Intelligent SPARQL Endpoints: Optimizing Execution Performance by Automatic Query Relaxation and Queue Scheduling

Ana I. Torre-Bastida[1], Esther Villar-Rodriguez[1], Miren Nekane Bilbao[2], and Javier Del Ser[1,2,3]

[1] TECNALIA. OPTIMA Unit, E-48160 Derio, Spain
{isabel.torre,esther.villar,javier.delser}@tecnalia.com
[2] University of the Basque Country UPV/EHU, 48013 Bilbao, Spain
{nekane.bilbao,javier.delser}@ehu.eus
[3] Basque Center for Applied Mathematics (BCAM), 48009 Bilbao, Spain

Abstract The Web of Data is widely considered as one of the major global repositories populated with countless interconnected and structured data prompting these linked datasets to be continuously and sharply increasing. In this context the so-called SPARQL Protocol and RDF Query Language is commonly used to retrieve and manage stored data by means of SPARQL endpoints, a query processing service especially designed to get access to these databases. Nevertheless, due to the large amount of data tackled by such endpoints and their structural complexity, these services usually suffer from severe performance issues, including inadmissible processing times. This work aims at overcoming this noted inefficiency by designing a distributed parallel system architecture that improves the performance of SPARQL endpoints by incorporating two functionalities: 1) a queuing system to avoid bottlenecks during the execution of SPARQL queries; and 2) an intelligent relaxation of the queries submitted to the endpoint at hand whenever the relaxation itself and the consequently lowered complexity of the query are beneficial for the overall performance of the system. To this end the system relies on a two-fold optimization criterion: the minimization of the query running time, as predicted by a supervised learning model; and the maximization of the quality of the results of the query as quantified by a measure of similarity. These two conflicting optimization criteria are efficiently balanced by two bi-objective heuristic algorithms sequentially executed over groups of SPARQL queries. The approach is validated on a prototype and several experiments that evince the applicability of the proposed scheme.

Keywords: SPARQL, query rewriting, Linked Open Data, ontology management, multiobjective optimization.

1 Introduction and Motivation

It will be soon a decade since the so-called Linked Open Data (LOD) paradigm, along with several related projects and initiatives, became the main technology enabler for the expansion of the Semantic Web, whose *raison d'être* was an intrinsic information technologies revolution centered on enriching the published data and coping with the inherent inability of machines to understand websites [1]. Over the last decade the increasing adoption of LOD led to the development of a distributed mesh of globally interlinked knowledge capable of providing a pioneering method to traverse the web and interpret its contents: the Web of Data. This huge, distributed, diverse database is deployed on manifold domains and a wide range of subjects such as government, libraries, life science and media, among many others. It allows for the execution of exploratory and selective queries over a enormous set of updated, comprehensive and pertinent data. The prevalent semantic query language for these repositories is SPARQL, which provides a full set of query operations and functionalities. Notwithstanding, in order to fully unleash the Semantic Web potential SPARQL users are forced to dominate the syntax of the SPARQL language. On this purpose the community has devoted considerable research effort towards deriving sophisticated yet friendly tools to help users properly exploit the vast amount of available data and achieve a satisfactory performance in terms of accuracy. Under this rationale, the systems and engines where SPARQL endpoints are deployed have become the primary target where to allocate specialized resources and intelligent software procedures to enhance the quality of service commonly jeopardized and called into question due to significant delays, specially when dealing with large datasets [2].

The contribution of this research work gravitates on three main axes to improve the performance of SPARQL systems: the performance prediction of SPARQL queries prior to their processing, their relaxation and the planning of run queues in processing engines. In the field of performance prediction there is a large number of works in the field of the SQL query language for relational databases, which have traditionally revolved around statistical or heuristics costs estimation. In regards to the prediction of the query execution time, supervised learning models have positioned themselves as the *off-the-shelf* estimators in recent years (see e.g. [3,4]). To the best of our knowledge there are very scarce studies that extrapolate this acquired knowledge with relational databases to the LOD repositories. The main difference between these two areas resides on the absence of an schematic structure in the RDF standard, as well as on the shortage of statistics of the datasets compounding the LOD environment. Justifiably, the current generation of SPARQL query cost estimation approaches that inspire from those derived for relational databases have been proven to be inadequate for the task of performance prediction. This is the rationale for the brand new direction started in [3] and subsequently followed in [4,5] that resorts to machine learning techniques to extract SPARQL performance metrics from already executed queries. Despite the good predictive scores reported in these references (with the latest work in [5] scoring an average R^2 of 0.94 with Support Vector Machines), we will show throughout this manuscript that there is still room for improvement in terms of the learning model and the set of features.

