Diefficiency Metrics: Measuring the Continuous Efficiency of Query Processing Approaches

Maribel Acosta[1], Maria-Esther Vidal[2,3], and York Sure-Vetter[1]

[1] Institute AIFB, Karlsruhe Institute of Technology
{maribel.acosta,york.sure-vetter}@kit.edu
[2] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
vidal@cs.uni-bonn.de
[3] Universidad Simón Bolívar

Abstract. During empirical evaluations of query processing techniques, metrics like execution time, time for the first answer, and throughput are usually reported. Albeit informative, these metrics are unable to quantify and evaluate the efficiency of a query engine over a certain time period – or diefficiency –, thus hampering the distinction of cutting-edge engines able to exhibit high-performance gradually. We tackle this issue and devise two experimental metrics named *dief@t* and *dief@k*, which allow for measuring the diefficiency during an elapsed time period t or while k answers are produced, respectively. The *dief@t* and *dief@k* measurement methods rely on the computation of the area under the curve of answer traces, and thus capturing the answer concentration over a time interval. We report experimental results of evaluating the behavior of a generic SPARQL query engine using both metrics. Observed results suggest that *dief@t* and *dief@k* are able to measure the performance of SPARQL query engines based on both the amount of answers produced by an engine and the time required to generate these answers.

1 Introduction

Reproducibility and replicability are two important issues to be addressed in the validation of experimental results. Testbeds and benchmarks are bedrocks towards achieving these issues, and the Semantic Web community has made important contributions in this direction, e.g., in the series of workshops on Evaluation of Ontology-based Tools (EON) [4] and the activities of the Ontology Alignment Evaluation Initiative (OAEI). Moreover, testbeds and benchmarks such as LUBM [9], Feasible [15], or WatDiv [3], have become building blocks for the evaluation of existing Semantic Web technologies.

Metrics provide measurement methods for reporting and analyzing experimental results, thus representing an important premise to allow for reproducibility and replicability of experimental studies. Particularly, in the area of the Semantic Web, metrics provide measurement methods for quantifying the behavior of Semantic Web technologies, e.g., introducing semantics-aware metrics and normalization for ontologies [19], measuring link discovery approaches [12], or the performance [7, 16] or scalability [6] of query engines over RDF datasets.

Specifically, to assess the performance of query processing techniques, metrics like execution time, time for the first answer, answer completeness, and throughput are usually reported. These metrics provide measurement methods to quantify the performance of a query processing technique at a given point in time, i.e., either when the first or all answers are produced. However, these metrics are unable to quantify and evaluate the efficiency of a query engine over a certain time period – or diefficiency[4] –. In consequence, continuous high-performance engines cannot be clearly distinguished from those which only exhibit high-performance at certain discrete points in time or that produce answers at a slower rate.

In this paper, we tackle the problem of measuring the continuous efficiency of query processing techniques and propose two experimental metrics named *dief@t* and *dief@k*. The proposed metrics capture the diefficiency during an elapsed time period t or while k answers are produced. Specifically, the *dief@t* and *dief@k* measurement methods rely on the computation of the area under the curve of answer traces to capture the concentration of answers over a time interval.

We evaluate the effectiveness of *dief@t* and *dief@k* on different configurations of a SPARQL query engine which produces results incrementally at different rates. The observed measurements capture complementary information about the continuous behavior of the studied engine. More importantly, the reported results allow for uncovering properties of the engine. For instance, these results provide valuable insights about the engine configuration that continuously produces more answers over time or at a slower rate. None of these properties could be either detected or explained if only traditional metrics would be measured.

In summary, the contributions of this work are as follows:

- Novel experimental metrics, *dief@t* and *dief@k*, that measure the continuous efficiency of query engines.
- Formal properties and proofs that demonstrate the theoretical soundness of the proposed metrics.
- An empirical evaluation that indicates that *dief@t* and *dief@k* allow for uncovering particularities in the performance of query processing approaches that could not be captured with metrics defined in the literature.

The remainder of this paper is structured as follows: Related work is presented in Section 2. We motivate our work in Section 3 by illustrating the processing of two SPARQL queries against the DBpedia dataset using a typical query engine. In Section 4 we formally introduce the two new experimental metrics *dief@t* and *dief@k* for measuring the performance or incremental query processing approaches. We evaluate our approach in Section 5 by conducting an empirical study where we evaluate the performance of different SPARQL engines with the proposed metrics. Finally, the paper concludes in Section 6.

[4] We propose the term *diefficiency* as the combination of the Greek prefix *di(a)-* (which means "through" or "across") and *efficiency*.

2 Related Work

The Database and Semantic Web communities have actively worked on the definition of metrics to provide computational methods to measure the behavior of knowledge and data management approaches. For example, ontologies can be measured during the whole ontology life cycle, and metrics as the one proposed by Vrandecic and Sure [19] allow for semantics-aware metrics which, e.g., can be used for ontology assessment and for tracking ontology evolution. Further, metrics for evaluating the quality of ontology matching and alignment techniques have been defined by benchmarking activities of the Ontology Alignment Evaluation Initiative (OAEI)[5] which resulted in [8].

Metrics also capture the behavior of knowledge and database systems in terms of different dimensions, e.g., efficiency or effectiveness. Table 1 summarizes metrics included in existing benchmarks [3, 5, 9, 11, 14–17, 20], and commonly used to evaluate one-time and continuous query processing tools [2, 10, 13, 18], i.e., query engines over persistent datasets or against volatile data streams.

