Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

LEARNING ARBITRARY RDF DATASET ENRICHMENT GRAPHS USING PRE- & POSTCONDITION BROADCASTING

Leipzig, Juli 2019

vorgelegt von
Kevin Dreßler
Studiengang Informatik

Betreuender Hochschullehrer:
Prof. Dr. Axel-Cyrille Ngonga Ngomo

Zweitgutachter:
Prof. Dr. Martin Middendorf

ABSTRACT

The impact of Linked Data in applications on the World Wide Web as well as in businesses has been tremendous in the recent decade. Linked Data Integration is of paramount importance for this tendency to continue. Despite there being a great amount of literatue on lifting, interlinking and fusion of RDF Datasets, the field of RDF dataset enrichment has seen little attention. Specifically, there exist no accessible tools to power this step in the Linked Data Lifecycle.

This thesis attempts to address this gap by extending on a previous approach in RDF dataset enrichment to enable successful RDF dataset enrichment for a larger set of use cases. Since such tools require extensive configuration of their components in order to lead to satisfactory results, it is often not feasible for non experts to use them. Therefore, an efficient machine learning algorithm using novel improvements to enable the use of our tool by novice user is proposed. Evaluation suggests that our approach can successfully learn a larger class of RDF dataset enrichment specifications than the state of the art, using only a single training example.

ZUSAMMENFASSUNG

Linked Data konnten in der letzten Dekade in Anwendungen des World Wide Webs und der Wirtschaft große Erfolge erringen. Die Integration von Linked Data ist ein essentieller Bestandteil um diesen Erfolg fortzuführen. Obwohl das Liften, Verlinken und Fusionieren von RDF-Datensätzen in großem Umfang in der Literatur behandelt wird, hat das Gebiet der RDF-Datensatzanreicherung wenig Beachtung gefunden. Insbesondere existieren keine zugänglichen Werkzeuge, um diesen Schritt im Linked Data Lifecycle zu unterstützen.

Diese Arbeit versucht, eben jene Lücke zu schließen, indem sie einen bestehenden Ansatz der RDF-Datensatzanreicherung mit dem Ziel erweitert, die RDF-Datensatzanreicherung in einer größeren Klasse von Anwendungsfällen zu ermöglichen. Da solche Werkzeuge eine umfangreiche Konfiguration ihrer Komponenten erfordern, um zu zufriedenstellenden Resultaten zu führen, ist es für Laien häufig zu schwer, diese zu verwenden. Daher wird auch ein effizienter Algorithmus für maschinelles Lernen entwickelt, der neuartige Verbesserungen verwendet, um die Verwendung unseres Werkzeuges für unerfahrene Benutzer zu ermöglichen. Die Evaluation legt nahe, dass dieser Ansatz in der Lage ist, eine größere Klasse von RDF-Datensatzanreicherungsspezifikationen erfolgreich zu erlernen als bestehende vergleichbare Ansätze. Zudem benötigt er dafür nur ein einziges Trainingsbeispiel.

If you really want to escape
the things that harass you,
what you're needing is not
to be in a different place
but to be a different person.

— Lucious Annaeus Seneca

ACKNOWLEDGMENTS

I would like to thank everyone who supported me through the past three years.

First and foremost, thanks go to my supervisor Dr. Mohamed Ahmed Sherif for his professional as well as personal support, patience and believe in my abilities.

I also thank Prof. Dr. Axel-Cyrille Ngonga Ngomo for introducing me to research years ago, giving me the benefit of the doubt and enabling me to have my first journal paper before even handing in my bachelors thesis.

Further thanks goes to everyone in the DICE research group that answered my questions or helped me getting around organizational or software issues when running my experiments.

Last but not least, I most genuinely thank my parents for their unconditional love and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

RDF	Resource Description Framework
SHACL	Shapes Constraint Language
W3C	World Wide Web Consortium
DEER	RDF Dataset Enrichment Framework
DEER2	RDF Dataset Enrichment Framework 2
DAG	Directed Acyclic Graph
GA	Genetic Algorithm
GP	Genetic Programming
PRNG	Pseudo Random Number Generator
MEP	Multi Expression Programming
LOD	Linked Open Data
HTTP	Hypertext Transport Protocol
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
IRI	International Resource Identifier
DL	Description Logic
XSD	XML Schema Definition
RDFS	RDF Schema
OWL	Web Ontology Language
SCBD	Simplified Concise Bounded Description
JDK	Java Development Kit
PF4J	Plugin Framework for Java
NER	Named Entity Recognition

FOX	Federated knOwledge eXtraction Framework
LIMES	Link Discovery Framework for Metric Spaces

1 INTRODUCTION

1.1 MOTIVATION

The Web of Data, also known as the Semantic Web, is growing from year to year[1], giving leeway to a vast amount of applications to harvest the knowledge cotained within the Linked Open Data (LOD) Cloud.

With growing numbers of datasets, we also see a growing number of domains being represented in the LOD Cloud, leading to the need for a growing number of novel ontologies and vocabularies. While some of these ontologies and vocabularies are well known and standardized by e.g. the World Wide Web Consortium (W3C), most are distributed over the web, hard to find and potentially model the same domain, therefore being redundant vocabularies.

This leads to a retrieval problem of ontologies and vocabularies for dataset curators which the term „Ontology Dowsing"[a] was coined[2] in order to capture the problematic unscientific guessing nature which is most common today when trying to locate a suitable ontology or vocabulary for modeling data in the Resource Description Framework (RDF). As a result, applications that comsume the Web of Data also often define their own specific vocabularies as it is not feasible to support, let alone be aware of, all potentially applicable ontologies and vocabularies for the specific application domain.

[a] *Dowsing is the practice of searching for ground water or metal ores using a Y-shaped rod.*

Moreover, limited resources on clients mean that the large datasets in the LOD Cloud, which are often schemaless due to the underlying Open World Assumption, have to be filtered and distilled before they can be used by applications.

We refer to the processes needed to solve the above mentioned problems as RDF dataset enrichment. RDF dataset enrichment is a quintessential part of Linked Data Integration, which also consists of the *linkage* and *fusion* of RDF datasets.

While there has been a lot of work on the automatic linkage and fusion, RDF dataset enrichment has been paid little attention

1 https://lod-cloud.net
2 https://www.w3.org/wiki/Ontology_Dowsing

to, despite there being a critical need for better solutions in order to truly enable Semantic Web powered applications.

1.2 OBJECTIVES

In this thesis we address this shortcoming by extending RDF Dataset Enrichment Framework (DEER) [51], the only existing approach to automated RDF dataset enrichment we are aware of. While DEER implements a fixed set of so called enrichment functions and only allows chaining them lineraly in a pipeline, we argue that this approach is too limited to of use to the very specialized needs of real-world RDF dataset enrichment. Therefore, our first objective will be to build an extension of DEER, called DEER2, which should be (1) highly modular, meaning that the framework should be easily extendable by third party developers in order to create specialized enrichment functions and (2) allow to represent the enrichment process as a Directed Acyclic Graph (DAG) of modular operations. These extensions should provide enough flexibility for dataset curators as well as application developers to use DEER2 in real world RDF dataset enrichment workflows.

The original DEER publications main contribution was the introduction of a refinement operator-based learning algorithm which enabled novice users to define adequate RDF dataset enrichment workflows. As highly modular applications in general require a lot of manual configuration of their components and therefore presume expert knowledge to precisely define how the modules operate and interact with each other, a machine learning based approach to automatic configuration will be the second and main objective of this paper.

Since introducing DAG-shaped RDF dataset enrichment workflows in DEER2, the complexity of the learning problem is greatly increased in comparison to DEER. We will therefore base our approach on Genetic Programming (GP) instead of refinement operators, since GP is known for its ability to find good solutions for hard symbolic regression problems, albeit at the cost of being non-deterministic.

1.3 DESIGN GOALS AND RESEACH QUESTIONS

We set the following goals for the design of DEER2:

(G1) DEER2 should be highly modular

(G2) DEER2 should represent RDF dataset enrichment workflows efficiently as DAGs

(G3) DEER2 should include an optimized, GP based learning algorithm for automatic configuration of RDF dataset enrichment workflows

(G4) DEER2 should improve on all of the identified shortcomings of DEER.

In order to measure the success of our learning approach we will aim to answer the following research questions:

(Q1) What is the optimal set of hyperparameters?

(Q2) How does our approach perform on real world datasets?

1.4 STRUCTURE OF THIS THESIS

The remainder of this thesis is structured as follows: In Chapter 2 we explore the State of the Art for fields relevant to this work and introduce some of the basic concepts required to understand this thesis. After that, we present our approach DEER2 in Chapter 3. We evaluate our approach and answer the posed research questions in Chapter 4. Finally, we conclude in Chapter 5.

2 BACKGROUND

2.1 GENETIC ALGORITHMS

A Genetic Algorithm (GA) is a non-deterministic population-based global search metaheuristic that reformulates optimization problems in a vocabulary that makes heavy usage of metaphors on biological evolution, thus also being called a nature-inspired metaheuristic.

Genetic Algorithms typically represent solutions to a problem as bit vectors which correspond to the *genotype* in biological evolution. These genotypes encode a set of parameters which are passed to a so called *fitness function* which promotes a survival of the fittest and is thus the target of minimization or maximization in the underlying optimization problem.

A fixed size population of genotypes, in this context also called individuals is initially generated at random using a Pseudo Random Number Generator (PRNG). This population of genotypes, parameterized by the population size (μ), is then iteratively evolved where the iterations are typically referred to as *generations*. In each generation, a subset of the population is selected to join a so called recombination pool, which is parameterized per the *recombination pool size* (λ). The selection of individuals is carried out using a so called selection operator. An important property of a selection operator is the *selective pressure*, which is informally described as the emphasis of selection on the best individual, where a high (small) selective pressure means a strong (weak) emphasis. We will discuss two of the most prominent selection operators further down in this text.

The individuals selected for recombination are then passed in pairs to a crossover operator, which generates a pair of childs by swapping the bit vectors representing the individuals at a given crossover point[b]. The childs are then inserted into the next generations population together with the survivors, which are selected from the current population also using a selection operator.

A mutation operator is then applied to each individual in the next generation with a certain *mutation probability* (σ). The classic mutation operator mutates each selected individual by chang-

[b] *Note that the literature contains a vast number of variations on the crossover operator such as multi-point crossover*[54] *and uniform crossover* [58]*, however they are of no particular interest to this work. There has also been a debate over the usefulness of crossover operators which resulted in several extensive theoretical analyses with minimalistic toy problems with the result that they „can provably be useful"*[13, 53].

ing the bit value of a given bit in its bit vector representation with a probability denoted by the *mutation rate* (ρ), i.e. a mutation rate of $\rho = \frac{1}{2}$ means that on average, half of the bits are changed to their opposing state.

These steps of selection, crossover and mutation are repeated until a certain termination criterion holds, which could include a maximum number of generations, a target fitness value or a test of population convergence.

Note that in the sense as defined above, Genetic Algorithms were reportedly[33] first conceived by John Holland in 1960 as an extension of general $(\mu + \lambda)$ Evolutionary Algorithms, the novelty being the introduction of crossover and selection operators whereas previous iterations of Evolutionary Algorithms only made use of mutation operators.

SELECTION OPERATORS While the literature contains a great number of selection operators, for sake of brevity we will only introduce the two selection operators used in the remainder of this text, namely the Elitist Selection[33] and the Tournament Selection[32].

The Elitist Selection is a simple selection procedure which includes the N best performing individuals in the next generation. It is typically used in conjunction with another selection operator, as it would just lead to a stochastic hill climbing algorithm when used as only selection operator and there are much more efficient hill climbing algorithms, such as [59] for discrete or [3] for continuous optimization problems.

The Tournament Selection is one of the most established selection operators, as it has been proven able to adjust the selective pressure independently from the population size and fitness function scaling[22]. It is parameterized by the tournament size k and the selection probability p. Initially, k individuals from the given population are randomly selected to enter a tournament. Then, the ith best individual in the tournament w.r.t. the fitness function is selected to win the tournament with the probability $p(1-p)^{i-1}$. This means that setting the probablity to $p = 1$ will result in a deterministic behaviour while setting $k = 1$ results in random selection.

2.1.1 *Genetic Programming*

Genetic Programming (GP) is a subfield of Genetic Algorithms which originated from the application of a GA in order to evolve computer programs in 1988 by John Koza[29]. While traditional GAs are commonly used for numerical optimization and search problems, GP can be used for symbolic regression and classification.

In Genetic Programming, programs are usually encoded as trees of operations and terminals, but other encodings have also been proposed. For example, Graff et al.[23] use DAGs to encode python programs, and Kvasnièka and Pospíchal[30] introduced a condensed encoding of DAGs, dubbed „Column Tables", which we will adapt and use in this work (see Section 3.1.2).

2.1.2 *Multi Expresssion Programming*

Multi Expression Programming (MEP), originally introduced by Ferreira[18] as „Gene Expression Programming" denotes a special kind of Genetic Programming where the genotype of an individual represents multiple solutions or subsolutions that has gained traction in recent years. Only the best amongst these multiple solutions is then used as the effective solution, also called the *phenotype* of the individual. This technique is used in problems where it does incur no additional cost to keep track of the fitness of subsolutions, such as is generally the case with the evaluation of a program represented as a tree: it can be easily seen that when evaluating such a tree-shaped program, one has to visit each node anyway in order to compute its result.

It can be used to evolve programs with the same complexity as traditional GP but much more efficient and excels in multiple-output problems. To this end it has been successfully applied to the TSP[43], data prediction[62], software effort estimation[2], on-the-fly hyperparameter optimization for Evolutionary Algorithms[44] and digital curcuit design[42].