Concerning the second aspect that can be leveraged so as to improve the performance of endpoints, the optimization of SPARQL queries has hitherto mainly focused on rewriting the query at hand based on different objectives, such as the minimization of the execution time or the reduction of its structural complexity. We classify these studies into three categories depending on the utilized optimization technique: cost-based [6–8], heuristics-based [9–11] and machine learning techniques [12, 13]. Cost-based schemes suffer from the aforementioned low availability of statistics in the LOD. Heuristic approaches assume that structurally simple queries are in general less expensive, but this is not the case in SPARQL due to the inference and variant extensional information contained in a SPARQL arrangement. The work by Bicer et al. in [12] introduce the concept of Relational Kernel Machines, which simplify the problem of extracting features from the complex structure of semantic data and hence improving naïve approximations based on Support Vector Machines. Likewise, in [13] long-running queries (detected by predicting its computational costs) are relaxed by applying a Genetic Algorithm (GA) based rewriting approach so as to yield a faster rewritten query. In our work we will take a step further so as to consider in the determination of the query relaxation policy the inherent Pareto trade-off between the quality of the results returned by the query and the relative running time with respect to its original version. This Pareto-optimal balance between both objectives will be shown to be tractable via evolutionary multi-objective heuristics.

Finally, the third axis refers to the scheduling of run queues to organize and coordinate query executions, around which our literature survey has identified a single, recent yet relevant contribution for SPARQL endpoints [14]. In this paper the authors explain that guaranteeing a consistently good quality of service in SPARQL endpoints is a difficult task to accomplish, for which the use of an scheduler is proposed to optimally manage the execution of queries in SPARQL endpoints. We go one step beyond the simple schedulers explored in this reference by proposing a novel approach in which we optimize the scheduling criterion based on the previously mentioned SPARQL relaxation policies.

Our software system blends together the three aspects commented above to improve the runtime performance of a SPARQL endpoint. The problem is that many of the queries processed by such systems can not be executed within a reasonable time for the user. To address this issue a bi-objective algorithm is designed to obtain the optimal set of relaxation rules on this dataset without disregarding the quality of the query result. By applying such a Pareto-optimal set of relaxation rules the execution time of the queries is reduced while keeping the quality degradation of their results to a minimum. Such rule sets can be further exploited by implementing a set of processing queues in the SPARQL endpoint, so that the optimization algorithm determines the adequate set of relaxation rules, the allocation of queries over the pool of processing queues and the execution order of the queries assigned to every queue. In summary, the main goal of this paper is the design of a software system capable of enhancing the performance of a SPARQL endpoint by combining optimized run queues, adequate query relaxation policies and SPARQL query run time predictions. Schematically the novel technical ingredients of this research work are as follows:

1. The derivation of new predictive features for the design of a runtime estimator for SPARQL queries, which can be divided in query language algebra and vocabulary features defining the terms of the query.
2. The design and implementation of a system based on run queues to improve the performance of SPARQL endpoints, which to our knowledge is the first one proposed in the literature.
3. A query relaxation optimization algorithm guided by two objectives: the maximization of the query quality (quantified in terms of similarity) and the minimization of the run time of the query.
4. The use of parallelizable evolutionary meta-heuristic solvers to the performance improvement of SPARQL endpoints in the particular aspects of query relaxation and run queue scheduling mentioned previously.

The rest of the manuscript is structured as follows: Section 2 overviews the general architecture of the proposed system and formulates the optimization problem that mathematically defines its operation. Subsections 2.1 and 2.2 delve into the design of estimators for the query running time and quality on which the aforementioned optimization problem is based. Next, Section 3 elaborates on the meta-heuristic optimization algorithm designed to efficiently implement the proposed system, including relevant aspects such as the solution encoding and the design of the operators. Section 4 reports on the experimental evaluation of the proposed scheme and conclusions are drawn in Section 5.

2 Architecture Overview and Problem Formulation

In this section we briefly introduce key concepts and notation used throughout the rest of the paper. SPARQL is the standard query language for RDF. Let I be the set of all IRIs (Internationalized Resource Identifiers), L be the set of RDF *literals*, and B be the set of RDF *blank nodes*. These three infinite sets are pairwise disjoint. An RDF *triple* is a tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$; s is called the *subject*, p is the *predicate*, and o stands for the *object* of the triple, respectively. An RDF *graph* is a finite set of triples. For the purpose of this paper, a *dataset* D is an RDF graph. Given a dataset D, we refer to the set $voc(\mathrm{D}) \subseteq (I \cup L)$ of IRIs and literals occurring in D as the *vocabulary* of D. We use the words *term* or *resource* to refer to elements in $I \cup L$.

The core of a SPARQL query is a *basic graph pattern*, which is used to match an RDF graph in order to search for the required answers. A *triple pattern* is a triple, without blank nodes, where a variable may occur in any place of the triple. A *graph pattern* is an expression recursively defined as follows: 1) a triple pattern is a graph pattern; 2) if P_1, P_2 are graph patterns, then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, and $(P_1 \text{ OPT } P_2)$ are graph patterns; and c) if P is a graph pattern and C is SPARQL constraint, then $(P \text{ FILTER } C)$ is a graph pattern. With these definitions in mind, a query is defined by $Q = (D, \delta)$ where D is the dataset to be used during the pattern matching and δ is the graph pattern of the query.

Figure 1 shows an overview of the proposed system, which is conceived as an intermediate manager between the users submitting their queries and the pool of parallel processing queues that compound the SPARQL endpoint. Several modules can be found in this diagram: first it is important to remark that the relaxation policies and the mapping to processing queues are optimized at the level of previously clustered query groups, so that queries within the same cluster undergo the same relaxation rules and are assigned to the same processing queue. This cluster analysis module is based on the methodology presented in [15] that follow these steps: data generation mimicking an input data source, query log mining, clustering and SPARQL feature analysis. As a result P query sets $\{\mathbf{Q}_p\}_{p=1}^{P} = \{\{Q_p^n\}_{n=1}^{N_p}\}_{p=1}^{P}$ (clusters) are produced with $Q_p^n = (D, \delta_p^n)$.