LUBM [9] is an exemplar benchmark to evaluate OWL applications with different reasoning capabilities and storage mechanisms. LUBM includes both data and query generators, as well as precise measurement methods to evaluate efficiency and effectiveness of OWL systems. Effectiveness represents the quality of the answers produced by the evaluated query processing technique, and it is measured in terms of the metric of answer completeness and soundness. LUBM also proposes to measure data processing efficiency in terms of two metrics: load time and query response time. Further, a combined metric enables to quantify the behavior of a query processing engine as the harmonic mean of query response time, and answer completeness and soundness. The Berlin Benchmark [5] (BSBM) is a benchmark generator tailored to evaluate SPARQL query engines. In addition to data and query, BSBM makes available a test driver able to execute sequences of queries over the system under test (SUT), thus simulating concurrent access of multiple clients. Additionally, BSBM presents a set of metrics to measure the SUT efficiency. The proposed measurement methods allow for quantifying queries executed per hour or second, load time, query execution time, loading time, and overall runtime. In addition, time required for a query engine to produce the first answer is commonly reported in experimental studies that evaluate the incremental behavior of the query engine [2, 10].

Metrics have also been proposed to capture the behavior of query processing systems over continuous data. SRBench is a benchmark for streaming SPARQL query processing. Besides datasets and queries, SRBench proposes metrics to evaluate effectiveness in terms of correctness, and efficiency according to throughput and response time. Finally, Sharaf et al.[18] define two metrics to evaluate the performance of data stream management systems: (a) the average response time captures a system's output rate, and (b) the average slowdown is the average of the ratio of a system's response time to its ideal processing time.

[5] `http://oaei.ontologymatching.org/`

Table 1. Characteristics of metrics. Metrics are characterized according to the type of query processing for which they have been defined, i.e., one-time or continuous, and in terms of the metric interpretation: higher is better (HB) or lower is better (LB). Different measurement methods have been proposed to calculate similar metrics, e.g., Query Response Time [9], Response Time [20], and Execution Time [10].

	Metrics	Characteristics		
		One-Time Query Processing	Continuous Query Processing	Metric Interpretation
Effectiveness	Answer Completeness and Soundness [9]	✓		HB
	Correctness [20]		✓	HB
	Answer Completeness [10]	✓		HB
Efficiency	Query Response Time [9]	✓		LB
	Loading Time [9]	✓		LB
	Throughput[20]		✓	HB
	Response Time [20]		✓	LB
	Queries per Second [5]	✓		HB
	Queries Mixes per Hour [5]	✓		HB
	Min/Max Query Execution Time [5]	✓		LB
	Overall Runtime [5]	✓		LB
	Composite Query Execution Time [5]	✓		LB
	Avg. Execution Time All Queries [5]	✓		LB
	Average Response Time [18]		✓	LB
	Average Slowdown [18]		✓	LB
	Response Time of Joined Tuples [18]		✓	LB
	Slowdown of Joined Tuples [18]		✓	LB
	Time for the First Tuple[2]	✓		LB
	Source Selection Time [10]	✓		LB
	Execution Time [10]	✓		LB
Combined	Combined Metric [9]	✓		HB

These metrics enable the evaluation of a query engine at a given point in time, e.g., either when the first or all answers are produced, as well as the quality of the produced answers. Nevertheless, none of these metrics are able to quantify the continuous behavior of either a one-time or a continuous query engine over a time period, i.e., diefficiency. The *dief@t* and *dief@k* measurement methods overcome this limitation, and provide complementary information to existing metrics that enables to quantify the answer generation rate in a time interval.

3 Motivating Example

Consider the SPARQL queries Query 1 from Listing 1.1 and Query 2 from Listing 1.2 to be executed against the DBpedia dataset using a query engine.[6] To

[6] Prefixes are used as in `http://prefix.cc/`.

Table 2. Query performance measured using metrics from the literature. The performance of the approaches nLDE Not Adaptive (**NA**), nLDE Selective (**Sel**), and nLDE Random (**Ran**) are compared. For each SPARQL query and metric, highlighted cells indicate the approach that exhibits the best performance in that metric.

Metrics	Query 1			Query 2		
	NA	**Sel**	**Ran**	**NA**	**Sel**	**Ran**
Time First Answer (sec.)	28.438	27.058	7.641	0.371	0.242	0.333
Execution Time (sec.)	300.328	300.404	300.137	10.593	12.210	9.304
Throughput (answers/sec.)	46.416	24.334	77.031	486.274	421.868	553.659
Completeness	50.31%	26.38%	83.44%	100%	100%	100%

illustrate, we selected the nLDE engine [1], and executed both queries using three different configurations of nLDE: Not Adaptive, Selective, and Random.

Listing 1.1. Query 1: Retrieve information about resources classified as DBpedia places and infrastructures.

```
SELECT * WHERE {
 ?d1 a dbo:Place .
 ?d2 a dbo:Infrastructure .
 ?d1 dbp:r2LengthF ?o .
 ?d2 dbp:lats ?o .
 ?d1 geo:point ?o1 .
 ?d2 dbp:coordinatesRegion ?o2 . }
```

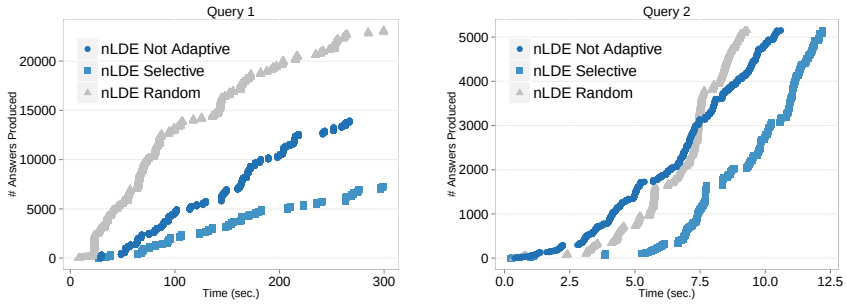
Listing 1.2. Query 2: Retrieve information about resources classified as DBpedia alcohol and Yago alcohol.

```
SELECT * WHERE {
 ?d1 dc:subject dbc:Alcohols .
 ?d1 dbp:routesOfAdministration ?o .
 ?d1 dbp:smiles ?s .
 ?d2 a dbyago:Alcohols .
 ?d2 dbp:routesOfAdministration ?o .
 ?d2 dbp:molecularWeight ?w . }
```

Table 2 reports the performance achieved by the three nLDE approaches using conventional query processing metrics.[7] Based on Table 2, we can conclude the following about the performance of the approaches when executing Query 1: (1) nLDE Random clearly outperforms the other approaches in all the metrics, and (2) nLDE Selective exhibits the worst performance in this case. Regarding the performance achieved by the nLDE approaches when executing Query 2, we can conclude that: (3) nLDE Random achieves the best performance in terms of execution time and throughput, (4) nLDE Selective is able to produce the first answer faster than the other approaches, and (5) nLDE Selective, however, exhibits the worst performance when considering execution time and throughput.

