTo ?xSeg2 .
  ?xSeg1 base:connectsTo ?xSeg3 .
}
```

(b) SPARQL graph pattern

```
protected void locals3() {
  Collection<Segment> candidates1 = getSegments();
  Top:
  for (Segment s1 : candidates1) {
    Collection<Segment> candidates2 = getSegmentsConnectedTo(s1);
    for (Segment s2 : candidates2) {
      if (!s2.getConnectsTo().contains(s1))
        continue;
      Collection<Segment> candidates3 = getSegmentsConnectedTo(s2);
      for (Segment s3 : candidates3) {
        if (!s1.getConnectsTo().contains(s3))
          continue;
        if (!s3.getConnectsTo().contains(s2))
          continue;
        if (!s3.getConnectsTo().contains(s1))
          continue;
        foundInvalid(s1);
        continue Top;
      }
    }
  }
}
```

(c) Java code

Fig. 8: Pattern schemas for `Locals_3` query

The benchmark machine contains two quad core Intel Xeon L5420 (2.50GHz) CPU, 32 GBs of RAM and a SAS disk formatted to ext4 for storing the models. In order to alleviate disturbance of a running measurement and minimize noise in the results, a bare metal 64 bit Ubuntu 12.04 OS was installed with unnecessary services (like cron) turned off. OpenJDK JVM version 1.6.0_24 is used as the Java environment and Eclipse Juno Modeling 64 bit for Linux for development, and for the EMF-IncQuery and Java dependencies.

The performance measurements of a tool for a given query-model pair was independent from the others, i.e. for every tool only its codebase was loaded, and every scenario measurement (see Fig. 2) was run in a different JVM. Before the execution, OS file cache was cleared, and swap was disabled to avoid this kind of thrashing. Each test (including all phases) must be run within 15 minutes, otherwise it was killed.

To obtain faithful execution times, we implemented a benchmarking framework, which accounted for the time measurements with nanosec precision (that can have less accuracy!), and the clear separation of phases enforced by the defined interfaces. These model loading and querying functions were implemented using functionally equivalent calls of a given tool. See Table III for the model management format and query language of the benchmarked tools. The benchmarks were realized as Java applications.

TABLE III: Measured tools

Tool	Model	Query Language	Version
Java	EMF	Java	6.0
EMF-IncQuery	EMF	IQPL	0.7.0
Sesame	RDF	SPARQL	2.5.0

Before acquiring memory usage (free heap space) from the JVM, GC calls were triggered five times to sweep unfreed objects from the RAM. For a JVM, 25 GB heap limit was specified, but to compensate 64 bit pointers, OOPS (ordinary object pointers) compression was also turned on: (`-XX:MaxPermSize=256m -XX:+UseCompressedOops -Xmx25g`).

In the benchmark all cases were run ten times, and the results were dumped into files, then processed by a spreadsheet software. Finally they are analyzed and visualized using the R statistical framework.

B. Method of Analysis

Our long-term goal is providing a catalog of reliable benchmark data that, based on metrics of the model and the queries, will allow the engineer to extrapolate the expected performance characteristics of various query technologies, which in turn will support making an informed decision on the solution to apply. In scope of this paper, we attempt to provide a necessary prerequisite with a narrower focus of investigation: finding out which model and query metrics are useful for predicting the performance of the various tools, over a wide range of different queries and model structures.

A given metric can only be a useful predictor of a certain performance indicator of a certain tool if they are strongly correlated. Therefore our analysis investigated correlations of metrics and performance indicators. Neither previous experience nor current measurement data supported a linear relationship between these variables; therefore we opted to abandon the commonly used Pearson correlation coefficient that is associated with linear regression. We rely instead on Kendall's τ rank correlation coefficient, ranging from -1 to $+1$; without assuming a linear model, τ characterises the degree to which it can be said that larger values of the metric correspond to larger values of the performance indicator.

Correlation coefficients may lead to incorrect conclusions if the sample of models and queries is too small. Therefore, whenever measuring an absolute value of τ, we additionally conducted the associated statistical test (τ-test) to decide whether the detected correlation between the variables is statistically significant ($p < 0.001$). Note that any such statistical result is conditional to the uniform sampling of the selected queries and models.