2.1.3 *Semantic Genetic Operators*

Semantic Genetic Operators are a special kinds of crossover and mutation operators which take into account the semantics of a genotype rather than just discerning it as a bit vector without any further interpretation. They are therefore aware of the specific genotype encoding and manipulate the individuals w.r.t. the solution space rather than the encoding space.

Their recent successful application to a number of problems[11, 20, 45, 57] indicates that taking semantics into account is a promising novel direction for GP.

Table 2.1: The Five Stars of Linked Open Data

Stars	Requirements	Example
☆☆☆☆★	Published with an Open License	PDF, PNG, TXT
☆☆☆★★	Machine-Readable Structured Data	XLS, ODF
☆☆★★★	Non-proprietary File Format	XML, CSV, JSON
☆★★★★	Use URIs to identify things	RDF
★★★★★	Linked to other data	RDF

2.2 LINKED DATA

Linked Data[5, 7, 8] is a term coined by Tim Berners-Lee[5] in 2006. It refers to data available on the Web that is (1) structured, (2) uses URIs as identifiers and (3) is interlinked with other data on the Web. Linked Data is enabled by the use of standard Web technologies such as the Hypertext Transport Protocol (HTTP), the RDF and the SPARQL Protocol and RDF Query Language (SPARQL). It is closely related to the term Semantic Web[6, 48], which describes a Web of Data which, in opposition to the human-readable Web of Documents is specifically engineered to be machine-readable. The Semantic Web promotes a high interconnectedness of datasets by interlinking them with eath other, thereby enabling the ability to consume data from multiple datasets at once using semantic queries, e.g. using the SPARQL protocol.

When considering Linked Data published under open licenses, we speak of Linked Open Data (LOD). A quality scheme for LOD, namely the five stars of LOD, was given by Tim Berners-Lee in 2010[5] and is summarized in Table 2.1.

2.2.1 *The Resource Description Framework*

The Resource Description Framework (RDF) is a „standard model for data interchange on the Web"[24]. RDF models data in an abstract syntax which is defined in terms of an abstract graph-based data model. The nodes in this graph-based model belong to one of three atomic base types: *IRI resources*, *literals* and *blank nodes*. IRI resources and blank nodes denote a particular subject in the world, while literals only contain values which are subject to further interpretation. As blank nodes lack the notion of a global identifier, they serve as existential variables in the data model.

An RDF graph is a collection of *statements*, also called *triples*, which basically correspond to simple sentences in natural language. Each statement therefore consists of a subject, a predicate and an object. Every subject is either a IRI resource or a blank node, all predicates are IRI resources and objects can be either of the three basic types.

In the following, we will give a formal specification of an RDF dataset on which we will rely in the remainder of this work.

RDF DATASET: Let $\mathcal{R}$ be the set of all RDF IRI resources, $\mathcal{B}$ be the set of all blank nodes and $\mathcal{L}$ be the set of all literals.

We call a set $D := \{(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})\}$ of triples an **RDF dataset**.

Note that for sake of simplicity, we will not distinguish between an RDF dataset and its abstract graph and therefore we from now on resort to referring to both as RDF datasets.

2.2.2 *Vocabularies, Ontologies and Shapes*

While RDF specifies a basic set of vocabulary, it is not very expressive on its own and offers no mechanism to define vocabularies or data schemes.

In order to make statements about the nature of the data that is expressed, other standards which extend on the RDF semantics have to be used.

This is a concious decision as it allows to use multiple semantics, i. e. Description Logics (DLs) of different expressiveness to be used in conjunction with the RDF data model.

The two main standards that are used to that end are RDF Schema (RDFS) and Web Ontology Language (OWL).

Contrary to the typical notion of „schema", as for example in XML Schema Definition (XSD), RDFS is not used to define a closed schema in order to force adherence to it. Rather, it provides a set of standard predicates and resources that enable composing a simple ontology with mostly statements about class and predicate hierarchies, instances of classes and relationships between classes and predicates. It thereby enables the use of efficient reasoners for basic inference as well as more standardized tools for applications to discover and reason about what the data is about and how it is potentially shaped. The „potentially" in the last sentence is owed to the fact that RDF as well as RDFS semantics follow the *open-world assumption*, i. e. the absence of a fact is not enough to prove its negation holds true.

The Web Ontology Language is a standard which allows for much more expressiveness compared to RDFS. There are three subsets of OWL: OWL Lite $\subset$ OWL DL $\subset$ OWL Lite. Their expressiveness increases from OWL Lite to OWL Full, which naturally comes at the cost of availability of efficient or even effective reasoning algorithms.

Another standard that can be used to accompany the RDF data model is the Shapes Constraint Language (SHACL). Contrary to the previously introduced standards, SHACL semantics follow the *Closed World Assumption* which on one hand decreases its effective usability alongside RDFS and OWL but on the other enables an important application that can not be realized with the latter two, namely that of schema validation.

2.3 LINKED DATA INTEGRATION

Generally speaking, Linked Data Integration denotes a process of processing data whose goal it is to return one or many RDF dataset(s) for a specific use case, e. g. publication to the LOD Cloud or preparation for further usage by specific applications.

The input to Linked Data Integration can, but not necessarily must be Linked Data itself. Typical processing steps in Linked Data Integration include, but are not limited to (1) lifting, (2) linking, (3) fusion, (4) enrichment. Any one step could be left out, depending on the use case.

In the following, we will define each of these processing steps and give a narrow overview of the State of the Art.

2.3.1 *Lifting*

In the Linked Data community, lifting denotes the act of materializing RDF datasets from sources of non-linked data. These sources may be relational databases, proprietary file formats, measurements from sensor networks and any other structured or unstructured data that is not Linked Data.

The term is closely related to RDF mapping, whereas in mapping the focus is often on ad-hoc transformation of an evolving non-linked data source in contrast to the materializing approach in lifting

After lifting, the data is expected to be at least four-star data according to the scheme presented in Table 2.1.

In [56], the authors attempt to lift event data from the interaction with Web pages. Bizer et al.[9] introduced a framework for discovering mappings to relational data sources on the Web. In [34], [4], [47] and [19], the authors proposed approaches for lifting metadata on data portals, entries in the Docker registry, entries in file systems and whole spreadsheets, respectively. The approach in [25] considers a framework for continuously lifting sensor data to Linked Data in order to exploit the semantic interconnectedness of the sensor measurings for situation awareness.

2.3.2 *Linking*

The linking of RDF datasets, sometimes also called instance matching, amounts to finding a mapping M between two RDF datasets, commonly called the source dataset S and the target dataset T, such that the pairs $(s, t) \in M$ identified by the mapping abide to a given relation r.

This field is closely related to the instance matching and deduplication in databases. The main challenge here is that due to the very large number of instances in the Web of Data, the quadratic run time complexity of a brute force search can not be accepted.

To this end, it is necessary to derive algorithms that discard large numbers of potential pairs, preferably without losing the completeness or correctness property, i. e. all pairs (s, t) that abide by $r(s, t)$ should be in M and no pair in M should not abide by r.

A plethora of such approaches has been developed for textual data[15, 17, 27, 28, 60, 61], geospatial data[16, 49, 52] as well as temporal data[21].

Another challenge lies in the automatic configuration of so called link discovery tools.

A number of different approaches have been proposed.

The approaches in [26, 36, 38] use Genetic Algorithm in an unsupervised, an active learning setting and both, respectively. In [37], the authors use an altered grid search for unsupervised learning and finally [50], the approach outperforming all of the previously mentioned, uses refinement operators for unsupervised as well as supervised learning.

2.3.3 *Fusion*

The fusion of RDF datasets is a task in which two interlinked datasets that contain information about the same set of instances are merged to produce a joint dataset. To this end, an ontology mapping has to be generated in a first step. Subsequently, the fusion is carried out following a set of so called fusion rules that are either predefined by the system, the user or dynamically learned.

In [39–41], the authors use a fusion architecture called *Kno-Fuss* which handles the end-to-end fusion of RDF datasets. The apporach in [31] combines quality assessment and fusion of RDF datasets. A probabilistic framework is applied to the fusion problem in[14] and [10] is an approach to automatic learning of fusion rules in the setting of cross-language fusion from the Wikipedia corpus.

2.3.4 *Enrichment*

Enrichment is the least well-defined step in the Linked Data Integration lifecycle. Informally, enrichment corresponds to applying a set of modifications to a given RDF dataset such that it abides by certain use case-specific predefined criteria. We will call this set of modifications, i. e. addition and deletion of triples from the source dataset, an enrichment workflow in the remainder of this text. Examples of such modifications include the dereferencing of information from other datasets, the application of quality assurance methods to the dataset, the conformation of the used vocabulary or the filtering of its contents.

In practice this means, that RDF dataset enrichment is a problem that greatly differs from use case to use case. Notable applications of enrichment in the literature include [1], which considers the automatic extraction of interests from Twitter posts in order to enrich user profiles on the Social Web and [12], which applies a ranking method to the Youtube tag space in order to enrich datasets on the LOD Cloud with links to Youtube videos.

However, to our best knowledge, the only effort directed towards a general framework of RDF dataset enrichment together with an automatic learning approach can be found in [51], where the authors propose a framework and learning algorithm called DEER. We will thus have a deeper analysis of DEER in the following section, where we will try to identify potential shortcomings of the framework in order to underline the necessity for our contribution in this work, DEER2.

2.3.5 *DEER*

As we chose to build our approach based on the existing Open Source software DEER, we are going to highlight specific areas in the original paper resp. software, where we see chances of substantial improvement.

On Restrictiveness of Sequential Chaining

The RDF Dataset Enrichment Framework models RDF dataset enrichment workflows as ordered sequences of enrichment functions. These enrichment functions take as input exactly one RDF dataset and return an altered version of it by virtue of addition and deletion of triples. It is argued that formally, any modification on a given dataset can be represented as a set of additions and deletions.

While this is certainly true, the application of two independent enrichment functions, that formally have no knowledge of each other, could be problematic as one function might delete triples which the other relies on in order to successfully apply the desired enrichment.

We therefore suggest replacing the simple sequential pipelining approach with a more sophisticated one based on arranging the enrichment functions in a DAG. In order to better distinguish our approach from the former, we will call atomic enrichment functions which allow for multiple inputs and multiple outputs *enrichment operators*.

On Modularity and Attractiveness of the System

The framework consists of five atomic enrichment functions that can be combined to form so called enrichment pipelines of arbitrary size. The language used implies that these enrichment functions are only examples and potentially more could be implemented. However, no implicit facility for easy extension of the framework is given and exploration of the code base suggests that the ability to implement new operators is restricted in the sense that it would require recompilation of the whole application.

In order to improve the real-world suitability of our contribution DEER2, we therefore will aim to attract developer interest by providing plugin facilities with low barriers to entry.

On Appropriateness of Fitness Function

The refinement operator-based learning approach in DEER is defined as the task of finding an appropriate enrichment pipeline given an input RDF dataset K and a target RDF dataset K'.

As the enrichment functions typically require manual configuration, the problem is twofold: it is not only the correct sequence of enrichment functions that is subject to the learning process but also the correct configuration of the enrichment functions that are part of the pipeline.

This is done by a combination of iterative construction of the enrichment pipeline and self-configuration of enrichment functions at each iteration.

That is, in order to decide which enrichment function should be next added to the current pipeline, each available enrichment function is self configured using the supplied training data. It is then applied to the result of the previous enrichment function and its result evaluated using a fitness function over the provided training data.

Note that one of the results of the evaluation of DEER was that it is sufficient to provide only a single training example. While it seems unusual for a learning method to rely on a single training example, it is a valid assertment in this specific case, as enrichment can be assumed to deal with more or less regularly shaped RDF datasets.

The dataset produced by the intermediate pipeline in each iteration is denoted as K_i with i being the position of the enrichment function producing K_i in the pipline.

The fitness function used is the F_1-score[46] over the set of triples in K_i and K'.

We argue that this choice of fitness function is not optimal in a modular setting where solutions can be expected to be constructed in an iterative manner. We consider the following short example to illustrate the problem.

Let us assume without loss of generality that there exists one triple $t = (s, p, o) \in K$ for which we can intuitively identify at least one corresponding triple $t = (s', p', o') \in K'$. Then a first enrichment function might only alter our input triples predicates, yielding the intermediate result $t_1 = (s, p', o) \in K_1$. Now another enrichment function might alter the subject, yielding $t_1 = (s', p', o) \in K_2$ before we apply the final enrichment function and get to the desired result $t_3 = t' = (s', p', o') \in K_3$. While intuitively there are clearly improvements in each intermediate result w.r.t. our target triple t', a fitness function only measuring the correspondence between whole triples will not be able to detect these improvements.

We will therefore use an evenly weighted linear combination of the F_1-scores between all the subjects, all the predicates, all the objects and additionally all the whole triples in our input and target datasets.

On Information Masking

A more subtle problem with the refinement operator-based approach in DEER lies in the combination of deterministic learning and self configuration. By assuming that the training example contains the necessary information for learning the pipeline, which is a sane assumption to make for regularly shaped input datasets, it is also implicitly assumed that it contains enough information for the *self-configuration* of *all* the enrichment functions.

As a trivial counterexample, consider an enrichment pipeline where we filter out all triples that contain a certain predicate p_f using the enrichment function e_3. This means that no triple (s, p_f, o) will be present in the target training data.

Let us assume that some triple (s', p', o') present in the target training data were constructed by another enrichment function e_2 from a triple $t_{masked} = (s, p_f, o)$ that was previously constructed by yet another enrichment function e_1.

Without the triple t_{masked} in the training data, e_1 might not be able to determine its own configuration and therefore will never get picked. In the end, our pipeline will never get learned using the deterministic bottom-up approach in DEER. We call the underlying problem information masking.