In order to further inspect the behavior of the approaches, let us consider now the continuous performance achieved by the nLDE engine when executing the SPARQL queries. Figure 1 depicts the answer trace of each approach. Answer traces (cf. Definition 1) records the progression of answers produced over time by an engine. Regarding Query 1, we observe in Figure 1(a) that the three approaches exhibit a uniform behavior over time, i.e., one of the approaches exhibits the best performance in all the reported metrics. For instance, nLDE Random steadily outperforms the other approaches by continuously producing

[7] For simplicity, we assume that the reported results are significantly different.



(a) Query 1: The approaches exhibit uniform behavior over time.

(b) Query 2: The approaches exhibit irregular behavior over time.

Fig. 1. Answer traces: continuous query performance. Answers produced (y-axis) as a function of time (x-axis). (a) nLDE Random continuously outperforms the other approaches. (b) The answer trace reveals that in the first 7.45 sec. of execution, nLDE Not Adaptive outperforms other approaches by producing more answers per time unit. However, this behavior is not captured by the metrics reported in Table 2.

more answers over time. Moreover, the answer trace shows that nLDE Selective produced answers at a slower rate in comparison with the other approaches. These findings are consistent with the results (1) and (2).

Regarding the execution of Query 2, in Figure 1(b) we can observe that the approaches exhibit an irregular behavior over time. For example, the answer trace reveals that nLDE Not Adaptive exhibits a better performance than nLDE Random during the first 7.45 seconds of query execution. This new result provides complementary information about the performance of nLDE Random, which was not captured by the analysis (3) using the metrics from Table 2. Furthermore, the high slope of the answer trace of nLDE Selective indicates that it produces over $1,000$ answers at a higher rate than the other approaches from seconds 5 to 7.45 of execution. This finding uncovers novel properties about the behavior of nLDE Selective that were not visible with the metrics reported in Table 2 and to some extent invalidates the conclusion derived in (5).

In this section, we have presented a brief qualitative analysis of the performance of different settings of the nLDE engine over time by manually inspecting the answer trace of the engine when executing two queries. However, to enable reproducibility and replicability of experimental studies, we need quantitative methods to measure the continuous efficiency of query processing approaches.

4 The Diefficiency Metrics

Analyses of answer traces provide valuable insights about the continuous efficiency – or diefficiency – of query engines. Therefore, we devise diefficiency

metrics that rely on answer traces to measure the performance of SPARQL query engines. The answer trace records the exact point in time when an engine produces a query answer and can be formally defined as follows.

Definition 1 (Answer Trace). *Let ρ be an approach, Q a query, and Ω the set of query answers produced by ρ when executing Q. The answer trace of ρ when executing Q, denoted $A_{\rho,Q}$, is defined as a sequence of pairs $(t_1, \mu_1), \ldots, (t_n, \mu_n)$ where $\mu_i \in \Omega$ is the ith answer, $t_i \in \mathcal{R}$ is the timestamp that indicates the point in time when μ_i is produced, and $t_i \leq t_{i+1}$ for all $1 \leq i < n$.*

For example, consider that the engine α executes a query Q and produces an answer μ_1 at the second 1.0 followed by another answer $\mu 2$ at the second 2.0. Then, the answer trace of α is as follows: $A_{\alpha,Q} = \langle (1.0, \mu_1), (2.0, \mu_2) \rangle$. Note that when a SPARQL engine ρ produces no answers ($\Omega = \emptyset$) for a query Q, then the answer trace $A_{\rho,Q}$ correspond to the empty sequence.

Based on the answer trace, we can determine whether a SPARQL engine follows an incremental or a blocking approach during query execution. Incremental approaches are able to produce results over time, i.e., answers are produced at different points in time. In our example, $A_{\alpha,Q}$ indicates that α produces answers incrementally while executing Q. In contrast, blocking approaches produce all query answers at a single point in time – usually at the end of query execution. For instance, the answer trace $A_{\beta,Q} = \langle (2.0, \mu_1), (2.0, \mu_2) \rangle$ of an engine β indicates that β corresponds to a blocking approach while executing Q. In the following, we define incremental and blocking approaches.

Definition 2 (Incremental Approach). *Let ρ be an approach and $A_{\rho,Q}$ its answer trace when executing a query Q. ρ is an incremental approach if there exists (t_j, μ_j) in $A_{\rho,Q}$ such that $t_j < t_{j+1}$.*

Definition 3 (Blocking Approach). *Let ρ be an approach and $A_{\rho,Q}$ its answer trace when executing a query Q. If $t_j = t_{j+1}$ for all $1 \leq j < n$, then ρ is considered a non-incremental or blocking approach.*

Besides characterizing incremental and blocking approaches, answer traces allow for determining the distribution of answers over time, i.e., the answer rate of the engine in intervals of time. For instance, based on the answer traces $A_{\alpha,Q}$ and $A_{\beta,Q}$ from the running example, Figure 2 depicts the answer distribution for the engines α (incremental) and β (blocking): at t=1, α has produced one answer while β has produced no answers until that point.