A limitation of the approach is that two strongly correlated variables (e.g. `countEdges` and `countTriples`) may show up as equally good predictors, but in reality they can not be used as two independent signals for predicting performance (as most triples in our models are edges). The simple correlation-based analysis presented here is intended as preliminary feature selection; we intend to follow up with redundancy-reducing feature selection techniques such as mRMR [36] that can take into account which metrics convey independent information. Afterwards, predicting the best choice of technology based on the metric values is a problem of multivariate regression / classification, for which we plan to employ advanced machine learning techniques (e.g. ID3 [37] decision trees or MARS [38] regression model) in future work.

C. Results

For each tool performance indicator and each metric, we conducted Kendall's correlation test to see whether the data available is sufficient to form a statistically significant support of correlation between the performance indicator and the metric. For metrics that were found to correlate, the absolute value of Kendall's τ correlation coefficient is displayed on a spider chart specific to the performance indicator of the tool (see Figure 9); positive correlation values are displayed as red triangles, while negative ones as blue squares.

1) Evaluation: Consistently with previously published results, the data shows that model size is a strong predictor of both model loading time and memory consumption, regardless of the technology.

a) Tool-specific Observations: However, check times show a more diverse picture. The check times for the Java implementation (being a dominantly search-intensive approach) are additionally correlated with the query-on-model metrics as well, with the strongest correlation shown by the *absDifficulty* metric.

Interestingly, the best Sesame check time predictor turned out to be the number of pattern variables, and there is no significant correlation with any direct model metrics.

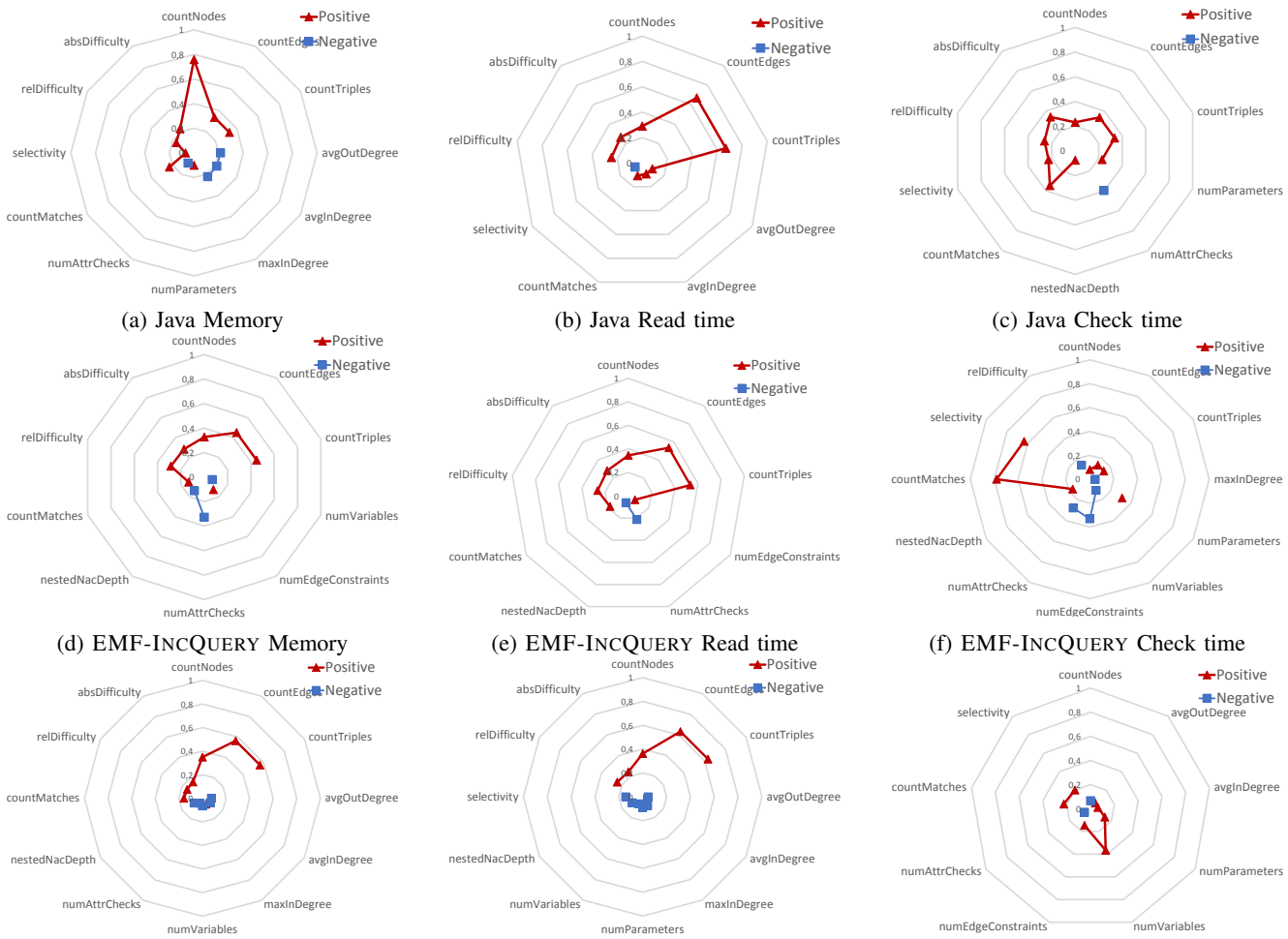
Check times in EMF-INCQUERY are very strongly correlated to the number of matches - which is to be expected of an incremental tool whose check phase consists of just enumerating the cached match set. As the incremental indexes are constructed during the load time, the model-on-query metrics become correlated with EMF-INCQUERY read time. It can also be highlighted that EMF-INCQUERY seems not to be sensitive towards the "difficulty" of the query (in any phase) or the model size (during the check phase) due to the very small correlations with corresponding metrics.