As a result of this analysis, we will use a non-deterministic approach based on GP in DEER2. We will complement this with an idea dubbed pre- and postcondition broadcasting which is developed in order to increase the chance of successful self configuration in the light of information masking.

2.4 ADDITIONAL PRELIMINARY DEFINITIONS

In this section we will introduce some additional definitions required to understand the remainder of this work which did not fit into one of the previous sections.

2.4.1 *Simplified Concise Bounded Descriptions*

Let D be an RDF dataset and $r \in D$ be an IRI resource in that dataset.

We define the **neighbourhood** $neighbours(r)$ of r in D as the set of triples that contain r as their subject. Formally, $neighbours(r) := \{(s,p,o) \in D \mid s \equiv r\}$.

Let $k \in \mathbb{N} > 0$. The Simplified Concise Bounded Description (SCBD) $SCBD(r,k)$ of a resource r in D is then recursively defined per Equation 2.1 where $SCBD(r,0) := \emptyset$ and $SCBD(r,1) := neighbours(r)$.

$$SCBD(r,k) := SCBD(r,k-1) \cup \left(\bigcup_{(s,p,o) \in SCBD(r,k-1)} neighbours(o) \right) \tag{2.1}$$

2.4.2 *The F_β-Score*

The F_β-score is a measure of a tests accuracy, or more abstract, a measure of correspondence between two sets. Let A, B be two sets, where we call A the evaluated set and B the reference set. We define the precision and the recall of A w.r.t. B per Equation 2.2 and Equation 2.3, respectively. The F_β-score is then defined as the weighted harmonic mean of recall and precision as per Equation 2.4.

$$precision(A,B) = |A \cap B| / |A| \tag{2.2}$$

$$recall(A,B) = |A \cap B| / |B| \tag{2.3}$$

$$F_\beta(A,B) = (1+\beta^2) \cdot \frac{precision(A,B) \cdot recall(A,B)}{(\beta^2 \cdot precision(A,B)) + recall(A,B)} \tag{2.4}$$

2.4.3 *Directed Acyclic Graphs*

A directed graph is a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ denotes a finite set, called vertices, and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ denotes a set of 2-tuples, called edges. An entry $(u,v) \in \mathbf{E}$ represents a directed edge from u to v.

A sequence of edges $((u_1, v_1), \dots, (u_n, v_n)) \in \mathbf{E}^n$ for which $v_k = u_{k+1}\ \forall 1 \leq k \leq n-1$ holds, is called a walk of length n

from u_1 to v_n in $\mathbf{G}$.

We call a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ Directed Acyclic Graph (DAG) iff:

1. $\forall (u, v) \in \mathbf{E} : u \neq v$ and
2. there exists no walk from v to itself

Note that the second criterion is equivalent to the the demand that all walks in $\mathbf{G}$ are of finite length.

3 APPROACH

After the analysis of the DEER framework, which inspired this work, we now have a clear understanding of the design goals according to which we shall now develop our approach. The presentation of our approach in this chapter is structured into three parts. In the first part we will develop a formal specification of our approach. The second part contains implementation details about the system we implemented following our formal specification. Finally, in the third part, we describe the design of a GP-based learning algorithm based on our previous formal specification.

3.1 FORMAL SPECIFICATION

3.1.1 *Dataset Operator*

Let $\mathcal{D}$ be the set of all RDF datasets. A function

$$\begin{aligned} \mathbb{O}_{(n,m)} &: \mathcal{D}^n \times \mathcal{D} \to \mathcal{D}^m \\ \left(\left(D_1^{(\text{in})}, \dots, D_n^{(\text{in})}\right), P\right) &\mapsto \left(D_1^{(\text{out})}, \dots, D_m^{(\text{out})}\right) \end{aligned} \tag{3.1}$$

is called a **dataset operator**. We call n the in-degree and m the out-degree of $\mathbb{O}_{(n,m)}$ and will resort to writing just $\mathbb{O}$ when n, m are clear or if the lack of their specification will incur no loss of generality. The set of all dataset operators is denoted as $\mathbb{O}$

Informally, a dataset operator takes a parameter datset and n input datasets as arguments and returns m output datasets. We specify the following naming scheme for dataset operators:

- $\mathbb{O}_{(0,1)}$ is called a **dataset emitter**,
- $\mathbb{O}_{(1,0)}$ is called a **dataset acceptor**,
- $\mathbb{O}_{(n,m)}$ is called an **enrichment operator** for $n, m > 0$ and
- $\mathbb{O}_{(n,1)}$ is called a **confluent enrichment operator**.

Moreover, a function

$$\begin{aligned} \mathbb{P}_{(n,m)} &: \mathcal{D}^n \to \mathcal{D}^m \\ \left(D_1^{(\text{in})}, \dots, D_n^{(\text{in})}\right) &\mapsto \left(D_1^{(\text{out})}, \dots, D_m^{(\text{out})}\right) \end{aligned} \tag{3.2}$$

is called a **parameterized dataset operator**. We denote the set of all parameterized dataset operators as $\mathbb{P}$.

We define a *configuration function*

$$\begin{aligned} \mathbb{c}\colon \mathbb{O} \times \mathcal{D} &\to \mathbb{P} \\ \left(\mathbb{o}_{(n,m)}, P\right) &\mapsto \mathbb{p}_{(n,m)} \end{aligned} \tag{3.3}$$

which takes a given dataset operator $\mathbb{o}$ together with a parameter dataset P and returns a parameterized dataset operator $\mathbb{p}$, which operates under the same semantics as $\mathbb{o}$ if it receives P as its last argument.

$$\begin{aligned} \mathbb{O}_{(n,m)}\colon \mathcal{D}^n \times \mathcal{D} &\to \mathcal{D}^m \\ \left(\left(D_1^{(\text{in})}, \dots, D_n^{(\text{in})}\right), P\right) &\mapsto \left(D_1^{(\text{out})}, \dots, D_m^{(\text{out})}\right) \end{aligned}$$

3.1.2 *Enrichment Graph*

$$\mathbf{V}_r := \{v \in \mathbf{V} \mid \forall u \in \mathbf{V} (u, v) \notin \mathbf{E}\} \tag{3.4}$$
$$\mathbf{V}_l := \{v \in \mathbf{V} \mid \forall u \in \mathbf{V} (v, u) \notin \mathbf{E}\} \tag{3.5}$$
$$\mathbf{V}_i := \mathbf{V} \backslash (\mathbf{V}_r \cup \mathbf{V}_l) \tag{3.6}$$

An enrichment graph $G = (\mathbf{V}, \mathbf{E}, \mathbf{L}, \Phi, \Psi)$ is a Directed Acyclic Labeled Multigraph. We call the subsets of vertices per Equations 3.4 - 3.6 root vertices, leaf vertices and intermediate vertices, respectively.

$$v \in \mathbf{V}_r \land \mathbb{o}_{(n,m)} = \Phi(v) \to n = 0 \land m = 1 \tag{3.7}$$
$$v \in \mathbf{V}_l \land \mathbb{o}_{(n,m)} = \Phi(v) \to n = 1 \land m = 0 \tag{3.8}$$
$$v \in \mathbf{V}_i \land \mathbb{o}_{(n,m)} = \Phi(v) \to n > 0 \land m > 0 \tag{3.9}$$

The mapping $\Phi\colon \mathbf{V} \to \mathbb{O}$ maps vertices to dataset operators, while $\Psi\colon \mathbf{V} \to \mathcal{D}$ maps vertices to configuration datasets The set of axioms defined per Equations 3.7 - 3.9 hold for enrichment graphs. Informally, they mean that all root vertices represent dataset emitters, all leaf vertices represent dataset acceptors and all intermediate vertices represent enrichment operators w.r.t. Φ.

$$\begin{aligned} \mathbf{L}\colon E &\to 2^{(\mathbb{N} \times \mathbb{N})} \\ e = (u, v) &\mapsto \{(i_1, j_1), \dots, (i_n, j_n)\} \end{aligned} \tag{3.10}$$

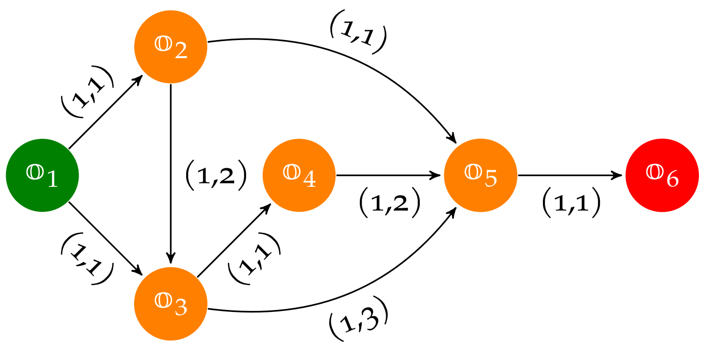


Figure 3.1: Example of an inherently confluent enrichment graph. Dataset emitters are colored in green, enrichment nodes are colored in orange and

An edge $(u, v) \in \mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ represents flow of data. Consider the definitions given in Equation 3.10. The label function $\mathbf{L}$ induces a mapping from the components of images of $\Phi(u)$ to the arguments of $\Phi(v)$. That is, for $e = (u, v)$, an entry of the label multiset $l \in \mathbf{L}(e) = (i, j)$ establishes a flow of data from the ith component in the image of $\Phi(u)$ to the jth argument of $\Phi(v)$.

$$\begin{aligned} \forall e_1, e_2 \in \mathbf{E}\colon e_1 = (u_1, v_1) \wedge e_2 = (u_2, v_2) \wedge v_1 = v_2 \\ \rightarrow \forall l_1 \in \mathbf{L}(e_1) \forall l_2 \in \mathbf{L}(e_2)\colon l_1 = (i_1, j_1) \wedge l_2 = (i_2, j_2) \wedge j_1 \neq j_2 \end{aligned} \tag{3.11}$$

In order to be deemed valid, the label function has to abide by the criterion defined in Equation 3.11. Informally, this criterion forbids multiple mappings to the same argument of the same dataset operator.

We present a categorization of enrichment graphs as follows. Let $G = (\mathbf{V}, \mathbf{E}, \mathbf{L}, \Phi, \Psi)$ be an enrichment graph.

- If $|\mathbf{V}_r| = |\mathbf{V}_l| = 1$ and for all vertices $u, v \in \mathbf{V}$ with $u \neq v$ there exists only a single walk from u to v, then G is called a linear enrichment graph.
- If $|\mathbf{V}_r| = |\mathbf{V}_l| = 1$ and there is a pair of vertices $u, v \in \mathbf{V}$, $u \neq v$ for which there there exist multiple walks from u to v, then G is called a semi-linear enrichment graph.
- If $|\mathbf{V}_r| > 1 \wedge |\mathbf{V}_l| = 1$, then G is called a confluent enrichment graph.
- Otherwise, G is called a general enrichment graph.

We call a confluent enrichment graph in which only confluent enrichment operators are allowed, i.e. $\forall v \in \mathbf{V}_i \colon \mathbb{O}_{(n,m)} = \Phi(v) \to m = 1$, an inherently confluent enrichment graph. An example of such an enrichment graph is given in Figure 3.1.

In order to evaluate an enrichment graph $G = (\mathbf{V}, \mathbf{E}, \mathbf{L}, \Phi, \Psi)$, we start with obtaining the RDF datasets as output by the dataset emitters in $\Phi(\mathbf{V}_r)$. These datasets then flow throught the graph as specified by the semantics we associated with the edge set $\mathbf{E}$ and the label multiset $\mathbf{L}$ above. Whenever a dataset operator $\mathbb{O}_{(n,m)} = \Phi(v), v \in \mathbf{V}_i$ has received all its n input datasets, it is evaluated using $\Psi(v)$ as its last argument. The flow through the graph continues until eventually all vertices have been evaluated.

3.1.3 *Enrichment Table*

An enrichment table, in the following denoted by $\mathbb{T}$, is a condensed linear representation for inherently confluent enrichment graphs.

$$N(O) := \max\left\{k \mid \mathbb{O}_{(n,m)} \in O, k = n\right\} \tag{3.12}$$

Let O be a set of dataset operators. Moreover, let $N(O)$ denote the maximum in-degree in O as given in 3.12. Now, an enrichment table is a table with $2 + N(O)$ columns and $|O|$ rows, where each row corresponds to one dataset operator.

The idea behind this representation is that we can go through this table from top to bottom and evaluate the corresponding dataset operators using only the results of the above rows. Since dataset acceptors produce no output, they are ommited in this representation for sake of simplicity. Note that this idea was taken from[30], where the authors call these tables „Column Tables".

The structure of the enrichment table is defined as follows. The first column contains dataset operators. The second column contains configuration datasets and the third column contains the in-degree of the dataset operator in the first column. The rest of the columns contain the indices of the rows used as input to the corresponding dataset operator.

We use the notation $\mathbb{T}_i$ to refer the the ith row of the enrichment table $\mathbb{T}$. Furthermore, the notation $\mathbb{T}_{i,j}$ denotes the jth entry in the ith row of $\mathbb{T}$.

Table 3.1: An example enrichment table which is equivalent to the inherently confluent enrichment graph in Figure 3.1. For better readability we prepended the row number followed by a colon in each row.

1:	$\mathbb{O}_1$	P_1	0	0	0	0
2:	$\mathbb{O}_2$	P_2	1	1	0	0
3:	$\mathbb{O}_3$	P_3	2	1	2	0
4:	$\mathbb{O}_4$	P_4	1	3	0	0
5:	$\mathbb{O}_5$	P_5	3	2	4	3

The example in Table 3.1 depicts the enrichment table representation of the enrichment graph given in Figure 3.1.

A method to obtain an enrichment table from a given inherently confluent enrichment graph can be taken from [30] and we will not reformulate the whole method here.

