Abstract: This deliverable provides an overview of the novel scalable version of GeoLift. After a study of the use of parallel architectures for link discovery, we show how LIMES and FOX were extended to make use of the results of this study to ensure the scalability of GeoLift.

The information in this document reflects only the author’s views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided “as is” without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/her sole risk and liability.

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)
History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>18/08/2014</td>
<td>First draft created</td>
<td>Axel-Cyrille Ngonga Ngomo</td>
</tr>
<tr>
<td>0.1</td>
<td>23/08/2014</td>
<td>First complete version created</td>
<td>Mohamed Ahmed Sherif</td>
</tr>
<tr>
<td>0.2</td>
<td>25/08/2014</td>
<td>First review</td>
<td>Axel-Cyrille Ngonga Ngomo</td>
</tr>
<tr>
<td>0.3</td>
<td>27/08/2014</td>
<td>Revised version</td>
<td>Mohamed Ahmed Sherif</td>
</tr>
<tr>
<td>0.4</td>
<td>28/09/2014</td>
<td>Second review</td>
<td>Giorgos Giannopoulos</td>
</tr>
<tr>
<td>1.0</td>
<td>30/09/2014</td>
<td>Final version</td>
<td>Mohamed Ahmed Sherif</td>
</tr>
</tbody>
</table>

Author List

<table>
<thead>
<tr>
<th>Organization</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFAI</td>
<td>Axel-Cyrille Ngonga Ngomo</td>
<td><a href="mailto:ngonga@informatik.uni-leipzig.de">ngonga@informatik.uni-leipzig.de</a></td>
</tr>
<tr>
<td>INFAI</td>
<td>Mohamed Ahmed Sherif</td>
<td><a href="mailto:sherif@informatik.uni-leipzig.de">sherif@informatik.uni-leipzig.de</a></td>
</tr>
<tr>
<td>INFAI</td>
<td>René Speck</td>
<td><a href="mailto:speck@informatik.uni-leipzig.de">speck@informatik.uni-leipzig.de</a></td>
</tr>
</tbody>
</table>
Executive Summary

Processing the large amount of geo-spatial data available in the Linked Open Data Cloud as well as in company repositories requires approaches that scale to millions of resources. To address these requirements, data integration frameworks must implement time-efficient and parallel algorithms that allow dealing with such large amounts of data. In this deliverable, we begin by studying different parallel architectures for the time-efficient integration of linked data sets. Our results (which won the Best Research Paper award at ESWC 2013) suggest that using cloud-based approaches is actually inferior to relying on local resources. We thus present scalability extensions of our integration framework GeoLift which are based on local parallel architectures. To this end, we first present the extensions of the GeoLift workflow. Thereafter, we present and evaluate parallel implementations of Orchid, the first geo-linking algorithm. We then present our scalable implementation of the Jaro-Winkler metric, a measure that was designed especially to compare names, which build the second most important component of link specifications for geo-spatial data. We also present LIMES’ novel planner, which allows our link discovery framework to run in 50% of its previous runtime on geo-spatial data. Finally, we present our extension of the FOX framework with respect to both accuracy and runtime. Overall, our extensions can clearly scale to the millions of resources that need to be processed when dealing with large geo-spatial datasets provided they are given enough memory to run. An evaluation of the framework on the evaluation partners’ data will be presented in D3.1.3.

The new versions of the framework can be found at the following URLs:

- GeoLift: http://github.com/GeoKnow/GeoLift
- LIMES: http://github.com/AKSW/LIMES
- FOX: http://github.com/AKSW/FOX
### Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBD</td>
<td>Concise Bounded Description</td>
</tr>
<tr>
<td>FOX</td>
<td>Federated knOwledge eXtraction Framework</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>NER</td>
<td>Named entity recognition</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>LIMES</td>
<td>Link Discovery Framework for Metric Spaces</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
</tbody>
</table>
Contents

1 Introduction ................................................................. 10

2 When to Reach for the Cloud: A Quantitative Study ......................... 11
   2.1 Preliminaries ................................................................. 11
   2.2 Link Discovery on GPUs .................................................. 12
      2.2.1 General-Purpose Computing on GPUs ......................... 12
      2.2.2 GPU-based $HR^3$ Implementation ................................ 13
   2.3 MapReduce-based Link Discovery ....................................... 15
      2.3.1 $HR^3$ with MapReduce ............................................. 15
      2.3.2 $HR^3$ with Load Balancing ........................................ 16
   2.4 Evaluation ...................................................................... 18
      2.4.1 Experimental Setup .................................................... 18
      2.4.2 Performance Comparison ............................................. 19
      2.4.3 Scalability: Data Size ................................................ 20

3 GeoLift ............................................................................ 21
   3.1 Introduction .................................................................. 21
   3.2 GeoLift Components ...................................................... 21
      3.2.1 Modules .................................................................. 21
         3.2.1.1 Dereferencing Module ......................................... 21
         3.2.1.2 Linking Module .................................................. 21
         3.2.1.3 NLP Module ....................................................... 22
         3.2.1.4 Conformation Module ......................................... 22
         3.2.1.5 Filter Module ...................................................... 23
      3.2.2 Operators .................................................................. 23
         3.2.2.1 Split Operator ...................................................... 23
         3.2.2.2 Merge Operator .................................................. 23
   3.3 GeoLift RDF Specification Paradigm .................................... 23
      3.3.1 GeoLift Execution Workflow ....................................... 24
      3.3.2 RDF Specification Resources ....................................... 24
         3.3.2.1 Dataset Resource ................................................ 24
         3.3.2.2 Module Resource ................................................ 24
         3.3.2.3 Operator Resource ............................................... 26
4 Scalability Improvements

4.1 Orchid

4.1.1 Overview

4.1.2 Load Balancing Algorithms

4.1.2.1 Naïve Load Balancer

4.1.2.2 Greedy Load Balancer

4.1.2.3 Pair-Based Load Balancer

4.1.2.4 Particle Swarm Optimization (PSO)

4.1.2.5 DPSO Load Balancer

4.1.3 Evaluation

4.1.3.1 Experimental Setup

4.1.3.2 Orchid vs. Parallel Orchid

4.1.3.3 Parallel Load balancing Algorithms Evaluation

4.1.3.4 DPSO Evaluation

4.2 Jaro-Winkler

4.2.1 The Jaro-Winkler Similarity

4.2.2 Improving the Runtime of Bounded Jaro-Winkler

4.2.2.1 Length-based filters

4.2.2.2 Filtering ranges by length

4.2.2.3 Filtering by character frequency

4.2.3 Evaluation

4.2.3.1 Runtime Evaluation

4.2.3.2 Scalability Evaluation

4.2.3.3 Comparison with existing approaches

4.3 HELIOS

4.3.1 Formal Specification

4.3.1.1 Normal Form for Link Specifications

4.3.1.2 Execution Plans

4.3.2 HELIOS Components

4.3.2.1 The HELIOS Rewriter

4.3.2.2 The HELIOS Planner

4.3.3 Evaluation
## List of Figures

1. Example dataset containing 11 places from Leipzig. To identify all points with a maximum Euclidean distance $\theta = 0.02$, the space is virtually tiled into hypercubes with an edge length of $\Delta = \theta/4$. A cube is identified by its coordinates $(c_1, ..., c_n)$. The gray-shadowed cells indicate the cubes whose points are compared with $B$, i.e., $\{C' | \text{index}(C(B), C') \leq \alpha^p\}$.

2. OpenCL memory model

3. Result index computation for $\mathcal{HR}^3$ on GPU hardware

4. Overview of the MR-based $\mathcal{HR}^3$ implementation with load balancing (left) and the cube population matrix for the example dataset with $m = 2$ (right)

5. Match task creation and reduce task assignment with/without splitting of large tasks (left). Example data flow for second MR job (right)

6. Datasets used for evaluation

7. Comparison of runtimes for Experiment 1

8. Comparison of runtimes on DS4

9. Content Negotiation as used by GeoLift (courtesy of W3C)

10. Example of GeoLift RDF specification (Turtle serialization)

11. Graph representation of configuration file presented by listing 10

12. GeoHR3 vs. Parallel GeoHR3

13. Runtimes of applying different parallel load balancing algorithms to DBpedia and LGD

14. DPSO 10 iteration evaluation on DBpedia and LGD

15. Runtime comparison on input size 1000, scaling threshold

16. Comparison of number of similarity computations on input size 1000, scaling threshold

17. Runtimes on sample of DBpedia labels with size $10^4$, scaling threshold

18. Runtimes on sample of DBpedia labels with size $10^5$, scaling threshold

19. Runtimes with multiple thresholds $\theta$ for growing input sizes

20. A LS for linking the datasets Person1 and Person2. The filter nodes are rectangles while the operator nodes are circles. $\text{eucl}(s.\text{age}, t.\text{age}) = (1 + |s.\text{age} - t.\text{age}|)^{-1}$. This LS can be transliterated $i(\cap(f(\text{edit}(\text{label, label}), 0.3), f(\text{eucl}(\text{age, age}), 0.5)), 0.5)$.

21. Leaf generation rule for linear combinations

22. Rule for minimum. In the corresponding rule for maximum, the mapping union is used

23. Example of propagation of dependencies. The dashed arrows represent dependencies. The dependencies from the left figure are first (using rule $p_1$). Then, the reduction rule $r_2$ is carried out, leading to the specification on the right.

24. Runtimes achieved by PPJoin+ (trigrams) and EDJoin (levenshtein) for $\theta = 0.5$. The x-axis of the heatmap show $|S|$ in thousands, while the y-axis shows $|T|$ in thousands. The color bars show the runtime in ms.

25. Cumulative runtimes on DBLP-ACM and LGD-LGD.
26 Workflow chart of the evaluation pipeline. .......................................................... 56
27 News and News* dataset. .................................................................................. 66
28 Web and Reuters dataset. ................................................................................. 67
29 All dataset. ....................................................................................................... 68
List of Tables

1  Bounds for distinct string lengths (θ = 0.9) ................................................................. 38
2  Calculation of $m_{\text{max}}$ .......................................................................................... 39
3  Runtimes (in seconds) of our approach (OA) and SILK 2.6.0 ........................................... 42
4  Exemplary graphs. .......................................................................................................... 44
5  Plans for the specification shown in Figure 20 .................................................................. 46
6  Comparison of runtimes on manual specifications. The top portion of the table shows runtimes of specifications of size 1 while the bottom part shows runtimes on specifications of size 3. EVT stands for Eventseeer, DF for DogFood, (P) stands for person, (A) stands for airports, (U) stands for universities, (E) stands for events. The best runtimes are in bold. ................................................................. 53
7  Summary of the results on on automatically generated specifications. $|L|$ shows for the average size ± standard deviation of the specifications in the experiment. $F_1$ shows the F-measure achieved by EAGLE on the dataset. The runtimes in four rightmost columns are the average runtimes in seconds. ..................................................................................................................... 54
8  Number of entities separated according entity types and in total. .................................... 57
9  News$^*$ token-based. ...................................................................................................... 59
10 News$^*$ entity-based. ...................................................................................................... 59
11 News token-based. .......................................................................................................... 60
12 News entity-based. .......................................................................................................... 60
13 Web token-based. .......................................................................................................... 61
14 Web entity-based. .......................................................................................................... 61
15 Reuters token-based. ...................................................................................................... 62
16 Reuters entity-based. ...................................................................................................... 62
17 All token-based. ............................................................................................................ 63
18 All entity-based. ............................................................................................................ 63
19 F-score of the best 3 classifiers on class level token-based. ............................................. 64
20 F-score of the best 3 classifiers on class level entity-based. ............................................ 65
21 Runtime evaluation of FOX. ........................................................................................... 65
1 Introduction

One of the main aims of GeoKnow is to develop scalable approaches for processing geo-spatial data. With respect to data integration, this means developing an integration framework that can deal with millions of resources. The goal of this deliverable is to present exactly such a framework, i.e., the new version of GeoLift. In addition to supporting very complex workflows based on a novel ontology, the new version of GeoLift relies on LIMES 0.6.3 to achieve geo-spatial link discovery. This new version of LIMES brings two major upgrades: First, the algorithms used for geo-spatial link discovery were implemented in parallel. The paradigm upon which we rely for the parallel implementation is based on a preliminary study presented in [42]. Second, LIMES implements the first planner for link discovery, which was shown in [41] to improve the overall runtime of geo-spatial link discovery by approximately 50%. In addition, the new version of LIMES includes implementations of the Jaro and Jaro-Winkler similarities, which have been shown to achieve high accuracies on names (such as geo-location names) as presented in [13] of the named entity recognition framework used by GeoLift, i.e., FOX [61] has now been extended to achieve better F-measures as well as to scale better. To this end, a parallel implementation was also used.

In the following, we begin by presenting our study of different means to achieve scalable data integration through parallel processing (Section 2). Here, we compare the runtimes of parallel processing approaches based on cloud infrastructures (Amazon), threads and GPUs. Our results amened us to decide for opt for a thread-based infrastructure for all other developments. We then present the new version of GeoLift (Section 3) including the new vocabulary that underlies the new GeoLift-specification paradigm. The key developments w.r.t. scalability and partly accuracy are then presented. We begin by presenting the parallel implementation of ORCHID (Section 4.1) and show that our parallel implementation improves the runtime of ORCHID significantly. In addition, we present our novel implementation of bounded Jaro-Winkler measures (Section 4.2). Thereafter, we present Helios, a novel rewriter and planner for link specifications that can further reduce the runtime of link discovery (Section 4.3). We round up the scalability extensions by presenting the new version of FOX and its accuracy and runtime improvements [61].
2 When to Reach for the Cloud: A Quantitative Study

In this section, we study the use of parallel architectures for RDF data integration. The complete study was presented in [42] and won the best Research Paper Award as it is the first study to quantify when we actually need to use the Cloud for scalability. We focused on LD in the study because it is of central importance for data integration as well as for realizing the fourth Linked Data principle [3]. With the growth of the Web of Data, the complexity of LD problems has grown considerably. For example, linking places from LinkedGeoData and DBpedia requires the comparison of hundreds of thousands of instances. Over the last years, several time-efficient algorithms such as LIMES [48], MultiBlock [26], HR³ [46] and Orchid [40] have been developed to address the problem of the a-priori quadratic runtime of LD approaches. In general, these algorithms aim at minimizing the number of unnecessary similarity computations to carry out. While these approaches have been shown to outperform naive LD implementations by several orders of magnitude, the sheer size of the number of links can still lead to unpractical runtimes. Thus, cloud implementations of some of these algorithms (e.g., LIMESMR [24] and Silk MapReduce¹) have been recently developed. The speed-up of these implementations is, however, limited by a considerable input-output overhead that can lead to worse runtimes than on single machines. Interestingly, the use of standard parallel hardware has recently been shown to have the potential to outperform cloud computing techniques [23].

The multiplicity of available hardware solutions for carrying out LD led us to ask the following fundamental question: When should which type of hardware be used to optimize the runtime of LD processes? Providing an answer to this question promises to enable the development of highly flexible and scalable LD and RDF integration frameworks that can adapt to the available hardware environment. It will allow to decide intelligently upon when to reach for remote computing services such as cloud computing services in contrast to using local resources such as graphics processing units (GPUs) or multi-processor and multi-core technology. To answer our research question, we compared the runtimes of several implementations of HR³ for several geo-spatial datasets and found break-even points for different hardware. We chose the HR³ algorithm because it is the first algorithm with a guaranteed reduction ratio [46]. Thus, it promises to generate less overhead than other LD algorithms for comparable problems. Moreover, this algorithm can be used in manifold scenarios including LD, finding geographically related data (radial search) as well as search space reduction for other LD algorithms. The main contributions of this work are:

- We present the first implementation of a LD approach for GPUs. It relies on the GPU for fast parallel indexing and on the CPU for the computation of distances.
- We show how load-balancing for Map-Reduce can be carried out for LD approaches in affine spaces.
- We obtain guidelines for the use of different parallel hardware for LD by the means of a comparative evaluation of different implementations on real-world datasets from the Linked Open Data Cloud.

2.1 Preliminaries

The specification of link discovery adopted herein is tantamount to the definition proposed in [46]. Given a formal relation² R and two (not necessarily disjoint) sets of instances S and T, the goal of link discovery is to find the set \( M = \{(s, t) \in S \times T : R(s, t)\} \). Given that the explicit computation of R is usually a very complex endeavor, most frameworks reduce the computation of M to that of the computation of an approximation \( \hat{M} = \{(s, t) : \delta(s, t) \leq \theta\} \), where \( \delta \) is a (complex) distance function and \( \theta \) is a distance threshold. Note that when \( S = T \) and \( R = \text{owl:sameAs} \), the link discovery task becomes a deduplication

¹https://www.assembla.com/spaces/silk/wiki/Silk_MapReduce
²For example, http://dbpedia.org/property/near
task. Naive approaches to computing $M$ have a quadratic time complexity, which is impracticable on large datasets. Consequently, a large number of approaches has been developed to reduce this time complexity (see [46] for an overview). Most of these approaches achieve this goal by optimizing their reduction ratio. In newer literature, the $HR^3$ algorithm [44] has been shown to be the first algorithm which guarantees that it can achieve any possible reduction ratio.

$HR^3$ builds upon the HYPPO algorithm presented in [43]. The rationale of $HR^3$ is to maximize the reduction ratio of the computation of $M$ in affine spaces with Minkowski measures. To achieve this goal, $HR^3$ computes an approximation of $M$ within a discretization of the space $\Omega = S \cup T$. Each point $\omega = (\omega_1, \ldots, \omega_n) \in \Omega$ is mapped to discrete coordinates $(|\omega_1/\Delta|, \ldots, |\omega_n/\Delta|)$, where $\Delta = \theta/\alpha$ and $\alpha \in \mathbb{N}\{0\}$ is called the granularity parameter. An example of such a discretization is shown in Figure 1: The point $B$ with coordinates $(12.3436, 51.3339)$ is mapped to the discrete coordinates $(2468, 10226)$. The set of all points with the same discrete coordinates forms a hypercube (short: cube) of width $\alpha$ in the space $\Omega$. The cube that contains $\omega$ is called $C(\omega)$. We call the vector $(c_1, \ldots, c_n) = (|\omega_1/\Delta|, \ldots, |\omega_n/\Delta|) \in \mathbb{N}^n$ the coordinates of $C(\omega)$.

Given the distance threshold $\theta$ and the granularity parameter $\alpha$, $HR^3$ computes the set of candidates $t \in T$ for each $s \in S$ by using the index function given in Eq. 1.

$$\text{index}(C, C') = \begin{cases} 0, & \text{if } \exists i : |c_i - c'_i| \leq 1 \text{ with } i \in \{1, \ldots, n\}, \\ \sum_{i=1}^n (|c_i - c'_i| - 1)^p & \text{else}. \end{cases}$$ (1)

where $C = C(s)$ and $C' = C(t)$ are hypercubes and $p$ is the order of the Minkowski measure used in the space $\Omega$.

Now, all source instances $s$ are only compared with the target instances $t$ such that $\text{index}(C(s), C(t)) \leq \alpha^p$. In our example, this is equivalent to computing the distance between $B$ and all points contained in the gray-shadowed area on the map. Overall, $HR^3$ achieve a reduction ratio of $\approx 0.82$ on the data in Figure 1 as it only performs 10 comparisons instead of 55.

2.2 Link Discovery on GPUs

2.2.1 General-Purpose Computing on GPUs

GPUs were originally developed for processing image data. Yet, they have been employed for general-purpose computing tasks in recent years. Compared to CPUs the architecture of GPU hardware exhibits a large number of simpler compute cores and is thus referred to as massively parallel. A single compute core typically contains several arithmetic and logic units (ALU) that execute the same instruction on multiple data streams (SIMD).

Parallel code on GPUs is written as compute kernels, the submission of which is orchestrated by a host program executed on the CPU. Several frameworks exist for performing general purpose computing on GPUs. In this work we use OpenCL\(^3\), a vendor-agnostic industry standard. The memory model as exposed to OpenCL kernels is depicted in Figure 2: An instance of a compute kernel running on a device is called a work item or simply thread\(^4\). Work items are combined into work groups. All items within the same group have access to low-latency local memory and the ability to synchronize load/store operations using barriers. Thus, the actual number of kernel instances running in parallel is often limited by register and local memory usage. Each work item is assigned a globally (among all work items) and locally (within a work group) unique identifier, which also imposes a scheduling order. Typically those identifiers are used to compute local and global memory

---

\(^3\)http://www.khronos.org/opencl/

\(^4\)We use the terms work item and thread interchangeably in this work.
Figure 1: Example dataset containing 11 places from Leipzig. To identify all points with a maximum Euclidean distance $\theta = 0.02$, the space is virtually tiled into hypercubes with an edge length of $\Delta = \theta/4$. A cube is identified by its coordinates $(c_1, ..., c_n)$. The gray-shadowed cells indicate the cubes whose points are compared with $B$, i.e., $\{C' \mid \text{index}(C(B), C') \leq \alpha\}$.

offsets for loading and storing data items that a given thread works on. Data transfer between host program and compute device is done via global device memory to which all work items have access, albeit with higher latency.