b) Metrics: Overall, it can be said that model-only metrics are useful in predicting the performance of model persistence operations. However, query-based and combined metrics (such as our proposed *abs-* and *relDifficulty*) are necessary to provide a more thorough picture. Note that since only statistically significant correlations are included, a low magnitude correlation does not necessarily mean a measurement error. It is possible that there is a true link between the two variables, but the τ value is lowered by other metrics that strongly influence the performance indicator.

2) Threats to Validity: Regarding the technological foundations and methodology of our measurements, the most important threats to validity stem from *time measurement uncertainty* and distortions due to transient effects such as *garbage collection* in the JVM and *thrashing* due to heap size exhaustion. Such effects were mitigated by using the most accurate Java time measurement method (`System.nanoTime`), allocating as much heap as possible, and using a timeout mechanism to identify and exclude cases affected by thrashing from the results. Additionally, it is also possible that there is sampling bias in our choice of models and metrics; we believe that this is sufficiently mitigated by our systematic choice of model generation strategies and the design principles of the queries. To improve the magnitude of correlation we increased the sample size by running the benchmarks ten times.

VI. CONCLUSION AND FUTURE WORK

In this paper, our aim was to develop a methodology whereby the performance of model queries can be systematically evaluated, and relevant performance predictor metrics can be identified. Based on an analysis of MDSE scalability issues and previously proposed model and query metrics, we devised a set of metrics that estimate the complexity of a modeling tool



(a) Java Memory (b) Java Read time (c) Java Check time

(d) EMF-IncQuery Memory (e) EMF-IncQuery Read time (f) EMF-IncQuery Check time

(g) Sesame Memory (h) Sesame Read time (i) Sesame Check time

Fig. 9: $|\tau|$ of correlating ($p < 0.001$) metrics for each performance indicator

workload based on combined characteristics of instance models and model queries. We designed a benchmark to effectively differentiate various model representation and query evaluation approaches with respect to their scalability. To verify our approach with real-life MDSE tools, we conducted an initial experimental evaluation and found that our methodology is useful for distinguishing key tool characteristics (such as batch vs. incremental evaluation). In addition to model size, we identified several additional metrics that correlate with query performance, which may be useful for the design and optimization of query-intensive modeling applications.