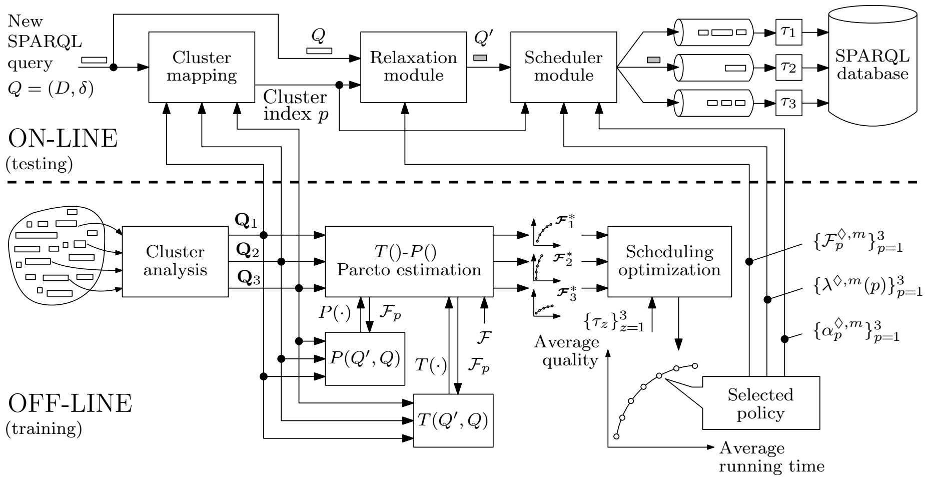


Figure 1: Overview of the proposed architecture assuming $P = 3$ query profiles $\{\mathbf{Q}_p\}_{p=1}^{3}$ and $Z = 3$ processing queues at the endpoint. The lower part of the plot corresponds to the processing stages that are performed off-line based on a historic record of queries submitted to the endpoint, whereas the upper part illustrates the entire relaxation and scheduling procedure applied to a new incoming query submitted to the endpoint.

Prior to its online working regime the SPARQL endpoint must decide the set of relaxation policies, the queue and the priority within the queue for each of such clusters. Let $f_r(Q)$ be the generic definition for a relaxation rule, drawn from a R-sized vocabulary $\mathcal{F} = \{f_r(Q)\}_{r=1}^{R}$ of possible relaxation operators. It is important to note that $f_r(Q)$ may only impact on a certain triple (s, p, o) within Q or, instead, involve more terms within its expression. Three kind of rules have been considered in the setup:

1. Deletion rules, which consist of eliminating a triple (s, p, o), filter, terms, union and/or optional clauses from the query.
2. Addition rules, which add a restrictive clause to the query, e.g. a limit operator.
3. Hierarchical rules, by which a term of the query is substituted by its descendant or ascendant in the ontological hierarchy of the queried dataset.

The complete list of possible rules $\mathcal{F}$ is sorted by their estimated degree of degradation on the results of the relaxed query. Under this notation $\mathcal{F}_p \subseteq \mathcal{F}$ will denote the sequence of relaxation operators that will be applied to the queries belonging to cluster p, whereas $Q_p^{n,\prime}$ will denote the relaxed version of query Q_p^n after the application of the rules in $\mathcal{F}_p$.

The determination of $\{\mathcal{F}_p\}_{p=1}^P$ will be done under a twofold criteria: we seek to optimally balance the impact of the relaxation policy on the average running time and quality of the results associated to the query; the more relaxed the query $Q_p^{n,\prime}$ is, the faster it will be executed at the endpoint, but the less precise the returned results will be with respect to the original, unrelaxed query Q_p^n. Such objectives will be represented by two functions $T(Q_p^{n,\prime}, Q_p^n)$ and $P(Q_p^{n,\prime}, Q_p^n)$, both $\in [0,1]$, corresponding to the relative running time and quality of the relaxed query $Q_p^{n,\prime}$ w.r.t. Q_p^n. In mathematical terms the relaxation module in Figure 1 seeks, for each query profile $\mathbf{Q}_p$, a group of policies $\boldsymbol{\mathcal{F}}_p^*$ composed by several relaxation rule sets $\{\mathcal{F}_p^{*,m}\}_{m=1}^{M_r}$ such that

$$\mathcal{F}_p^{*,m} = \arg_{\mathcal{F}_p \subseteq \mathcal{F}} \left\{ \min \frac{1}{N_p} \sum_{n=1}^{N_p} T(Q_p^{n,\prime}, Q_p^n), \max \frac{1}{N_p} \sum_{n=1}^{N_p} P(Q_p^{n,\prime}, Q_p^n) \right\}, \quad (1)$$

subject to $Q_p^{n,\prime}$ being the query resulting from the successive application of the relaxation rules $f \in \mathcal{F}_p$ to Q_p^n. For each $m \in \{1, \ldots, M_r\}$ a different set of rules $\mathcal{F}_p^{*,m}$ balances differently both fitness objectives when applied over the reference query profile $\mathbf{Q}_p$. Subsections 2.1 and 2.2 will elaborate on the estimation of the value for $T(Q_p^{n,\prime}, Q_p^n)$ and $P(Q_p^{n,\prime}, Q_p^n)$ prior to the execution of the query itself.