Threads on modern GPUs do not run in isolation. They are scheduled in groups of 64 or 32 work items depending on the hardware vendor. All threads within such a group execute the same instruction in lock-step. Any code path deviations due to control flow statements need to be executed by all items, throwing away unnecessary results (predication). It is therefore essential that each work item in such a group performs the same amount of work. The OpenCL framework does not expose the size of such groups to the API user. An upper bound is given by the work group size, which is always an integer multiple of the schedule group size.

2.2.2 GPU-based $\mathcal{HR}^3$ Implementation

For GPU-based computation all data must be copied to the device via the PCIe bus. We therefore only perform expensive computations on the device that benefit from the massive parallelism. In the case of $\mathcal{HR}^3$ this is the computation of the index function that determines which hypercubes a given cube needs to be compared with. Since GPUs work best with regular memory access patterns a few preparation steps are needed. These are performed serially on the host. First, we discretize the input space $\Omega = S \cup T$, resulting in a set of hypercubes. All hypercubes are then sorted component-wise. The number of hypercubes determines the global work size. That is, each thread is assigned a hypercube (called pivot cube) determined by its global id. The work to be done by each thread is then to compute all those hypercubes that abide by the bound on indexes set by $\mathcal{HR}^3$. 
A naive implementation would have each thread compare its pivot cube to all other cubes, resulting in an amount of work quadratic in the number of hypercubes. A better approach is to minimize the amount of cube comparisons while maintaining an even work distribution among threads within the same group. Since hypercubes are globally sorted and fetched by work items in increasing schedule order, the ordering is maintained also locally. That is, let $g = k + 1$ be the local work group size. The work item with the least local id per group is assigned the smallest pivot cube $C_0$ while the last work item having the highest local id operates on the largest cube $C_k$ as its pivot. Both work items therefore can determine a lower and upper bound for the whole group as follows. The first item computes the cube $C_0 - \alpha = (c_0^1 - \alpha, \ldots, c_0^n - \alpha)$ and the last item computes the cube $C_k + \alpha = (c_k^1 + \alpha, \ldots, c_k^n + \alpha)$, where $c_0^i$ and $c_k^i$ are the coordinates of the respective pivot cubes. Thread 0 then determines $i_l$, the index of the largest cube not greater then $C_0 - \alpha$ while thread $k$ computes $i_u$, the index of the smallest cube that is greater than $C_k + \alpha$. After a barrier synchronization that ensures all work items in a group can read the values stored by threads 0 and $k$, all work items compare their pivot cube to cubes at indices $i_l, \ldots, (i_u - 1)$ in global device memory. Since all work items access the same memory locations fetches can be efficiently served from global memory cache.

In OpenCL kernels dynamic memory management is not available. That is, all buffers used during a computation must be allocated in advance by the host program. In particular, the size of the result buffer must be known before submitting a kernel to a device. We therefore cannot simply write the resulting cubes to...
an output vector. Instead, we compute results in two passes. During the first pass each thread writes the number of results it needs to produce to an output vector. A prefix sum over this vector yields at each index the accumulated number of results of threads with a lower id. This value can be used as an index into the final output vector at which each thread can start writing its results.

As an example consider Figure 3. It shows four threads \(0 \ldots 3\), each of which loads a single cube from the sorted source cubes vector. The index from which each threads loads its cube is given by its id\(^{5}\). In this example we assume a granularity factor of \(\alpha = 4\). For thread 1 the smallest cube its pivot cube needs to be compared with is \(C(D) = (2464, 10265)\) while the largest is \(C(B) = (2468, 10266)\). It therefore writes 2 into an output vector, again using its thread id as an index. Thread 0 as well as 2 and 3 do the same, which results in the result size vector as depicted in Figure 3. In order to determine the final indexes each thread can use for storing its results in the result vector, an exclusive prefix sum is computed over the result size vector. This operation computes at each index \(i\) the sum of the elements at indexes \(0 \ldots (i - 1)\), resulting in the accumulated result size vector. A result vector of the appropriate size is allocated and in a second kernel run each thread can now write the cube coordinates starting at the index computed in the previous step. Indexing results are then copied back to the host where comparison of the actual input points is carried out. Since this operation is dominated by the construction of the result it cannot be significantly improved on parallel hardware.

### 2.3 MapReduce-based Link Discovery

In this section we present an implementation of \(\mathcal{HR}^3\) with MapReduce (MR), a programming model designed for parallelizing data-intensive computing in cluster environments \([10]\). MR implementations like Apache Hadoop rely on a distributed file system (DFS) that can be accessed by all nodes. Data is represented by key-value pairs and a computation is expressed employing two user-defined functions, map and reduce, which are processed by a fixed number of map (\(m\)) and reduce tasks (\(r\)). For each intermediate key-value pair produced in the map phase, a target reduce task is determined by applying a partitioning function that operates on the pair’s key. The reduce tasks first sort incoming pairs by their intermediate keys. The sorted pairs are then grouped and the reduce function is invoked on all adjacent pairs of the same group.

We describe a straightforward realization of \(\mathcal{HR}^3\) as well as an advanced approach that considers skew handling to guarantee load balancing and to avoid unnecessary data replication. In favor of readability, we consider a single dataset only.

#### 2.3.1 \(\mathcal{HR}^3\) with MapReduce

\(\mathcal{HR}^3\) can be implemented with a single MR job. The main idea is to compare the points of two related cubes within a single reduce call. We call two cubes \(C, C'\) related iff \(\text{index}(C, C') \leq \alpha^p\). For each input point \(\omega\), the map function determines the surrounding cube \(C(\omega)\) and the set of related cubes \(RC\), which might contain points within the maximum distance. For each cube \(C' \in RC\), map outputs a \((\text{cid}_1 \odot \text{cid}_2 \odot \text{flag}, (p, \text{flag}))\) pair with a composite key and the point itself as value. The first two components of the key identify the two involved cubes using textual cube ids: \(\text{cid}_1 = \min\{C(\omega).id, C'.id\}\) and \(\text{cid}_2 = \max\{C(\omega).id, C'.id\}\). The flag indicates whether \(\omega\) belongs to the first or to the second cube. The repartitioning of the output key-value pairs is done by applying a hash function on the first two key components. This assigns all points of \(C(\omega) \cup C'\) to the same reduce task. All key-value pairs are sorted by their complete keys. Finally, the reduce function is invoked on all values whose first two key components are equal. In reduce, the actual distance computation takes place. Due to the sorting, it is ensured that all points of the cube with the smaller cube id are processed first.

\(^{5}\)For means of readability we show only one id per thread that serves as both its local and global id.
Algorithm 1: Basic $HR^3$ - Map

map($k_{in}$=unused, $v_{in}$ = $\omega$)

$\Delta \leftarrow \theta/\alpha$

cid$_1$ $\leftarrow$ getCubeId($C(\omega)$),

foreach $C' \in RC$ do

$\text{cid}_2$ $\leftarrow$ getCubeId($C'$),

if $\text{cid}_1 \leq \text{cid}_2$ then

output($\text{cid}_1.\text{cid}_2.0$, ($\omega$, 0));

else

output($\text{cid}_2.\text{cid}_1.1$, ($\omega$, 1));

// part = hash($\text{cid}_1, \text{cid}_2$) mod $r$

// sort component-wise by entire key

// group by $\text{cid}_1, \text{cid}_2$

Algorithm 2: Basic $HR^3$ - Reduce

reduce($k_{tmp}$=cid$_1$, cid$_2$, $v_{tmp}$=list< $\omega$, flag >)

$buf \leftarrow \{}$

if $\text{cid}_1 = \text{cid}_2$ then

foreach ($\omega$, flag) $\in v_{tmp}$ do

foreach $\omega'$ $\in buf$ do

compare($\omega$, $\omega'$);

buf $\leftarrow$ buf $\cup\{\omega\}$;

else

foreach ($\omega$, flag) $\in v_{tmp}$ do

if flag=0 then

buf $\leftarrow$ buf $\cup\{\omega\}$;

else

foreach $\omega'$ $\in buf$ do

compare($\omega$, $\omega'$);

// CPM

Figure 4: Overview of the MR-based $HR^3$ implementation with load balancing (left) and the cube population matrix for the example dataset with $m = 2$ (right)

allowing for an efficient comparison of points of different cubes. The pseudo-code of the $HR^3$ implementation is shown in Algorithms 1 and 2.

The described approach has two major drawbacks. First, a map task operates only on a fraction of the input data without global knowledge about the overall data distribution. Thus, each point is replicated and repartitioned $|RC|$ times, independently of whether there are points in the related cubes or not. Second, this approach is vulnerable to data skew, i.e., due to the inherent quadratic time complexity varying cube sizes can lead to severe load imbalances of the reduce tasks. Depending on the problem size and the granularity of the space tiling, the scalability of the described approach might be limited to a few nodes only. We provide an advanced approach that addresses these drawbacks in the next section.

### 2.3.2 $HR^3$ with Load Balancing

The advanced approach borrows ideas from the load balancing approaches for Entity Resolution presented in [33]. An overview is shown in Figure 4(a). The overall idea is to schedule a light-weight analysis MR job that linearly scans the input data in parallel and collects global data statistics. The second MR job utilizes these
Figure 5: Match task creation and reduce task assignment with/without splitting of large tasks (left). Example data flow for second MR job (right)

statistics for a data-driven redistribution of points ensuring evenly loaded reduce tasks.

**Data Analysis Job.** The first job calculates the cube index of each point in the map phase and sums up the number of points per (non-empty) cube in reduce. The output is a cube population matrix (CPM) of size $c \times m$ that specifies the number of points of $c$ cubes across $m$ input partitions. For our running example, an analysis job with $m = 2$ map tasks would read data from two input partitions $\Pi_0$ and $\Pi_1$ (cf. table in Figure 1) and produce the CPM shown in Figure 4(b).

**Distance Computation Job.** The second MR job is based on the same number of map tasks and the same partitioning of the input data. At initialization, each map task reads the CPM. Similar to the basic approach, the reduce function processes pairs of related cubes. Because the CPM allows for an easy identification of empty cubes, the number of intermediate key-value pairs can be reduced significantly. As an example, for point $B$ of the running example, the map function of the basic approach would output 77 key-value pairs. With the knowledge encoded in the CPM, this can be reduced to two pairs only, i.e., for computing $B$’s distances to the points $C$ and $D$, respectively.

Before processing the first input point, each map tasks constructs a list of so-called match tasks. A match task is a triple $(C_i, C_j, w)$, where $C_i, C_j$ are two related cubes and $w = |C_i| \cdot |C_j| \cdot (|C_i| - 1)/2$ for $i = j$ is the corresponding workload. The overall workload $W$ is the sum of the workload of all match tasks. To determine each match task’s target reduce task, the list is sorted in descending order of the workload. In this order, match tasks are assigned to the $r$ reduce tasks following a greedy heuristic, i.e., the current match task is assigned to the reduce task with the currently lowest overall workload. The resulting match tasks are shown on the top of Figure 5(a). Obviously, the reduce tasks are still unevenly loaded, because a major part of the overall workload is made up by the match task $C_4 - C_4$. To address this, for each large match task $M = (C_i, C_j, w)$ with $w > W/r$, both cubes are split according to their input partitioning into $m$ subcubes. Consequently, $M$ is split into a set of smaller subtasks, each comprising a pair of split subcubes before the sorting and reduce task assignment takes place. The bottom of Figure 5(a) illustrates the splitting of the large match task $C_4 - C_4$. Because its workload $w = 6$ exceeds the average reduce task workload of $9/2 = 4.5$, $C_4$ is split into two subcubes $C_{4,0}$ (containing $E, F$) and $C_{4,1}$ (containing $G, H$). This results in three subtasks $(C_{4,0}, C_{4,0}, 1)$, $(C_{4,1}, C_{4,1}, 1)$, and $(C_{4,0}, C_{4,1}, 4)$ that recompose the original match task. Thus, both reduce tasks compute approximately the same number of distances indicating a good load balancing for the example.
After the initial match task creation, map task \( i \) builds an index that maps a cube to a set of corresponding match tasks. Thereby, only cubes of whom the input partition \( i \) actually contains points, need to be considered. For each input point \( \omega \) and each match task of the cube \( C(\omega) \), the map function outputs a \((\text{red}_\text{task} \odot \text{match}_\text{task} \odot \text{flag}, (\omega, \text{flag}))\) pair. Again, the flag indicates to which of the match task’s (possibly split) cubes \( \omega \) belongs to. The partitioning is only based on the reduce task index. The sorting is performed on the entire key, whereas the grouping is done by match task index. Figure 5(b) illustrates the dataflow for the running example. Note, that due to the enumeration of the match tasks and the sorting behavior, it is ensured that the largest match tasks are processed first. This makes it unlikely that larger delays occur at the end of the computation when most nodes are already idle.

### 2.4 Evaluation

The aim of our evaluation was to discover break-even points for the use of parallel processor, GPU and cloud implementations of LD algorithms. For this purpose, we compared the runtimes of the implementations of HR\(^3\) presented in the previous sections on four data sets within two series of experiments. The goal of the first series of experiment was to compare the performance of the approaches for link discovery problems of common size. Thereafter, we carried out a scalability evaluation on a large dataset to detect break-even points of the implementations. In the following, we present the datasets we used as well as the results achieved by the different implementations.

#### 2.4.1 Experimental Setup

We utilized the four datasets of different sizes shown in Figure 6. The small dataset DS\(_1\) contains place instances having three elevation features. The medium-sized datasets DS\(_2\) and DS\(_3\) contain instances with geographic coordinates. For the scalability experiment we used the large dataset DS\(_3\) and varied its size up to \(6 \cdot 10^6\). Throughout all experiments we considered the Euclidean distance. Given the spectrum of implementations at hand, we ran our experiments on three different platforms. The **CPU experiments** (Java, Java\(_2\), Java\(_4\), Java\(_8\) for 1, 2, 4 and 8 cores) were carried out on a 32-core server running JDK1.7 on Linux 10.04. The processors were 8 quad core AMD Opteron 6128 clocked at 2.0 GHz. The **GPU experiments** (GPU) were performed on an average consumer workstation. The GPU was a AMD Radeon 7870 GPU with 20 compute units, each of which has the ability to schedule up to 64 parallel hardware threads. The host program was executed on a Linux workstation running Ubuntu 12.10 and AMD APP SDK 2.8. The machine had an Intel Core i7 3770 CPU and 8 GB of RAM. All C++ code was compiled with gcc 4.7.2. Given that C++ and Java are optimized differently, we also ran the Java code on this machine and computed a runtime ratio that allowed our results to remain compatible. The **MapReduce experiments** (basic: MR, load balanced: MR\(_l\)) were performed with the **Dedoop prototype** \([32]\) on Amazon EC2 in EU-west location. For the first experiment we used 10 nodes of type c1.medium (2 virtual cores, 1.7 GB memory). For the large data set we employed 20 nodes of type c1.xlarge (8 virtual cores, 7 GB memory).
2.4.2 Performance Comparison

The results of our performance comparison are shown in Figure 7. While the parallel implementation of HR$^3$ on CPUs scales linearly for uniformly distributed data, the considerable skew in the DS$^3$ data led to the 8-core version being only 1.6 times faster than the mono-core implementation with a threshold of 1\(^\circ\). This impressively demonstrates the need for load balancing in all parallel link discovery tasks on skewed data. This need is further justified by the results achieved by MR and MR$^l$ on DS$^3$. Here, MR$^l$ clearly outperforms MR and is up to 2.7 times faster. Still, the most important result of this series of experiments becomes evident after taking a look at the GPU and Java runtimes on the workstation.

Most importantly, the massively parallel implementation outperforms all other implementations significantly. Especially, the GPU implementation outperforms the MR and MR$^l$ by one to two orders of magnitude. Even the Java$^8$ implementation is outperformed by up to one order of magnitude. The performance boost of the GPU is partly due to the different hardware used in the experiments. To measure the effect of the hardware, we ran the server Java program also on the workstation. A comparison of the runtimes achieved during this rerun shows that the workstation is between 2.16 and 7.36 times faster than the server. Still, our results suggest that our massively parallel implementation can make an effective use of the underlying architecture to outperform all other implementations in the indexing phase. The added efficient implementation of float operations for the distance computation in C++ leads to an overall superior performance of the GPU. Here, the results can be regarded as conclusive with respect to MR and MR$^l$ and clearly suggest the use of local parallelism when dealing with small to average-sized link discovery problems.
The key observation that leads to conclusive results when comparing GPU and CPU results is that the generation of the cube index required between 29.3% (DS$_1$, $\theta = 50m$) and 74.5% (DS$_3$, $\theta = 1^\circ$) of the total runtime of the algorithm during the deduplication tasks. Consequently, while running a parallel implementation on the CPU is advisable for small datasets with small thresholds for which the index computation makes up a small percentage of the total computation, running the approach on medium-sized datasets or with larger thresholds should be carried out on the GPU. This conclusion is yet only valid as long as the index fits into the memory of the GPU, which is in most cases 4 to 8 times smaller than the main memory of workstations. Medium-sized link discovery tasks that do not fit in the GPU memory should indeed be carried out on the CPUs. Our experiments suggest a break-even point between CPU and GPU for result set sizes around $10^8$ pairs for 2-dimensional data. For higher-dimensional data where the index computation is more expensive, the break-even point is reached even for problems smaller than DS$_1$.

2.4.3 Scalability: Data Size

The strengths of the cloud are revealed in the second series of experiments we performed (see Figure 8). While the DFS and data transfer overhead dominates the total runtime of the LD tasks on the small datasets, running the scalability experiments on 20 nodes reveals that for tasks which generate more than 12 billion pairs as output, MR$_l$ outperforms our local Java implementation. Moreover, we ran further experiments with more than 20 nodes on the 6 million data items. Due to its good scalability, the cloud implementation achieves the runtime of the GPU or performs even better for more nodes, e.g., for 30 (50) nodes MR$_l$ requires approx. 32min (23min).

Overall, we can derive from these results that the upload bottleneck engendered by the use of Cloud services can only be accounted for when dealing with billions of results. While such result sets will definitely play a role in the future, they do not reflect the current scales at which we need to integrate data. We thus opted for developing our scalability extensions in a thread mode as we await results which do not fit in the memory of a GPU.
3 GEO Lift

3.1 Introduction

Manifold RDF data contain implicit references to geographic data. For example, music datasets such as Jamendo include references to locations of record labels, places where artists were born or have been, etc. The aim of the spatial mapping component, dubbed GeoLift\(^6\), is to retrieve this information, make it explicit and integrate it into data sources according to the specifications of the user. To this end, GeoLift relies on a simple yet powerful workflow system that consists of two main components: modules and operators. Modules implement functionality for processing the content of a dataset (e.g., applying named entity recognition to a particular property). Thus, they take a dataset as input and return a dataset as output. Operators work at a higher level of granularity and combine datasets. Thus, they take sets of datasets as input and return sets of datasets. In the following, we present the modules and operators implemented by GeoLift. The scalability of the framework was improved by improving the scalability of the modules, as these are the runtime bottleneck of the framework. The updates to the modules are presented in the subsequent sections of this report.

3.2 GEO Lift Components

GeoLift was designed to be a modular tool which can be easily extended and re-purposed. GeoLift provides two main types of artifacts: modules and operators. Thus, in case a user knows the type of enrichment that is to be carried out, he can define the sequence of modules/operators that must be used to process his dataset.

3.2.1 Modules

By Modules we mean these artifacts in charge of generating geographical data based on RDF data. The input for such a module is an RDF dataset. The output is also an RDF dataset enriched with geographical information. Formally, a module can thus be regarded as a function \( \mu: \mathcal{R} \rightarrow \mathcal{R} \), where \( \mathcal{R} \) is the set of all RDF datasets. Currently, GeoLift implements five modules: dereferencing, linking, NLP, conformation and filter.

3.2.1.1 Dereferencing Module

For datasets which contain similarity proprieties links (e.g. owl:sameAs), we dereference all links from the dataset to other datasets by using a content negotiation on HTTP as shown in Figure 9. This returns a set of triples that needs to be filtered for relevant geographical information. Here, we use a predefined list of attributes that links to geographical information. Amongst others, we look for \texttt{geo:lat}, \texttt{geo:long}, \texttt{geo:lat_long}, \texttt{geo:line} and \texttt{geo: polygon}. The list of retrieved property values can be configured.