As a primary direction for future work, we plan to extend the benchmark into a comprehensive evaluation of real-life large models of varying size and a wide spectrum of tools (including traditional, object-oriented and graph database systems, as well as various MDSE technologies). Using these results decision trees could be built, the Goal Question Metric [39] method could be applied to improve software quality or more sophisticated statistical analysis could be performed, aiding domain engineers to choose the right tools and languages for their task. We will also extend the benchmark with a case specifically tuned for incremental re-evaluation, with metrics that take workload-specific model change characteristics as well as long-term query re-execution performance into account. Since the challenges of "big MDSE" are characteristically very similar to the issues of emerging graph database systems, we believe that our methodology can be generalized to graph search problems outside of the MDSE world. Our long term goal is to improve on existing modeling and graph search benchmarks in order to increase their relevance and applicability to real-life engineering tasks.

Acknowledgment

This work was partially supported by the CERTIMOT (ERC_HU-09-01-2010-0003) project, and the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4. A/-11-1-2012-0001 "National Excellence Program".

REFERENCES

[1] Dimitris Kolovos and Richard Paige and Fiona Polack, "The Grand Challenge of Scalability for Model Driven Engineering," in *Models in Software Engineering*, ser. Lecture Notes in Computer Science, Chaudron, Michel, Ed. Springer Berlin / Heidelberg, 2009, vol. 5421, pp. 48–53.

[2] The Eclipse Project, "Eclipse Modeling Framework," http://www.eclipse.org/emf/.

[3] ——, "The CDO Model Repository," http://eclipse.org/cdo/.

[4] J. Espinazo Pagan, J. Sanchez Cuadrado, and J. García Molina, "Morsa: A scalable approach for persisting and accessing large models," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, J. Whittle, T. Clark, and T. Kühne, Eds. Springer Berlin / Heidelberg, 2011, vol. 6981, pp. 77–92, 10.1007/978-3-642-24485-8_7. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24485-8_7

[5] "Neo Technology: Neo4j," http://neo4j.org/.

[6] "OpenLink Software: Open Virtuoso," http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/.

[7] "Sesame: RDF API and Query Engine," http://www.openrdf.org/.

[8] Eclipse Model Development Tools Project, "Eclispe OCL website," 2011, http://www.eclipse.org/modeling/mdt/?project=ocl.

[9] Eclipse Modeling Project, "EMF model query," http://www.eclipse.org/modeling/emf/?project=query.

[10] M. Scheidgen, A. Zubow, J. Fischer, and T. H. Kolbe, "Automated and transparent model fragmentation for persisting large models," in *Model Driven Engineering Languages and Systems*. Springer, 2012, pp. 102–118.

[11] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, "Incremental evaluation of model queries over EMF models," in *Model Driven Engineering Languages and Systems, 13th International Conference, MODELS'10*, Springer. Springer, 10/2010 2010, acceptance rate: 21%.

[12] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *International Journal On Semantic Web and Information Systems*, vol. 5, no. 2, 2009.

[13] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP2Bench: A SPARQL performance benchmark," in *Proc. of the 25th International Conference on Data Engineering*. Shanghai, China: IEEE, 2009, pp. 222–233.

[14] "Transformation tool contest," planet-sl.org/ttc2013/, 2013.

[15] World Wide Web Consortium, "Resource Description Framework (RDF)," http://www.w3.org/standards/techs/rdf/.

[16] ——, "SPARQL Query Language for RDF," http://www.w3.org/TR/rdf-sparql-query/.

[17] O. Görlitz, M. Thimm, and S. Staab, "SPLODGE: Systematic generation of SPARQL benchmark queries for Linked Open Data," in *The Semantic Web – ISWC 2012*, ser. Lecture Notes in Computer Science, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds. Springer Berlin Heidelberg, 2012, vol. 7649, pp. 116–132. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35176-1_8

[18] G. Bergmann, Á. Horváth, I. Ráth, and D. Varró, "A benchmark evaluation of incremental pattern matching in graph transformation," in *Proc. 4th International Conference on Graph Transformations, ICGT 2008*, ser. Lecture Notes in Computer Science, H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, Eds., vol. 5214, Springer. Springer, 2008, pp. 396–410, acceptance rate: 40%.