Once such Pareto estimations have been produced off-line for each query profile, the scheduler module exploits this information to determine 1) which processing queue should be assigned to an incoming query associated to a certain cluster $p \in \{1, \ldots, P\}$; 2) which relaxation policy should be applied to the query among those in $\boldsymbol{\mathcal{F}}_p^*$; and 3) the execution order of the queries (i.e. their priority) in the case several of them are assigned to the same queue. Without loss of generality computing power differences between processing queues are assumed to yield factors $\{\tau_z\}_{z=1}^Z$ (with $\tau_z \in (0,1]$ and Z denoting the number of queues) such that the time taken by queue z to process Q_p^n is reduced by $100 \cdot \tau_z$ %. The queue allocation to be decided at this module will be denoted as a non-surjective, non-injective mapping function $\lambda : \{1, \ldots, P\} \mapsto \{1, \ldots, Z\}$, such that $\lambda(p)$ will stand for the queue to which the queries associated to profile $p \in \{1, \ldots, P\}$ will be forwarded. Priorities within queue $z \in \{1, \ldots, Z\}$ will be denoted as a real-valued variable $\alpha_p \in \mathbb{R}$ such that if $\lambda(p) = \lambda(p')$ (i.e. profiles p and p' are assigned the same processing queue), the queries in profile p will be executed first if $\alpha_p \leq \alpha_{p'}$. Conversely, if $\alpha_p > \alpha_{p'}$ queries belonging to cluster p' will be granted a higher execution priority level than those in p. The criterion to determine the optimal mapping $\lambda^\diamondsuit(p)$, relaxation policies $\{\mathcal{F}_p^\diamondsuit\}_{p=1}^P$ and priority factors $\boldsymbol{\alpha}^\diamondsuit = \{\alpha_p^\diamondsuit\}_{p=1}^P$ at the scheduler module will again rely on the aforementioned time-quality Pareto trade-off, but incorporating a subtle yet relevant aspect: queries within the same processing queue interact in terms of their completion time, i.e. both the relative order of queries within a given queue and the different processing capabilities of the queues themselves are meaningful for the overall evaluation of the average execution time taken by the endpoint to process incoming queries. In other words, a vector of mapping functions $\boldsymbol{\lambda}^{\Diamond}(\cdot) \doteq \{\lambda^{\Diamond,m}(\cdot)\}_{m=1}^{M_s}$, relaxation policies $\boldsymbol{\mathcal{F}}_p^{\Diamond} \doteq \{\mathcal{F}_p^{\Diamond,m}\}_{m=1}^{M_s}$ and priority levels $\boldsymbol{A}^{\Diamond}(\cdot) \doteq \{\boldsymbol{\alpha}^{\Diamond,m}\}_{m=1}^{M_s} = \{\{\alpha_p^{\Diamond,m}\}_{p=1}^{P}\}_{m=1}^{M_s}$ will balance the following Pareto:

$$\lambda^{\Diamond,m}(\cdot),\ \boldsymbol{\alpha}^{\Diamond,m}, \mathcal{F}_p^{\Diamond,m} = \arg_{\substack{\lambda\in\Lambda\\ \boldsymbol{\alpha}\in\mathbb{R}^P\\ \mathcal{F}_p\subseteq\mathcal{F}}} \left\{ \min \frac{1}{P}\sum_{p=1}^{P} \frac{\tau_{\lambda(p)}}{N_p} \sum_{n=1}^{N_p} T(Q_p^{n,\prime}, Q_p^n) \right.$$

$$\sum_{\substack{\varrho=1\\ \varrho\neq p}}^{P} \frac{1}{N_\varrho} \sum_{\eta=1}^{N_\varrho} \tau_{\lambda(\varrho)} T(Q_\varrho^{\eta,\prime}, Q_\varrho^{\eta}) \mathbb{I}(\alpha_\varrho \leq \alpha_p) \mathbb{I}(\lambda(\varrho) = \lambda(p)), \tag{2}$$

$$\left. \max \frac{1}{P}\sum_{p=1}^{P} \frac{1}{N_p} \sum_{n=1}^{N_p} P(Q_p^{n,\prime}, Q_p^n) \right\}, \tag{3}$$

where $\mathbb{I}(\cdot)$ is an auxiliary indicator function taking value 1 if its argument is true and 0 otherwise; $\lambda(p) \in \{1,\ldots,Z\}$ denotes the index of the queue to which the queries in cluster p are assigned; and $Q_p^{n,\prime}$ is the result of relaxing query Q_p^n through policy $\mathcal{F}_p$. In words, Expression (2) denotes the time taken by the queries Q_p^n within cluster p, which depends not only on the assigned queue through $\tau_{\lambda(p)}$, but also on the average time taken by queries belonging to other clusters $\varrho \in \{1,\ldots,p-1,p+1,\ldots,P\}$ provided that they are assigned to the same processing queue and granted higher priority. Finally, Expression (3) poses the mean quality score averaged over all the considered query clusters.