3.2.1.2 Linking Module

Links to geographical resources do not occur in several knowledge bases. Here, we rely on the metrics implemented in the Limes framework\(^7\) \([49, 46, 51]\) to link the resources in the input dataset with geographical datasets. Limes, the Link Discovery Framework for Metric Spaces, is a framework for discovering links between entities contained in Linked Data sources. Limes is a hybrid framework \([46]\) that combines the mathematical

---

\(^6\)
\[^{http://aksw.org/Projects/GeoLift}\]

\(^7\)
\[^{http://limes.sf.net}\]
characteristics of metric spaces as well prefix-, suffix- and position filtering to compute pessimistic approximations of the similarity of instances. These approximations are then used to filter out a large amount of those instance pairs that do not suffice the mapping conditions. By these means, Limes can reduce the number of comparisons needed during the mapping process by several orders of magnitude and complexity without loosing a single link.

Linking using Limes [46, 45] can be achieved in three ways:

1. Manually, by the means of a link specification [46], which is an XML-description of (1) the resource in the input and target datasets that are to be linked and (2) of the similarity measure that is to employed to link these datasets.

2. Semi-automatically based on active learning [52, 53, 54]. Here, the idea is that if the user is not an expert and thus unable to create a link specification, he can simply provide the framework with positive and negative examples iteratively. Based on these examples, Limes can compute links for mapping resources with high accuracy.

3. Automatically based on unsupervised machine learning. Here, the user can simply specify the sets of resources that are to be linked with each other. Limes implements both a deterministic and non-deterministic machine-learning approaches that optimize a pseudo-F-measure to create a one-to-one mapping.

3.2.1.3 NLP Module

The geographical information hidden in datatype properties is retrieved by using Named Entity Recognition (NER). GeoLift- as an expandable modeller framework - can inject any NER framework to implement the NLP module. In the current version of GeoLift, we rely on the FOX framework. For more details about the FOX framework see subsection 4.4.

3.2.1.4 Conformation Module

The idea of the conformation module is to uniform the enriched triples found by GeoLift under one authority. The conformation module changes a specified source URI authority to a specified target URI authority.

3.2.1.5 Filter Module

The idea behind the filter module is to extract only a set of desired triples from the input dataset. Filter module takes a set of triples patterns and a dataset as inputs. Applying the triple patterns against the input dataset, it filters the input dataset and produces the filtered triples as output dataset.

For example, running triple pattern "?s <http://dbpedia.org/ontology/abstract> ?o" against an input dataset containing the Concise Bounded Description (CBD)⁸ of Berlin “http://dbpedia.org/resource/Berlin” will generate an output dataset that contains only Berlin's abstracts.

3.2.2 Operators

The idea behind operators is to enable users to define a workflow for processing their input dataset. GeoLift operators combine the enrichment modules and allow defining a workflow for processing information. Note that the format of the input and output of operator is identical. Thus, the user is empowered to create workflows of arbitrary complexity by simply connecting modules and/or operators. Formally, an operator can be regarded as a function \( \varphi : \mathcal{R}^* \rightarrow \mathcal{R}^* \).

3.2.2.1 Split Operator

The idea behind the split operator is to enable parallel execution of different modules in the same dataset. The split operator takes one dataset as input and produces \( n \geq 2 \) output datasets, which are all identical to the input dataset. Each of the output datasets of the split operator has its own workflow (as to be input to any other module or operator). Thus, GeoLift is able to execute all workflows of output datasets in parallel.

3.2.2.2 Merge Operator

The idea behind the merge operator is to enable combining datasets. The merge operator takes a set of \( n \geq 2 \) input datasets and merges them into one output dataset containing all the input datasets’ triples. As in case of split operator, the merged output dataset has its own workflow (as to be input to any other module or operator).

3.3 GeoLift RDF Specification Paradigm

In the current version of GeoLift we introduce our new RDF based specification paradigm. The main idea behind this new paradigm is to enable the processing execution of specifications in an efficient way. To this end, we first decided to use RDF as language for the specification. This has the main advantage of allowing for creating specification repositories which can be queried easily with the aim of retrieving accurate specifications for the use cases at hand. Moreover, extensions of the specification language do not require a change of the specification language due to the intrinsic extensibility of ontologies. The third reason for choosing RDF as language for specifications is that we can easily check the specification for correctness by using a reasoner, as the specification ontology allows for specifying the restrictions that specifications must abide by.

Figure 10 introduces an example of RDF specification file presented in turtle serialization, which will be used as a guiding example through the next subsections. For the sake of clarity, we represent the configuration file in form of graph in figure 11.

---

⁸ For more details about CBD see [http://www.w3.org/Submission/CBD/](http://www.w3.org/Submission/CBD/)
3.3.1 GeoLIFT Execution Workflow

First of all, GeoLIFT determines the set of output datasets \( D = \{d_1, \ldots, d_n\} \), i.e. datasets which are included as output of some modules/operators but not as input to any other modules/operators. In our example it is only \( d_8 \). Then, for each dataset \( d_i \) in \( D \), GeoLIFT tries to find \( d_i \) either trivially from file/URI/endpoint or recursively by solving for modules/operators that generate \( d_i \) as output.

Going back to our example, GeoLIFT first tries to read \( d_8 \) trivially but it fail as there is no direct way to read it. Then, GeoLIFT recursively goes for solving the Conformation module as it generates \( d_8 \) as output. This recursive procedure continues until GeoLIFT reads the dataset \( d_1 \) from the endpoint. Afterwords, going back in the recursion stack GeoLIFT produces the triples of \( d_8 \).

3.3.2 RDF Specification Resources

GeoLIFT RDF specification file may contain four main resource types (dataset, module, operator and parameter). In the following subsections we will describe each of them along with our example shown in Figure 10.

3.3.2.1 Dataset Resource

As its name implies, a Dataset resource represents a dataset. For an example see Figure 10 (lines 4:6), in which the RDF specification represents a dataset with URI http://dbpedia.org/resource/Berlin and endpoint http://dbpedia.org/sparql. GeoLIFT extracts the CBD of the Berlin resource from DBpedia endpoint.

Beside defining a resource as a dataset (e.g. Figure 10 (lines 4)), a dataset resource may also contain the following predicates:

- `rdfs:label` setting label for the dataset.
- `:hasUri` setting URI of the dataset resource, to read resource through content negotiation (e.g. Figure 10 (line 5)).
- `:fromEndPoint` endpoint to read the CBD from. Note: must be used in conjunction with `:hasUri` predicate, otherwise an error will be generated by GeoLIFT (e.g. Figure 10 (line 6)).
- `:inputFile` input file to load the dataset from.
- `:outputFile` output file to save the dataset to (e.g. Figure 10 (lines 14)).
- `:outputFormat` output format to save the dataset in it (e.g. Figure 10 (lines 14)), Turtle format will be used as default format if this property is not set.

3.3.2.2 Module Resource

Beside defining a resource as a module (e.g. Figure 10 (line 16)), a Module resource may also contain the following predicates:

- `rdfs:label` setting label for the module (e.g. Figure 10 (line 17)).
- `:hasInput` setting input datasets for the module (e.g. Figure 10 (line 18)).
Figure 10: Example of GeoLift RDF specification (Turtle serialization).
• :hasOutput setting output datasets for the module (e.g. Figure 10 (line 19)).
• :hasParameter setting parameters for the module (e.g. Figure 10 (line 20)).

3.3.2.3 Operator Resource

Beside defining a resource as an operator (e.g. Figure 10 (line 24)), a Module resource may also contain the following predicates:

• rdfs:label setting a label for the operator (e.g. Figure 10 (line 25)).
• :hasInput setting input datasets for the operator (e.g. Figure 10 (line 26)).
• :hasOutput setting output datasets for the operator (e.g. Figure 10 (line 27)).
• :hasParameter setting parameters for the operator.

3.3.2.4 Parameter Resource

Beside defining a resource as a parameter (e.g. Figure 10 (line 21)), a Module resource may also contain the following predicates:

• rdfs:label setting label for the parameter.
• :hasKey setting the parameter key (e.g. Figure 10 (line 22)).
• :HasValue setting the parameter value (e.g. Figure 10 (lines 23)).

The values of (key, values) pairs are modules/operators dependent.⁹

Figure 11: Graph representation of configuration file presented by listing 10
4 Scalability Improvements

In this section, we present the different scalability improvements that we carried out to ensure that GeoLift can deal with large datasets. We begin by presenting the parallel implementation of Orchid. Thereafter, we present the novel algorithm we devised to process Jaro-Winkler similarities. The Helios planner, which allows determining time-efficient plans for processing link specifications, is the next approach we present. Finally, we present the FOX enhancements that underly the new version of GeoLift.

4.1 Orchid

4.1.1 Overview

Geo-spatial data is most commonly described by using two categories of object properties: geometry descriptions (polygons, latitude and longitude, etc.) and textual entries. In D.3.1.1, we presented the Orchid algorithm, which allows comparing geometry information efficiently. While there are several approaches for computing the distance between two polygons \( \delta \), our approach relies on the Hausdorff distance \( \text{hd} \):

\[
\text{hd}(s, t) = \max_{s_i \in s} \min_{t_j \in t} \{ \delta(s_i, t_j) \},
\]

where \( \delta \) is the metric associated to the affine space within which the polygons are defined. We assume that the earth is a perfect ball with radius \( R = 6378 \text{ km} \). It is important to notice that the Hausdorff distance is a metric in the mathematical sense of the term in any metric space. Moreover, the orthodromic distance is also known to be a metric, leading to the problem formulated above being expressed in a metric space.

Several approaches have addressed the time-efficient computation of Hausdorff distances throughout literature (see \[56\] for a good overview). Yet, so far, these approaches have not been concerned with the problem of only finding those triples \((s, t, \text{hd}(s, t))\) with \( \text{hd}(s, t) \leq \theta \).

In D3.1.1, we presented and evaluated the approaches implemented by Orchid to achieve this goal, including naive, bound and indexed approaches. Orchid implements a space tiling algorithm for orthodromic spaces \[47\]. To achieve this goal, Orchid creates a grid on the surface of the planet and maps the polygons (representing the spatial resource) to at least one square on the grid. At runtime, Orchid only compares certain pairs of grids. We call each comparison of source elements from a given grid \( g \) with target elements from a given grid \( g' \) (note that \( g \neq g' \) is not required here) a task. In the following, we introduce different approaches to achieve a load balancing of Orchid’s tasks within parallel implementations of Orchid.

4.1.2 Load Balancing Algorithms

The main idea behind load balancing techniques is to utilize parallel processing to distribute the tasks necessary to generate the solution to a problem across several processing units. Throughput maximization, response time minimization and resources overloading avoidance are the main purposes of any load balancing technique. We devised, implemented and evaluated five different load balancing approaches for Orchid ranging from a naive load balancer to a particle-swarm-optimization-based approach.

4.1.2.1 Naïve Load Balancer

The idea behind the naïve load balancer is to divide all tasks between all processors equally regardless of their complexity. Let \( n \) be the number of tasks to tackle and \( m \) be the number of processors at hand. Then, each of
the $m$ processor is assigned $\lfloor \frac{n}{m} \rfloor$ and $\lceil \frac{n}{m} \rceil$ tasks. Algorithm 3 shows the pseudocode of our implementation of a naïve load balancing approach in which tasks are assigned to processors in the order of the input set.

**Algorithm 3: Naïve Load Balancer**

```
input : $T \leftarrow \{t_0, ..., t_{n-1}\}$ : set of tasks of size $|T| \leftarrow n$
        $m$ : number of processors

output: $B \leftarrow \{b_0, ..., b_{m-1}\}$ : a partition of $T$ to a list of $m$ blocks of tasks

$i \leftarrow 0$;

foreach task $t$ of $T$ do
    add($B$, $i$, $t$);
    $i \leftarrow (i + 1) \mod m$;
```

4.1.2.2 Greedy Load Balancer

The main idea behind the greedy load balancing [8] technique is to sort each of the input tasks in descending order based on their complexity. Then, starting from the most complex task, the greedy load balancer assigns tasks to processors in order. This approach is basically a heuristic that aims at achieving an even distribution of the total task complexity over all processors. The pseudo code of the greedy load balancer technique is presented in Algorithm 4.

**Algorithm 4: Greedy Load Balancer**

```
input : $T \leftarrow \{t_0, ..., t_{n-1}\}$ : set of tasks of size $|T| \leftarrow n$
        $m$ : number of processors

output: $B \leftarrow \{b_0, ..., b_{m-1}\}$ : a partition of $T$ to a list of $m$ blocks of balanced tasks

$C \leftarrow \text{computeTasksComplexity}(T)$;
$C \leftarrow \text{sortDescending}(C)$;
$i \leftarrow 0$;

foreach task $t$ of $C$ do
    add($B$, $i$, $t$);
    $i \leftarrow (i + 1) \mod m$;
```

4.1.2.3 Pair-Based Load Balancer

The main idea of the pair-based load balancing [34] is to assign processors tasks in pairs of the form (very complex, not complex). In order to get most homogeneous pairs, the algorithm first sort all input tasks according to tasks’ complexities. Afterwards, from the sorted list of tasks, the pair based algorithm generates the $n/2$ pairs of tasks where the pair $k$ is computed by selecting tasks $k$ and $n - k$ from the sorted list of tasks. The pseudo code of the pair based technique is introduced at Algorithm 5.

4.1.2.4 Particle Swarm Optimization (PSO)

Here we consider tasks as the particles to be optimized. Particles are partitioned into as many blocks as there are processors. The aim is here to to balance the size of the blocks (i.e., the total complexity of each block)
**Algorithm 5: Pair Based Load Balancer**

<table>
<thead>
<tr>
<th>Algorithm 5: Pair Based Load Balancer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong>:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>output</strong>:</td>
</tr>
<tr>
<td>$C \leftarrow \text{computeTasksComplexity}(T)$;</td>
</tr>
<tr>
<td>$C \leftarrow \text{sort}(C)$;</td>
</tr>
<tr>
<td>$i \leftarrow 0$;</td>
</tr>
<tr>
<td><strong>for</strong> $i \leq \lceil n/2 \rceil$ <strong>do</strong></td>
</tr>
<tr>
<td>$l \leftarrow \text{get}(C, i)$;</td>
</tr>
<tr>
<td>add($B$, $i$, $l$);</td>
</tr>
<tr>
<td>$r \leftarrow \text{get}(C, n-i-1)$;</td>
</tr>
<tr>
<td>add($B$, $i$, $r$);</td>
</tr>
<tr>
<td>$i \leftarrow i + 1$;</td>
</tr>
</tbody>
</table>

as well as possible. Therefore, we defined the **fitness function** for the procedure as the complexity difference between the most overloaded task block and least underloaded task block. Formally,

$$F \leftarrow \min\{|B_{\text{max}}| - |B_{\text{min}}|\},$$

(3)

where the $|B_{\text{max}}|$, $|B_{\text{min}}|$ are the sizes of the most and least loaded blocks respectively.

The PSO load balancer initially divides all particles into the task blocks regardless of tasks’ complexity (i.e. each block now contains the same number of particles). Then, particles are saved in what we dubbed as **Best Known Positions (BKP)**. Afterwards, PSO computes the fitness function to the initial BKP and saves it as **Best Known Fitness (BKF)**. Until a termination criterion is met, the PSO moves the particles between different task blocks using a random velocity and computes the fitness function of the new particle distribution. If a better BKF found, the PSO updates both BKF and BKP.

The PSO has two termination criteria: (1) **minimum fitness threshold** and (2) **maximum number of iterations**. If the minimum fitness threshold is reached in any iteration the algorithm terminates instantly and the BKP is returned. Otherwise the BKF is returned after reaching the maximum number of iterations. The pseudo-code of the PSO load balancer algorithm in presented in **algorithm 6**.

**4.1.2.5 DPSO Load Balancer**

The PSO load balancer (see section 4.1.2.4) has a main drawback of being a non-deterministic approach. This drawback is inherited from the fact that PSO is a heuristic algorithm that depends up on a random selection of velocity for moving particles. In order to overcome this drawback, we propose the **Deterministic PSO (DPSO)**.

The DPSO starts in the same way as the PSO by partitioning all the tasks to $m$ task blocks, where $m$ equals the number of processors. In this stage, each block contains the same number of tasks regardless of tasks’ complexities.

Until reaching a termination criterion, the DPSO works as:

1. Find the most overloaded block $O$ and least underloaded block $U$, compute respective complexities $O_{\text{complexity}}$ and $U_{\text{complexity}}$.

\[10\] The minimum fitness threshold in zero by default
Algorithm 6: Particle Swarm Optimization Load Balancer

**input** : $T \leftarrow \{t_0, ..., t_{n-1}\}$ : set of tasks of size $|T| \leftarrow n$

$m$ : number of processors and

$F_{\text{threshold}}$ : is the fitness function threshold (zero by default)

$C$ : number of iterations $\leftarrow C$

**output** : $\text{BKP} \leftarrow \{b_0, ..., b_{m-1}\}$ : the Best Known particles’ Positions as a list of $m$ blocks of balanced tasks

Initialize particles’ best known position $\text{BKP}$

$i \leftarrow 0$;

**foreach** task $t$ of $T$ do

$\text{add}(\text{BKP}, i, t)$;

$i \leftarrow (i + 1) \mod m$;

Initialize best known Fitness $\text{BKF}$

$\text{BKF} \leftarrow \text{getMostOverloadedBlock}(\text{BKP}) - \text{getLeastUnderloadedBlock}(\text{BKP})$;

$c \leftarrow 0$;

**while** $c < C$ do

$P \leftarrow \text{BKP}$;

Move each particle $p$ to new position using a random particle velocity $v$

**foreach** block $b \in \text{BKP}$ do

**foreach** particle $p \in b$ do

$v \leftarrow \text{getRandomVelocity}(m)$;

$q \leftarrow \text{get}(\text{BKP}, v)$;

if $q \neq b$ then

$\text{remove}(P, b, p)$;

$\text{add}(P, v, p)$;

If better fitness achieved update result

$F \leftarrow \text{getMostOverloadedBlock}(P) - \text{getLeastUnderloadedBlock}(P)$;

if $F = F_{\text{threshold}}$ then

return $\text{BKF}$;

if $F > \text{BKF}$ then

$\text{BKF} \leftarrow F$;

$\text{BKP} \leftarrow P$;

$c \leftarrow c + 1$;


2. Compute fitness function as \( O_{\text{complexity}} - U_{\text{complexity}} \).

3. Sort \( O \), perform task migration by moving tasks in order from \( O \) to \( U \) as long as a better fitness function achieved.

The DPSO has two termination criteria: (1) minimum fitness threshold and (2) maximum number of iterations. The pseudo code of the DPSO load balancer algorithm is presented in Algorithm 7.

The deterministic nature of DPSO comes from the fact that (1) DPSO only moves tasks from most over-loaded block to the least underloaded block, (2) DPSO sorts the most overloaded block before it starts the task migration process. Sorting insures migration of smaller tasks first in which away an optimal load balancing between most and least loaded blocks is achieved in each iteration.

```
Algorithm 7: DPSO Load Balancer

input : \( T \leftarrow \{t_0, \ldots, t_{n-1}\} \) : set of tasks of size \( |T| \leftarrow n \)
\( m \) : number of processors and
\( C \) : number of iterations \( \leftarrow C \)

output: \( B \leftarrow \{b_0, \ldots, b_{m-1}\} \) : a partition of \( T \) to a list of \( m \) blocks of balanced tasks

\( i \leftarrow 0; \)
initially, divide tasks equally between processors task blocks

foreach task \( t \) of \( T \) do
    \( \text{add}(B, i, t); \)
    \( i \leftarrow (i + 1) \mod m; \)

\( c \leftarrow 0; \)
while \( c < C \) do
    Compute fitness function \( F \) as the difference between most and least loaded tasks
    \( O \leftarrow \text{getMostOverloadedBlock}(B); \)
    \( U \leftarrow \text{getLeastUnderLoadedBlock}(B); \)
    \( O_{\text{complexity}} \leftarrow \text{getComplexity}(O); \)
    \( U_{\text{complexity}} \leftarrow \text{getComplexity}(U); \)
    \( F \leftarrow |O_{\text{complexity}} - U_{\text{complexity}}|; \)
    Balance the most and least loaded tasks
    \( O \leftarrow \text{sort}(O); \)

foreach task \( t \in O \) do
    \( O_{\text{complexity}} \leftarrow O_{\text{complexity}} - \text{getComplexity}(t); \)
    \( U_{\text{complexity}} \leftarrow U_{\text{complexity}} + \text{getComplexity}(t); \)
    \( F_{\text{new}} \leftarrow |O_{\text{complexity}} - U_{\text{complexity}}|; \)
    if \( F_{\text{new}} < F \) then
        remove \( O, t \);
        add \( U, t \);
        \( F \leftarrow F_{\text{new}}; \)
    else
        break;

\( c \leftarrow c + 1; \)
```

4.1.3 Evaluation

4.1.3.1 Experimental Setup

We used two publicly available datasets for our experiments. The first dataset, DBpedia\textsuperscript{11}, contains all the 731,922 entries from DBpedia that possess geometry entries. We chose DBpedia because it is commonly used in the Semantic Web community. The second dataset, LGD\textsuperscript{12}, contains all 3,836,119 geo-spatial objects from http://linkgeodata.org that are instances of the class Way.\textsuperscript{12} Further details to the datasets can be found in [47].