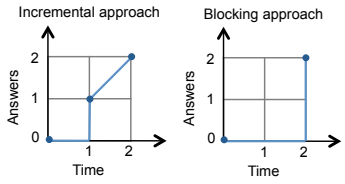


Fig. 2. Answer distribution

The answer distribution corresponds to the number of answers produced in function of time. In the case of incremental approaches, the answer distribution function is computed by applying a linear interpolation between the time points recorded in the answer trace. For blocking approaches, the answer distribution

function corresponds to the maximum of number of answers produced at a point in time. Formally, the answer distribution function is defined as follows.

Definition 4 (Answer Distribution Function). *Let ρ be a SPARQL query engine and $A_{\rho,Q} = \langle (t_1, \mu_1), \ldots, (t_n, \mu_n) \rangle$ its answer trace when executing a query Q. The answer distribution function of ρ when executing Q, denoted $X_{\rho,Q}$, is defined as a function $X_{\rho,Q} : [0; t_n] \rightarrow \mathcal{N}$. For a given $t \in [0; t_n]$, with $0 \leq t < t_1$, $X_{\rho,Q}(t) = 0$. For t such that $t_i \leq t \leq t_{i+1}$, with $1 \leq i \leq n$, $X_{\rho,Q}(t)$ is as follows:*

$$X_{\rho,Q}(t) = \begin{cases} i + \frac{t - t_i}{t_{i+1} - t_i} & , t_i \neq t_{i+1} \\ max(\{j \mid (t_j, \mu_j) \in A_{\rho,Q}, t_j = t_i\}) & , t_i = t_{i+1} \end{cases}$$

In order to measure the continuous efficiency of engines while producing query answers over time, we propose metrics that rely on the answer distribution function. The proposed metrics comprise two novel measurement methods, *dief@t* and *dief@k*. Both *dief@t* and *dief@k* compute the area under the curve of the answer distribution function, which allows for measuring the diefficiency achieved by engines during query execution. Furthermore, in the following sections we will show that *dief@t* and *dief@k* capture the irregular behavior of incremental approaches, such as the one reported in Figure 1(b).

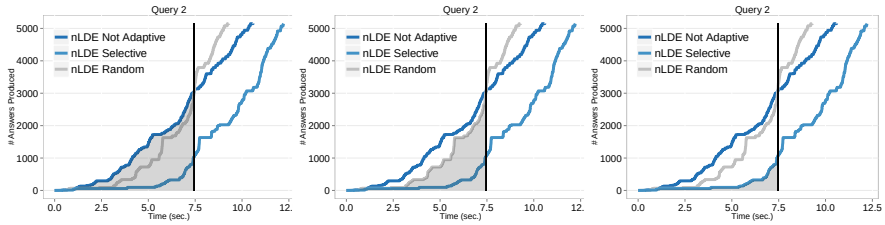
4.1 *dief@t*: Diefficiency at Time *t*

The measurement method *dief@t* measures the diefficiency of an engine in the first t time units of query execution. To do so, *dief@t* computes the area under the curve of the answer distribution function until t. Formally, this area can be defined as the definite integral of the answer distribution function from the moment the engine starts executing a query until the time t.

Definition 5 (*dief@t*). *Let ρ be an approach, Q a query, and $X_{\rho,Q}$ the answer distribution function when ρ executes Q. Consider that ρ produces n answers, i.e., $X_{\rho,Q}$ is defined for the interval $[0; t_n]$. Given $t \in \mathcal{R}$ a point in time such that $t \in [0; t_n]$, the diefficiency of ρ in the first t time units of execution, denominated dief@t, is computed as follows:*

$$dief_{\rho,Q}@t := \int_0^t X_{\rho,Q}(x)\ dx$$

To illustrate the application of *dief@t* in diefficiency analysis, consider the answer traces of the nLDE variants from the motivating example. Figure 3 depicts the computation of *dief@t* at t=7.45 seconds. Intuitively, approaches that produce answers at a higher rate in a certain period of time must exhibit high diefficiency values. Figure 3 indicates that nLDE Not Adaptive achieves the best performance among the other approaches. The *dief@t* values reported in Figures 3(a) to 3(c) confirm that nLDE Adaptive exhibits the highest diefficiency followed by nLDE Random, while nLDE Selective is the least efficient approach. In the following property, we formally state the interpretation of *dief@t*.



(a) nLDE Not Adaptive
dief@t = 7323.46

(b) nLDE Random
dief@t = 5031.90

(c) nLDE Selective
dief@t = 1148.63

Fig. 3. *dief@t* of the nLDE engine for Query 2 at t=7.45. (a), (b), and (c) highlight the area under the curve of the answer distribution function that is computed for measuring the diefficiency of the approaches at the given t. nLDE Not Adaptive exhibits the best performance for the given interval, followed by nLDE Random and lastly nLDE Selective. The reported *dief@t* values are congruent with these observations.

Property 1. Let $\rho 1$ and $\rho 2$ be approaches that execute a query Q. Given $t \in \mathcal{R}$ for which $X_{\rho 1,Q}$ and $X_{\rho 2,Q}$ are defined. If $dief_{\rho 1,Q}@t > dief_{\rho 2,Q}@t$ then $\rho 1$ exhibits a better performance than $\rho 2$ until t in terms of diefficiency.

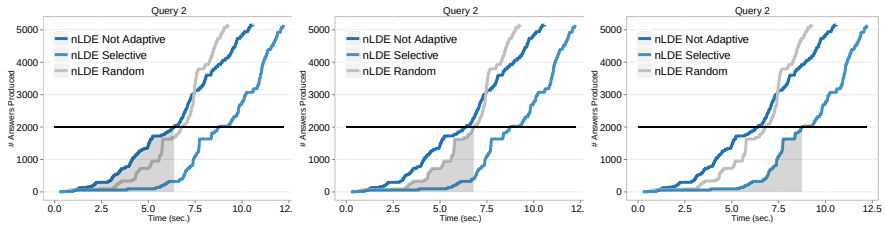
***dief@t* interpretation**: Higher is better.