In essence, the authors state that a Column Table can be easily obtained from the adjacency matrix of a given DAG $G = (\mathbf{V}, \mathbf{E})$ [c] when an indexing $\phi\colon \{1, \dots, |\mathbf{V}|\}$ of its vertices is provided such that $\forall (v, v') \in \mathbf{E}\colon \phi(v) > \phi(v')$.

[c] *Please note that our defintion of inherently confluent enrichment graphs corresponds to general DAGs without loss of generality.*

In order to obtain the results of a given enrichment table $\mathbb{T}$, it just has to be evaluated from top to bottom. The last result is considered to be the result of the whole table.

The evaluation result of a row $\mathbb{T}_i$ is denoted as $\mathbf{T}_i$. We will now show this process exemplarily for the enrichment table given in Table 3.1.

$$
\begin{aligned}
\mathbf{T}_1 &= \mathbb{O}_1\,(P_1) \\
\mathbf{T}_2 &= \mathbb{O}_2\,(\mathbf{T}_1, P_2) \\
\mathbf{T}_3 &= \mathbb{O}_3\,(\mathbf{T}_1, \mathbf{T}_2, P_3) \\
\mathbf{T}_4 &= \mathbb{O}_4\,(\mathbf{T}_3, P_4) \\
\mathbf{T}_5 &= \mathbb{O}_5\,(\mathbf{T}_2, \mathbf{T}_4, \mathbf{T}_3, P_5)
\end{aligned}
$$

3.2 IMPLEMENTATION DETAILS

We implement DEER2[3] as a Java application using the Java Development Kit (JDK) version 9. DEER2 is based on a general engine for parallel data transformation workflows with the name

3 https://github.com/dice-group/deer

FARADAY-CAGE, which we did also develop[4]. This engine enables DEER2 to automatically distribute the execution of a given enrichment graph to the available CPU cores.

The RDF-related features in FARADAY-CAGE and DEER2 are implemented using the Apache Jena Library[5]. In order to be applicable to a wide domain of enrichment workflows, DEER2 implements eleven standard enrichment operators.

For the configuration of enrichment workflows in DEER2, we use RDF files in the Turtle serialization format[6]. To this end, we developed two RDF vocabularies, namely the `fcage`[7] and the `deer`[8] vocabularies. The `fcage` vocabulary supplies predicates and base classes for the definition of DAG-shaped data transformation workflows, including but not limited to our notion of an enrichment graph. The `deer` vocabulary is used complementary and supplies a RDFS class hierarchy for all standard implementations of dataset operators in DEER2 as well as predicates for configuring instances of these classes.

We implement two user friendly backends, namely a command line interface and a RESTful Web service.

Since this thesis does not serve the purpose of a technical documentation, we will ommit a detailed discussion of most of these parts here. Rather, we will focus on the mechanisms used to ensure the modularity of DEER2, since this was one of our leading design goals. Moreover, we will give a short overview of seven enrichment operators, since a basic understanding of their functionality is necessary to follow the development of our learning approach in Section 3.3.

3.2.1 *Modularity*

In order to allow other developers to supply custom enrichment operators to DEER2, we chose to use the Plugin Framework for Java (PF4J) as a foundation of our plugin system. PF4J allows to annotate certain interfaces as so called extension points and supplies an automatic discovery system for classes implementing these interfaces using the Java reflection language feature.

4 https://github.com/dice-group/faraday-cage
5 https://jena.apache.org
6 https://www.w3.org/TR/turtle/
7 https://w3id.org/fcage
8 https://w3id.org/deer

Therefore, we do not make any assumptions about the enrichment operators that are available at runtime. We supply mechanisms for developers to retrieve a list of the currently loaded enrichment operators at runtime using a command line interface and a RESTful Web service.

Since each of the plugins can have their own configuration vocabulary, we implemented an approach to configuration validation using SHACL. The same SHACL shape graphs which we use for configuration validation could also be used in order to let frontends dynamically generate forms for convenient user interaction in the future.

Finally, we supply a dummy Apache Maven[9] project[10] that includes the necessary architecture and configurations for compiling custom code to enrichment operator plugins.

3.2.2 *Overview of Implemented Enrichment Operators*

In this section we will give a short descriptions of the functionality of each of the standard enrichment operators implemented in DEER2 that will be used in our learning approach.

Please note that all of these operators are based on the ones as defined in the original DEER publication[51] and share the same general functionality, despite being completely reimplemented.

Unless otherwise mentioned, their in-degree and out-degree amount to 1 and we denote their input and output RDF datasets $D^{(\text{in})}$ and $D^{(\text{out})}$, respectively.

Filter Operator

The idea of the filter operator is to select a specific set of the input dataset triples. Then, the selected set of triples are generated by the filter operator as its output.

The `deer:selector` parameter accepts a resource with at least one of the three basic selectors types: `deer:subject`, `deer:predicate` and `deer:object`, which can be combined as required. For simple filtering tasks, e.g. if the task is to filter out everything but triples containing a few given properties, this is easier to set up than to write a SPARQL query.

9 `https://maven.apache.org`
10 `https://github.com/dice-group/deer/tree/master/examples/simple-plugin-example`

Authority Conformation Operator

The idea of the authority conformation operator is to change a specified source IRI authority to a specified target IRI authority. By the authority of an IRI we mean the part of the IRI before the last slash or hash sign, which is also sometimes called the namespace.

One authority conformation operation is represented by the `deer:operation` property. The objects of `deer:operation` need to be blank nodes with the properties `deer:sourceAuthority` and `deer:targetAuthority` for specifying the source and target authorities, respectively.

Any number of authority conformation operations can be specified by declaring multiple `deer:operation` parameters.

Formally, given two IRIs called source authority (a_s) and a target authority (a_t), this operator will first set $D^{(\text{out})} := D^{(\text{in})}$. It will then find all triples $(s, p, o) \in D^{(\text{in})}, s \in \mathcal{R}$ for which a_s is a prefix of the IRI representing s. For each triple $t = (s, p, o)$ found that way, the operator will create a new triple $t' = (s', p, o)$ where s' is obtained by replacing a_s with a_t in s. It will then perform the set operation $D^{(\text{out})} := (D^{(\text{out})} \backslash \{t\}) \cup \{t'\}$ on the output dataset.

Predicate Conformation Operator

The idea of the predicate conformation operator is to replace all instances of specified source predicate to a specified target predicate with the same subject and object values.

One predicate conformation operation is represented by the `deer:operation` property. The objects of `deer:operation` must be blank nodes with the properties `deer:sourcePredicate` and `deer:targetPredicate` for specifying the source and target predicates, respectively.

Any number of predicate conformation operations can be specified by declaring multiple `deer:operation parameters`.

Formally, given two IRI resources called source predicate (p_s) and a target predicate (p_t), this operator will first set $D^{(\text{out})} := D^{(\text{in})}$. It will then find all triples $(s, p, o) \in D^{(\text{in})}, s \in \mathcal{R}$ for which $p_s = p$. For each triple $t = (s, p_s, o)$ found that way, the operator will create a new triple $t' = (s, p_t, o)$ and perform the set operation $D^{(\text{out})} := (D^{(\text{out})} \backslash \{t\}) \cup \{t'\}$ on the output dataset.

Dereferencing Operator

For datasets which contain IRI resources from remote RDF datasets, the idea of the dereferencing enrichment operator is to deference a set of triples that contain a desired predicate from the remote dataset and add them to the output dataset using content negotiation on HTTP.

One dereferencing operation is represented by the `deer:operation` parameter, which takes as object a blank node with the following properties:

- `deer:lookUpPrefix` specifies the IRI prefix to identify the resources from the desired remote RDF dataset.
- `deer:dereferencingProperty` specifies the predicate used to identify the desired triples from the remote RDF dataset.
- `deer:importProperty` specifies the predicate used when importing the identified triples to the output dataset.

One can specify any number of `deer:operation` parameters and they will all be carried out and in case of overlapping lookup prefix, they will be carried out in a single HTTP request to save bandwidth.

Named Entity Recognition Operator

The idea of the Named Entity Recognition (NER) operator is to extract IRI resources from string literals using Federated knOwledge eXtraction Framework (FOX)[55], which is a NER framework for the Semantic Web. To that end, it uses a lookup predicate p_l to identify triples $(s, p_l, o), o \in \mathcal{L}$ containing interesting string literals in the input dataset. For each such triple, it extracts the textual data from the string literal o and sends it to an instance of the FOX Web service. The IRI resources referencing named entities in the string literal are returned by FOX and added to the output dataset using a configurable import predicate p_i

This operator specifies the following configuration properties:

- `deer:literalProperty` specifies the lookup predicate.
- `deer:importProperty` specifies the import predicate.
- `deer:foxUrl` specifies the Uniform Resource Locator (URL) of the FOX Web service.

Merge Operator

The merge operator has an in-degree of 2 and an out-degree of 1. The very simple idea of it is to merge two RDF datasets. Formally, for given input datasets $D_1^{(\text{in})}$, $D_2^{(\text{in})}$, this operator returns $D^{(\text{out})} := D_1^{(\text{in})} \cup D_2^{(\text{in})}$.

Linking Operator

The linking operator has an in-degree of 2 and an out-degree of 1. Its idea is to perform Link Discovery on two input RDF datasets using the Link Discovery Framework for Metric Spaces (LIMES) [35]. Note that the LIMES Java library is integrated into DEER2.

One can specify the following configuration properties for this operator:

- `deer:linkSpecification` specifies the link specification to be passed to LIMES.
- `deer:threshold` specifies the linking threshold to be passed to LIMES.
- `deer:linkingPredicate` specifies the predicate to be used for the generated links.

Given two input datasets $D_1^{(\text{in})}$, $D_2^{(\text{in})}$ and the RDF dataset of links generated by the invocation of LIMES D_L, this operator returns $D^{(\text{out})} := D_1^{(\text{in})} \cup D_2^{(\text{in})} \cup D_L$

3.3 LEARNING APPROACH

We have seen that even the most simplistic standard enrichment operators already require a fair amount of configuration. Moreover, one has to be very well versed with the set of available enrichment operators in order to specify how they should be combined to an adequate enrichment graph for the specific use case at hand.

Together with the additional complexity induced by the modular design of DEER2, we can assert that it requires expert knowledge to operate it.

This however goes against our initial motivation of presenting an approach accessible also to novice users.

In this section, we will therefore develop a GP-based approach for the automatic learning of enrichment graphs. To that end, we will first define our learning problem in Section 3.3.1. After that, we will introduce our baseline algorithm in Section 3.3.2 which is subject to subsequent tweaks and extensions described in Section 3.3.3 through 3.3.4.

3.3.1 *The Learning Problem*

The problem under study is that of finding an adequate enrichment graph for a given training example.

Since for this work we do not wish to consider multi-objective learning, we restrict ourselves to learning the subclass of inherently confluent enrichment graphs. We will furthermore restrict our study to enrichment graphs where the maximum in-degree of the involved enrichment operators and the number of involved dataset emitters are at most two.

Therefore, a training example will consist of a list of source training datasets $\left(D_{s_1}, \dots, D_{s_n}\right) \in \mathcal{D}^n$ with $n \in \{1, 2\}$ as well as a single target training dataset $D_t \in \mathcal{D}$.

The output of RDF enrichment is commonly expected to have a regular structure, which more formally means that the output dataset will consist of a number of approximately isomorphic RDF subgraphs.

Therefore, we will regard the source and target datasets as Simplified Concise Bounded Descriptions (SCBDs) of sufficient depth to representatively capture the desired enrichment on just one of these approximately isomorphic subgraphs. This is in accordance with the observation that a single SCBD is sufficient for the training of enrichment pipelines in [51].

3.3.2 *Baseline Algorithm*

In order to solve the learning problem defined above, we will use a population-based Genetic Programming (GP) algorithm and additionally employ techniques from Multi Expression Programming (MEP).

We choose enrichment tables as the encoding of the programs (i. e.enrichment graphs) that are to be learned by our approach. In order to tailor our encoding fit for use with MEP we will first define the notion of a *multi-expressive enrichment table*.

3.3.2.1 *Multi-Expressive Enrichment Table*

We call a row within an enrichment table an *output row*, if it is not used as input to a subsequent row.

Please verify that output rows by definition always correspond to dataset acceptors and that our previous definition of enrichment tables allows for only a single output row, as they must be isomorphic to inherently confluent enrichment graphs.

We now define a multi-expressive enrichment table as an enrichment table which has more than one *output row*.

A normal enrichment table $\mathbb{T}^{(\text{norm})}$ can be obtained from a multi-expressive enrichment table in a recursive fashion $\mathbb{T}$ as follows:

- Let $\mathbb{T}^{(\text{norm})}$ be an empty enrichment table.
- Let $\mathcal{I}$ be an empty set.
- Let $\zeta : \mathbb{N} \to \mathbb{N}$ denote a bijective mapping of index numbers, which initially only contains the mapping $0 \mapsto 0$.
- We select one of the output rows with index r as reference. This row is added together with its index to the list $\mathcal{I}$ as $\mathcal{I} := \mathcal{I} \circ (\mathbb{T}_r, r)$.
- When adding a row $\mathbb{T}_i$ to $\mathcal{I}$, we also add all the row-index pairs $\left(\left(\mathbb{T}_{\mathbb{T}_{i,4}}, \mathbb{T}_{i,4}\right), \dots, \left(\mathbb{T}_{\mathbb{T}_{i,\mathbb{T}_{i,3}}}, \mathbb{T}_{i,\mathbb{T}_{i,3}}\right)\right)$ to $\mathcal{I}$.

Table 3.2: Example of a multi-expressive enrichment table. Output rows are marked in red color. When selecting the 8th output row as reference we can recursively reconstruct the normal enrichment table given in Table 3.1. For better readability we prepended the row number followed by a colon in each row.