All experiments were carried out on a 64-core server running OpenJDK 64-Bit Server 1.6.0_27 on Ubuntu 12.04.2 LTS. The processors were 12 Hexa-core AMD Opteron 6128 clocked at 2.0 GHz. Unless stated otherwise, each experiment was assigned 16 GB of memory and was ran 5 times.

4.1.3.2 Orchid vs. Parallel Orchid

Here we evaluate the speed up gained by using parallel implementation of Orchid algorithm (GeoHR3). We ran two experiment one using the normal implementation of GeoHR3 [47] and the other using parallel implementation of GeoHR3 with Naive load balancer (2 threads), we used increasing number of polygons of DBpedia as input for our experiments. As mentioned earlier, each experiment was repeated 5 times. Figure 12 shows that parallel implementation of GeoHR3 using Naive load balancer achieves speedup more than the normal GeoHR3 by a factor of more than 5 times.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure12}
\caption{GeoHR3 vs. Parallel GeoHR3}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure12}
\caption{GeoHR3 vs. Parallel GeoHR3}
\end{figure}

\textsuperscript{11}We used version 3.8 as available at http://dbpedia.org/Datasets.

\textsuperscript{12}We used the RelevantWays dataset (version of April 26th, 2011) of LinkedGeoData as available at http://linkedgeodata.org/Datasets.
4.1.3.3 Parallel Load balancing Algorithms Evaluation

Figure 13 introduces a comparison between different parallel implementations of Orchid using the aforementioned algorithms. Using only two threads with increasing number of DBpedia polygons, Figure 13 (a) shows that the naïve, PSO and DPSO approaches outperform the greedy and pair based algorithms. Figure 13 (b) introduces the relative mean squared error of execution of Figure 13 (a).

Using four threads to run the same experiment Figure 13 (c) and Figure 13 (d)) did not yield any better performance as more threads require more overhead thread synchronization time.

Figure 13: Runtimes of applying different parallel load balancing algorithms to DBpedia and LGD

4.1.3.4 DPSO Evaluation

The idea behind our proposed DPSO load balancer inherited from the PSO load balancer, in which we move tasks between blocks to get better fitness function. The main advantage of DPSO over PSO is being totally deterministic unlike the PSO nature which came from random velocity picking. As DPSO being an iterative procedure, we need to find an upper limit for DPSO iteration number. To this end, we ran DPSO against both DBpedia and LGD whole datasets for ten iterations.

We performed two identical experiments using the whole DBpedia dataset as input to the first one and
the whole LGD dataset as input to the second one. Applying DPSO for each dataset, we measured both run time and MSE for each iteration. Figure 14 shows the results of the two experiments. We came out with two results: (1) DPSO achieves better run time starting from the second iteration and (2) at most 5 iterations are enough to reach acceptable load balancing.

![DBpedia Runtime](image1)
![DBpedia MSE](image2)
![LGD Runtime](image3)
![LGD MSE](image4)

Figure 14: DPSO 10 iteration evaluation on DBpedia and LGD

### 4.2 Jaro-Winkler

The integration of large numbers of geo-spatial resources requires devising time-efficient algorithms for comparing their location and labels. With Orchid, we developed a scalable algorithm for dealing with the locations of resources. As the Jaro-Winkler similarities were devised especially for comparing names of resources [27], we integrated this measure in LIMES to ensure the accurate linking of geo-spatial entities. However, no time-efficient approach for bounded Jaro-Winkler distances has designed to this point. We address the efficiency of the execution of bounded Jaro-Winkler measures¹³ by deriving equations that allow discarding a large number of computations while executing bounded Jaro-Winkler comparisons with high thresholds. In particular, we derive length- and range-based filters that allow reducing the number of strings \( t \) from the target dataset \( T \) that are compared with a string \( s \) from the source dataset \( S \). We also present a character-based filter that allows

---

¹³We use bounded measures in the same sense as [40], i.e., to mean that we are only interested in pairs of strings whose similarity is greater than or equal to a given lower bound.
detecting whether two strings \( s \) and \( t \) share enough resemblance to be similar according to the Jaro-Winkler measure. We evaluate our approach w.r.t. to its runtime and its scalability with several threshold settings and dataset sizes. This work is to be regarded as preliminary and will be extended within the course of WP3.

4.2.1 The Jaro-Winkler Similarity

Let \( \Sigma \) be the set of all the strings that can be generated by using an alphabet \( A \). The Jaro measure \( d_j : \Sigma \times \Sigma \rightarrow [0, 1] \) is a string similarity measure approach which was developed originally for name comparison in the U.S. Census. This measure takes into account the number of character matches \( m \) and the ratio of their transpositions \( t \):

\[
d_j = \begin{cases} 
0 & \text{if } m = 0 \\
\frac{1}{3} \left( \frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{otherwise}
\end{cases}
\] (4)

Here two characters are considered to be a match if and only if (1) they are the same and (2) they are at most at a distance \( w = \lceil \max(|s_1|, |s_2|) \rceil / 2 \) from each other. For example, for \( s_1 = ”Spears” \) and \( s_2 = ”Pears” \), the second \( s \) of \( s_1 \) matches the \( s \) of \( s_2 \) while the first \( s \) of \( s_1 \) does not match the \( s \) of \( s_2 \).

The Jaro-Winkler measure [63] is an extension of the Jaro distance. This extension is based on Winkler’s observation that typing errors occur most commonly in the middle or at the end of a word, but very rarely in the beginning. Hence, it is legitimate to put more emphasis on matching prefixes if the Jaro distance exceeds a certain “boost threshold” \( b_t \), originally set to 0.7.

\[
d_w = \begin{cases} 
d_j & \text{if } d_j < b_t \\
d_j + (\ell p(1 - d_j)) & \text{otherwise}
\end{cases}
\] (5)

Here, \( \ell \) denotes the length of the common prefix and \( p \) is a weighting factor. Winkler uses \( p = 0.1 \) and \( \ell \leq 4 \). Note that \( \ell p \) must not be greater than 1.

For the strings \( s_1 = ”DEMOCRACY” \), \( s_2 = ”DEMOGARPHY” \) (with \( s_2 \) being intentionally misspelled) we get the following output of the Jaro-Winkler measure.

\[
\cdot \ |s_1| = 9, |s_2| = 10 \\
\cdot \ |s_1| = 4 \\
\cdot \ m = 7 \\
\cdot \ t = 1 \\
\cdot \ d_j = \frac{1}{3} \left( \frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) = \frac{1}{3} \left( \frac{7}{9} + \frac{7}{10} + \frac{6}{7} \right) = 0.778 \\
\cdot \ d_w = d_j + \ell p (1 - d_j) = 0.867
\]

4.2.2 Improving the Runtime of Bounded Jaro-Winkler

The main principle behind reducing the runtime of the computation of measures is to reduce their reduction ratio. Here, we use a sequence of filters that allow discarding similarity computations while being sure that they would have led to a similarity score which would have been less than our threshold \( \theta \). To this end, we regard the problem as that of finding filters that return an upper bound estimation \( \theta_e(s_1, s_2) \geq d_w(s_1, s_2) \) for some properties of the input strings that can be computed in constant time. For a given threshold \( \theta \), if \( \theta_e(s_1, s_2) \leq \theta \), then we can safely ignore the input \( (s_1, s_2) \).
4.2.2.1 Length-based filters

In the following, we denoted the length of a string \( s \) with \(|s|\). Our first filter is based on the insight that large length differences are a guarantee for poor similarity. For example, the strings "a" and "alpha" cannot have a Jaro-Winkler similarity of 1 by virtue of their length difference. We can formalize this idea as follows: Let \( s_1 \) and \( s_2 \) be strings with respective lengths \(|s_1|\) and \(|s_2|\). Without loss of generality, we will assume that \(|s_1| \leq |s_2|\). Moreover, let \( m \) be the number of matches across \( s_1 \) and \( s_2 \). Because \( m \leq |s_1| \), we can substitute \( m \) with \(|s_1|\) and gain the following upper bound estimation for \( d_j(s_1, s_2) \):

\[
d_j = \frac{1}{3} \left( \frac{m}{|s_1|} + \frac{m - t}{|s_2|} + \frac{m - t}{m} \right) \leq \frac{1}{3} \left( 1 + \frac{|s_1|}{|s_2|} + \frac{|s_1| - t}{|s_1|} \right) \tag{6}
\]

Now the lower bound for the number \( t \) of transpositions is 0. Thus, we obtain the following equation.

\[
d_j \leq \frac{1}{3} \left( 1 + \frac{|s_1|}{|s_2|} + 1 \right) \leq \frac{2}{3} + \frac{|s_1|}{3|s_2|} \tag{7}
\]

The application of this approximation on Winkler’s extension is trivial:

\[
d_w = d_j + \ell \cdot p \cdot (1 - d_j) \leq \frac{2}{3} + \frac{|s_1|}{3|s_2|} + \ell \cdot p \cdot \left( \frac{1}{3} - \frac{|s_1|}{3|s_2|} \right) = \theta_c \tag{8}
\]

Consider the pair \( s_1 = "bike" \) and \( s_2 = "bicycle" \) and a threshold \( \theta = 0.9 \). Applying the estimation for Jaro we get \( d_j \leq \frac{2}{3} - \frac{4}{37} = 0.857 \). This exceeds the boost threshold, so we use equation (8) to compute \( \theta_c(s_1, s_2) = 0.885 \). Now we do not have to actually compute \( d_w(s_1, s_2) \), since \( \theta_c(s_1, s_2) < \theta \).

By using this approach we can decide in \( O(1) \) if a given pairs score is greater than a given threshold, which saves us the much more expensive score computation for a big number of pairs, provided that the input strings sufficiently vary in length.

4.2.2.2 Filtering ranges by length

The approach described above can be reversed to limit the number of pairs that we are going iterate over. To this end, we can construct a index : \( \mathbb{N} \rightarrow 2^\mathbb{N} \) which maps strings lengths \( l \in \mathbb{N} \) to all strings \( s \) with \(|s| = l\). With the help of this index, we can now determine the set of strings \( t \) that should be compared with the subset \( S(l) \) of \( S \) that only contains strings of length \( l \). We go about using this insight by computing the upper and lower bound for the length of a string \( t \) that should be compared with a string \( s \). This is basically equivalent to asking what is the minimum length difference \(||s| - |t||\) so that \( \theta \geq \theta_c(s, t) \) is satisfied. We transpose equation (8) to the following for our lower bound:

\[
|t| \geq \left| 3|s| \frac{\theta - \ell p}{1 - \ell p} - 2|s| \right| \tag{9}
\]

Analogously, we can derive the following upper bound:

\[
|t| \leq \left| \frac{|s|}{3^{\theta - \ell p} - 2} \right| \tag{10}
\]

\(^{14}\)In most programming languages, especially Java (which we used for our implementation), the length of string is stored in a variable and can thus be accessed in constant time.
<table>
<thead>
<tr>
<th></th>
<th>(s)</th>
<th>(s)</th>
<th>(s)</th>
<th>sizes in range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>(4, 7)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>14</td>
<td>(4, 7, 11)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>22</td>
<td>(7, 11, 18)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>36</td>
<td>(11, 18)</td>
<td></td>
</tr>
</tbody>
</table>

For example, consider a list of strings \(S\) with equally distributed, distinct string lengths \((4, 7, 11, 18)\). Using Equation 9 and Equation 10 we obtain Table 1. Taking into account the last column of the table, we will save a total of \(\frac{3}{8}\) comparisons.

### 4.2.2.3 Filtering by character frequency

An even more fine-grained approach can be chosen to filter out computations. Let \(e : \Sigma \times A \rightarrow \mathbb{N}\) be the function with returns the number of occurrences of a given character \(c\) in a string \(s\). For the strings \(s_1\) and \(s_2\), the number of maximum possible matches \(m_{\text{max}}\) can be expressed as

\[
m_{\text{max}} = \sum_{c \in s_1} \min(e(s_1, c), e(s_2, c)) \geq m
\]

Consequently, we can now substitute \(m\) for \(m_{\text{max}}\) in the Jaro distance computation:

\[
d_j(s_1, s_2) = \frac{1}{3} \left( \frac{m_{\text{max}}}{|s_1|} + \frac{m_{\text{max}}}{|s_2|} + \frac{m_{\text{max}} - t}{m_{\text{max}}} \right) \leq \frac{1}{3} \left( \frac{m_{\text{max}}}{|s_1|} + \frac{m_{\text{max}}}{|s_2|} + 1 \right)
\]

We can thus derive that \(d_j(s_1, s_2) \geq \theta\) iff

\[
m_{\text{max}} \geq \frac{(3\theta - 1)|s_1||s_2|}{|s_1| + |s_2|}.
\]

For instance, let \(s_1 = "\text{astronaut}"\), \(s_2 = "\text{astrochimp}"\). The retrieval of \(m_{\text{max}}\) is shown in Table 2.

The question that remains to answer is how well do these filters perform on real data. We answer this question empirically in the subsequent section.

### 4.2.3 Evaluation

The aim of our evaluation was to study how well our approach performs on real data. We chose DBpedia 3.9 as a source of data for our experiments as it contains data pertaining to 1.1 million persons and thus allows for both fine-grained evaluations and scalability evaluations. All experiments where deduplication experiments, i.e., \(S = T\). We considered the list of all \(\text{rdfs:label}\) in DBpedia in our runtime evaluation and scalability experiments. We also computed the runtime of our approach on up to \(10^5\) labels for our scalability experiments. All experiments were performed on a 2.5 GHz Intel Core i5 machine with 16GB RAM running OS X 10.9.3.
Table 2: Calculation of $m_{\text{max}}$

<table>
<thead>
<tr>
<th>$c$</th>
<th>$e(s_1, c)$</th>
<th>$e(s_2, c)$</th>
<th>$\text{min}(e(s_1, c), e(s_2, c))$</th>
<th>$m_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>r</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>t</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 15: Runtime comparison on input size 1000, scaling threshold
4.2.3.1 Runtime Evaluation

In our first series of experiments, we evaluated the runtime of all filter combinations against the naive approach on a small dataset containing 1000 labels from DBpedia. The results of our evaluation are shown in Figure 15. This evaluation suggests that all filters outperform the naive approach. Moreover, the combination of all filters leads to the best overall runtime in most cases. Interestingly, the character-based filter leads to a significant reduction of the number of comparisons (see Figure 16) by more than 2 orders of magnitude. However, the runtime improvement is not as substantial. This result seems to indicate that the lookup in the character indexes is very time-demanding. We will thus aim to improve our character indexing in future work. Overall, the results on this dataset already shows that we outperform the naive approach by more than an order of magnitude when $\theta$ is high. The runtimes on a larger sample of size $10^4$ show an even better improvement (see Figure 17). This suggests that the relative improvement of our approach improves with the size of the problem.

![Figure 16: Comparison of number of similarity computations on input size 1000, scaling threshold](image)

4.2.3.2 Scalability Evaluation

The aim of the scalability evaluation was to measure how well our approach deals with datasets of growing size datasets. In our first set of experiments, we looked at the growth of the runtime of our approach on datasets of growing sizes. Our results suggest that our approach grows linearly with the number of labels contained in $S$ and $T$ (see Figure 19). This suggests that the runtime of our approach can be easily predicted for large datasets, which is of importance when asking users to wait for the results of the computation. In the second series of scalability experiments we looked at the runtime behaviour of our approach on a large dataset with $10^5$ labels. Our results suggest that the runtime of our approach falls superlinearly with an increase of the threshold $\theta$ (see Figure 18). This behaviour suggest that our approach is especially useful on clean datasets, where high thresholds can be used for link discovery.
Figure 17: Runtimes on sample of DBpedia labels with size $10^4$, scaling threshold

Figure 18: Runtimes on sample of DBpedia labels with size $10^5$, scaling threshold
4.2.3.3 **Comparison with existing approaches**

We compared our approach with SILK 2.6.0. To this end, we retrieved all rdfs:label of instances of subclasses of Person. We only compared with SILK on small datasets (i.e., on classes with small numbers of instances) as the results on these small datasets already showed that we outperform SILK consistently.¹⁵ Our results are shown in Table 3. They suggest that the absolute difference in runtime grows with the size of the datasets. Thus, we did not consider testing larger datasets against SILK as in the best case, we were already 4.7 times faster than SILK (Architect dataset, θ = 0.95).

<table>
<thead>
<tr>
<th>DBpedia Class</th>
<th>Size</th>
<th>OA(0.8)</th>
<th>OA(0.9)</th>
<th>OA(0.95)</th>
<th>SILK(0.8)</th>
<th>SILK(0.9)</th>
<th>SILK(0.95)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>9509</td>
<td>15.07</td>
<td>10.13</td>
<td>6.38</td>
<td>27</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Architect</td>
<td>3544</td>
<td>5.58</td>
<td>5.48</td>
<td>2.32</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Criminal</td>
<td>5291</td>
<td>11.54</td>
<td>7.77</td>
<td>4.52</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

4.3 **HELIOS**

Current declarative frameworks for link discovery such as LIMES [46] and SILK [25] take a link specification (short: LS, also called linkage rule [25]) as input and convert it internally into a sequence of operations which is then executed. While relying on time-efficient algorithms (e.g., PPJoin+ [65] and H/RE³ [45]) for single operation has been shown to be very time-efficient [46], the optimization of the execution of the whole LS within this paradigm has been payed little attention to.

We address this potential gap in scalability by presenting Helios, the (to the best of our knowledge) first execution optimizer for LD. Helios aims to reduce the costs necessary to execute a LS. To achieve this goal, our

¹⁵We ran SILK with -Dthreads = 1 for the sake of fairness.
The user interface for the tool can be accessed via the SAIM interface at 
operators of atomic components:

Formally, most LD tools aim to discover the set

\[ \{ (s, t) \in S \times T : R(s, t) \} \]

of source resources (for example descriptions of persons) and a set \( T \) of target resources. To achieve this goal, declarative LD frameworks rely on LS, which describe the conditions under which \( R(s, t) \) can be assumed to hold for a pair \((s, t) \in S \times T\). Several grammars have been used for describing LS in previous works \[45, 25, 55\]. In general, these grammars assume that LS consist of two types of atomic components: similarity measures \( m \), which allow comparing property values of input resources and operators \( op \), which can be used to combine these similarities to more complex specifications.

Without loss of generality, a similarity measure \( m \) can be defined as a function \( m : S \times T \rightarrow [0, 1] \). We use mappings \( M \subseteq S \times T \times [0, 1] \) to store the results of the application of a similarity function to \( S \times T \) or subsets thereof. We also store the results of whole link specifications in mappings. The set of all mappings is denoted by \( \mathcal{M} \). We call a measure atomic iff it relies on exactly one similarity measure \( \sigma \) (e.g., the edit similarity, dubbed \( \text{edit} \))\(^{17}\) to compute the similarity of a pair \((s, t) \in S \times T\) with respect to the (list of) properties \( p_s \) of \( s \) and \( p_t \) of \( t \) and write \( m = \sigma(p_s, p_t) \). A similarity measure \( m \) is either an atomic similarity measure or the combination of two similarity measures via a metric operator such as \( \text{max} \), \( \text{min} \) or linear combinations. For example, \( \text{edit}(s, \text{label}, t, \text{label}) \) is an atomic measure while \( \max(\text{edit}(s, \text{label}, t, \text{label}), \text{edit}(s, \text{age}, t, \text{age})) \) is a complex similarity measure.

We define a filter as any function which maps a mapping \( M \) to another mapping \( M' \). Similarity filters \( f(m, \theta) \) return \( f(m, \theta, M) = \{(s, t, r') : \exists r : (s, t, r) \in M \land m(s, t) \geq \theta \land \text{edit}(s, \text{age}, t, \text{age}) \} \). Threshold filters \( i(\theta) \) return \( i(\theta, M) = \{(s, t, r) \in M : r \geq \theta \} \). Note that \( i(0, M) = M \) and that we sometimes omit \( M \) from similarity filters for the sake of legibility.

\(^{16}\)Helios was implemented in the LIMES framework. All information to the tool can be found at http://limes.sf.net. A graphical user interface for the tool can be accessed via the SAIM interface at http://aksw.org/projects/SAIM.