Lastly, it is important to note that *dief@t* and throughput are not the same metrics. For example, until t=7.45, nLDE Not Adaptive and nLDE Random have produced nearly the same amount of answers,[8] in consequence, the throughput values of both approaches are also almost the same.[9] However, we can observe that nLDE Not Adaptive continuously produced more answers than nLDE Random and this is captured by *dief@t*. In contrast to throughput that considers the total number of answers produced at a single point in time, *dief@t* accounts for the progression of answers over an entire time interval.

4.2 *dief@k*: Diefficiency at *k* Answers

The metric *dief@k* measures the diefficiency of an engine while producing the first k answers of a query, after the first answer was produced. *dief@k* computes the area under the curve of the answer distribution until the point in time t_k when the engine produces the kth answer, as recorded in the answer trace. Formally, this area corresponds to the definite integral of the answer distribution function from the moment the engine starts the query execution until t_k.

[8] Until t=7.45, nLDE Not Adaptive produced 3075 answers and nLDE Random 3067.
[9] The throughput values achieved by nLDE Not Adaptive and nLDE Random are 410 and 408.93 (answers/sec.), respectively.



(a) nLDE Not Adaptive
dief@k = 4686.30

(b) nLDE Random
dief@k = 3517.85

(c) nLDE Selective
dief@k = 3235.67

Fig. 4. *dieft@k* of the nLDE engine for Query 2 at k=2000. (a), (b), and (c) highlight the area under the curve of the answer distribution function that is computed for measuring the diefficiency of the approaches while producing the first k answers. The slope of the answer distributions indicate that nLDE Selective produces the first 2000 answers at a higher rate, followed by nLDE Random and then by nLDE Not Adaptive. The reported *dief@k* values are congruent with these observations.

Definition 6 (*dief@k*). *Let ρ be an approach, Q a query, and $A_{\rho,Q}$ and $X_{\rho,Q}$ the answer trace and answer distribution function when ρ executes Q, respectively. Consider that ρ produces n answers. Given $k \in \mathcal{N}$ such that $0 < k \leq n$, the diefficiency of ρ while producing the first k answers, denominated dief@k, is computed as follows:*

$$dief_{\rho,Q}@k := \int_0^{t_k} X_{\rho,Q}(x)\ dx$$

where $t_k \in \mathcal{R}$ is the point in time when ρ produces the kth answer of Q, i.e., $(t_k, \mu_k) \in A_{\rho,Q}$.

In Figure 4, we illustrate the application of *dief@k* based on the answer traces of nLDE. Intuitively, engines that require a short amount of time to produce k answers are considered more efficient. In consequence, the lower the value of *dief@k*, the higher the continuous efficiency (or diefficiency) of the engine. Figure 4 indicates that nLDE Selective achieves the best performance among the other approaches when producing the first 2000 answers. Figures 4(a) to 4(c) report the values of *dief@k*, which confirm that nLDE Selective achieves the highest diefficiency, while nLDE Not Adaptive is the least efficient approach in terms of the rate while producing the first 2000 answers. In the following property, we formally state the interpretation of *dief@k*.

Property 2. Let $\rho 1$, $\rho 2$ be approaches, and Q a query. Let $n1$, $n2$ be the number of answers produced by $\rho 1$ and $\rho 2$ when executing Q, respectively. Given $k \in \mathcal{N}$, $k \leq n1$ and $k \leq n2$, if $dief_{\rho 1,Q}@k < dief_{\rho 2,Q}@k$ then $\rho 1$ exhibits a better performance than $\rho 2$ in terms of diefficiency while producing the first k answers.

***dief@k* interpretation**: Lower is better.

4.3 Extensions and Properties of *dief@t* and *dief@k*

Measuring Diefficieny at Any Time Interval So far, we have defined metrics to measure the diefficiency since the approach starts the execution of a query until the last answer is produced. Nonetheless, the metric *dief@t* can be extended to measure the diefficiency of a querying approach at any given time interval $[t_a; t_b]$ defined in the answer trace. Intuitively, the diefficiency in $[t_a; t_b]$ corresponds to the area under the curve of the answer distribution in that interval. By applying the additive property of integration on intervals, this area can be computed with the metric *dief@t* at t_a and t_b as follows:

$$\begin{aligned} \int_0^{t_a} X_{\rho,Q}(x)\ dx + \int_{t_a}^{t_b} X_{\rho,Q}(x)\ dx &= \int_0^{t_b} X_{\rho,Q}(x)\ dx \\ \int_{t_a}^{t_b} X_{\rho,Q}(x)\ dx &= \int_0^{t_b} X_{\rho,Q}(x)\ dx - \int_0^{t_a} X_{\rho,Q}(x)\ dx \\ \int_{t_a}^{t_b} X_{\rho,Q}(x)\ dx &= dief_{\rho,Q}@t_b - dief_{\rho,Q}@t_a \end{aligned} \quad (1)$$

The diefficieny of an approach in the interval $[t_a; t_b]$ is $dief@t_b - dief@t_a$.

Measuring Diefficieny Between the k_a*th* and the k_b*th* Answers The metric *dief@k* can also be used to measure the diefficiency of a query engine during the production of the k_ath and the k_bth answers, with $k_a \leq k_b$. From the answer trace of the approach, it is possible to obtain t_{ka} and t_{kb}, the points in time when the k_ath and the k_bth answers are produced, respectively. By definition of the answer trace (cf. Definition 1), it holds that $t_{ka} \leq t_{kb}$. In this case, the diefficiency of the engine corresponds to the area under the curve of the answer distribution in the interval $[t_{ka}; t_{kb}]$. By applying the additive property of integration on intervals, this area can be computed with *dief@k* as follows:

$$\begin{aligned} \int_0^{t_{ka}} X_{\rho,Q}(x)\ dx + \int_{t_{ka}}^{t_{kb}} X_{\rho,Q}(x)\ dx &= \int_0^{t_{kb}} X_{\rho,Q}(x)\ dx \\ \int_{t_{ka}}^{t_{kb}} X_{\rho,Q}(x)\ dx &= \int_0^{t_{kb}} X_{\rho,Q}(x)\ dx - \int_0^{t_{ka}} X_{\rho,Q}(x)\ dx \\ \int_{t_{ka}}^{t_{kb}} X_{\rho,Q}(x)\ dx &= dief_{\rho,Q}@k_b - dief_{\rho,Q}@k_a \end{aligned} \quad (2)$$

The diefficieny of an approach while producing the k_ath and the k_bth answers is $dief@k_b - dief@k_a$.