[19] G. Varró, A. Schürr, and D. Varró, "Benchmarking for graph transformation," in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 05)*. Dallas, Texas, USA: IEEE Press, September 2005, pp. 79–88.

[20] ——, "Benchmarking for graph transformation," Budapest University of Technology and Economics, Tech. Rep. TUB-TR-05-EE17, March 2005, http://www.cs.bme.hu/~gervarro/publication/TUB-TR-05-EE17.pdf.

[21] R. Geiß and M. Kroll, "On improvements of the Varro benchmark for graph transformation tools," Universität Karlsruhe, IPD Goos, Tech. Rep. 2007-7, 12 2007, iSSN 1432-7864. [Online]. Available: http://www.info.uni-karlsruhe.de/papers/TR_2007_7.pdf

[22] The AGTIVE Tool Contest, "official website," 2007, http://www.informatik.uni-marburg.de/~swt/agtive-contest.

[23] GraBaTs - Graph-Based Tools: The Contest, "official website," 2008, http://www.fots.ua.ac.be/events/grabats2008/.

[24] Albert Zündorf, "AntWorld benchmark specification, GraBaTs 2008," 2008, http://is.tm.tue.nl/staff/pvgorp/events/grabats2009/cases/grabats2008performancecase.pdf.

[25] A. Reder and A. Egyed, "Incremental consistency checking for complex design rules and larger model changes," in *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems*, ser. MODELS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 202–218. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33666-9_14

[26] J.-R. Falleri, X. Blanc, R. Bendraou, M. Aurélio, A. da Silva, and C. Teyton, "Incremental inconsistencies detection with low memory overhead," *Software: Practice and Experience*, vol. 43, 2013.

[27] J. Stárka, M. Svoboda, and I. Mlynkova, "Analyses of RDF triples in sample datasets," in *COLD*, 2012.

[28] G. Varró, D. Varró, and K. Friedl, "Adaptive graph pattern matching for model transformations using model-sensitive search plans," in *GraMot 2005, International Workshop on Graph and Model Transformations*, ser. ENTCS, G. Karsai and G. Taentzer, Eds., vol. 152, Elsevier. Elsevier, 2006, p. 191–205. [Online]. Available: http://www.inf.mit.bme.hu/FTSRG/Publications/varro/2005/gramot05_vvf.pdf

[29] I. Kollia and B. Glimm, "Cost based query ordering over OWL ontologies," in *The Semantic Web – ISWC 2012*, ser. Lecture Notes in Computer Science, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds. Springer Berlin Heidelberg, 2012, vol. 7649, pp. 231–246. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35176-1_15

[30] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 788–799, May 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2311906.2311907

[31] A.-L. Barabasi and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nat Rev Genet*, vol. 5, no. 2, pp. 101–113, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1038/nrg1272

[32] T. Angskun, G. Bosilca, and J. Dongarra, "Self-healing in binomial graph networks," in *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II*, ser. OTM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1032–1041. [Online]. Available: http://dl.acm.org/citation.cfm?id=1780453.1780490

[33] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[34] B. Vladimir and B. Ulrik, "Efficient generation of large random networks," *Physical Review E*, vol. 71, no. 3, p. 036113, 04 2005.

[35] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró, "A Graph Query Language for EMF models," in *Theory and Practice of Model Transformations, ICMT 2011*. Springer, 2011.

[36] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 8, pp. 1226–1238, 2005.

[37] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Online]. Available: http://dx.doi.org/10.1007/BF00116251

[38] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Statist.*, vol. 19, no. 1, pp. 1–141, 1991, with discussion and a rejoinder by the author. [Online]. Available: http://dx.doi.org/10.1214/aos/1176347963

[39] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Wiley, 1994.