\(^{17}\)We define the edit similarity of two strings \( s \) and \( t \) as \((1 + \text{lev}(s, t))^{-1}\), where \( \text{lev} \) stands for the Levenshtein distance.
We call a specification \textit{atomic} when it consists of exactly one filtering function. For example, applying the atomic specification \( f(\text{edit}(\text{ex:label}, \text{ex:label}), 1) \) to our input data leads to the mapping \( \{(\text{ex1:P3, ex2:P4, 1}), (\text{ex1:P2, ex2:P2, 1}), (\text{ex1:P4, ex2:P4, 1})\} \). A complex specification can be obtained by combining two specifications \( L_1 \) and \( L_2 \) by (1) a mapping operator (that allows merging the mappings which result from \( L_1 \) and \( L_2 \)) and (2) a subsequent filter that allows postprocessing the results of the merging.\footnote{We rely on binary operators throughout this work because \( n \)-ary set operators can always be mapped to a sequence of binary operators.} In the following, we limit ourselves to the operators based on \( \cup \), \( \cap \) and \( \setminus \) (set difference), as they are sufficient to describe any operator based on set operators. We extend these operators to mappings as follows:

\[
\begin{align*}
M_1 \cap M_2 &= \{(s, t, r) : \exists a, b (s, t, a) \in M_1 \land (s, t, b) \in M_2 \land r = \min(a, b)\}. \\
M_1 \cup M_2 &= \{(s, t, r) : (\neg \exists(s, t, a) \in M_1 \land (s, t, r) \in M_2) \lor (\neg \exists(s, t, b) \in M_2 \land (s, t, r) \in M_1) \lor \\
& (\exists(s, t, a) \in M_1 \land \exists(s, t, b) \in M_2 \land r = \max(a, b))\}. \\
M_1 \setminus M_2 &= \{(s, t, r) \in M_1 : \neg \exists(s, t, a) \in M_2\}.
\end{align*}
\]

For example, if \( M_1 = \{(\text{ex1:P1, ex2:P2, 1}), (\text{ex1:P1, ex2:P3, 1})\} \) and \( M_2 = \{(\text{ex1:P1, ex2:P2, 0.5})\} \) then \( M_1 \cup M_2 = M_1, M_1 \cap M_2 = M_2 \) and \( M_1 \setminus M_2 = \{(\text{ex1:P1, ex2:P3, 1})\} \).

Based on this grammar, we can regard all LS as \textit{bi-partite directed trees} \( L = (V(L), E(L)) \) which abide by the following restrictions:

1. The vertices of \( L \) can be either filter nodes \( f \in F \) or operator nodes \( op \in OP \), i.e., \( V(L) = F \cup OP \). The leaves and the root of \( L \) are always filter nodes. The leaves are filters that run on \( S \times T \).

2. Edges in \( L \) can only exist between filters and operators, i.e., \( E(L) \subseteq (F \times OP) \cup (OP \times F) \).

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Persons1 graph & Persons2 graph \\
\hline
ex1:P1 ex:label "Anna"@en . & ex2:P1 ex:label "Ana"@en . \\
ex1:P1 ex:age "12"8sd:integer . & ex2:P1 ex:age "12"8sd:integer . \\
ex1:P1 a ex:Person . & ex2:P1 a ex:Person . \\
ex1:P2 ex:label "Jack"@en . & ex2:P2 ex:label "Jack"@en . \\
ex1:P2 ex:age "15"8sd:integer . & ex2:P2 ex:age "14"8sd:integer . \\
ex1:P2 a ex:Person . & ex2:P2 a ex:Person . \\
ex1:P3 ex:label "John"@en . & ex2:P3 ex:label "Joe"@en . \\
ex1:P3 ex:age "16"8sd:integer . & ex2:P3 ex:age "16"8sd:integer . \\
ex1:P3 a ex:Person . & ex2:P3 a ex:Person . \\
ex1:P4 a ex:Person . & ex2:P4 a ex:Person . \\
\hline
\end{tabular}
\caption{Exemplary graphs.}
\end{table}
Figure 20: A LS for linking the datasets Person1 and Person2. The filter nodes are rectangles while the operator nodes are circles. \( \text{eucl}(s.\text{age}, t.\text{age}) = (1 + |s.\text{age} - t.\text{age}|)^{-1} \). This LS can be transliterated \( i(\cap(f(\text{edit}(\text{label}, \text{label}), 0.3), f(\text{eucl}(\text{age}, \text{age}), 0.5)), 0.5) \).

An example of a LS is shown in Figure 20. We call this representation of LS their NF. In the rest of this section, we deal exclusively with finite specifications, i.e., specifications such that their NF contains a finite number of nodes. We call the number of filter nodes of a specification \( L \) the size of \( L \) and denote it \( |L| \). For example, the size of the specification in Figure 20 is 3. We dub the direct child of \( L \)'s root the operator of \( L \). For example, the operator of the specification in Figure 20 is \( \cap \). We call a LS \( L' \) a sub-specification of \( L \) (denoted \( L' \subseteq L \)) if \( L' \)'s NF is a sub-tree of \( L \)'s NF that abides by the definition of a specification (i.e., if the root of \( L' \)'s NF is a filter node and the NF of \( L' \) contains all children of \( L' \) in \( L \)). For example, \( f(\text{edit}(\text{label}, \text{label}), 0.3) \) is a sub-specification of our example. We call \( L' \) a direct sub-specification of \( L \) (denoted \( L' \subset L \)) if \( L' \) is a sub-specification of \( L \) whose root node a grandchild of the \( L \)'s root. For example, \( f(\text{edit}(\text{label}, \text{label}), 0.3) \) is a direct sub-specification of the LS shown in Figure 20. Finally, we transliterate LS by writing \( f(m, \theta, \text{op}(L_1, L_2)) \) where \( f(m, \theta) \) is \( L \)'s root, \( \text{op} \) is \( L \)'s operator, \( L_1 \subset L \) and \( L_2 \subset L \).

4.3.1.2 Execution Plans

We define an execution plan \( P \) as a sequence of processing steps \( p_1, \ldots, p_n \) of which each is drawn from the set \( \mathcal{A} \times \mathbb{R} \times \mathcal{T} \times \mathcal{M} \times \mathcal{M} \), where:

1. \( \mathcal{A} \) is the set of all actions that can be carried out. This set models all the processing operations that can be carried out when executing a plan. These are:
   - \( \text{run} \), which runs the computation of filters \( f(m, \theta) \) where \( m \) is an atomic measure. This action can make use of time-efficient algorithms such as \( \mathcal{HR}^3 \).
   - \( \text{filter} \), which runs filters \( f(m, \theta) \) where \( m \) is a complex measure.
   - \( \text{filterout} \), which runs the negation of \( f(m, \theta) \).
   - Mapping operations such as \( \text{union} \), \( \text{intersection} \) and \( \text{minus} \) (mapping difference) and
   - \( \text{return} \), which terminates the execution and returns the final mapping.

   The result of each action (and therewith of each processing step) is a mapping.

2. \( \mathbb{R} \) is the set of all complex measures as described above united with the \( \emptyset \)-measure, which is used by actions that do not require measures (e.g., \( \text{return} \)).

3. \( \mathcal{T} \) is the set of all possible thresholds (generally \([0, 1]\)) united with the \( \emptyset \)-threshold for actions that do not require any threshold (e.g., \( \text{union} \)) and

4. \( \mathcal{M} \) is the set of all possible mappings, i.e., the powerset of \( S \times T \times [0, 1] \).

We call the plan \( P \) atomic if it consists of exactly one processing step. An execution planner \( EP \) is a function which maps a LS to an execution plan \( P \). The canonical planner \( EP_0 \) is the planner that runs specification
Table 5: Plans for the specification shown in Figure 20

<table>
<thead>
<tr>
<th>Canonical Plan</th>
<th>Abbreviated Canonical Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1 = \langle \text{run, edit(label,label)}, 0.3, \emptyset, \emptyset \rangle$</td>
<td>$M_1 = \langle \text{run, edit(label,label)}, 0.3 \rangle$</td>
</tr>
<tr>
<td>$M_2 = \langle \text{run, eucl(age,age)}, 0.5, \emptyset, \emptyset \rangle$</td>
<td>$M_2 = \langle \text{run, eucl(age,age)}, 0.5 \rangle$</td>
</tr>
<tr>
<td>$M_3 = \langle \text{intersection,} \emptyset, \emptyset, M_1, M_2 \rangle$</td>
<td>$M_3 = \langle \text{intersection,} M_1, M_2 \rangle$</td>
</tr>
<tr>
<td>$M_4 = \langle \text{return,} \emptyset, \emptyset, M_3, \emptyset \rangle$</td>
<td>$M_4 = \langle \text{return,} M_3 \rangle$</td>
</tr>
</tbody>
</table>

Alternative Plan 1 (abbreviated)  

| $M_1 = \langle \text{run, edit(label,label)}, 0.3 \rangle$ | $M_1 = \langle \text{run, eucl(age,age)}, 0.5 \rangle$ |
| $M_2 = \langle \text{filter, eucl(age,age)}, 0.5, M_1 \rangle$ | $M_2 = \langle \text{filter, edit(label,label)}, 0.3, M_1 \rangle$ |
| $M_3 = \langle \text{return,} M_2 \rangle$ | $M_3 = \langle \text{return,} M_2 \rangle$ |

Alternative Plan 2 (abbreviated)

In postorder, i.e., by traversing the NF of LS in the order left-right-root. The approach currently implemented by LIMES [46] is equivalent to $EP_0$. For example, the plan generated by $EP_0$ for Figure 20 is shown in the left column of Table 5. For the sake of brevity and better legibility, we will use abbreviated versions of plans that do not contain $\emptyset$ symbols. The abbreviated version of the plan generated by $EP_0$ for the specification in Figure 20 is shown in the right column of Table 5. We call two plans equivalent when they return the same results for all possible $S$ and $T$. We call a planner complete when it always returns plans that are equivalent to those generated by $EP_0$.

The insight behind our work is that equivalent plans can differ significantly with respect to their runtime. For example, the canonical plan shown in Table 5 would lead to 32 similarity computations (16 for edit and 16 for euclidean) and one mapping intersection, which can be computed by using 16 lookups. If we assume that each operation requires 1ms, the total runtime of this plan would be 48ms. The alternative plan 1 shown in Table 5 is equivalent to the plans in Table 5 but only runs 16 computations of edit (leading to $M_1$ of size 6) and 6 computations of euclidean on the data contained in $M_1$. The total runtime of this plan would thus be 22ms. Detecting such runtime-efficient and complete plans is the goal of Helios.

4.3.2 HELOIS Components

HELOIS is an optimizer for LS which consists of two main components: a rewriter (denoted RW) and a planner (denoted HP). Each LS $L$ to be processed is first forwarded to RW, which applies several algebraic transformation rules to transform $L$ into an equivalent LS $L'$ that promises to be more efficient to execute. The aim of HP is then to derive a complete plan $P$ for $L'$. This plan is finally sent to the execution engine, which runs the plan and returns a final mapping. In the following, we present each of these components. Throughout the formalization, we use $\rightarrow$ for logical implications and $\Rightarrow$ to denote rules.

4.3.2.1 The HELOIS Rewriter

RW implements an iterative rule-based approach to rewriting. Each iteration consists of three main steps that are carried out from leaves towards the root of the input specification. In the first step, sub-graphs of the input specification $L$ are replaced with equivalent sub-graphs which are likely to be more efficient to run. In
a second step, dependencies between nodes in $L$ are determined and propagated. The third step consists of removing portions of $L$ which do not affect the final results of $L$’s execution. These three steps are iterated until a fixpoint is reached.

**Step 1: Rewriting** Given a LS $L$, RW begins by rewriting the specification using algebraic rules dubbed leaf generation rules.

\[
\begin{align*}
\alpha m_1 + \beta m_2, \theta & \Rightarrow \alpha m_1 + \beta m_2, \theta \\
\end{align*}
\]

Figure 21: Leaf generation rule for linear combinations.

The leaf generation rules ($LR$) make use of relations between metric operators and specification operators to transform leaf nodes with complex measures into graphs whose leaves contain exclusively atomic measures. For example, the rule shown in Figure 21 transforms a filter that relies on the linear combinations of 2 measures into a LS with three filters whose leaves only contain atomic measures as described in [46]. While it might seem absurd to alter the original filter in this manner, the idea here is that we can now run specialized algorithms for $m_1$ and $m_2$, then compute the intersection $M$ of the resulting mapping and finally simply check each of the $(s, t) \in S \times T$ for whether it abides by the linear combination in the root filter. This approach is usually more time-efficient than checking each $(s, t) \in S \times T$ for whether it abides by the linear combination in the original specification. Similar rules can be devised for $\min$ (see Figure 22), $\max$ and the different average functions used in LD frameworks. After $L$ has been rewritten by the rules in $LR$, each of its leaves is a filter with atomic measures.

\[
\begin{align*}
\min(m_1, m_2), \theta & \Rightarrow \text{identity, } \theta \\
\end{align*}
\]

Figure 22: Rule for minimum. In the corresponding rule for maximum, the mapping union is used.

**Step 2: Dependency Detection and Propagation** The idea behind the use of dependencies is to detect and eliminate redundant portions of the specification. Consequently, RW implements two types of dependency-based rules: dependency detection rules and dependency propagation rules. Formally, we say that $L_1$ depends on $L_2$ (denoted $\text{depends}(L_1, L_2)$) if the mapping resulting from $L_1$ is a subset of the mapping resulting from $L_2$ for all possible $S$ and $T$. RW generates dependencies between leaves (which now only contain atomic measures) by making use of

\[
L_1 = f(m, \theta_1) \land L_2 = f(m, \theta_2) \land \theta_1 \geq \theta_2 \Rightarrow \text{depends}(L_1, L_2).
\]

(14)

Moreover, RW makes use of dependencies between several similarity and distance measures that are commonly used in literature. For example, the authors of [65] show that for two non-empty strings $x$ and $y$, $\text{jaccard}(x, y) \geq \theta \rightarrow \text{overlap}(x, y) \geq \frac{\theta}{1 + \theta}(|x| + |y|)$. Given that $|x| \geq 1$ and $|y| \geq 1$, we can infer that

\[
\text{jaccard}(x, y) \geq \theta \rightarrow \text{overlap}(x, y) \geq \frac{2\theta}{1 + \theta}.
\]

(15)
Thus, if \( L_1 = f(\text{jaccard}(p_1, p_2), \theta_1) \) and \( L_2 = f(\text{overlap}(p_1, p_2), \theta_2) \) with \( \theta_2 \leq \frac{2\theta_1}{1+\theta_1} \), then 
\( \text{depends}(L_1, L_2) \) holds. Currently, RW implements dependencies between the \text{overlap}, \text{trigrams} and the \text{jaccard} similarities discussed in [65].

Leaf-level dependencies can be propagated towards the root of the specification based on the following rules:

\[
\begin{align*}
\text{p1:} & \quad L = i(\theta, \text{op}(L_1, L_2)) \land L_1 = f(m, \theta_1, \text{op}_1(L_{11}, L_{12})) \land L_2 = f(m, \theta_2, \text{op}_2(L_{21}, L_{22})) \land \theta_1 \geq \theta \land \theta_2 \geq \theta \\
\text{p2:} & \quad \text{depends}(L_1, L') \land \text{depends}(L_2, L') \land L = f(m, \theta, \cap(L_1, L_2)) \Rightarrow \text{depends}(L, L') \quad \text{(if all children of a conjunction depend on } L' \text{ then the father of this conjunction depends on } L').
\end{align*}
\]

\[
\begin{align*}
\text{p3:} & \quad L = f(m, 0, \cup(L_1, L_2)) \land (\text{depends}(L', L_1) \lor \text{depends}(L', L_2)) \Rightarrow \text{depends}(L', L) \quad \text{(if } L' \text{ depends on one child of a disjunction and the father of the disjunction has the threshold 0 then } L' \text{ depends on the father of the disjunction).}
\end{align*}
\]

**Step 3: Reduction** Given two specifications \( L_1 \subseteq L \) and \( L_2 \subseteq L \) with \text{depends}(L_1, L_2), we can now reduce the size of \( L = \text{filter}(m, \theta, \text{op}(L_1, L_2)) \) by using the following rules:

\[
\begin{align*}
\text{r1:} & \quad L' = \text{filter}(m, \theta, \cap(L_1, L_2)) \land \text{depends}(L_1, L_2) \Rightarrow L' := \text{filter}(m, \theta, L_1), \\
\text{r2:} & \quad L' = \text{filter}(m, \theta, \cup(L_1, L_2)) \land \text{depends}(L_1, L_2) \Rightarrow L' := \text{filter}(m, \theta, L_1), \\
\text{r3:} & \quad L' = \text{filter}(m, \theta, \setminus(L_1, L_2)) \land \text{depends}(L_1, L_2) \Rightarrow L' := \emptyset \text{ where := stands for overwriting.}
\end{align*}
\]

An example that elucidates the ideas behind \text{DR} is given in Figure 23. Set operators applied to one mapping are assumed to not alter the mapping.

The leaf generation terminates after at most as many iterations as the total number of atomic specifications used across all leaves of the input LS \( L \). Consequently, this step has a complexity of \( O(|L'|) \) where \( L' = \mathcal{L} \mathcal{R}(L) \). The generation of dependencies requires \( O(|L'|^2) \) node comparisons. Each time a reduction rule is applied, the size of the \( L' \) decreases, leading to reduction rules being applicable at most \(|L'|\) times. The complexity of the reduction is thus also \( O(|L'|) \). In the worst case of a left- or right-linear specification, the propagation of dependencies can reach the complexity \( O(|L'|^2) \). All three steps of each iteration thus have a complexity of at most \( O(|L'|^2) \) and the specification is at least one node smaller after each iteration. Consequently, the worst-case complexity of the rewriter is \( O(|L'|^3) \).

**4.3.2.2 The HELIOS Planner**

The goal of the HELIOS planner HP is to convert a given LS into a plan. Previous work on query optimization for databases have shown that finding the optimal plan for a given query is exponential in complexity [60]. The complexity of finding the perfect plan for a LS is clearly similar to that of finding a plan for a given query. To circumvent the complexity problem, we rely on the following optimality assumption: Given \( L_1 \subseteq L \) and \( L_2 \subseteq L \) with \( L = f(m, \theta, \text{op}(L_1, L_2)) \), a good plan for \( L \) can be derived from plans for \( L_1 \) and \( L_2 \). In the following, we begin by explaining core values that HP needs to evaluate a plan. In particular, we explain how HP evaluates atomic and complex plans. Thereafter, we present the algorithm behind HP and analyze its complexity.
To achieve this goal, we used the following exact solution to linear regression:

\[ \text{Note that the first entry of all these values because they be computed in linear time.} \]

Based on these assumptions, we derived the following selectivities for the input mappings selectivity of a plan \( P \) to be the probability that a pair \( (s, t) \) is returned after the execution of \( P \). Moreover, we assumed the input mappings \( M_1 \) (selectivity: \( s_1 \)) resp. \( M_2 \) (selectivity: \( s_2 \)) to be the results of independent computations. Based on these assumptions, we derived the following selectivities for \( \text{op}(M_1, M_2) \):

\[ \text{op} = \cap \rightarrow s(\text{op}) = s_1 s_2. \]

Figure 23: Example of propagation of dependencies. The dashed arrows represent dependencies. The dependencies from the left figure are first (using rule \( p_1 \)). Then, the reduction rule \( r_2 \) is carried out, leading to the specification on the right.

**Plan Evaluation HP** uses two values to characterize any plan \( P \): (1) the approximate runtime of \( P \) (denoted \( \gamma(P) \)) and (2) the selectivity of \( P \) (dubbed \( s(P) \)), which encodes the size of the mapping returned by \( P \) as percentage of \( |S \times T| \).

**Computing \( \gamma(P) \) and \( s(P) \) for atomic LS:** Several approaches can be envisaged to achieve this goal. In our implementation of HP, we used approximations based on sampling. The basic assumption behind our feature choice was that LD frameworks are usually agnostic of \( S \) and \( T \) before the beginning of the LD. Thus, we opted for approximating the runtime of atomic plans \( P \) by using \( |S| \) and \( |T| \) as parameters. We chose these values because they be computed in linear time.\(^{20}\) To approximate \( \gamma(P) \) for atomic plans, we generated source and target datasets of sizes 1000, 2000, \ldots, 10000 by sampling data from the English labels of DBpedia 3.8. We then stored the runtime of the measures implemented by our framework for different thresholds \( \theta \) between 0.5 and 1.\(^{21}\) The runtime of the \( i^{th} \) experiment was stored in the row \( y_i \) of a column vector \( Y \). The corresponding experimental parameters \( (1, |S|, |T|, \theta) \) were stored in the row \( r_i \) of a four-column matrix \( R \).