Based on the definitions of *dief@t* and *dief@k*, it is possible to establish: the analytical relationship between the proposed metrics (Proposition 1), the diefficiency of blocking approaches at any point in time (Theorem 1), and the total diefficiency – from the moment the approach starts the query execution until it produces the last answer – of incremental approaches (Theorem 2).

Proposition 1 (Analytical Relationship Between *dief@t* and *dief@k*). *Let ρ be an approach, Q a query, and $X_{\rho,Q}$ the answer distribution function when ρ executes Q. Consider that $X_{\rho,Q}$ is defined for the interval $[0; t_n]$. The following condition holds for all k such that $1 \leq k \leq n$:*

$$dief_{\rho,Q}@t_k = dief_{\rho,Q}@k$$

Theorem 1. *The diefficiency of blocking approaches is always zero.*

Proof. Consider β a blocking approach and $X_{\beta,Q}$ its answer distribution function when executing a query Q. Assume that $X_{\beta,Q}$ is defined for the interval $[0; t_n]$. Without loss of generality, assume that diefficiency is measured with *dief@t*. We will show that the $dief_{\beta,Q}@t'$ is zero for all t' in the interval $[0; t_n]$. By definition of blocking approach (cf. Definition 2), it holds that $t_i = t_n$ with $1 \leq i \leq n$. With Definition 4, we obtain that $X_{\beta,Q}(t) = 0$, for all $0 \leq t < t_n$, then $\int_0^t X_{\beta,Q}(x)\ dx = 0$, i.e., $dief_{\beta,Q}@t' = 0$. Lastly, $X_{\beta,Q}$ is greater than zero only for $t' = t_n$, for which we obtain that $\int_{t_n}^{t_n} X_{\beta,Q}(x) = 0$, i.e., $dief_{\beta,Q}@t_n = 0$.

Theorem 2. *In queries where the number of answers is higher than one, the total diefficiency of incremental approaches is higher than zero.*

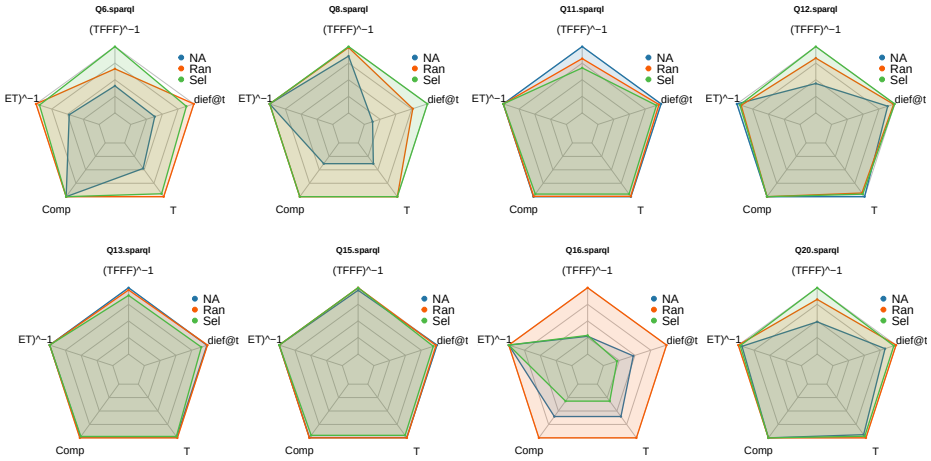
Proof. Consider α an incremental approach, $X_{\alpha,Q}$ its answer distribution and $n \in \mathcal{N}$ the number of answers when executing a query Q. By hypothesis, $n > 1$. By contradiction, assume that the total diefficiency of α is zero. Without loss of generality, assume that the diefficiency of α is measured with *dief@k*, therefore, $dief_{\alpha,Q}@n = 0$. Since $n > 1$ and α is incremental, there exists an answer μ_j produced by α, such that $1 \leq j < n$ and $t_j < t_n$. Without loss of generality, lets assume that $j = n{-}1$. The diefficiency of α between the n-1th and the nth answer can be measured as $dief_{\alpha,Q}@n - dief_{\alpha,Q}@n\text{-}1$. Since $X_{\alpha,Q}$ is non-decreasing, it holds that $dief_{\alpha,Q}@n \geq dief_{\alpha,Q}@n\text{-}1$. By hypothesis $dief_{\alpha,Q}@n = 0$, therefore $dief_{\alpha,Q}@n\text{-}1 = 0$. Applying Equation (2) we obtain that $\int_{t_{n\text{-}1}}^{t_n} X_{\alpha,Q}(x)\ dx = 0$. Note that, by Definition 4, $X_{\alpha,Q}(x) > 0$ for all x with $t_1 \leq t_{n\text{-}1} \leq x \leq t_n$. In consequence, if $\int_{t_{n\text{-}1}}^{t_n} X_{\alpha,Q}(x)\ dx = 0$ then $t_{n\text{-}1} = t_n$. However, this contradicts the hypothesis that α is incremental.