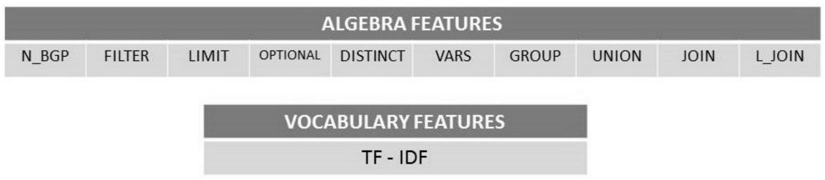
Before proceeding with the algorithmic solution proposed to efficiently solve the above problems, it should be noted that in practice the relaxation and scheduling modules might be conceived and formulated as a single optimization problem driven by the objective functions in Expressions (2) and (3). However, by decoupling both modules a deeper understanding of the flexibility of the clusters with respect to the set of relaxation operators can be acquired, with further potential applications beyond the one addressed in this paper (e.g. optimizing a distributed deployment of the database at hand).

2.1 SPARQL Query Run-time Prediction

As shown in Figure 1 and argued above, an estimation of the running time required to complete a given relaxed query Q is needed when the proposed approach operates in both off-line and on-line modes. Such an estimation must be produced without executing the query itself. Therefore, a supervised learning model is included in order to predict execution times of generic SPARQL queries based on a historic set of already executed queries. The adopted approach is similar to the one presented by Hassan et al. in [5], but with novel ingredients:

the learning model itself and the set of features extracted from the expression of the SPARQL queries to build the training dataset. As such, this dataset consist of a set of previously executed queries and the observed performance metric values (execution times) for those queries in their native, unrelaxed form. The goal is to extract proper features from the syntax of the queries to construct a prediction model that provide us with an accurate estimation of the execution time that can be generalized to new, possibly relaxed query sets. The proposed set of features are classified as:

1. Algebra features, which represent the syntax of the SPARQL query, its operators and structural information. First we transform a query into an algebra expression tree, from which we extract the following features: number of basic graph patterns, filter operator presence, type of filter, limit operator presence, optional operator presence, distinct operator presence, number of projected variables, group operator presence, number of union, number of joins and number of left joins.
2. Dataset vocabulary features, for which we use the dataset terms involved in the SPARQL query definition to extract intensional and semantic information about them. First we compute the overall set of terms, and with this *bag of words* we compute the TF-IDF frequency [16] as a quantitative score of the importance of the terms of the query (*words*) in the dataset (*document*).



ALGEBRA FEATURES									
N_BGP	FILTER	LIMIT	OPTIONAL	DISTINCT	VARS	GROUP	UNION	JOIN	L_JOIN

VOCABULARY FEATURES
TF - IDF

Figure 2: SPARQL query features vector.

Regarding the supervised learning model we opt for a so-called Random Forest Classifier [17], a widely utilized ensemble model characterized by its good generalization properties and low tendency to overfit. In short Random Forests exploit the principle of bagging by randomly splitting the data into chunks, selecting a feature subset and training a weak learner (tree) on each of them, from where the predicted output is given by voting (classification) or averaging (prediction) the individual outputs of the aforementioned weak models.

2.2 SPARQL Quality Estimation

The estimation of the Pareto-optimal set of relaxation policies in the proposed system also requires an *a priori* estimation of the quality of the relaxed query with respect to its original version. This estimated function, heretofore denoted as $P(Q', Q)$, must be computed based exclusively on the query itself, i.e. without submitting any request to the endpoint. Within the scope of this paper it has been noted that a query can be defined by a pair $Q = (D, \delta)$. Based on this notation, the application of a relaxation policy to this query produces a relaxed query $Q' = (D, \delta')$, with δ defining the RDF graph pattern of the query. To this end, a similarity metric $S(Q', Q)$ has been designed to provide a quantitative estimation of the similarity between queries Q' and Q. This similarity metric is designed to replace the generic function $P(Q', Q)$. Basically, $S(Q', Q)$ maps two SPARQL queries to a real value in the closed interval $[0, 1]$, such that higher values indicate that queries are more similar to each other. Mathematically, $S : \mathcal{Q} \times \mathcal{Q} \mapsto [0, 1]$, with $\mathcal{Q}$ denoting the set of all possible SPARQL queries that can be defined over the dataset D.

To implement the similarity function, we follow the approach introduced in [18] by which similarity between Q' and Q depends on the similarity between their graph patterns δ' and δ. We can consider a graph pattern δ as composed by a number of triple patterns $\{P_1, \ldots, P_K\}$, each composed by three terms (s_k, p_k, o_k). In general a term t (either subject, predicate or object) can be assigned different *similarity factors* $\phi(t', t) \in \mathbb{R}[0, 1]$. These factors are meant to reflect a similarity measure between the term t of the original query pattern and its counterpart t' after the relaxation process.