Note that the first entry of all \( r_i \) is 1 to ensure that we can learn possible constant factors. We finally computed the vector \( \Gamma = (\gamma_0, \gamma_1, \gamma_2, \gamma_3)^T \) such that

\[ \gamma(P) = \gamma_0 + \gamma_1 |S| + \gamma_2 |T| + \gamma_3 \theta. \quad (16) \]

To achieve this goal, we used the following exact solution to linear regression: \( \Gamma = (R^T R)^{-1} R^T Y \). The computation of \( s(P) \) was carried out similarly with the sole difference that the entries \( y_i \) for the computation of \( s(P) \) were \( \frac{M_i}{|S| \times |T|} \), where \( M_i \) is the size of the mapping returned by the \( i^{th} \) experiment. Figure 24 shows a sample of the results achieved by different algorithms in our experiments. The plan returned for the atomic \( LSf(m, \theta) \) is \( (\text{run}, m, \theta) \).

**Computing \( \gamma(P) \) and \( s(P) \) for complex LS:** The computation of the costs associated with atomic filter, filterout and operators was computed analogously to the computation of runtimes for atomic LS. For filters, the feature was the size of the input mapping. For non-atomic plans \( P \), we computed \( \gamma(P) \) by summing up the \( \gamma(p_i) \) for all the steps \( p_i \) included in the plan. The selectivity of operators was computed based on the selectivity of the mappings that served as input for the operators. To achieve this goal, we assumed that the selectivity of a plan \( P \) to be the probability that a pair \( (s, t) \) is returned after the execution of \( P \). Moreover, we assumed the input mappings \( M_1 \) (selectivity: \( s_1 \)) resp. \( M_2 \) (selectivity: \( s_2 \)) to be the results of independent computations. Based on these assumptions, we derived the following selectivities for \( \text{op}(M_1, M_2) \):

\[ \text{op} = \cap \rightarrow s(\text{op}) = s_1 s_2. \]

\(^{20}\) Other values can be used for this purpose but our results suggest that using \( |S| \) and \( |T| \) is sufficient in most cases.

\(^{21}\) We used the same hardware as during the evaluation.
D3.1.2 - v. 1.0

(a) Runtimes for trigrams.

(b) Heatmap for trigrams.

(c) Runtimes for levenshtein.

(d) Heatmap for levenshtein.

Figure 24: Runtimes achieved by PPJoin+ (trigrams) and EDJoin (levenshtein) for $\theta = 0.5$. The x-axis of the heatmap show $|S|$ in thousands, while the y-axis shows $|T|$ in thousands. The color bars show the runtime in ms.

- $op = \cup \rightarrow s(op) = 1 - (1 - s_1)(1 - s_2)$.
- $op = \setminus \rightarrow s(op) = s_1(1 - s_2)$.

The HP algorithm The core of the approach implemented by HP is shown in Algorithm 8. For atomic specifications $f(m, \theta)$, HP simply returns $(\text{run}, m, \theta)$ (getBestPlan method in Algorithm 8). If different algorithms which allow running $m$ efficiently are available, HP chooses the implementation that leads to the smallest runtime $\gamma(P)$. Note that the selectivity of all algorithms that allow running $m$ is exactly the same given that they must return the same mapping. If the specification $L = (m, \theta, op(L_1, L_2))$ is not atomic, HP’s core approach is akin to a divide-and-conquer approach. It first devises a plan for $L_1$ and $L_2$ and then computes the costs of different possible plans for $op$. For $\cap$ for example, the following three plans are equivalent:

1. Canonical plan. This plan simply consists of merging (via the concatenate method in Algorithm 8) the results of the best plans for $L_1$ and $L_2$. Consequently, the plan consists of (1) running the best plan for $L_1$ (i.e., $Q_1$ in Algorithm 8), (2) running the best plan for $L_2$ (i.e., $Q_2$ in Algorithm 8), then (3) running the intersection action over the results of $Q_1$ and $Q_2$ and finally (4) running filter over the result of the intersection action.
2. **Filter-right plan.** This plan uses \( f(m_2, \theta_2) \) as a filter over the results of \( Q_1 \). Consequently, the plan consists of (1) running the best plan for \( L_1 \), then (2) running the filter action with measure \( m_2 \) and threshold \( \theta_2 \) over the results of \( Q_1 \) and finally (3) running filter with measure \( m \) and threshold \( \theta \) over the previous result.

3. **Filter-left plan.** Analogous to the filter-right plan with \( L_1 \) and \( L_2 \) reversed.

Similar approaches can be derived for the operators \( \cup \) and \( \setminus \) as shown in Algorithm 8. HP now returns the least costly plan as result (getLeastCostly method in Algorithm 8). This plan is finally forwarded to the execution engine which runs the plan and returns the resulting mapping.

Given that the alternative plans generated by HP are equivalent and that HP always chooses one of this plan, our algorithm is guaranteed to be complete. Moreover, HP approximates the runtime of at most 3 different plans per operator and at most \( k \) different plans for each leaf of the input specification (where \( k \) is the maximal number of algorithms that implements a measure \( m \) in our framework). Consequently, the runtime complexity of HP is \( O(\max\{k, 3\} \times |L|) \).

**Algorithm 8:** The plan method

```plaintext
if L is atomic then
    P = getBestPlan(L);
else
    if L = f(m, θ, op(L1)) then
        P = getBestPlan(L1)
    else
        Q1 := PLAN(L1)
        Q2 := PLAN(L2)
        if L = f(m, θ, ∩(L1, L2)) then
            P1 := CONCATENATE(intersection, Q1, Q2)
            P2 := CONCATENATE(filter(m2, θ2), Q1)
            P := getLeastCostly(P0, P1, P2)
        else if L = f(m, θ, ∪(L1, L2)) then
            P0 := CONCATENATE(union, Q1, Q2)
            P1 := CONCATENATE(union, filter(m2, θ2), S × T1), Q2)
            P2 := CONCATENATE(union, filter(m1, θ1, S × T1), Q1)
            P := getLeastCostly(P0, P1, P2)
        else if L = f(m, θ, \setminus(L1, L2)) then
            P0 := CONCATENATE(minus, Q1, Q2)
            P1 := CONCATENATE(filterout(m2, θ2), Q2)
            P := getLeastCostly(P0, P1)
        end if
    end if
end if
a0 := filter(m, θ)
P := CONCATENATE(a0, P)
return P
```

### 4.3.3 Evaluation

#### 4.3.3.1 Experimental Setup

The aim of our evaluation was to measure the runtime improvement of Helios for the overall runtime of LS. We thus compared the runtimes of \( EP_0 \) (i.e., LIMES), RW (i.e., RW + \( EP_0 \)), HP and Helios (i.e., RW +HP) in our experiments. We chose LIMES because it has been shown to be very time-efficient in previous work [46]. We considered manually created and automatically generated LS. All experiments were carried out on server running Ubuntu 12.04. In each experiment, we used a single kernel of a 2.0GHz AMD Opteron processor with 10GB RAM.
The manually created LS were selected from the LATC repository. We selected 17 LS which relied on SPARQL endpoints that were alive or on data dumps that were available during the course of the experiments. The specifications linked 18 different datasets and had sizes between 1 and 3. The small sizes were due to humans tending to generate small and non-redundant specifications.

The automatic specifications were generated during a single run of specification learning algorithm EAGLE on four different benchmark datasets described in Table 7. The mutation and crossover rates were set to 0.6 while the number of inquiries per iteration was set to 10. The population size was set to 10. The sizes of the specifications generated by EAGLE varied between 1 and 11. We compared 1000 LS on the OAEI 2010 Restaurant and the DBLP-ACM dataset each, 80 specifications on the DBLP-Scholar dataset and 100 specifications on LGD-LGD. We chose to use benchmark datasets to ensure that the specifications used in the experiments were of high-quality w.r.t. the F-measure they led to. Each specification was executed 10 times. No caching was allowed. We report the smallest runtimes over all runs for all configurations to account for possible hardware and I/O influences.

4.3.3.2 Results on Manual Specifications

The results of our experiments on manual specifications are shown in Table 6 and allow deriving two main insights: First, Helios can improve the runtime of atomic specifications (which made up 62.5% of the manual LS). This result is of tremendous importance as it suggests that the overhead generated by Helios is mostly insignificant, even for specifications which lead to small runtimes (e.g., DBP-DataGov requires 8ms). Moreover, our experiments reveal that Helios achieves a significant improvement of the overall runtime of specifications with sizes larger than 1 (37.5% of the manual LS). In the best case, Helios is 49.5 times faster than EP0 and can reduce the runtime of the LS LDG-DBP (A) from 52.7s to 1.1s by using a filter-left plan. Here, we see that the gain in runtime generated by Helios grows with $|S| \times |T|$. This was to be expected as a good plan has more effect when large datasets are to be processed. Overall, Helios outperforms LIMES' canonical planner on all non-atomic specifications. On average, Helios is 4.3 times faster than the canonical planner on LS of size 3.

4.3.3.3 Results on Automatic Specifications

Overall, our results on automatic specifications show clearly that Helios outperforms the state of the art significantly. In Table 7, we show the average runtime of EP0, RW, HP and Helios on four different datasets of growing sizes. The overall runtime of Helios is clearly superior to that of EP0 on all datasets. As expected, the gain obtained by using Helios grows with the size of $|S| \times |T|$. In particular, the results on the very small Restaurant dataset support the results achieved on the manual specifications. While HP alone does not lead to a significant improvement, Helios leads to an improvement of the overall runtime by 6.35%. This improvement is mostly due to RW eliminating filters and therewith altering the plans generated by HP. These alterations allow for shorter and more time-efficient plans.

On the larger DBLP-ACM dataset, Helios achieves a runtime that is up to 185.8 times smaller than that of EP0 (e.g., for $f(\neg f(jaccard(authors, authors), 0.93), f(edit(venue, venue), 0.93))$, 0.53)). Yet, given that the runtime approximations are generic, Helios sometimes generated plans that led to poorer runtimes. In the worst case, a plan generated by Helios was 6.5 times slower than the plan generated by EP0 (e.g., for...
Table 6: Comparison of runtimes on manual specifications. The top portion of the table shows runtimes of specifications of size 1 while the bottom part shows runtimes on specifications of size 3. EVT stands for Eventseer, DF for DogFood, (P) stands for person, (A) stands for airports, (U) stands for universities, (E) stands for events. The best runtimes are in bold.

| Source - Target | $|S| \times |T|$ | $E_P_0$ (ms) | $RW$ (ms) | $HP$ (ms) | Helios (ms) | Gain (ms) |
|-----------------|------------------------|----------------|------------|------------|-------------|----------|
| DBP - Datagov   | $1.7 \times 10^3$      | 8             | 8          | 8          | 8           | 0        |
| RKB - DBP       | $2.2 \times 10^3$      | 1             | 1          | 1          | 1           | 0        |
| Epo - DBP       | $73.0 \times 10^3$     | 54            | 53         | 54         | 53          | 1        |
| Rail - DBP      | $133.2 \times 10^3$    | 269           | 268        | 268        | 268         | 1        |
| Stad - Rmon     | $341.9 \times 10^3$    | 25            | 23         | 15         | 14          | 11       |
| EVT - DF (E)    | $531.0 \times 10^3$    | 893           | 906        | 909        | 905         | -12      |
| Climb - Rail    | $1.9 \times 10^6$      | 41            | 40         | 40         | 40          | 1        |
| DBLP - DataSW   | $92.2 \times 10^6$     | 59            | 59         | 58         | 54          | 5        |
| EVT - DF (P)    | $148.4 \times 10^6$    | 2,477         | 2,482      | 2,503      | 2,434       | 43       |
| EVT - DBLP      | $161.0 \times 10^6$    | 9,654         | 9,575      | 9,613      | 9,612       | 42       |
| DBP - OpenEI    | $10.9 \times 10^3$     | 2             | 2          | 2          | 2           | 0        |
| DBP - GSpecies  | $94.2 \times 10^3$     | 120           | 119        | 120        | 119         | 1        |
| Climb - DBP     | $312.4 \times 10^3$    | 55            | 55         | 55         | 55          | 0        |
| DBP - LGD (E)   | $34.1 \times 10^6$     | 2,259         | 2,133      | 1,206      | 1,209       | 1,050    |
| Climb - LGD     | $215.0 \times 10^6$    | 24,249        | 24,835     | 3,497      | 3,521       | 20,728   |
| DBP - LGD (A)   | $383.8 \times 10^6$    | 52,663        | 59,635     | 1,066      | 1,064       | 51,599   |
| LGD - LGD       | $509.3 \times 10^9$    | 46,604        | 38,560     | 32,831     | 22,497      | 24,107   |

The best runtimes are in bold.

On average, Helios is 38.82% faster than $E_P_0$. Similar results can be derived from DBLP-Scholar, where Helios is 29.61% faster despite having run on only 80 specifications. On our largest dataset, LinkedGeoData, the time gain is even larger with HELIOS being 46.94% faster. Note that this improvement is very relevant for end users, as it translates to approximately 1h of runtime gain for each iteration of our experiments. Here, the best plan generated by Helios is 314.02 times faster than $E_P_0$. Moreover, we can clearly see the effect of RW with average runtime improvement of 5.1% (see Figure 25).

We regard our overall results as very satisfactory given that the algorithms underlying $E_P_0$ are in and of themselves already optimized towards runtime. Still, by combining them carefully, Helios can still cut down the overall runtime of learning algorithms and even of manually created link specifications. To ensure that our improvements are not simply due to chance, we compared the distribution of the cumulative runtimes of $E_P_0$ and RW, HP and Helios as well $E_P_0$ and Helios by using a Wilcoxon paired signed rank test at a significance level of 95%. On all datasets, all tests return significant results, which shows that the RW, HP and Helios lead to statistically significant runtime improvements.

4.4 FOX

One of the first research papers in the field of named entity recognition (NER) was presented in 1991 [39]. Today, more than two decades later, this research field is still highly relevant for manifold communities including Semantic Web Community, where the need to capture and to translate the content of natural language (NL) with the help of NER tools arises in manifold semantic applications [29, 19, 20, 15, 50]. The NER tools that resulted
Table 7: Summary of the results on automatically generated specifications. \(|L|\) shows for the average size ± standard deviation of the specifications in the experiment. \(F_1\) shows the F-measure achieved by EAGLE on the dataset. The runtimes in four rightmost columns are the average runtimes in seconds.

| \(|S| \times |T|\) | \(|L|\) | \(F_1\) | \(E\) | RW | HP | Helios |
|---|---|---|---|---|---|---|
| Restaurants | 72.3 \times 10^3 | 4.44±1.79 | 0.89 | 0.15 | 0.15 | 0.14 |
| DBLP-ACM | 6.0 \times 10^6 | 6.61±1.32 | 0.99 | 1.38 | 1.37 | 1.00 | 0.99 |
| DBLP-Scholar | 168.1 \times 10^6 | 6.42±1.47 | 0.91 | 17.44 | 17.41 | 13.54 | 13.46 |
| LGD-LGD | 5.8 \times 10^9 | 3.54±2.15 | 0.98 | 102.33 | 97.40 | 72.19 | 69.64 |

from more than 2 decades of research now implement a diversity of algorithms that rely on a large number of heterogeneous formalisms. Consequently, these algorithms have diverse strengths and weaknesses.

GeoLift can consume NL to generate RDF data and relies on the FOX framework to this end. While several other frameworks could be used to this end, most of the existing solutions rely solely on one of the formalisms developed for NER or simply merge the results of several tools (e.g., by using simple voting). By doing so, current approaches fail to make use of the diversity of current NER algorithms. On the other hand, it is a well-known fact that algorithms with diverse strengths and weaknesses can be aggregated in various ways to create a system that outperforms the best individual algorithms within the system [64]. This learning paradigm is known as ensemble learning. While previous works have already suggested that ensemble learning can be used to improve NER [50], no comparison of the performance of existing supervised machine-learning approaches for ensemble learning on the NER task has been presented so far.

We address this research gap by presenting FOX. In this evaluation, we use four state-of-the-art NER algorithms, fifteen different machine learning algorithms and five datasets. The statistical significance our results is ensured by using Wilcoxon signed-rank tests.

The goal of our evaluation is to answer the following questions:

1. Does NER based on ensemble learning achieve higher f-scores than the best NER tool within the system?
2. Does NER based on ensemble learning achieve higher f-scores than simple voting based on the results of the NER tools?
3. Which ensemble learning approach achieves the best f-score for the NER task?
4.4.1 Overview

4.4.1.1 Named Entity Recognition

NER encompasses two main tasks: (1) The identification of names²⁵ such as “Germany”, “University of Leipzig” and “G. W. Leibniz” in a given unstructured text and (2) the classification of these names into predefined entity types²⁶, such as Location, Organization and Person. In general the NER task can be viewed as the sequential prediction problem of estimating the probabilities

\[ P(y_i|x_{i-k}...x_{i+l}, y_{i-m}...y_{i-1}) \]

where \( x = (x_1, ..., x_n) \) is an input sequence (i.e., the preprocessed input text) and \( y = (y_1, ..., y_n) \) the output sequence (i.e., the entity types) [58].

4.4.1.2 Ensemble Learning

The goal of an ensemble learning algorithm \( S \) is to generate a classifier \( F \) with a high predictive performance by combining the predictions of a set of \( m \) basic classifiers \( C_1, \ldots, C_m \) [12]. One central observation in this respect, is that combining \( C_1, \ldots, C_m \) can only lead to a high predictive performance when these classifiers are accurate and diverse [66]. Several approaches have been developed to allow an efficient combination of basic classifiers. The simplest strategy is voting, where each input token is classified as belonging to the class that was predicted by the largest number of basic classifiers [12]. Voting can be extended to weighted voting, where each of the basic classifiers is assigned a weight and \( S \) returns the class with the highest total prediction weight. More elaborate methods try to ensure the diversity of the classifiers. Approaches that aim to achieve this goal include drawing random samples (with replacement) from the training data (e.g., bagging, [6]) or generating sequences of classifiers of high diversity that are trained to recognize each other’s mistakes (e.g., boosting, [59]). The results of all classifiers are finally combined via weighted voting.

Here, we consider ensemble learning for NER. Thanks to the long research tradition on the NER topic, the diversity and accuracy of the tools is already available and can be regarded as given. However, classical ensemble learning approaches present the disadvantage of relying on some form of weighted vote on the output of the classifiers. Thus, if all classifiers \( C_i \) return wrong results, classical ensemble learning approaches are bound to make the same mistake [12]. In addition, voting does not take the different levels of accuracy of classifiers for different entity types into consideration. Rather, it assigns a global weight to each classifier that describes its overall accuracy. Based on these observations, we decided to apply ensemble learning for NER based at entity-type level. The main advantage of this ensemble-learning setting is that we can now assign different weights to each tool-type pair.

Formally, we model the ensemble learning task at hand as follows: Let the matrix \( M^{mt \times n} \) (Equation 17) illustrate the input data for \( S \), where \( P^m_{n,t} \) are predictions of the \( m \)-th NER tool that the \( n \)-th token is of the \( t \)-th type.

\[
\begin{pmatrix}
P_{1,1} & \cdots & P_{1,t} & P_{1,1}^2 & \cdots & P_{1,t}^2 & \cdots & P_{1,1}^m & \cdots & P_{1,t}^m \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots \\
P_{m,1} & \cdots & P_{m,t} & P_{m,1}^2 & \cdots & P_{m,t}^2 & \cdots & P_{m,1}^m & \cdots & P_{m,t}^m \\
\end{pmatrix}
\]

(17)

The goal of ensemble learning for NER is to detect a classifier that leads to a correct classification of each of the \( n \) tokens into one of the types \( t \).

²⁵Also referred as instances.
²⁶Also referred as classes.
4.4.2 Evaluation

We performed a thorough evaluation of ensemble learning approaches by using five different datasets and running a 10-fold cross-validation for 15 algorithms. In this section, we present the pipeline and the setup for our evaluation as well as our results.

4.4.2.1 Pipeline

Figure 26 shows the workflow chart of our evaluation pipeline. In the first step of our evaluation pipeline, we preprocessed our reference dataset to extract the input text for the NER tools as well as the correct NEs, which we used to create training and testing data. In the second step, we made use of all NER tools with this input text to calculate the predictions of all entity types for each token in this input. At this point, we represented the output of the tools as matrix (see Equation 17). Thereafter, the matrix was randomly split into 10 disjoint sets as preparation for a 10-fold cross-validation. We trained the different classifiers at hand (i.e., $S$) with the training dataset (i.e., with 9 of 10 sets) and tested the trained classifier with the testing dataset (i.e., with the leftover set). To use each of the 10 sets as testing set once, we repeated training and testing of the classifiers 10 times and used the disjoint sets accordingly. Furthermore, the pipeline was repeated 10 times to deal with non-deterministic classifiers. In the last step, we compared the classification of the 10 testing datasets with the oracle dataset to calculate measures for the evaluation.