1:	$\mathbb{O}_1$	P_1	0	0	0	0
2:	$\mathbb{O}_2$	P_2	1	1	0	0
3:	$\mathbb{O}_3$	P_3	2	1	2	0
4:	$\mathbb{O}_4$	P_4	1	3	0	0
5:	$\mathbb{O}_6$	P_6	2	3	1	0
6:	$\mathbb{O}_7$	P_7	1	5	0	0
7:	$\mathbb{O}_8$	P_8	2	5	3	0
8:	$\mathbb{O}_5$	P_5	3	2	4	3
9:	$\mathbb{O}_9$	P_9	1	7	0	0

- We then sort the entries $(\mathbb{T}_k, i_k) \in \mathcal{I}$ according to the index i_k in ascending order.
- For each entry $(\mathbb{T}_k, i_k) \in \mathcal{I}$, $k \in (1, \ldots, |\mathcal{I}|)$, we add $\mathbb{T}_k$ to $\mathbb{T}^{(\text{norm})}$ and keep track of the mapping from the old row number to the new row number by adding the mapping $i_k \mapsto k$ to ζ.
- We then replace the input definition column entries $\mathbb{T}^{(\text{norm})}_{k,4}, \ldots, \mathbb{T}^{(\text{norm})}_{k,\mathbb{T}^{(\text{norm})}_{k,3}}$ with $\zeta\left(\mathbb{T}^{(\text{norm})}_{k,4}\right), \ldots, \zeta\left(\mathbb{T}^{(\text{norm})}_{k,\mathbb{T}^{(\text{norm})}_{k,3}}\right)$.

Consider the multi-expressive enrichment table given in Table 3.2. When selecting the 8th output row (with the first column entry $\mathbb{O}_5$) as reference, we reconstruct the normal enrichment table as layed out in Table 3.1 by following the method defined above.

3.3.2.2 *Population Initialization*

After having given a definition of multi-expressive enrichment tables as well as a method to reduce them to normal enrichment tables, we will now provide an algorithm for the randomized generation of multi-expressive enrichment tables which will be used for our population initialization. This algorithm will receive a set of prespecified dataset emitters $P^{\perp}$ and a corresponding set of configuration datasets $D^{\perp}$ as well as the number of rows it should generate.

We specify this algorithm using pseudo code in Listing 3.1.

Note that we initially set the configuration datasets of all the enrichment operators to the empty dataset. This is due to the fact that we employ a mechanism dubbed *heuristic self-configuration* in order to derive the configuration datasets of enrichment operators on-demand while evaluating the multi-expressive enrichment table.

3.3.2.3 *Heuristic Self-Configuration of Enrichment Operators*

When evaluation a multi-expressive enrichment table, we denote the datasets resulting from evaluation of a given row $\mathbb{T}_i$ as $\mathbf{T}_i$.

The heuristic self-configuration of enrichment operators for a given row $\mathbb{T}_i$ is done by supplying the result of all its input rows $\left(\mathbb{T}_{i,4}, \ldots, \mathbb{T}_{i,\mathbb{T}_{i,3}}\right)$ and D_t (the target training dataset) to a method $\mathbb{s} : \mathcal{D}^n \times \mathcal{D}, n \in \{1, 2\}$ that heuristically derives a configuration by searching for clues in the structture of the given RDF datasets.

Each enrichment operator must supply such a method. Deriving good heuristics for self-configuration is a very hard problem on its own and therefore will not be part of this thesis.

The enrichment operators we use here all specify a very simple heuristics for self-configuration and improvement of these methods should be part of future work in this field.

However, the complexity of the heuristics used here is not crucial for the evaluation of our approach in Chapter 4, since we will specifically engineer the test cases in a way such that our simple heuristics have the chance of deriving the proper configuration, *given that we just learn the proper structure of the graph*.

We therefore ommit a specification of heuristics to not burden the reader with a lot of formalism that has little to none utility in following the rest of this work.

Instead, now that we know about self-configuration, we will continue by describing the evaluation of a given genotype, i. e.a multi-expressive enrichment table.

3.3.2.4 *Fitness Function*

Let $\mathbf{S}(D)$, $\mathbf{P}(D)$, $\mathbf{O}(D)$ denote the sets of subjects, predicates and objects[d] of a given dataset D, respectively.

[d] *Note that we exclude blank nodes from* $\mathbf{S}(D)$ *and* $\mathbf{T}(D)$.

Algorithm 3.1: Random generation of multi-expressive enrichment tables. $P^{\perp}$ is the set of dataset emitters to be used. $D^{\perp}$ is the set of configuration datasets to be used for the dataset emitters. $o(m)$ is a function that returns a random enrichment operators with a maximum in-degree of $m \in [1,2]$. k is the number of rows to be generated. $\text{rnd}(a,b)$ is a function returning a random number in $[a,b]$

```
input: $P^{\perp}$ , $D^{\perp}$ , $k$
output: $\mathbb{T}$
begin
  i ← 1 % row counter
  $\mathbb{T} \leftarrow \emptyset$ % empty multi-expressive enrichment table
  while i ≤ k
    if ($k < |P_{\perp}|$) % insert dataset emitters
      $\mathbb{T}_i \leftarrow (P_i^{\perp}, D_i^{\perp}, 0, 0, 0)$
    else
      $d \leftarrow \text{rnd}(1, \max(2, i-1))$
      $\mathbb{o}_{(n,m)} \leftarrow o(d)$
      $d \leftarrow n$
      if ($d = 1$)
        $\mathbb{T}_i \leftarrow (\mathbb{o}, \emptyset, d, \text{rnd}(1, i-1))$
      else
        $\mathbb{T}_i \leftarrow (\mathbb{o}, \emptyset, d, \text{rnd}(1, i-1), \text{rnd}(1, i-1))$
      end
    end
    i ← i + 1
  end
  return res
end
```

We define a fitness function f as

$$\begin{aligned} f \colon \mathcal{D}^2 &\to \mathbb{R} \\ (D_r, D_t) &\mapsto \frac{1}{4}\left(F_1\left(D_r, D_t\right) + F_1\left(\mathbf{S}(D_r), \mathbf{S}(D_t)\right)\right. \\ &\quad \left. + F_1\left(\mathbf{P}(D_r), \mathbf{P}(D_t)\right) + F_1\left(\mathbf{O}(D_r), \mathbf{O}(D_t)\right)\right) \end{aligned} \tag{3.13}$$

where F_1 is the F_1-score as defined in Section 2.4.2.

Informally, f measures similarity in terms of overlapping IRI resources in subjects , overlapping IRI resources in predicates and overlapping IRI resources and literals in objects as well as overlapping triples.

3.3.2.5 *MEP-based Evaluation of Genotypes*

As our genotypes are represented as multi-expressive enrichment table $\mathbb{T}$, they correspond to multiple phenotypes, i. e.normal enrichment tables.

To this end, we regard each row in $\mathbb{T}$ as a potential output row. The evaluation of $\mathbb{T}$ is done row-wise and much similar to the evaluation of a normal enrichment table[e] with the notable exception that we apply self-configuration to obtain the configuration dataset[f].

[e] *as specified in Section 3.1.3*

[f] *as specified in Section 3.3.2.3*

Furthermore, we will compute the fitness for the result of each row as specified in Section 3.3.2.4.

The effective fitness of a genotype is the maximum fitness of all its rows. Moreover, we define the *effective phenotype* of a given genotype as the normal enrichment table obtained by reconstruction[g] beginning at the row with the highest fitness value.

[g] *as specified in Section 3.3.2.1*

3.3.2.6 *General Notes*

As we have seen previously, we use multi-expressive enrichment tables as genotypes. These will have a fixed number of rows, denoted r.

Our population also consists of a fixed number of genotypes, denoted p.

We use a single-point crossover operator, which will swap rows of two genotypes at a random crossover point.

For mutation, we use an operator which randomly manipulates a given multi-expressive enrichment table in way such that their invariants[h] stay intact.

Finally, in each generation we use elitist selection[i] in order to avoid decreasing the fitness of the best individual in the next generation.

[i] *The single best individual always survives.*

Tournament selection with a tournament size of 3 and a selection probability of 0.75 is applied for determining the mating pool as well as for selecting the survivors.

Both the offspring and the survivors are subject to mutation. We identify the offspring fraction $\frac{\mu}{\lambda}$, mutation probability σ and mutation rate ρ as the most important parameters for our approach. They shall hence be considered hyperparameters and are subject to experimental investigation in **??**.

The algorithm will stop when

1. a perfect solution ($f = 1$) is found,
2. a maximum number g of generations is exceeded, or
3. the optional convergence detection mechanism[j] commands termination.

[j] *See* **??**.

Now that we have given a complete overview of our baseline algorithm, we will present modifications of this baseline that we expect to improve the performance of the learning algorithm.

3.3.3 *Genotype Compaction*

The idea of enrichment table compaction is that we could speedup learning by restrucuring genotypes $\mathbb{T}$ in way such that we move the effective phenotype $\mathbb{T}_p$ of $\mathbb{T}$ to the front of the compacted genotype $\mathbb{T}'$.

This is done by first deriving $\mathbb{T}_p$ per the method described in Section 3.3.2.5. Let $|\mathbb{T}|$ denote the number of rows in $\mathbb{T}$. Furthermore, let $\bot(\mathbb{T})$ denote the first one or two rows corresponding to dataset emitters in a given genotype $\mathbb{T}$. We define $\top(\mathbb{T}) =$

$\mathbb{T} \backslash \bot(\mathbb{T})$. The compacted genotype $\mathbb{T}'$ is then defined row-wise by

$$\begin{aligned} \mathbb{T}'_i &:= \bot(\mathbb{T})_i && \iff && 1 \leq i \leq |\bot(\mathbb{T})| \\ \mathbb{T}'_i &:= \top(\mathbb{T}_p)_{i-|\bot(\mathbb{T})|} && \iff && |\bot(\mathbb{T})| < i \leq |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| \\ \mathbb{T}'_i &:= \texttt{rndrow()} && \iff && |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| < i \leq r \end{aligned} \tag{3.14}$$

where `rndrow()` means that in this case we generate a new row randomly in way that is reflected by lines 10 through 17 of Listing 3.1.

We apply genotype compaction randomly with a probability of 0.5 to the whole population in the beginning of each generation.

In order to further minimize the size of the compacted genotype, we detect so called *no-op* rows, remove them and repair the genotype.

As the reader should by now be familiar with our formalism, we ommit a formal specification of this process and instead limit ourselves to only defining how we *detect* a *no-op* row.

NO-OP DETECTION. Given a row $\mathbb{T}_i$, its output dataset $\mathbf{T}_i$ and its input datasets $I = \left(\mathbf{T}_{\mathbb{T}_{i,4}}, \dots, \mathbf{T}_{\mathbb{T}_{i,\mathbb{T}_{i,3}}}\right)$ we say that $\mathbb{T}_i$ is a *no-op row*, iff $\mathbf{T}_i = D$ for some $D \in I$.

3.3.4 *Semantic Genetic Operators*

We define three semantic genetic operators, which we will present in the following.

3.3.4.1 *Subgraph Merging Crossover*

The idea behind our subgraph merging crossover is to combine two genotypes $\mathbb{T}$, $\mathbb{T}'$ in a way that increases the probability of learning a graph where both $\mathbb{T}$ and $\mathbb{T}'$ are part of its effective phenotype.

This is motivated by the observation that two sufficiently distinct good performing inherently confluent enrichment graphs could be merged into one by redirecting the vertices which have edges to their dataset acceptors to a new vertice with in-degree two.

Let $\mathbb{T}_p$, $\mathbb{T}'_p$ be the effective phenotypes of $\mathbb{T}$, $\mathbb{T}'$, respectively. We apply this operator only if

$$|\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| \leq r$$

holds, otherwise we default to the single-point crossover.

In order to guarantee that the population size stays constant, the subgraph merging crossover needs to return two child genotypes.

To this end, we first compute both childs equally, but then we fill the $r - |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)|$ remaining rows randomly for each child genotype.

If $r - |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| \geq 1$, we will also insert a row that merges the two subgraphs in $\mathbb{T}_p$ and $\mathbb{T}'_p$ with a probability of 0.25. To this end, we will randomly aquire an enrichment operator with an in-degree of two and set the input columns to the positions of the output rows of $\mathbb{T}_p$ and $\mathbb{T}'_p$ in the newly created child genotype.

Let $\mathbb{o}^{\text{merge}}_{(2,1)}$ be a randomly chosen enrichment operator with in-degree two. Formally, a child genotype $\mathbb{T}^c$ is defined row-wise as

$$\begin{aligned}
\mathbb{T}^c_i &:= \bot(\mathbb{T})_i && \iff 1 \leq i \leq |\bot(\mathbb{T})| \\
\mathbb{T}^c_i &:= \top(\mathbb{T}_p)_{i-|\bot(\mathbb{T})|} && \iff |\bot(\mathbb{T})| < i \leq |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| \\
\mathbb{T}^c_i &:= \top(\mathbb{T}'_p)_{i-|\bot(\mathbb{T})|-|\top(\mathbb{T}_p)|} && \iff |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| < i \\
& && \quad \leq |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| \\
\mathbb{T}^c_i &:= \Big(\mathbb{o}^{\text{merge}}_{(2,1)}, \emptyset, 2, |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)|, && \iff i = |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| + 1 \\
& \qquad |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)|\Big) && \quad \leq r \wedge \mathrm{rnd}(1,4) = 1 \\
\mathbb{T}^c_i &:= \texttt{rndrow()} && \iff i = |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| + 1 \\
& && \quad \leq r \wedge \mathrm{rnd}(1,4) > 1 \\
\mathbb{T}^c_i &:= \texttt{rndrow()} && \iff |\bot(\mathbb{T})| + |\top(\mathbb{T}_p)| + |\top(\mathbb{T}'_p)| + 1 \\
& && \quad < i \leq r
\end{aligned} \tag{3.15}$$

3.3.4.2 *Precondition Broadcasting Mutation*

The idea of the precondition broadcasting mutation is to factor in a measure of *how well a particular enrichment operator could replace another one* in a given row.