Given a triple pattern $P = (s, p, o)$ and its relaxed counterpart $P' = (s', p', o')$, the vector $(\phi(s', s), \phi(p', p), \phi(o', o))$ can be conceived as a point in a three dimensional space, where the point $(1, 1, 1)$ represents the best (i.e. the triple P and its relaxed counterpart are equal to each other) and the point $(0, 0, 0)$ the worst. Therefore, the Euclidean distance between $(\phi(s', s), \phi(p', p), \phi(o', o))$ and $(1, 1, 1)$ can be deemed as a measure of pattern similarity $\sigma(P', P)$ given by

$$\sigma(P', P) = 1 - \sqrt{\frac{(1 - \phi(s', s))^2 + (1 - \phi(p', p))^2 + (1 - \phi(o', o))^2}{3}}. \quad (4)$$

Given an original query Q compounded by graph patterns $\{P_1, \ldots, P_K)$ and its relaxed version Q', their pattern-wise similarity values $\{\sigma(P'_k, P_k)\}_{k=1}^{K}$ (computed as in the above expression) can be likewise regarded as coordinates in a K-dimensional space. Again, the Euclidean distance between the points $(\sigma(P'_1, P_1), \ldots, \sigma(P'_K, P_K))$ and $(1, \ldots, 1)$ yields the sought normalized similarity measure $S(Q', Q)$ expressed as

$$S(Q', Q) = 1 - \sqrt{\frac{\sum_{k=1}^{K} (1 - \sigma(P'_k, P_k))^2}{K}}, \quad (5)$$

which serves as a predictive estimation of the quality of a given relaxed query Q' for the meta-heuristic optimization algorithm explained in what follows.

3 Proposed Meta-heuristic Solver

Following the mathematical formulation posed in Section 2 (Expressions (1) through (3)), it should be noticed that the two bi-objective optimization problems driving the relaxation and scheduling criteria of the proposed scheme are very similar to each other, the difference relying mainly on the alphabet of the variables involved. As such, the Pareto optimization of the relaxation policies $\mathcal{F}_p$ to be applied to a certain query profile $\mathbf{Q}_p$ can be encoded as a $|\mathcal{F}_p|$-sized vector of integers drawn from the set $\{1,\ldots,R\}$ (with R standing from the number of possible relaxation operators), each indexing a different rule within $\mathcal{F}$. On the other hand, the scheduling problem requires a mixed integer-real encoding strategy, as it implies optimizing not only – again – the relaxation policy to be applied for a given query profile, but also the relative ordering of queries along time when they belong to profiles assigned to the same processing queue. Such an encoding diversity imprints notable changes in the design of the solvers to face these problems.

In essence both optimization algorithms are based on approximate, self-learning approaches that allow evolving intermediate solutions towards areas of the solution space characterized by increasingly higher Pareto optimality. In this context there is a plethora of algorithmic alternatives in the literature to deal with multi-objective problems [19, 20], among which we will focus on those inspiring from behavioral patterns observed in Nature and Social Sciences. Specifically the bi-objective solver utilized in this work stands on the so-called Harmony Search (HS) algorithm [21], which emulates the process of music improvisation and composition observed in practice so as to yield a population-based meta-heuristic solver quite similar to other schemes from Evolutionary computation[1], but outperforming them in many practical scenarios [22].

In its original definition, the HS algorithm breaks down in three stochastically-driven operators (HMCR, PAR and RSR) that are sequentially applied to every compounding variable (referred to as *instrument*) of a φ-sized population of potential solutions to the problem (correspondingly, *harmonies*) until a given stop condition is satisfied. The value taken by a variable is therefore a *note*, whose entire pitch range depends on the alphabet of the represented variable. This is the rationale why the design of the HS operators is closely linked to the encoding approach that allows numerically representing the produced solutions, specially those that do not permute elements among the population but rather impose perturbations on the variables based on their respective alphabets.

When particularized to the problem tackled in this work, the integer encoding approach used to deal with the first relaxation problem posed in Expression (1) permits to resort to the nominal HS operators for integer-variable problems first introduced in [21] (namely, HMCR, PAR and RSR), with the variable alphabet $\mathcal{F}$ sorted in terms of the estimated impact of its compounding relaxation rules in the fitness functions to be optimized. Likewise, each harmony produced by HS as a potential solution to the scheduling problem is divided in two parts, isolated from each other in regards to the application of the HS operators, but coupled together by its participation in both metric functions: 1) a P-sized vector of

[1] In fact there have been controversial discussions lately around the originality of the naïve HS algorithm in regards to its close resemblance to the more traditional $(\mu+1)$ Evolution Strategies. Having said this, foregoing algorithmic descriptions will use the HS terminology impartially with respect to the aforementioned controversy.

integer elements from the alphabet $\{1, \ldots, M_r\}$, with M_r denoting the number of Pareto-optimal relaxation rulesets previously produced for each query profile p by the relaxation module; and 2) a P-sized vector of real-valued variables from $\mathbb{R}[0, Z]$, with Z denoting the number of processing queues. Each of the variables within this second part represents both the profile-to-queue mapping $\lambda(p)$ and the relative ordering α_p by virtue of a Random Keys strategy [23]; if X_p denotes the p-th variable of this part, $\lambda(p) = \lfloor X_p \rfloor$ and $\alpha_p = X_p - \lfloor X_p \rfloor$. By sorting queues in terms of their processing capabilities $\{\tau_z\}_{z=1}^{Z}$ a more suitable relative queue ordering can be achieved for its processing through the vicinity-based PAR operator of HS.