We ran our pipeline on 15 ensemble learning algorithms. We carried out both a token-based evaluation and an entity-based evaluation. In the token-based evaluation, we regarded partial matches of multi-word units as being partially correct. For example, our gold standard considered “Federal Republic of Germany” as being an instance of Location. If a tool generated “Germany” as being a location and omitted “Federal Republic of”, it was assigned 1 true positive and 3 false negatives. The entity-based evaluation only regarded exact matches as correct. In the example above, the entity was simply considered to be incorrect. To provide transparent results, we only used open-source libraries in our evaluation. Given that some of these tools at hand do not allow accessing their confidence score without any major alteration of their code, we considered the output of the tools to be binary (i.e., either 1 or 0).

We integrated four NER tools so far: the Stanford Named Entity Recognizer[27] (Stanford) [14], the Illinois Named Entity Tagger[28] (Illinois) [58], the Ottawa Baseline Information Extraction[29] (Balie) [38] and the Apache

---

[27] http://nlp.stanford.edu/software/CRF-NER.shtml (version 3.2.0)
[28] http://cogcomp.cs.illinois.edu/page/software_view/NETagger (version 2.4.0)
OpenNLP Name Finder\textsuperscript{30} (OpenNLP) \cite{4}. We only considered the performance of these tools on the classes \textbf{Location}, \textbf{Organization} and \textbf{Person}. To this end, we mapped the entity types of each of the NER tools to these three classes. We utilized the Waikato Environment for Knowledge Analysis (Weka) \cite{21} and the implemented classifiers with default parameters: AdaBoostM1 (ABM1) \cite{16} and Bagging (BG) \cite{6} with J48 \cite{57} as base classifier, Decision Table (DT) \cite{31}, Functional Trees (FT) \cite{18, 35}, J48 \cite{57}, Logistic Model Trees (LMT) \cite{35, 62}, Logistic Regression (Log) \cite{36}, Additive Logistic Regression (LogB) \cite{17}, Multilayer Perceptron (MLP), Naïve Bayes (NB) \cite{28}, Random Forest (RF) \cite{7}, Support Vector Machine (SVM) \cite{9} and Sequential Minimal Optimization (SMO) \cite{22}. In addition, we used voting at class level (CVote) and a simple voting (Vote) approach \cite{64} with equal weights for all NER tools. CVote selects the NER tool with the highest prediction performance for each type according to the evaluation and uses that particular tool for the given class. Vote as naive approach combines the results of the NER tools with the Majority Vote Rule \cite{30} and was the baseline ensemble learning technique in our evaluation.

\subsection{4.4.2.2 Experimental Setup}

We used five datasets and five measures for our evaluation. We used the recommended Wilcoxon signed-rank test to measure the statistical significance of our results \cite{11}. For this purpose, we applied each measurement of the ten 10-fold cross-validation runs for the underlying distribution and we set up a 95\% confidence interval.

\subsection{4.4.2.3 Datasets}

An overview of the datasets is shown in Table 8. The \textbf{Web} dataset consists of 20 annotated Web sites as described in \cite{58} and contains the most noise compared to the other datasets. The dataset \textbf{Reuters} consists of 50 documents randomly chosen out of the Reuters-21578 corpus\textsuperscript{31} \cite{5}. \textbf{News} is a small subset of the dataset \textbf{News} that consists of text from newspaper articles and was re-annotated manually by the authors to ensure high data quality. Likewise, \textbf{Reuters} was extracted and annotated manually by the authors. The last dataset, \textbf{All}, consists of the datasets mentioned before merged into one and allows for measuring how well the ensemble learning approaches perform when presented with data from heterogenous sources.

\begin{table}[h]
\centering
\caption{Number of entities separated according entity types and in total.}
\begin{tabular}{l|cccc}
\hline
Class & News & News$^*$ & Web & Reuters & All \\
\hline
Location & 5117 & 341 & 114 & 146 & 5472 \\
Organization & 6899 & 434 & 257 & 208 & 7467 \\
Person & 3899 & 254 & 396 & 91 & 4549 \\
Total & 15915 & 1029 & 767 & 445 & 17488 \\
\hline
\end{tabular}
\end{table}

\textsuperscript{30}http://opennlp.apache.org/index.html (version 1.5.3)

\textsuperscript{31}The Reuters-21578 corpus is available at: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html.
4.4.2.4 Measures

To assess the performance of the different algorithms, we computed the following values on the test datasets: The number of true positives $TP_t$, the number of true negatives $TN_t$, the number of false positives $FP_t$ and the number of false negatives $FN_t$. These numbers were collected for each entity type $t$ and averaged over the ten runs of the 10-fold cross-validations. Then, we applied the one-against-all approach to convert the multi-class confusion matrix of each dataset into a binary confusion matrix.

Subsequently, we determined with macro-averaging the classical measures recall ($rec$), precision ($pre$) and f-score ($F_1$) as follows:

$$rec = \frac{\sum_{t \in T} TP_t}{|T|}, \quad pre = \frac{\sum_{t \in T} TP_t}{|T|}, \quad F_1 = \frac{2pre \cdot rec}{pre + rec}.$$  \hspace{1cm} (18)

For the sake of completeness, we averaged the error rate ($error$) (Equation 19) and the Matthews correlation coefficient ($MCC$) \cite{37} (Equation 20) similarly.

$$error = \frac{\sum_{t \in T} FP_t + FN_t}{|T|}.$$  \hspace{1cm} (19)

$$MCC = \frac{\sum_{t \in T} TP_t TN_t - FP_t FN_t}{\sqrt{(TP_t + FP_t)(TP_t + FN_t)(TN_t + FP_t)(TN_t + FN_t)}}.$$  \hspace{1cm} (20)

The error rate monitors the fraction of positive and negative classifications for that the classifier failed. The Matthews correlation coefficient considers both the true positives and the true negatives as successful classification and is rather unaffected by sampling biases. Higher values indicating better classifications.

4.4.2.5 Results

Table 9–Table 18 show the results of our evaluation for the 15 classifiers we used within our pipeline and the four NER tools we integrated so far. The best results are marked bold and the NER tools are underlined. Figure 27–Figure 29 depict the f-scores separated according classes of the four NER tools, the simple voting approach Vote and the best classifier for the depicted dataset.

We reached the highest f-scores on the News$^*$ dataset (Table 9 and Table 10) for both the token-based and the entity-based evaluation. In the token-based evaluation, the MLP and RF classifiers perform best for precision (95.28%), error rate (0.32%) and Matthews correlation coefficient (0.951). MLP performs best for f-score (95.23%) with 0.04% more recall than RF. The baseline classifier (i.e., simple voting) is clearly outperformed by MLP by up to +5.21% recall, +12.31% precision, +9.31% f-score, -0.62% error rate and +0.094 MCC. Furthermore, the best single approach is Stanford and outperformed by up to +2.83% recall, +4.27% precision, +3.55% f-score, -0.21% error rate (that is a reduction by 40%) and +0.037 MCC. Slightly poorer results are achieved in the entity-based evaluation, where MLP is second to FT with 0.01% less f-score.

On the News dataset (Table 11-Table 12), which was the largest homogenous dataset in our evaluation, we repeatedly achieved high f-scores. The best approach w.r.t. the token-based evaluation is LMT with an f-score of 92.94%. Random Forest follows the best approach with respect to f-score again. Moreover, the best single tool Stanford and the baseline classifier Vote are repeatedly outperformed by up to +2.6% resp. +19.91%
Table 9: **News**\(^*\) token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F(_1)</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>95.19</td>
<td>95.28</td>
<td>95.23</td>
<td>0.32</td>
<td>0.951</td>
</tr>
<tr>
<td>RF</td>
<td>95.15</td>
<td>95.28</td>
<td>95.21</td>
<td>0.32</td>
<td>0.951</td>
</tr>
<tr>
<td>ABM1</td>
<td>94.82</td>
<td>95.18</td>
<td>95.00</td>
<td>0.33</td>
<td>0.948</td>
</tr>
<tr>
<td>SVM</td>
<td>94.86</td>
<td>95.09</td>
<td>94.97</td>
<td>0.33</td>
<td>0.948</td>
</tr>
<tr>
<td>J48</td>
<td>94.78</td>
<td>94.98</td>
<td>94.88</td>
<td>0.34</td>
<td>0.947</td>
</tr>
<tr>
<td>BG</td>
<td>94.76</td>
<td>94.93</td>
<td>94.84</td>
<td>0.34</td>
<td>0.947</td>
</tr>
<tr>
<td>LMT</td>
<td>94.68</td>
<td>94.95</td>
<td>94.82</td>
<td>0.34</td>
<td>0.946</td>
</tr>
<tr>
<td>DT</td>
<td>94.63</td>
<td>94.95</td>
<td>94.79</td>
<td>0.34</td>
<td>0.946</td>
</tr>
<tr>
<td>FT</td>
<td>94.30</td>
<td>95.15</td>
<td>94.72</td>
<td>0.35</td>
<td>0.945</td>
</tr>
<tr>
<td>LogB</td>
<td>93.54</td>
<td>95.37</td>
<td>94.44</td>
<td>0.37</td>
<td>0.943</td>
</tr>
<tr>
<td>Log</td>
<td>94.05</td>
<td>94.75</td>
<td>94.40</td>
<td>0.37</td>
<td>0.942</td>
</tr>
<tr>
<td>SMO</td>
<td>94.01</td>
<td>94.37</td>
<td>94.19</td>
<td>0.39</td>
<td>0.940</td>
</tr>
<tr>
<td>NB</td>
<td>94.61</td>
<td>92.64</td>
<td>93.60</td>
<td>0.42</td>
<td>0.934</td>
</tr>
<tr>
<td>Stanford</td>
<td>92.36</td>
<td>91.01</td>
<td>91.68</td>
<td>0.53</td>
<td>0.914</td>
</tr>
<tr>
<td>CVote</td>
<td>92.02</td>
<td>90.84</td>
<td>91.42</td>
<td>0.54</td>
<td>0.911</td>
</tr>
<tr>
<td>Vote</td>
<td>89.98</td>
<td>82.97</td>
<td>85.92</td>
<td>0.94</td>
<td>0.857</td>
</tr>
<tr>
<td>Illinois</td>
<td>82.79</td>
<td>87.35</td>
<td>84.95</td>
<td>0.92</td>
<td>0.845</td>
</tr>
<tr>
<td>Balie</td>
<td>77.68</td>
<td>82.05</td>
<td>79.80</td>
<td>1.21</td>
<td>0.792</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>71.42</td>
<td>90.47</td>
<td>79.57</td>
<td>1.13</td>
<td>0.797</td>
</tr>
</tbody>
</table>

Table 10: **News**\(^*\) entity-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F(_1)</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>93.95</td>
<td>92.27</td>
<td>93.10</td>
<td>0.30</td>
<td>0.930</td>
</tr>
<tr>
<td>MLP</td>
<td>94.10</td>
<td>93.09</td>
<td>93.00</td>
<td>0.30</td>
<td>0.929</td>
</tr>
<tr>
<td>LMT</td>
<td>94.08</td>
<td>92.07</td>
<td>92.97</td>
<td>0.31</td>
<td>0.928</td>
</tr>
<tr>
<td>RF</td>
<td>93.76</td>
<td>92.18</td>
<td>92.83</td>
<td>0.31</td>
<td>0.927</td>
</tr>
<tr>
<td>ABM1</td>
<td>93.30</td>
<td>91.65</td>
<td>92.47</td>
<td>0.33</td>
<td>0.923</td>
</tr>
<tr>
<td>J48</td>
<td>93.30</td>
<td>91.65</td>
<td>92.47</td>
<td>0.33</td>
<td>0.923</td>
</tr>
<tr>
<td>Log</td>
<td>93.42</td>
<td>91.39</td>
<td>92.37</td>
<td>0.33</td>
<td>0.922</td>
</tr>
<tr>
<td>LogB</td>
<td>92.89</td>
<td>91.68</td>
<td>92.27</td>
<td>0.33</td>
<td>0.921</td>
</tr>
<tr>
<td>SMO</td>
<td>92.55</td>
<td>91.26</td>
<td>91.90</td>
<td>0.36</td>
<td>0.917</td>
</tr>
<tr>
<td>DT</td>
<td>92.44</td>
<td>91.29</td>
<td>91.86</td>
<td>0.34</td>
<td>0.917</td>
</tr>
<tr>
<td>NB</td>
<td>94.08</td>
<td>88.26</td>
<td>91.01</td>
<td>0.40</td>
<td>0.909</td>
</tr>
<tr>
<td>Stanford</td>
<td>92.00</td>
<td>87.58</td>
<td>89.72</td>
<td>0.45</td>
<td>0.895</td>
</tr>
<tr>
<td>CVote</td>
<td>91.43</td>
<td>86.94</td>
<td>89.10</td>
<td>0.47</td>
<td>0.889</td>
</tr>
<tr>
<td>Vote</td>
<td>82.07</td>
<td>68.48</td>
<td>83.34</td>
<td>0.67</td>
<td>0.831</td>
</tr>
<tr>
<td>Illinois</td>
<td>91.42</td>
<td>76.52</td>
<td>82.67</td>
<td>0.83</td>
<td>0.829</td>
</tr>
<tr>
<td>Balie</td>
<td>81.54</td>
<td>79.66</td>
<td>80.48</td>
<td>0.79</td>
<td>0.801</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>69.36</td>
<td>85.02</td>
<td>75.78</td>
<td>0.88</td>
<td>0.760</td>
</tr>
</tbody>
</table>
### Table 11: News token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>$F_1$</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMT</td>
<td>93.73</td>
<td>92.16</td>
<td>92.94</td>
<td>0.51</td>
<td>0.927</td>
</tr>
<tr>
<td>RF</td>
<td>93.56</td>
<td>92.19</td>
<td>92.87</td>
<td>0.51</td>
<td>0.926</td>
</tr>
<tr>
<td>DT</td>
<td>93.64</td>
<td>92.10</td>
<td>92.86</td>
<td>0.51</td>
<td>0.926</td>
</tr>
<tr>
<td>J48</td>
<td>93.50</td>
<td>92.20</td>
<td>92.84</td>
<td>0.52</td>
<td>0.926</td>
</tr>
<tr>
<td>ABM1</td>
<td>93.49</td>
<td>92.17</td>
<td>92.83</td>
<td>0.52</td>
<td>0.926</td>
</tr>
<tr>
<td>BG</td>
<td>93.11</td>
<td>92.49</td>
<td>92.79</td>
<td>0.52</td>
<td>0.925</td>
</tr>
<tr>
<td>FT</td>
<td>93.44</td>
<td>92.15</td>
<td>92.79</td>
<td>0.52</td>
<td>0.925</td>
</tr>
<tr>
<td>MLP</td>
<td>93.22</td>
<td>92.26</td>
<td>92.73</td>
<td>0.52</td>
<td>0.925</td>
</tr>
<tr>
<td>SVM</td>
<td>92.19</td>
<td>92.49</td>
<td>92.31</td>
<td>0.54</td>
<td>0.920</td>
</tr>
<tr>
<td>SMO</td>
<td>92.15</td>
<td>91.90</td>
<td>92.01</td>
<td>0.57</td>
<td>0.917</td>
</tr>
<tr>
<td>Log</td>
<td>91.38</td>
<td>91.36</td>
<td>91.35</td>
<td>0.63</td>
<td>0.910</td>
</tr>
<tr>
<td>LogB</td>
<td>91.42</td>
<td>91.32</td>
<td>91.34</td>
<td>0.62</td>
<td>0.910</td>
</tr>
<tr>
<td>Stanford</td>
<td>92.70</td>
<td>88.09</td>
<td>90.34</td>
<td>0.68</td>
<td>0.900</td>
</tr>
<tr>
<td>CVote</td>
<td>92.70</td>
<td>88.09</td>
<td>90.34</td>
<td>0.68</td>
<td>0.900</td>
</tr>
<tr>
<td>NB</td>
<td>93.36</td>
<td>86.17</td>
<td>89.58</td>
<td>0.77</td>
<td>0.893</td>
</tr>
<tr>
<td>Illinois</td>
<td>82.43</td>
<td>78.11</td>
<td>80.20</td>
<td>1.37</td>
<td>0.795</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>75.21</td>
<td>74.41</td>
<td>73.71</td>
<td>2.06</td>
<td>0.732</td>
</tr>
<tr>
<td>Vote</td>
<td>83.13</td>
<td>69.14</td>
<td>73.03</td>
<td>2.36</td>
<td>0.735</td>
</tr>
<tr>
<td>Balie</td>
<td>70.81</td>
<td>72.86</td>
<td>71.54</td>
<td>1.90</td>
<td>0.707</td>
</tr>
</tbody>
</table>

### Table 12: News entity-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>$F_1$</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMT</td>
<td>92.95</td>
<td>88.84</td>
<td>90.84</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>BG</td>
<td>92.82</td>
<td>88.95</td>
<td>90.83</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>DT</td>
<td>92.89</td>
<td>88.88</td>
<td>90.83</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>ABM1</td>
<td>92.87</td>
<td>88.82</td>
<td>90.79</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>J48</td>
<td>92.87</td>
<td>88.82</td>
<td>90.79</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>FT</td>
<td>92.90</td>
<td>88.78</td>
<td>90.78</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>RF</td>
<td>92.84</td>
<td>88.77</td>
<td>90.74</td>
<td>0.44</td>
<td>0.906</td>
</tr>
<tr>
<td>MLP</td>
<td>92.83</td>
<td>88.69</td>
<td>90.70</td>
<td>0.44</td>
<td>0.905</td>
</tr>
<tr>
<td>SVM</td>
<td>91.56</td>
<td>89.22</td>
<td>90.33</td>
<td>0.45</td>
<td>0.901</td>
</tr>
<tr>
<td>SMO</td>
<td>91.13</td>
<td>88.36</td>
<td>89.69</td>
<td>0.49</td>
<td>0.895</td>
</tr>
<tr>
<td>Log</td>
<td>90.62</td>
<td>88.09</td>
<td>89.29</td>
<td>0.51</td>
<td>0.891</td>
</tr>
<tr>
<td>LogB</td>
<td>90.76</td>
<td>87.83</td>
<td>89.22</td>
<td>0.51</td>
<td>0.890</td>
</tr>
<tr>
<td>Stanford</td>
<td>91.78</td>
<td>83.92</td>
<td>87.66</td>
<td>0.58</td>
<td>0.875</td>
</tr>
<tr>
<td>CVote</td>
<td>91.78</td>
<td>83.92</td>
<td>87.66</td>
<td>0.58</td>
<td>0.875</td>
</tr>
<tr>
<td>NB</td>
<td>92.54</td>
<td>81.16</td>
<td>86.34</td>
<td>0.69</td>
<td>0.863</td>
</tr>
<tr>
<td>Illinois</td>
<td>81.66</td>
<td>72.50</td>
<td>76.71</td>
<td>1.11</td>
<td>0.763</td>
</tr>
<tr>
<td>Balie</td>
<td>71.58</td>
<td>68.67</td>
<td>69.66</td>
<td>1.42</td>
<td>0.692</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>72.71</td>
<td>67.29</td>
<td>67.89</td>
<td>1.80</td>
<td>0.681</td>
</tr>
<tr>
<td>Vote</td>
<td>82.71</td>
<td>61.30</td>
<td>67.10</td>
<td>2.19</td>
<td>0.686</td>
</tr>
</tbody>
</table>
Table 13: Web token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F₁</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM1</td>
<td>64.40</td>
<td>74.83</td>
<td>69.04</td>
<td>3.38</td>
<td>0.675</td>
</tr>
<tr>
<td>RF</td>
<td>64.36</td>
<td>74.57</td>
<td>68.93</td>
<td>3.38</td>
<td>0.674</td>
</tr>
<tr>
<td>MLP</td>
<td>63.86</td>
<td>75.11</td>
<td>68.81</td>
<td>3.33</td>
<td>0.674</td>
</tr>
<tr>
<td>FT</td>
<td>62.98</td>
<td>75.47</td>
<td>68.25</td>
<td>3.33</td>
<td>0.670</td>
</tr>
<tr>
<td>LMT</td>
<td>63.39</td>
<td>74.24</td>
<td>68.04</td>
<td>3.43</td>
<td>0.666</td>
</tr>
<tr>
<td>DT</td>
<td>62.80</td>
<td>74.18</td>
<td>67.85</td>
<td>3.43</td>
<td>0.664</td>
</tr>
<tr>
<td>SVM</td>
<td>62.94</td>
<td>73.45</td>
<td>67.60</td>
<td>3.49</td>
<td>0.662</td>
</tr>
<tr>
<td>LogB</td>
<td>60.47</td>
<td>77.48</td>
<td>67.57</td>
<td>3.40</td>
<td>0.665</td>
</tr>
<tr>
<td>Log</td>
<td>60.31</td>
<td>77.89</td>
<td>67.50</td>
<td>3.39</td>
<td>0.666</td>
</tr>
<tr>
<td>SMO</td>
<td>63.47</td>
<td>72.45</td>
<td>67.49</td>
<td>3.57</td>
<td>0.659</td>
</tr>
<tr>
<td>BG</td>
<td>61.06</td>
<td>76.19</td>
<td>67.46</td>
<td>3.34</td>
<td>0.663</td>
</tr>
<tr>
<td>J48</td>
<td>62.21</td>
<td>73.73</td>
<td>67.21</td>
<td>3.49</td>
<td>0.658</td>
</tr>
<tr>
<td>NB</td>
<td>71.19</td>
<td>63.42</td>
<td>66.88</td>
<td>4.42</td>
<td>0.647</td>
</tr>
<tr>
<td>Stanford</td>
<td>60.57</td>
<td>72.19</td>
<td>65.81</td>
<td>3.51</td>
<td>0.643</td>
</tr>
<tr>
<td>Illinois</td>
<td>69.64</td>
<td>60.56</td>
<td>64.44</td>
<td>5.09</td>
<td>0.621</td>
</tr>
<tr>
<td>Vote</td>
<td>66.90</td>
<td>54.75</td>
<td>58.59</td>
<td>6.02</td>
<td>0.567</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>45.71</td>
<td>58.81</td>
<td>49.18</td>
<td>5.93</td>
<td>0.477</td>
</tr>
<tr>
<td>Balie</td>
<td>38.63</td>
<td>43.83</td>
<td>40.15</td>
<td>7.02</td>
<td>0.371</td>
</tr>
</tbody>
</table>