To this end, the enrichment operators need to provide an additional method $\mathbb{a}$, which takes the same kind of arguments that the self-configuration method does. It then returns a real value between 0 and 1 which expresses the applicability of the enrichment operator, given the input and target datasets.

We ommit formal specifications for sake of brevity, as at this point it should be clear to the reader how these methods are defined. The functionality of the precondition broadcasting mutation is as follows.

When a row $\mathbb{T}_i$ has been marked for mutation in a given genotype $\mathbb{T}$, we evaluate $\mathbb{T}$ up to $\mathbb{T}_i$ and obtain the evaluation results of its input dataset(s). Subsequently we *broadcast* the input dataset(s) together with the target training dataset to the applicability methods $\mathbb{a}$ of all available enrichment operators. After the operators have returned their result, we initialize a roulette wheel selection where the portions of the virtual roulette wheel are proportional to the relative applicabilities of the operators.

Finally, we set $\mathbb{T}_{i,1}$ to the winner of the roulette wheel.

We chose the name *postcondition broadcasting* for this method, as the input datasets to an enrichment operator can be regarded as its preconditions w.r.t. applicability.

3.3.4.3 *Postcondition Broadcasting Mutation*

The postcondition broadcasting mutation works similar to the precondition broadcasting mutation but in the reverse direction.

Therefore, we need to evaluate our genotype top-down instead of bottom-up.

To that end, enrichment operators need to supply a method $\mathbb{o}^{-1}$ that heuristically computes their inverse, again using the same arguments as the self-configuration method $\mathbb{s}$.

For sake of brevity, we ommit the definition of these methods here, but the interested reader can look them up in our source code.

3.3.5 *Convergence Detection Mechanism*

Our convergence detection mechanism is used to (1) detect convergence of our population and (2) prohibit convergence into a local optima by temporarily adjusting σ and ρ.

In order to detect convergence, we keep track of the best fitness value and the standard deviation of fitness values in our generations. We assert convergence if the best fitness value has not changed and the standard deviation of fitness values has been under a certain threshold s_{min} for the last ten generations.

When convergence is detected, we attempt to escape the potential local optima by temporarily setting $\sigma = \rho = 1$. In the following generations, we lower them again by multiplication with a constant factor until they reach their initial values.

The algorithm finally terminates if after a fixed number of convergence detections and escape attempts no perfect solution is found.

4 EVALUATION

In this chapter we are going to define our experimental protocol as well as present and discuss our evaluation results.

4.1 EXPERIMENTAL SETUP

We performed two sets of experiments, one for determining the best set of hyperparemeters and one for evaluating the performance of our approach on large scale datasets derived from real-world RDF datasets.

All experiments were carried out on a 64-core 2.3 GHz server running *OpenJDK* 64-Bit Server 1.8.0_151 on *Ubuntu* 16.04.3 LTS. Each experiment was assigned 128 GB RAM.

4.2 INCREMENTAL HYPERPARAMETER OPTIMIZATION

In order to determine the best set of values for our hyperparameters *offspring fraction* $\alpha = \frac{\mu}{\lambda}$, *mutation probability* σ and *mutation rate* ρ, we incrementally performed a series of grid searches.

Equation 4.1 displays the values we considered in our first grid search.

$$\begin{aligned} \alpha &\in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\} \\ \sigma &\in \{0.1, 0.3, 0.5, 0.7, 0.9\} \\ \rho &\in \{0.1, 0.3, 0.5, 0.7, 0.9\} \end{aligned} \tag{4.1}$$

We started by setting up a minimal learning problem as follows. First, we constructed a very simple toy dataset consisting of a single SCBD. Afterwards, we manually defined a linear enrichment graph with only three enrichment operators, the authority conformation, predicate conformation and NER enrichment operators. We derived our training example by applying this linear enrichment graph to the toy dataset.

Note that we set up this problem in a way such that the particular order of these enrichment operators is of no importance. That is, as long as all three are present and are configured in the same way, the linear enrichment graph containing them will evaluate to the same result.

We set the number of rows in the enrichment tables to $r = 7$, the population size to $p = 30$ and the maximum number of generations to $g = 5000$.

In order to have a good display of how efficient our approach can find perfect results, we disabled the convergence detection mechanism. We empirically found a seed `seed` $= 54738$ for our PRNG for which the initial population does not include a perfect genotype.

As our approach is non-deterministic, we needed to repeat the experiment a sufficient number of times. This number was not obvious and therefore we empirically tried different orders of magnitudes $10^1, 10^2, 10^3$, where we repeated each series of repetitions ten times and measured the standard deviation of the average number of generations for each series of repetitions for all ten attempts. When we set the repetitions to 10^3, we could for the first time observe the standard deviation dropping to under 5% of the mean number of generations. We therefore sticked to this number of repetitions for all of the hyperparameter optimization experiments.

For this experiment we measured the mean, standard deviation, minimum and maximum of the generations until termination for each particular set of hyperparameters.

The results are shown in Figure 4.1. Note that the best set of hyperparameters we retrieved was $\alpha = 1.0$, $\sigma = 0.9$ and $\rho = 0.9$. The interpretation of those values is that for this particular experiment, the correct solutions are just found by chance, since this set of hyperparameters effectively maximizes the systems entropy.

This interpretation is further underlined by the high values we retrieved for the standard deviation of generations to termination, which can be seen as the coloring of the first plot in Figure 4.1.

Despite these results are anything but surprising, as completely random manipulation of enrichment graphs can only lead to completely random results, they serve a good baseline to measure the success of the subsequential refinements of our baseline al-

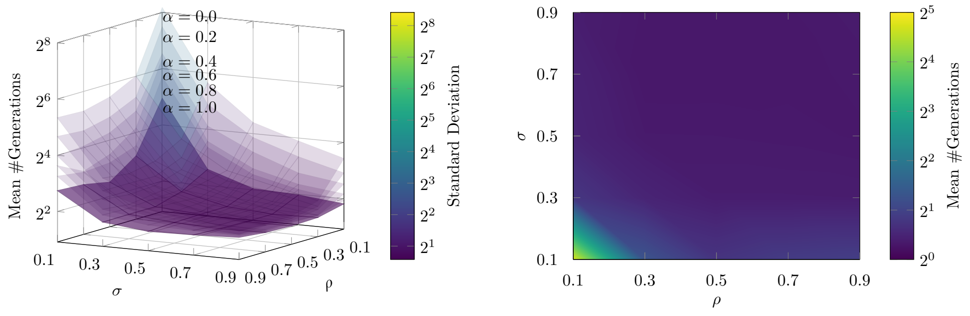


Figure 4.1: **First set of Hyperparameter Optimization Results.** The experiment was repeated one thousand times. We measured the average as well as the standard deviation of the generations to termination. On the left hand side we see the average number of generations to termination plotted against the mutation probability σ and the mutation rate ρ. The six planes correspond to the six values we tried for the offspring fraction *alpha*. We drew the best performing plane more opaque than the rest. On the right hand side we see a more detailed heat map of the best performing plane on the left. The results suggest that $\alpha = 1.0$, $\sigma = 0.9$ and $rho = 0.9$ are the best performing hyperparameters for this experiment.

gorithm.

We continued to study the set of best hyperparameters for the same setting, but this time with the addition of the genotype compaction as described in Section 3.3.3.

The results for this experiment are shown in Figure 4.2.

Note that the first plot in Figure 4.2 is rotated by 180 degrees w.r.t. the first plot in Figure 4.1. Therefore, we get a completely different result this time, where the best set of hyperparameter amounts to $\alpha = 0.0$, $\sigma = 0.1$ and $\rho = 0.1$.

However, despite these results looking very good, they are deceitful in that they are again just obtained by chance. As we hoped for, the genotype compaction does indeed speed up the learning process by lowering the average number of generations needed to find a perfect genotype. But since we always generate random rows to fill the genotype to its full capacity, we are actually just increasing the already considerable chance of finding a genotype that contains our set of enrichment operators in any

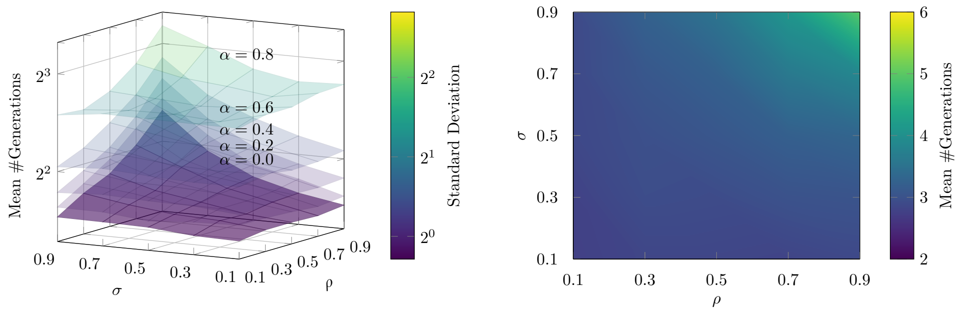


Figure 4.2: **Second set of Hyperparameter Optimization Results.** The experiment was repeated one thousand times. We measured the average as well as the standard deviation of the generations to termination. On the left hand side we see the average number of generations to termination plotted against the mutation probability σ and the mutation rate ρ. The five planes correspond to the five values we tried for the offspring fraction *alpha*. We drew the best performing plane more opaque than the rest. On the right hand side we see a more detailed heat map of the best performing plane on the left. The results suggest that $\alpha = 0.0$, $\sigma = 0.1$ and $rho = 0.1$ are the best performing hyperparameters for this experiment.

order.

We therefore continued our investigation by increasing the hardness of the learning problem. This time, we constructed two toy datsets as source training datsets. Furthermore, we specified an inherently confluent enrichment graph that consisted of five enrichment operators. This enrichment graph was developed in a way such that the enrichment operators did now depend on each other, so that it did not suffice to find an arbitrary permutation of them.

Note that with two dataset emitters and five enrichment operators, in order to find the perfect solution, our approach would need to work with its full capacity. We therefore set the number of rows in our genotypes to $r = 8$ to give the approach just a bit „space to breathe".

After constructing this much harder learning problem, we then continued our investigation. At first we wanted to have a good understanding of how much harder the new problem actually is compared to the old one. To that end, we executed 10^3 repetitions for the optimal set of hyperparameters $\alpha = 0.0$, $\sigma = 0.1$ and $\rho = 0.1$ found previously, where the average number of generations to termination amounted to 2.77. The result of this run was that with an average number of generations to termination of 73.80 the new problem was indeed much harder than the old one. In an extreme case, the perfect result was only obtained after 2636 generations.

Since due to the long runtimes it would have not been feasible to run a grid search for this problem with the current setting, we decided to instead introduce another addition to our algorithm, namely our graph merging crossover operator.

We left the rest of the setting untouched and ran the experiment again with an result of 36.99 generations to termination on average. Despite this being a substantial improvement, it was still not enough to run a complete grid search within an acceptable time, so we continued by introducing our precondition broadcasting mutation operator.

The new result looked very promising with an average number of generations to termination of only 15.23, but still the maximum number of generations until a perfect solution was found was 528.
In-depth analysis using the Java debugger made it evident that our approach was often stuck in local optima for extended periods of time.

We therefore enabled our convergence detection mechanism before we ran the next grid search. Note that with the convergence detection mechanism we no longer necessarily retreived a perfect solution in each run. Therefore, we measured the number of times that we actually did find a perfect solution. Furthermore, we restricted the grid search for σ and ρ to the values $0.1, 0.3$ and 0.5, as the previous results suggested that in order to achieve convergence, we should not introduce too much mutation into our system.

The final results, which are shown in Figure 4.3, suggest that the best set of hyperparameters are $\alpha = 1$, $\sigma = 0.5$ and $\rho = 0.5$.

It is notable that for these hyperparameters we achieve 100% perfect results despite enabling our convergence detection mechanism. Another interpretation of the data worth of mentioning is that the average number of generations to termination is lower

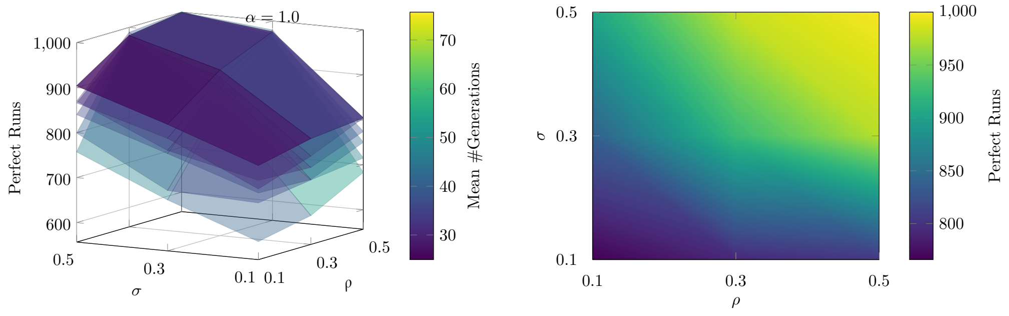


Figure 4.3: **Third set of Hyperparameter Optimization Results.** The experiment was repeated one thousand times. We measured the average generations to termination and the number of perfect runs. On the left hand side we see the number of perfect runs plotted against the mutation probability σ and the mutation rate ρ. The six planes correspond to the six values we tried for the offspring fraction *alpha*. We drew the best performing plane more opaque than the rest. On the right hand side we see a more detailed heat map of the best performing plane on the left. The results suggest that $\alpha = 1.0$, $\sigma = 0.5$ and $rho = 0.5$ are the best performing hyperparameters for this experiment.

for other settings, but these settings did not achieve this amount of perfect results and the difference in run time is neglectable.