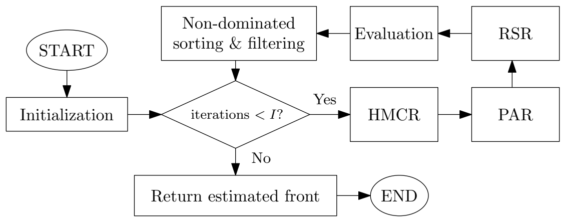


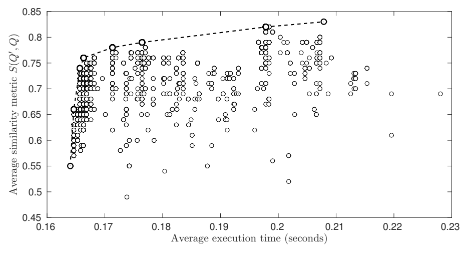
Figure 3: Flow diagram of the proposed bi-objective optimization algorithm.

To end with, Figure 3 schematically describes the 4 compounding steps of the utilized bi-objective HS solver:

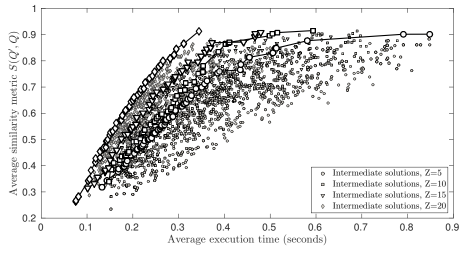
1. Initialization: the harmonies (potential solutions) of the population are filled with notes (values) drawn uniformly at random from the alphabets of their compounding instruments (variables).
2. Application of the operators: a new population of harmonies is produced by sequentially applying, to each note of the prevailing population, the stochastic HMCR, PAR and RSR operators defined in [21] based on probabilistic parameters $Pr(\text{HMCR})$, $Pr(\text{PAR})$ and $Pr(\text{RSR})$. For real-valued variables the PAR operator requires an additional bandwidth parameter $BW \in \mathbb{R}(0, \infty)$, such that the new value X_p^{new} for X_p given by the PAR operator is $X_p^{new} = X_p + BW \cdot \text{Uniform}(-1, 1)$.
3. Evaluation and update: fitness values for every newly produced potential solution are computed, concatenated to those from the previous iterations, sorted and filtered based on the well-known non-dominated sorting criterion utilized in other solvers (e.g. NSGA-II). The application of this criterion yields a φ-sized set of harmonies that are kept for the next iteration due to their higher Pareto optimality and wider Pareto span (crowding distance).
4. Termination: steps 2 and 3 are repeated for I iterations.

4 Experimental Evaluation

In this section we assess the performance of the two novel technical aspects of our system: the prediction of SPARQL query run-times (Subsection 2.1) and the bi-objective optimization algorithm (Section 3). To this end a distributed queue system based on Apache Kafka has been implemented. The SPARQL query and similarity computation module relies on the JENA framework, RDFLib, Wordnet Similarity (`code.google.com/p/ws4j/`) and SimMetrics (`sourceforge.net/projects/simmetrics/`), whereas the optimization and run-time prediction algorithms have been implemented in Python. The experiment discussed in what follows can be conceived as a SPARQL query rewriting scenario contextualized in the LOD semantic web. Specifically we select the so-called DBPSB benchmark, with DBpedia 3.5.1 as the RDF dataset. The training (70%), validation (25%), and test (5%) queries for the run-time prediction model are extracted from the 25 available DBPSB query templates in which RDF terms are assigned randomly from the DBpedia vocabulary, amounting to a total of 1260 queries. To measure its average execution time, each query is executed five times. This setup for the predictive model scores a 20-fold cross-validated average R^2 score of 0.977 with a Random Forest Classifier composed by 100 estimators, which is higher than the $R^2 = 0.94$ score reported in [5].



(a)



(b)

Figure 4: (a) Pareto front estimated by the relaxation module for a query profile $\mathbf{Q}_p$; (b) Pareto front produced by the bi-objective scheduler for $P = 10$ query profiles and $Z \in \{5, 10, 15, 20\}$ queues with varying processing capabilities.

We begin our discussion by Figure 4.a, which depicts the estimated Pareto front (in bold black) of the relaxation module corresponding to a hypothesized query profile $\mathbf{Q}_p$ composed by the entire set of 1260 DBPSB queries previously generated for the predictive model. Such compounding queries can be processed through a maximum of 10 relaxation operators from an overall alphabet of $R = 18$ possible relaxation rules. Values for the parameters of the HS solvers are $\varphi = 25$, $Pr(\text{HMCR}) = 0.5$, $Pr(\text{PAR}) = 0.1$, $Pr(\text{RSR}) = 0.01$ and $I = 100$ iterations, which have been optimized via a off-line grid search. This figure also includes the Pareto fronts estimated during the execution of the bi-objective algorithm marked in increasing levels of gray. As seen in the plot the derived HS-based algorithm succeeds at finding query relaxation policies that sacrifice the quality of the results returned by the queries for a lower execution time.