5 Empirical Study

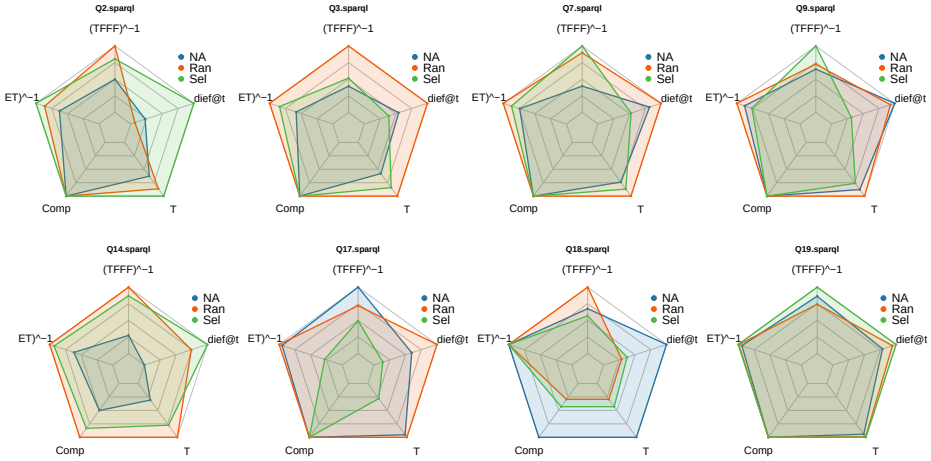
We empirically assess the effectiveness of our metrics *dief@t* and *dief@k* to measure the performance of SPARQL query engines. Experimental results are available at `https://doi.org/10.6084/m9.figshare.5008289` under CC BY 4.0.

Approaches and Implementations: We compare the performance of three configurations of the nLDE engine [1] to execute SPARQL queries, i.e., Not Adaptive, Random, and Selective. nLDE is implemented in Python 2.7.6. Experiments were run on a Debian Wheezy 64 bit machine with CPU: 2x Intel(R) Xeon(R) CPU E5-2670 2.60GHz (16 physical cores), and 256GB RAM. We set the query evaluation timeout to 300 sec. with random delays as specified in [1].

Dataset and Query Benchmark: We use the nLDE Benchmark 1 [1] that comprises 20 queries executed against the DBpedia dataset (v. 2015). Benchmark queries are composed of basic graph patterns of between 4 and 14 triple patterns; these queries are non-selective and produce a large number of intermediate results. We selected 16 queries for which all the approaches produce more than one answer to compute the area under curve of the answer distribution.



(a) Queries in which approaches exhibit a nearly "uniform" behavior



(b) Queries in which *dief@t* uncovers unknown performance patterns

Fig. 5. Performance per benchmark query of SPARQL query processing approaches: nLDE Not Adaptive (NA), nLDE Random (Ran), nLDE Selective (Sel). Axes correspond to the following experimental metrics: Inverse of execution time (ET^{-1}), inverse of time for the first tuple ($TFFT^{-}1$), completeness (Comp), throughput (T), and *dief@t*. Interpretation of all metrics (axes): higher is better.

5.1 Measuring the Performance of SPARQL Query Processing Approaches with Metrics from the Literature and *dief@t*

In this evaluation, we report on the performance achieved by the nLDE approaches when executing the benchmark queries using metrics defined in the literature (cf. Section 2) and our proposed metric *dief@t*. The values for each metric are defined and measured/computed as follows:

- Execution time (**ET**): Elapsed time spent by the approach to complete the execution of a query. ET is measured as the absolute wall-clock system time as reported by the Python `time.time()` function.
- Time for the first tuple (**TFFT**): Elapsed time spent by the approach to produce the first query answer. TFFT is measured as the absolute wall-clock system time as reported by the Python `time.time()` function.
- Completeness (**Comp**): Percentage of the total number of answers produced by the approach after executing a query.
- Throughput (**T**): Number of total answers produced by the approach after evaluating a query divided by its execution time (ET).
- ***dief@t:*** As in Definition 5, where t is the minimum execution time registered by one of the tested approaches when executing a query; *dief@t* is computed with the function `auc` from the `flux` package in `R`.

In Figure 5, we report on the results of the performance achieved by the nLDE configurations per query using radar plots. Radar plots allow for comparing the performance of the studied approaches in multiple dimensions, in this case, in several metrics. For the sake of readability, we transformed the axes of the plots such that all the metrics have the same interpretation: higher is better.

Figure 5(a) comprises the queries where the performance of the nLDE approaches is consistent in all the metrics, i.e., the approaches display a "uniform" behavior over time. The consistent performance can be easily observed when the polygon of an approach encloses the ones of the other approaches. For example, according to the plotted results in Q16, the approach nLDE Random exhibits the best performance in all the dimensions, followed by nLDE Random, and then by nLDE Selective. In cases like these, the metrics defined in the literature accurately capture the overall performance of the approaches. It is important to note that, for these queries, *dief@t* is consonant with the other metrics, thus, corroborating the steady outperformance of one approach.

Figure 5(b) groups the queries in which the approaches exhibit a fluctuating performance among the metrics. In Q2, for instance, the values of the metrics from the literature indicate that nLDE Random and nLDE Selective are competitive approaches. Yet, *dief@t* allows for uncovering that nLDE Selective is able to continuously produce answers at a faster rate than nLDE Random for this query. Conversely, in Q7, conventional metrics indicate that nLDE Not Adaptive is outperformed by the other approaches. Nonetheless, *dief@t* suggests that nLDE Not Adaptive is rather competitive in terms of diefficiency. In summary, *dief@t* allows for more comprehensive analyses of approaches, while characterizing the entire engine performance during query execution.

5.2 Measuring the Continuous Answer Rate of SPARQL Query Processing Approaches with *dief@k*

We now analyze the diefficiency achieved by the nLDE approaches while producing the first k answers for queries from Figure 5(b).[10] We compute *dief@k* as in

[10] Q14 was discarded since it produces only four answers.