4.3 PERFORMANCE EVALUATION

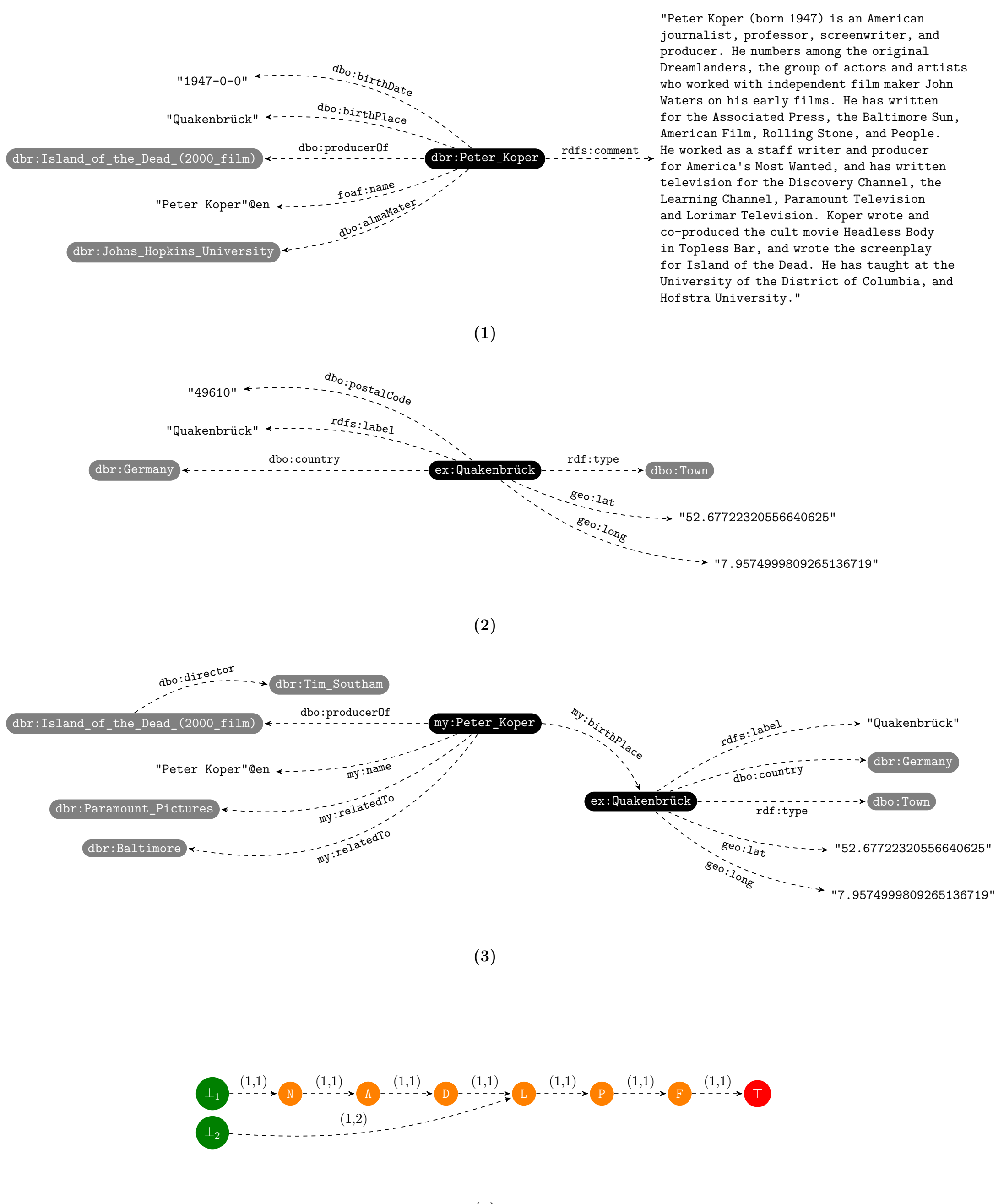
This experiment was devised from real-world data coming from DBpedia[11], where we obtained one dataset about 2, 230 persons born in Germany and another one about 1, 660 towns in Germany.

The source and target training datasets for this experiment are collectively shown in *Figure* 4.4 together with the enrichment graph that was used to compute the target training dataset.

We measured the performance of our Genetic Programming (GP) algorithm by averaging the best fitness found in each generation over a series of fifty repetitions of this experiment. Moreover, we measured run times of our algorithm.

In order to account for variance, we computed 95%-confidence intervals using Student's t-distribution.

11 https://dbpedia.org



"Peter Koper (born 1947) is an American journalist, professor, screenwriter, and producer. He numbers among the original Dreamlanders, the group of actors and artists who worked with independent film maker John Waters on his early films. He has written for the Associated Press, the Baltimore Sun, American Film, Rolling Stone, and People. He worked as a staff writer and producer for America's Most Wanted, and has written television for the Discovery Channel, the Learning Channel, Paramount Television and Lorimar Television. Koper wrote and co-produced the cult movie Headless Body in Topless Bar, and wrote the screenplay for Island of the Dead. He has taught at the University of the District of Columbia, and Hofstra University."

Figure 4.4: **Performance Evaluation Setup.** (1) shows the SCBD used as the first source training dataset. (2) shows the SCBD used as the second source training dataset. (3) shows the target training dataset. (4) is the enrichment graph used to obtain (3) from (1) and (2). Within it, green nodes are dataset emitters, orange nodes are enrichment operators and the red node is the dataset acceptor. The letters in the orange nodes representing enrichment operators are the first letters of the corresponding enrichment operator, i. e.`N` for the NER operator, `A` for the authority conformation operator, `D` for the dereferencing operator, `L` for the linking operator, `P` for the predicate conformation operator and finally `F` for the filter operator.

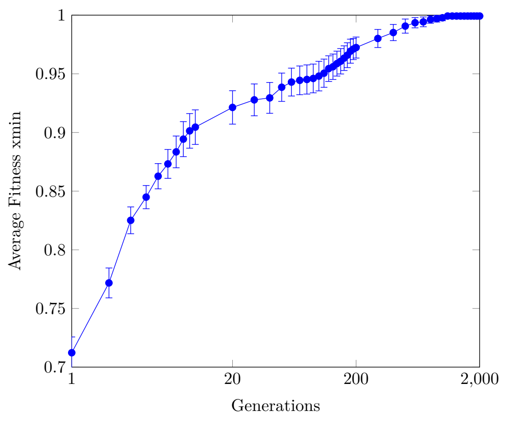


Figure 4.5: **Performance Evaluation Results.** We ran the performance evaluation experiment 50 times with a maximum number of generation of 2,000. The results suggest that the fitness generally grows rapidly over the first 20 generations. The generality of our results is underlined by the fact that the 95%-confidence intervals are always within 0.1 of the measured average fitness over all repetitions.

The results in Figure 4.5 suggest that our approach will on average solve this task with an overall solution quality of over 99% in under 500 generations. Note that our approach took on average 16 seconds to solve this problem with a minimum of 4 seconds and a maximum of 30 seconds.

5 CONCLUSION & FUTURE WORK

In this chapter we conclude this thesis by summarizing our contributions and verifying that we met the design goals and answered the research question we have defined in Section 1.3.

To remind the reader, our respective design goals and research questions were:

(G1) DEER2 should be highly modular

(G2) DEER2 should represent RDF dataset enrichment workflows efficiently as DAGs

(G3) DEER2 should include an optimized, GP based learning algorithm for automatic configuration of RDF dataset enrichment workflows

(G4) DEER2 should improve on all of the identified shortcomings of DEER.

(Q1) What is the optimal set of hyperparameters?

(Q2) How does our approach perform on real world datasets?

5.1 SUMMARY

In this thesis, we have developed a theory of DAG-shaped RDF dataset enrichment based on an analysis of a previous approach that did only consider linear enrichment specifications. Therefore, we regard design goal **(G2)** as met. Based on this theory, we implemented a tool for RDF dataset enrichment with a set of eleven standard enrichment operators. We demonstrated the modularity in accordance with the theory, which makes no assumptions on the nature of enrichment operators other than them operating on RDF data, thus satisfying design goal **(G1)**.

In order to narrow down the scope of this work, we introduced a classification of enrichment graphs and identified the subclass of inherently confluent enrichment graphs as the object of our study. Subsequently, we defined an efficient representation of *inherently confluent* enrichment graphs called enrichment tables, which we then used as starting point to define our Genetic Programming (GP)-based learning algorithm.

In order to improve the performance of our learning algorithm, we developed a method called genotype compaction alongside three semantic genetic operators, namely the graph merging crossover, the precondition mutation and the postcondition mutation, satisfying goal (**G3**). The pre- and postcondition broadcasting mutation operators cope with information masking and our novel fitness function can more adequatly measure small improvements in the learning process. Therefore, we also regard (**Q4**) as met.

In the evaluation chapter we rigorously identified the set of optimal hyperparameters offspring fraction $\alpha = 1.0$, mutation probability $\sigma = 0.5$ and mutation rate $\rho = 0.5$ for our full approach, therefore (**Q1**) is answered. Moreover, we have shown that our algorithm can perform well on real world datasets with an average execution time of about 16 seconds and a mean solution quality of 99% within a 95%-confidence interval of $\pm 0.6\%$ after 500 generations, thus giving an answer for (**Q2**).

5.2 FUTURE WORK

In future work, we will investigate research relating to the automated enrichment of RDF datasets in three directions.

The first research direction will be to develop a sound theory of self-configuration heuristics, which could e. g.make use of graph embedding techniques or variations on approximate solutions to the subgraph isomorphism problem.

Another direction of interest is to extend the class of enrichment graphs for which we investigate learning methods, eventually developing multi-objective algorithms. Such algorithms could be used in order to automate the provision of RDF datasets for different types of consumers given the same input datasets, thereby exploiting parallelism.

Finally, we would like to develop methods for improving the scalability of our approach in a big data setting.

BIBLIOGRAPHY

[1] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. "Semantic Enrichment of Twitter Posts for User Profile Construction on the Social Web." In: *The Semantic Web: Research and Applications*. Ed. by Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 375–389. ISBN: 978-3-642-21064-8.

[2] Najla Akram, Al-Saati, and Taghreed Riyadh Alreffaee. "Using Multi Expression Programming in Software Effort Estimation." In: *CoRR* abs/1805.00090 (2018). URL: https://dblp.org/rec/journals/corr/abs-1805-00090.

[3] M Altman. "A generalized gradient method of minimizing a functional." In: *Commentationes Mathematicae* 14.1 (1970).

[4] Ahmed Ben Ayed, Julien Subercaze, Frédérique Laforest, Tarak Chaari, Wajdi Louati, and Ahmed Hadj Kacem. "Docker2RDF: Lifting the Docker Registry Hub into RDF." In: *SERVICES*. IEEE Computer Society, 2017, pp. 36–39. DOI: 10.1109/SERVICES.2017.15. URL: https://dblp.org/rec/conf/services/AyedSLCLK17.

[5] Tim Berners-Lee. *Linked Data*. July 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html (visited on 07/12/2019).

[6] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The Semantic Web." In: *Scientific american* 284.5 (2001), pp. 28–37.

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked Data - The Story So Far." In: *Int. J. Semantic Web Inf. Syst.* 5.3 (2009), pp. 1–22. DOI: 10.4018/JSWIS.2009081901. URL: https://dblp.org/rec/journals/ijswis/BizerHB09.

[8] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. "Linked data on the web (LDOW2008)." In: *WWW*. ACM, 2008, pp. 1265–1266. DOI: 10.1145/1367497.1367760. URL: https://dblp.org/rec/conf/www/BizerHIB08.

[9] Christian Bizer and Andreas Schultz. "The R2R Framework: Publishing and Discovering Mappings on the Web." In: *Proceedings of the First International Workshop on Consuming Linked Data, Shanghai, China, November 8, 2010*. Ed. by Olaf Hartig, Andreas Harth, and Juan F. Sequeda. Vol. 665. CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[10] Volha Bryl and Christian Bizer. "Learning conflict resolution strategies for cross-language Wikipedia data fusion." In: *WWW (Companion Volume)*. ACM, 2014, pp. 1129–1134. DOI: 10.1145/2567948.2578999. URL: https://dblp.org/rec/conf/www/BrylB14.

[11] Qi Chen, Bing Xue, Yi Mei, and Mengjie Zhang. "Geometric Semantic Crossover with an Angle-Aware Mating Scheme in Genetic Programming for Symbolic Regression." In: *EuroGP*. Vol. 10196. Lecture Notes in Computer Science. 2017, pp. 229–245. DOI: 10.1007/978-3-319-55696-3_15. URL: https://dblp.org/rec/conf/eurogp/ChenXMZ17.

[12] Smitashree Choudhury, John G. Breslin, and Alexandre Passant. "Enrichment and Ranking of the YouTube Tag Space and Integration with the Linked Data Cloud." In: *The Semantic Web - ISWC 2009*. Ed. by Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 747–762. ISBN: 978-3-642-04930-9.

[13] Benjamin Doerr, Edda Happ, and Christian Klein. "Crossover can provably be useful in evolutionary computation." In: *Theor. Comput. Sci.* 425 (2012), pp. 17–33. DOI: 10.1016/J.TCS.2010.10.035. URL: https://dblp.org/rec/journals/tcs/DoerrHK12.

[14] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. "Knowledge vault: a web-scale approach to probabilistic knowledge fusion." In: *KDD*. ACM, 2014, pp. 601–610. DOI: 10.1145/2623330.2623623. URL: https://dblp.org/rec/conf/kdd/0001GHHLMSSZ14.