We finally proceed with the analysis of the results obtained by the proposed scheduler for $P = 10$ profiles as a function of the number of available processing queues $Z \in \{5, 10, 15, 20\}$. For simplicity profiles are generated by shuffling and splitting the set of DBPSB queries in 10 groups, but any other clustering scheme can be instead applied. For the sake of understandability of the results processing capabilities are forced to grow linearly with Z such that an increase of this parameter implies adding queues with lower τ_z to the simulation setup. Parameter values of the HS solver are as in the previous case, with $BW = 1$. As shown in Figure 4.b, as Z increases the Pareto fronts produced by the scheduler are better in the Pareto sense due to the availability of new queues with enhanced processing power. It is important to note that this trend also holds when $Z \leq P$, which evince that the proposed scheduler excels at relaxing profiles and allocating them to processing queues in scenarios with limited resources.

5 Concluding Remarks and Future Research Lines

This work has presented a novel approach to improve the performance of SPARQL endpoints by using an optimized run-queue system incorporating automatic query relaxation and intelligent queue scheduling functionalities. The proposed scheme hinges on a bi-objective optimization criterion, which permits the administrator of the endpoint to balance differently two conflicting objectives: the average run-time of queries incoming at the endpoint and the quality of such queries when relaxed with respect to their original versions. In order to efficiently solve this trade-off, a bi-objective solver based on the HS algorithm has been designed along with an encoding strategy aimed at handling mixed integer/real-valued variables. Besides this bi-objective formulation further novel aspects have been proposed in this manuscript: an improved feature set to predict the execution time of SPARQL queries and the use of semantic similarity to infer their quality. From a practical perspective the proposed system is useful in those cases when several users concurrently submit computationally expensive queries to SPARQL endpoints where other mechanism such as cost-based models are not effective due to the unavailability of data statistics.

There are several research directions to aim at from this study: to begin with the work can be extended to other benchmarks such as LUBM or BSBM, possibly incorporating real SPARQL endpoint logs in various domains such as UNIPROT (biological domain) or BNE (*Biblioteca Nacional de España*, bibliographic domain). There is also room for improvement in the optimization algorithm itself by leveraging other alternative bioinspired solvers. Other schemes for computing the semantic similarity can be explored such as the one proposed in [24].

References

1. Shadbolt, N., Hall, W., Berners-Lee, T.: The semantic web revisited. IEEE Intelligent Systems **21**(3) (2006) 96–101
2. Schmidt, M., Meier, M., Lausen, G.: Foundations of sparql query optimization. In: ACM International Conference on Database Theory. (2010) 4–33
3. Ganapathi, A., Kuno, H., Dayal, U., Wiener, J.L., Fox, A., Jordan, M., Patterson, D.: Predicting multiple metrics for queries: Better decisions enabled by machine learning. In: IEEE International Conference on Data Engineering. (2009) 592–603

4. Akdere, M., Çetintemel, U., Riondato, M., Upfal, E., Zdonik, S.B.: Learning-based query performance modeling and prediction. In: IEEE International Conference on Data Engineering. (2012) 390–401
5. Hasan, R., Gandon, F.: A machine learning approach to sparql query performance prediction. In: IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies. Volume 1. (2014) 266–273
6. Görlitz, O., Staab, S.: Splendid: Sparql endpoint federation exploiting void descriptions. COLD **782** (2011)
7. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: The Semantic Web. (2011) 601–616
8. Bernstein, A., Kiefer, C., Stocker, M.: OptARQ: A SPARQL optimization approach based on triple pattern selectivity estimation. Technical Report ifi-2007.03, University of Zurich (2007)
9. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.: Heuristics-based query optimisation for sparql. In: ACM International Conference on Extending Database Technology. (2012) 324–335
10. Gubichev, A., Neumann, T.: Exploiting the query structure for efficient join ordering in sparql queries. In: EDBT. (2014) 439–450
11. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: Sparql basic graph pattern optimization using selectivity estimation. In: ACM International Conference on World Wide Web. (2008) 595–604
12. Bicer, V., Tran, T., Gossen, A.: Relational kernel machines for learning from graph-structured rdf data. In: The Semantic Web. (2011) 47–62
13. Yamagata, Y., Fukuta, N.: An approach to dynamic query classification and approximation on an inference-enabled sparql endpoint. Journal of information processing **23**(6) (2015) 759–766
14. Maali, F., Hassan, I.A., Decker, S.: Scheduling for SPARQL endpoints. In: International Semantic Web Conference. (2014) 19–28
15. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.C.: Dbpedia sparql benchmark–performance assessment with real queries on real data. The Semantic Web (2011) 454–469
16. Robertson, S.: Understanding inverse document frequency: on theoretical arguments for idf. Journal of Documentation **60**(5) (2004) 503–520
17. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
18. Trillo, R., Gracia, J., Espinoza, M., Mena, E.: Discovering the semantics of user keywords. Journal of Universal Computer Science **13**(12) (2007) 1908–1935
19. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization **26**(6) (2004) 369–395
20. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation **8**(2) (2000) 173–195
21. Geem, Z.W., Kim, J.H., Loganathan, G.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2) (2001) 60–68
22. Manjarres, D., Landa-Torres, I., Gil-Lopez, S., Del Ser, J., Bilbao, M.N., Salcedo-Sanz, S., Geem, Z.W.: A survey on applications of the harmony search algorithm. Engineering Applications of Artificial Intelligence **26**(8) (2013) 1818–1831
23. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing **6**(2) (1994) 154–160
24. Pirró, G., Euzenat, J.: A feature and information theoretic framework for semantic similarity and relatedness. In: The Semantic Web. (2010) 615–630