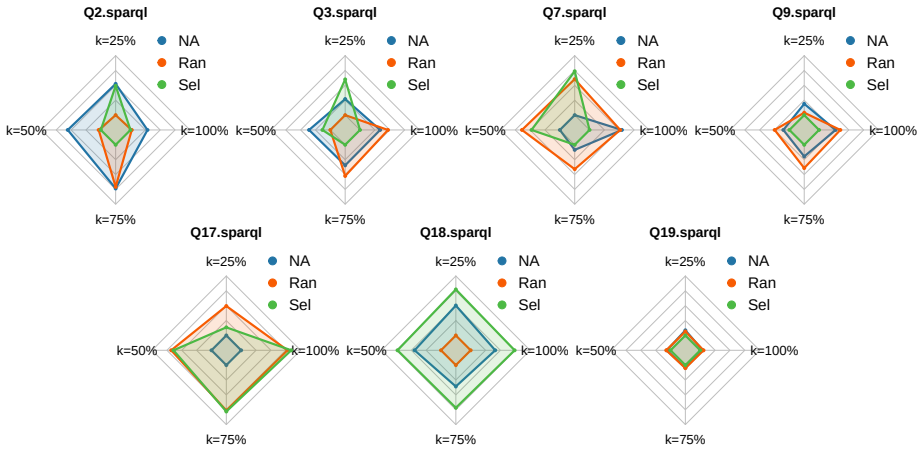


Fig. 6. Diefficiency while producing a portion k of the answers, per benchmark query and SPARQL query processing approaches: nLDE Not Adaptive (NA), nLDE Random (Ran), nLDE Selective (Sel). Performance is measured with *dief@k*, with k=25%, k=50%, k=75%, k=100%. Interpretation of the axes: lower is better.

Definition 6, and we set k to the minimum amount of answers produced among all the nLDE variants in each query. Figure 6 reports on the *dief@k* values of nLDE while producing the first 25%, 50%, 75%, and 100% of the query answers. We can observe that, in most of the cases, nLDE Not Adaptive maintains a steady answer rate over time (except for Q7 and Q9). Furthermore, *dief@k* values show that the behavior of nLDE Selective and nLDE Random fluctuates[11], even producing the first 25% of the answers up to 3 times slower than nLDE Not Adaptive (in Q7); this fluctuating behavior can be a consequence of the adaptive heuristics implemented by nLDE Random and nLDE Selective during query processing. With this study, we show how the metric *dief@k* can be used to explain and quantify continuous answer rates of query engines.

6 Conclusions and Future Work

In this paper, we tackled the problem of quantifying the continuous behavior of a query engine, and define two novel experimental metrics, *dief@t* and *dief@k*. Formal properties of these metrics demonstrate the analytical characteristics of the proposed measurement methods, as well as stating lower bounds according to the characteristics of the assessed query engine. Additionally, *dief@t* and *dief@k* were empirically evaluated, and observed results allow for uncovering patterns in the performance of the evaluated engines that could not be quantified with existing metrics. As proof of concept, *dief@t* and *dief@k* were used to evaluate SPARQL query approaches, however, both metrics will be able to capture and quantify the behavior of any method that produces results incrementally.

In the future, we plan to use *dief@t* and *dief@k* to evaluate state-of-the-art SPARQL query engines, and compare observed patterns with the ones quantified

[11] This can be observed when the polygons in the plot are not perfect diamonds.

by metrics proposed by existing benchmarks. We hypothesize that *dief@t* and *dief@k* will allow for uncovering unknown characteristics of these engines.

References

1. M. Acosta and M. Vidal. Networks of linked data eddies: An adaptive web query processing engine for RDF data. In *ISWC*, pages 111–127, 2015.
2. M. Acosta, M.-E. Vidal, J. Castillo, T. Lampo, and E. Ruckhaus. ANAPSID: An adaptive query processing engine for SPARQL endpoints. In *ISWC*, pages 18–34, 2011.
3. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of RDF data management systems. In *ISWC*, pages 197–212, 2014.
4. J. Angele and Y. Sure, editors. *Proceedings of the EON Workshop*, volume 62 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2002.
5. C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
6. L. Cheng and S. Kotoulas. Efficient large outer joins over mapreduce. In *Euro-Par*, pages 334–346, 2016.
7. S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *SIGMOD*, pages 145–156, 2011.
8. J. Euzenat and P. Shvaiko. *Ontology Matching, Second Edition*. Springer, 2013.
9. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semant.*, 3(2-3):158–182, Oct. 2005.
10. G. Montoya, M. Vidal, Ó. Corcho, E. Ruckhaus, and C. B. Aranda. Benchmarking federated SPARQL query engines: Are existing testbeds enough? In *ISWC*, pages 313–324, 2012.
11. M. Morsey, J. Lehmann, S. Auer, and A. N. Ngomo. DBpedia SPARQL benchmark-performance assessment with real queries on real data. In *ISWC*, pages 454–469, 2011.
12. M. Nentwig, M. Hartung, A. N. Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.
13. D. L. Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC*, pages 370–388, 2011.
14. N. A. Rakhmawati, M. Saleem, S. Lalithsena, and S. Decker. QFed: query set for federated SPARQL query benchmark. In *iiWAS*, pages 207–211, 2014.
15. M. Saleem, Q. Mehmood, and A. N. Ngomo. FEASIBLE: A feature-based SPARQL benchmark generation framework. In *ISWC*, pages 52–69, 2015.
16. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. In *ISWC*, pages 585–600, 2011.
17. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^2Bench: A SPARQL performance benchmark. In *ICDE*, pages 222–233, 2009.
18. M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Trans. Database Syst.*, 33(1):5:1–5:44, 2008.
19. D. Vrandecic and Y. Sure. How to design better ontology metrics. In *ESWC*, pages 311–325, 2007.
20. Y. Zhang, M. Pham, Ó. Corcho, and J. Calbimonte. SRBench: A streaming RDF/SPARQL benchmark. In *ISWC*, pages 641–657, 2012.