[15] Kevin Dreßler and Axel-Cyrille Ngonga Ngomo. "On the Efficient Execution of Bounded Jaro-Winkler Distances." In: *Proceedings of Ontology Matching Workshop*. 2014.

[16] Heshan Du, Natasha Alechina, Michael Jackson, and Glen Hart. “Matching geospatial instances.” In: *OM*. Vol. 1111. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 239–240. URL: https://dblp.org/rec/conf/semweb/DuAJH13.

[17] Jianhua Feng, Jiannan Wang, and Guoliang Li. “Trie-join: a trie-based method for efficient string similarity joins.” In: *VLDB J.* 21.4 (2012), pp. 437–461. DOI: 10.1007/S00778-011-0252-8. URL: https://dblp.org/rec/journals/vldb/FengWL12.

[18] Cândida Ferreira. “Gene Expression Programming: A New Adaptive Algorithm for Solving Problems.” In: *Complex Systems* 13.2 (2001). URL: https://dblp.org/rec/journals/compsys/Ferreira01.

[19] Manuel Fiorelli, Tiziano Lorenzetti, Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. “Sheet2RDF: a Flexible and Dynamic Spreadsheet Import&Lifting Framework for RDF.” In: *IEA/AIE*. Vol. 9101. Lecture Notes in Computer Science. Springer, 2015, pp. 131–140. DOI: 10.1007/978-3-319-19066-2_13. URL: https://dblp.org/rec/conf/ieaaie/FiorelliLPST15.

[20] Stefan Forstenlechner, David Fagan, Miguel Nicolau, and Michael O’Neill. “Semantics-Based Crossover for Program Synthesis in Genetic Programming.” In: *Artificial Evolution*. Vol. 10764. Lecture Notes in Computer Science. Springer, 2017, pp. 58–71. DOI: 10.1007/978-3-319-78133-4_5. URL: https://dblp.org/rec/conf/ae/ForstenlechnerF17.

[21] Kleanthi Georgala, Mohamed Ahmed Sherif, and Axel-Cyrille Ngonga Ngomo. “An Efficient Approach for the Generation of Allen Relations.” In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI) 2016, The Hague, 29. August - 02. September 2016*. 2016. URL: http://svn.aksw.org/papers/2016/ECAI_AEGLE/public.pdf.

[22] David E. Goldberg and Kalyanmoy Deb. “A Comparative Analysis of Selection Schemes Used in Genetic Algorithms.” In: *FOGA*. Morgan Kaufmann, 1990, pp. 69–93. URL: https://dblp.org/rec/conf/foga/GoldbergD90.

[23] M. Graff, E. S. Tellez, S. Miranda-Jiménez, and H. J. Escalante. “EvoDAG: A semantic Genetic Programming Python library.” In: *2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*. Nov. 2016, pp. 1–6. DOI: 10.1109/ROPEC.2016.7830633.

[24] RDF Working Group. *Resource Description Framework (RDF)*. Feb. 2014. URL: https://www.w3.org/RDF/ (visited on 07/12/2019).

[25] Souleiman Hasan, Edward Curry, Mauricio Banduk, and Sean O'Riain. "Toward situation awareness for the semantic sensor web: Complex event processing with dynamic linked data enrichment." In: *Proceedings of the 4th International Conference on Semantic Sensor Networks-Volume 839*. CEUR-WS. org. 2011, pp. 69–82.

[26] Robert Isele. "Learning expressive linkage rules for entity matching using genetic programming." PhD thesis. Universität Mannheim, 2013. URL: https://dblp.org/rec/books/daglib/0033457.

[27] Robert Isele, Anja Jentzsch, and Christian Bizer. "Efficient Multidimensional Blocking for Link Discovery without losing Recall." In: *WebDB*. 2011. URL: https://dblp.org/rec/conf/webdb/IseleJB11.

[28] Alexandros Karakasidis and Vassilios S. Verykios. "A Highly Efficient and Secure Multidimensional Blocking Approach for Private Record Linkage." In: *ICTAI*. IEEE Computer Society, 2012, pp. 428–435. DOI: 10.1109/ICTAI.2012.65. URL: https://dblp.org/rec/conf/ictai/KarakasidisV12.

[29] John R Koza. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Vol. 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

[30] Vladimír Kvasnièka and Jiøí Pospíchal. "Simple Implementation of Genetic Programming by Column Tables." In: *Soft Computing in Engineering Design and Manufacturing*. Ed. by P. K. Chawdhry, R. Roy, and R. K. Pant. London: Springer London, 1998, pp. 48–56. ISBN: 978-1-4471-0427-8.

[31] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. "Sieve: linked data quality assessment and fusion." In: *EDBT/ICDT Workshops*. ACM, 2012, pp. 116–123. DOI: 10.1145/2320765.2320803. URL: https://dblp.org/rec/conf/edbt/MendesMB12.

[32] Brad L. Miller and David E. Goldberg. "Genetic Algorithms, Tournament Selection, and the Effects of Noise." In: *Complex Systems* 9.3 (1995). URL: https://dblp.org/rec/journals/compsys/MillerG95.

[33] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998. URL: https://dblp.org/rec/books/daglib/0019083.

[34] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. "Lifting Data Portals to the Web of Data." In: *LDOW@WWW*. CEUR Workshop Proceedings. CEUR-WS.org, 2017. URL: https://dblp.org/rec/conf/www/NeumaierUP17.

[35] Axel-Cyrille Ngonga Ngomo and Sören Auer. "LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data." In: *IJCAI*. IJCAI/AAAI, 2011, pp. 2312–2317. DOI: 10.5591/978-1-57735-516-8/IJCAI11-385. URL: https://dblp.org/rec/conf/ijcai/NgomoA11.

[36] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. "EAGLE: Efficient Active Learning of Link Specifications using Genetic Programming." In: *Proceedings of ESWC*. 2012.

[37] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. "Unsupervised learning of link specifications: deterministic vs. non-deterministic." In: *Proceedings of the Ontology Matching Workshop*. 2013.

[38] Axel-Cyrille Ngonga Ngomo, Klaus Lyko, and Victor Christen. "COALA—Correlation-Aware Active Learning of Link Specifications." In: *Proceedings of ESWC*. 2013.

[39] Andriy Nikolov, Victoria S. Uren, and Enrico Motta. "KnoFuss: a comprehensive architecture for knowledge fusion." In: *K-CAP*. ACM, 2007, pp. 185–186. DOI: 10.1145/1298406.1298446. URL: https://dblp.org/rec/conf/kcap/NikolovUM07.

[40] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. "Handling Instance Coreferencing in the KnoFuss Architecture." In: *IRSW*. Vol. 422. CEUR Workshop Proceedings. CEUR-WS.org, 2008. URL: https://dblp.org/rec/conf/esws/NikolovUMR08.

[41] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. "Integration of Semantically Annotated Data by the KnoFuss Architecture." In: *EKAW*. Vol. 5268. Lecture Notes in Computer Science. Springer, 2008, pp. 265–274. DOI: 10.1007/978-3-540-87696-0_24. URL: https://dblp.org/rec/conf/ekaw/NikolovUMR08.

[42] Mihai Oltean and D. Dumitrescu. “Evolving TSP Heuristics Using Multi Expression Programming.” In: *International Conference on Computational Science*. Vol. 3037. Lecture Notes in Computer Science. Springer, 2004, pp. 670–673. DOI: 10.1007/978-3-540-24687-9_99. URL: https://dblp.org/rec/conf/iccS/OlteanD04.

[43] Mihai Oltean and Dumitru Dumitrescu. “Evolving TSP heuristics using Multi Expression Programming.” In: *CoRR* abs/1509.02459 (2015). URL: https://dblp.org/rec/journals/corr/OlteanD15.

[44] Mihai Oltean and Crina Grosan. “Evolving Evolutionary Algorithms Using Multi Expression Programming.” In: *ECAL*. Vol. 2801. Lecture Notes in Computer Science. Springer, 2003, pp. 651–658. DOI: 10.1007/978-3-540-39432-7_70. URL: https://dblp.org/rec/conf/ecal/OlteanG03.

[45] Dunlu Peng, Shaohong Wu, and Cong Liu. “MPSC: A Multiple-Perspective Semantics-Crossover Model for Matching Sentences.” In: *IEEE Access* 7 (2019), pp. 61320–61330. DOI: 10.1109/ACCESS.2019.2915937. URL: https://dblp.org/rec/journals/access/PengWL19.

[46] Yutaka Sasaki et al. “The truth of the F-measure.” In: *Teach Tutor mater* 1.5 (2007), pp. 1–5.

[47] Bernhard Schandl and Niko Popitsch. “Lifting File Systems into the Linked Data Cloud with TripFS.” In: *LDOW*. Vol. 628. CEUR Workshop Proceedings. CEUR-WS.org, 2010. URL: https://dblp.org/rec/conf/www/SchandlP10.

[48] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. “The Semantic Web Revisited.” In: *IEEE Intelligent Systems* 21.3 (2006), pp. 96–101. DOI: 10.1109/MIS.2006.62. URL: https://dblp.org/rec/journals/expert/ShadboltBH06.

[49] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. “RADON - Rapid Discovery of Topological Relations.” In: *Proceedings of The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. 2017. URL: https://svn.aksw.org/papers/2017/AAAI_RADON/public.pdf.

[50] Mohamed Ahmed Sherif, Axel-Cyrille Ngonga Ngomo, and Jens Lehmann. “Wombat - A Generalization Approach for Automatic Link Discovery.” In: *ESWC (1)*. Vol. 10249. Lecture Notes in Computer Science. 2017, pp. 103–119. DOI:

10.1007/978-3-319-58068-5_7. URL: https://dblp.org/rec/conf/esws/SherifNL17.

[51] Mohamed Ahmed Sherif, Axel-Cyrille Ngonga Ngomo, and Jens Lehmann. "Automating RDF Dataset Transformation and Enrichment." In: *12th Extended Semantic Web Conference, Portorož, Slovenia, 31st May - 4th June 2015*. Springer, 2015.

[52] Vivek R. Shivaprabhu, Booma Sowkarthiga Balasubramani, and Isabel F. Cruz. "Ontology-based Instance Matching for Geospatial Urban Data Integration." In: *UrbanGIS@SIGSPATIAL*. ACM, 2017, 8:1–8:8. DOI: 10.1145/3152178.3152186. URL: https://dblp.org/rec/conf/gis/ShivaprabhuBC17.

[53] William M. Spears and Vic Anand. "A Study of Crossover Operators in Genetic Programming." In: *ISMIS*. Vol. 542. Lecture Notes in Computer Science. Springer, 1991, pp. 409–418. DOI: 10.1007/3-540-54563-8_104. URL: https://dblp.org/rec/conf/ismis/SpearsA91.

[54] William M. Spears and Kenneth A. De Jong. "An Analysis of Multi-Point Crossover." In: *FOGA*. Morgan Kaufmann, 1990, pp. 301–315. URL: https://dblp.org/rec/conf/foga/SpearsJ90.

[55] René Speck and Axel-Cyrille Ngonga Ngomo. "Named Entity Recognition using FOX." In: *International Semantic Web Conference (Posters & Demos)*. Vol. 1272. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 85–88. URL: https://dblp.org/rec/conf/semweb/SpeckN14a.

[56] Roland Stühmer, Darko Anicic, Sinan Sen, Jun Ma, Kay-Uwe Schmidt, and Nenad Stojanovic. "Lifting Events in RDF from Interactions with Annotated Web Pages." In: *International Semantic Web Conference*. Vol. 5823. Lecture Notes in Computer Science. Springer, 2009, pp. 893–908. DOI: 10.1007/978-3-642-04930-9_56. URL: https://dblp.org/rec/conf/semweb/StuhmerASMSS09.

[57] Ranyart Rodrigo Suárez, Mario Graff, and Juan J. Flores. "Semantic Crossover Operator for GP based on the Second Partial Derivative of the Error Function." In: *Research in Computing Science* 94 (2015), pp. 87–96. URL: https://dblp.org/rec/journals/rcs/SuarezGF15.

[58] Gilbert Syswerda. "Uniform Crossover in Genetic Algorithms." In: *ICGA*. Morgan Kaufmann, 1989, pp. 2–9. URL: https://dblp.org/rec/conf/icga/Syswerda89.

[59] Diane E. Vaughan, Sheldon H. Jacobson, Shane N. Hall, and Laura A. McLay. "Simultaneous Generalized Hill-Climbing Algorithms for Addressing Sets of Discrete Optimization Problems." In: *INFORMS Journal on Computing* 17.4 (2005), pp. 438–450. DOI: `10.1287/IJOC.1040.0064`. URL: `https://dblp.org/rec/journals/informs/VaughanJHM05`.

[60] Chuan Xiao, Wei Wang, and Xuemin Lin. "Ed-Join: an efficient algorithm for similarity joins with edit distance constraints." In: *PVLDB* 1.1 (2008), pp. 933–944. DOI: `10.14778/1453856.1453957`. URL: `https://dblp.org/rec/journals/pvldb/XiaoWL08`.

[61] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. "Efficient similarity joins for near-duplicate detection." In: *ACM Trans. Database Syst.* 36.3 (2011), 15:1–15:41. DOI: `10.1145/2000824.2000825`. URL: `https://dblp.org/rec/journals/tods/XiaoWLYW11`.

[62] Qingke Zhang, Bo Yang, Lin Wang, and Jianzhang Jiang. "An improved multi-expression programming algorithm applied in function discovery and data prediction." In: *IJICT* 5.3/4 (2013), pp. 218–233. DOI: `10.1504/IJICT.2013.054952`. URL: `https://dblp.org/rec/journals/ijict/ZhangYWJ13`.

ERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, 15. Juli 2019

Kevin Dreßler