Table 14: Web entity-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F₁</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM1</td>
<td>64.95</td>
<td>61.86</td>
<td>63.36</td>
<td>1.99</td>
<td>0.624</td>
</tr>
<tr>
<td>Stanford</td>
<td>64.80</td>
<td>61.31</td>
<td>62.83</td>
<td>1.95</td>
<td>0.619</td>
</tr>
<tr>
<td>LogB</td>
<td>61.25</td>
<td>64.10</td>
<td>62.60</td>
<td>1.94</td>
<td>0.616</td>
</tr>
<tr>
<td>FT</td>
<td>63.67</td>
<td>61.10</td>
<td>62.21</td>
<td>2.09</td>
<td>0.612</td>
</tr>
<tr>
<td>SVM</td>
<td>62.36</td>
<td>57.26</td>
<td>59.57</td>
<td>2.15</td>
<td>0.586</td>
</tr>
<tr>
<td>J48</td>
<td>63.21</td>
<td>59.72</td>
<td>61.39</td>
<td>2.12</td>
<td>0.603</td>
</tr>
<tr>
<td>BG</td>
<td>64.04</td>
<td>59.10</td>
<td>61.30</td>
<td>2.13</td>
<td>0.603</td>
</tr>
<tr>
<td>Log</td>
<td>64.15</td>
<td>55.88</td>
<td>59.69</td>
<td>2.27</td>
<td>0.587</td>
</tr>
<tr>
<td>SMO</td>
<td>62.36</td>
<td>57.26</td>
<td>59.57</td>
<td>2.15</td>
<td>0.586</td>
</tr>
<tr>
<td>DT</td>
<td>61.92</td>
<td>57.05</td>
<td>59.34</td>
<td>2.17</td>
<td>0.583</td>
</tr>
<tr>
<td>LMT</td>
<td>61.25</td>
<td>56.89</td>
<td>58.96</td>
<td>2.19</td>
<td>0.579</td>
</tr>
<tr>
<td>NB</td>
<td>74.18</td>
<td>49.20</td>
<td>58.55</td>
<td>3.17</td>
<td>0.586</td>
</tr>
<tr>
<td>Illinois</td>
<td>69.31</td>
<td>45.85</td>
<td>54.25</td>
<td>3.82</td>
<td>0.541</td>
</tr>
<tr>
<td>Vote</td>
<td>67.42</td>
<td>37.77</td>
<td>47.12</td>
<td>4.84</td>
<td>0.477</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>46.94</td>
<td>46.78</td>
<td>43.99</td>
<td>3.71</td>
<td>0.437</td>
</tr>
<tr>
<td>Balie</td>
<td>38.07</td>
<td>32.92</td>
<td>35.07</td>
<td>3.63</td>
<td>0.334</td>
</tr>
</tbody>
</table>

f-score. Once again, the entity-based results are approximately 2% poorer, with LMT leading the table like in the token-based evaluation.

On the Web dataset (Table 13-Table 14), which is the worst-case dataset for NER tools as it contains several incomplete sentences, the different classifiers reached their lowest values. For the token-based evaluation, AdaBoostM1 with J48 achieves the best f-score (69.04%) and Matthews correlation coefficient (0.675) and is followed by Random Forest again with respect to f-score. Naïve Bayes performs best for recall (96.64%), Logistic Regression for precision (77.89%) and MLP and RF for the error rate (3.33%). Simple voting is outperformed by ABM1 by up to +3.5% recall, +20.08% precision, +10.45% f-score, -2.64% error rate and +0.108 MCC, while Stanford (the best tool for this dataset) is outperformed by up to +3.83% recall, +2.64% precision, +3.21% f-score, -0.13% error rate and +0.032 MCC. Similar insights can be won from the entity-based evaluation, with some classifiers like RF being approximately 10% poorer that at token level.

On the Reuters dataset (Table 15-Table 16), which was the smallest dataset in our evaluation, Support Vector Machine performs best. In the token-based evaluation, SVM achieves an f-score of 87.78%, an error rate of 0.89% and a Matthews correlation coefficient of 0.875%. They are followed by Random Forest with respect to f-score once again. Naïve Bayes performs best for recall (86.54%). In comparison, ensemble learning
### Table 15: Reuters token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>$F_1$</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>84.57</td>
<td>91.75</td>
<td>87.78</td>
<td>0.89</td>
<td>0.875</td>
</tr>
<tr>
<td>RF</td>
<td>86.11</td>
<td>89.24</td>
<td>87.58</td>
<td>0.90</td>
<td>0.872</td>
</tr>
<tr>
<td>MLP</td>
<td>85.89</td>
<td>89.46</td>
<td>87.55</td>
<td>0.90</td>
<td>0.871</td>
</tr>
<tr>
<td>LMT</td>
<td>84.41</td>
<td>91.08</td>
<td>87.43</td>
<td>0.89</td>
<td>0.871</td>
</tr>
<tr>
<td>J48</td>
<td>84.64</td>
<td>90.70</td>
<td>87.33</td>
<td>0.93</td>
<td>0.870</td>
</tr>
<tr>
<td>Log</td>
<td>84.33</td>
<td>90.85</td>
<td>87.27</td>
<td>0.89</td>
<td>0.870</td>
</tr>
<tr>
<td>LogB</td>
<td>84.22</td>
<td>91.01</td>
<td>87.22</td>
<td>0.90</td>
<td>0.870</td>
</tr>
<tr>
<td>ABM1</td>
<td>84.51</td>
<td>90.47</td>
<td>87.15</td>
<td>0.93</td>
<td>0.868</td>
</tr>
<tr>
<td>BG</td>
<td>84.70</td>
<td>90.16</td>
<td>87.14</td>
<td>0.94</td>
<td>0.868</td>
</tr>
<tr>
<td>FT</td>
<td>85.25</td>
<td>88.75</td>
<td>86.87</td>
<td>0.95</td>
<td>0.864</td>
</tr>
<tr>
<td>DT</td>
<td>84.41</td>
<td>89.00</td>
<td>86.43</td>
<td>0.99</td>
<td>0.861</td>
</tr>
<tr>
<td>SMO</td>
<td>84.45</td>
<td>88.49</td>
<td>86.28</td>
<td>0.98</td>
<td>0.859</td>
</tr>
<tr>
<td>Illinois</td>
<td>83.74</td>
<td>88.27</td>
<td>85.35</td>
<td>1.09</td>
<td>0.851</td>
</tr>
<tr>
<td>NB</td>
<td>86.54</td>
<td>83.18</td>
<td>84.77</td>
<td>1.10</td>
<td>0.842</td>
</tr>
<tr>
<td>CVote</td>
<td>81.96</td>
<td>88.66</td>
<td>84.64</td>
<td>1.14</td>
<td>0.844</td>
</tr>
<tr>
<td>Stanford</td>
<td>81.57</td>
<td>84.85</td>
<td>82.85</td>
<td>1.20</td>
<td>0.824</td>
</tr>
<tr>
<td>Vote</td>
<td>80.11</td>
<td>81.15</td>
<td>79.41</td>
<td>1.43</td>
<td>0.793</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>67.94</td>
<td>82.08</td>
<td>73.96</td>
<td>1.76</td>
<td>0.736</td>
</tr>
<tr>
<td>Balie</td>
<td>64.92</td>
<td>68.61</td>
<td>64.78</td>
<td>2.62</td>
<td>0.645</td>
</tr>
</tbody>
</table>

### Table 16: Reuters entity-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>$F_1$</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>81.37</td>
<td>88.85</td>
<td>84.71</td>
<td>0.69</td>
<td>0.846</td>
</tr>
<tr>
<td>ABM1</td>
<td>80.60</td>
<td>88.72</td>
<td>84.15</td>
<td>0.73</td>
<td>0.840</td>
</tr>
<tr>
<td>LMT</td>
<td>80.80</td>
<td>87.92</td>
<td>83.96</td>
<td>0.73</td>
<td>0.838</td>
</tr>
<tr>
<td>J48</td>
<td>80.41</td>
<td>88.50</td>
<td>83.95</td>
<td>0.73</td>
<td>0.838</td>
</tr>
<tr>
<td>BG</td>
<td>80.55</td>
<td>87.70</td>
<td>83.75</td>
<td>0.75</td>
<td>0.836</td>
</tr>
<tr>
<td>Illinois</td>
<td>82.77</td>
<td>85.73</td>
<td>83.74</td>
<td>0.72</td>
<td>0.836</td>
</tr>
<tr>
<td>LogB</td>
<td>80.70</td>
<td>86.23</td>
<td>83.32</td>
<td>0.75</td>
<td>0.830</td>
</tr>
<tr>
<td>DT</td>
<td>81.11</td>
<td>85.20</td>
<td>82.95</td>
<td>0.79</td>
<td>0.827</td>
</tr>
<tr>
<td>RF</td>
<td>80.08</td>
<td>86.11</td>
<td>82.86</td>
<td>0.78</td>
<td>0.826</td>
</tr>
<tr>
<td>Log</td>
<td>80.01</td>
<td>85.51</td>
<td>82.62</td>
<td>0.78</td>
<td>0.823</td>
</tr>
<tr>
<td>Illinois</td>
<td>82.77</td>
<td>85.73</td>
<td>83.74</td>
<td>0.72</td>
<td>0.836</td>
</tr>
<tr>
<td>Log</td>
<td>80.01</td>
<td>85.51</td>
<td>82.62</td>
<td>0.78</td>
<td>0.823</td>
</tr>
<tr>
<td>SMO</td>
<td>80.27</td>
<td>84.09</td>
<td>81.98</td>
<td>0.83</td>
<td>0.817</td>
</tr>
<tr>
<td>MLP</td>
<td>80.27</td>
<td>84.09</td>
<td>81.98</td>
<td>0.83</td>
<td>0.817</td>
</tr>
<tr>
<td>SMO</td>
<td>79.62</td>
<td>83.21</td>
<td>81.36</td>
<td>0.88</td>
<td>0.809</td>
</tr>
<tr>
<td>FT</td>
<td>80.00</td>
<td>82.71</td>
<td>81.32</td>
<td>0.85</td>
<td>0.809</td>
</tr>
<tr>
<td>CVote</td>
<td>77.86</td>
<td>85.42</td>
<td>81.00</td>
<td>0.85</td>
<td>0.809</td>
</tr>
<tr>
<td>NB</td>
<td>83.80</td>
<td>77.68</td>
<td>80.61</td>
<td>0.92</td>
<td>0.802</td>
</tr>
<tr>
<td>Stanford</td>
<td>77.56</td>
<td>82.38</td>
<td>79.68</td>
<td>0.90</td>
<td>0.794</td>
</tr>
<tr>
<td>Vote</td>
<td>80.35</td>
<td>76.25</td>
<td>77.37</td>
<td>1.03</td>
<td>0.773</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>66.85</td>
<td>80.33</td>
<td>72.89</td>
<td>1.18</td>
<td>0.726</td>
</tr>
<tr>
<td>Balie</td>
<td>68.90</td>
<td>70.14</td>
<td>68.71</td>
<td>1.39</td>
<td>0.684</td>
</tr>
</tbody>
</table>
Table 17: **A11** token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F&lt;sub&gt;1&lt;/sub&gt;</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>91.58</td>
<td>90.97</td>
<td>91.27</td>
<td>0.64</td>
<td>0.909</td>
</tr>
<tr>
<td>LMT</td>
<td>91.67</td>
<td>90.86</td>
<td>91.26</td>
<td>0.64</td>
<td>0.909</td>
</tr>
<tr>
<td>ABM1</td>
<td>91.49</td>
<td>90.99</td>
<td>91.24</td>
<td>0.64</td>
<td>0.909</td>
</tr>
<tr>
<td>J48</td>
<td>91.46</td>
<td>90.98</td>
<td>91.22</td>
<td>0.64</td>
<td>0.909</td>
</tr>
<tr>
<td>DT</td>
<td>91.59</td>
<td>90.84</td>
<td>91.21</td>
<td>0.64</td>
<td>0.909</td>
</tr>
<tr>
<td>FT</td>
<td>91.49</td>
<td>90.82</td>
<td>91.16</td>
<td>0.65</td>
<td>0.908</td>
</tr>
<tr>
<td>BG</td>
<td>91.25</td>
<td>91.00</td>
<td>91.12</td>
<td>0.65</td>
<td>0.908</td>
</tr>
<tr>
<td>MLP</td>
<td>90.94</td>
<td>91.05</td>
<td>90.99</td>
<td>0.66</td>
<td>0.907</td>
</tr>
<tr>
<td>SVM</td>
<td>90.15</td>
<td>91.24</td>
<td>90.67</td>
<td>0.67</td>
<td>0.903</td>
</tr>
<tr>
<td>SMO</td>
<td>90.13</td>
<td>90.48</td>
<td>90.27</td>
<td>0.71</td>
<td>0.899</td>
</tr>
<tr>
<td>Log</td>
<td>88.69</td>
<td>90.57</td>
<td>89.59</td>
<td>0.76</td>
<td>0.892</td>
</tr>
<tr>
<td>LogB</td>
<td>88.92</td>
<td>90.21</td>
<td>89.53</td>
<td>0.76</td>
<td>0.892</td>
</tr>
<tr>
<td>Stanford</td>
<td>90.75</td>
<td>87.73</td>
<td>89.21</td>
<td>0.78</td>
<td>0.888</td>
</tr>
<tr>
<td>CVote</td>
<td>90.75</td>
<td>87.73</td>
<td>89.21</td>
<td>0.78</td>
<td>0.888</td>
</tr>
<tr>
<td>NB</td>
<td><strong>92.00</strong></td>
<td>85.27</td>
<td>88.46</td>
<td>0.89</td>
<td>0.881</td>
</tr>
<tr>
<td>Illinois</td>
<td>81.66</td>
<td>77.61</td>
<td>79.54</td>
<td>1.48</td>
<td>0.788</td>
</tr>
<tr>
<td>Vote</td>
<td>81.85</td>
<td>69.96</td>
<td>72.90</td>
<td>2.44</td>
<td>0.733</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>72.63</td>
<td>75.60</td>
<td>72.65</td>
<td>2.19</td>
<td>0.723</td>
</tr>
<tr>
<td>Balie</td>
<td>67.75</td>
<td>71.65</td>
<td>69.40</td>
<td>2.09</td>
<td>0.685</td>
</tr>
</tbody>
</table>

On the **A11** dataset for token-based evaluation (Table 17), the Random Forest approach performs best for f-score (91.27%), error rate (0.64%) and Matthews correlation coefficient (0.909). Support Vector Machine achieves the best precision (91.24%) and Naïve Bayes the best recall (91.00%) again. In comparison, ensemble learning outperformed Vote with RF by up to +9.71% recall, +21.01% precision, +18.37% f-score, -1.8% error rate and +0.176% MCC and Stanford, the best tool for this dataset, by up to +0.83% recall, +3.24% precision, +2.06% f-score, -0.14% error rate and +0.021% MCC. Again, entity-based evaluation (Table 18) compared to token-based evaluation, the f-score of J48, the best ensemble learning approach here, is approximately 1% poorer with higher recall but lower precision. In Figure 29, we clearly see a learning effect for RF and J48 at class level.

Table 18: **A11** entity-based.

<table>
<thead>
<tr>
<th>S</th>
<th>rec</th>
<th>pre</th>
<th>F&lt;sub&gt;1&lt;/sub&gt;</th>
<th>error</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48</td>
<td>92.68</td>
<td>88.62</td>
<td><strong>90.59</strong></td>
<td>0.44</td>
<td>0.904</td>
</tr>
<tr>
<td>ABM1</td>
<td>92.66</td>
<td>88.59</td>
<td>90.56</td>
<td>0.44</td>
<td>0.904</td>
</tr>
<tr>
<td>LMT</td>
<td>92.59</td>
<td>88.50</td>
<td>90.48</td>
<td>0.45</td>
<td>0.903</td>
</tr>
<tr>
<td>DT</td>
<td>92.56</td>
<td>88.44</td>
<td>90.44</td>
<td>0.45</td>
<td>0.902</td>
</tr>
<tr>
<td>FT</td>
<td>92.47</td>
<td>88.37</td>
<td>90.35</td>
<td>0.45</td>
<td>0.902</td>
</tr>
<tr>
<td>BG</td>
<td>92.17</td>
<td>88.55</td>
<td>90.31</td>
<td>0.45</td>
<td>0.901</td>
</tr>
<tr>
<td>MLP</td>
<td>92.07</td>
<td>88.60</td>
<td>90.28</td>
<td>0.45</td>
<td>0.901</td>
</tr>
<tr>
<td>SVM</td>
<td>90.91</td>
<td><strong>88.97</strong></td>
<td>89.88</td>
<td>0.46</td>
<td>0.897</td>
</tr>
<tr>
<td>SMO</td>
<td>90.94</td>
<td>87.31</td>
<td>89.00</td>
<td>0.52</td>
<td>0.888</td>
</tr>
<tr>
<td>Log</td>
<td>89.49</td>
<td>88.10</td>
<td>88.70</td>
<td>0.53</td>
<td>0.885</td>
</tr>
<tr>
<td>LogB</td>
<td>89.21</td>
<td>87.68</td>
<td>88.36</td>
<td>0.54</td>
<td>0.881</td>
</tr>
<tr>
<td>Stanford</td>
<td>92.00</td>
<td>84.48</td>
<td>88.05</td>
<td>0.56</td>
<td>0.879</td>
</tr>
<tr>
<td>CVote</td>
<td>92.00</td>
<td>84.48</td>
<td>88.05</td>
<td>0.56</td>
<td>0.879</td>
</tr>
<tr>
<td>NB</td>
<td><strong>92.69</strong></td>
<td>80.59</td>
<td>86.04</td>
<td>0.71</td>
<td>0.860</td>
</tr>
<tr>
<td>Illinois</td>
<td>81.43</td>
<td>71.82</td>
<td>76.25</td>
<td>1.12</td>
<td>0.759</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>71.29</td>
<td>69.44</td>
<td>67.66</td>
<td>1.80</td>
<td>0.682</td>
</tr>
</tbody>
</table>

outperforms Vote with SVM by up to +4.46% recall, +3.48% precision, +2.43% f-score, -0.54% error rate and +0.082 MCC. Moreover, the best NER tool for this dataset, Illinois, is outperformed by up to +0.83% recall, +3.48% precision, +2.43% f-score, -0.20% error rate and +0.024 MCC. In Figure 28(a), we barely see a learning effect as ABM1 is almost equal to one of the integrated NER tools assessed at class level especially for the class **Organization** on the Web dataset but in Figure 28(c) on the Reuters dataset we clearly see a learning effect for the class **Organization** and **Person** with the SVM approach.
Table 19: F-score of the best 3 classifiers on class level token-based.

<table>
<thead>
<tr>
<th>S</th>
<th>Class</th>
<th>News</th>
<th>News*</th>
<th>Web</th>
<th>Reuters</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>Location</td>
<td>92.12</td>
<td>94.96</td>
<td>54.58</td>
<td>82.25</td>
<td>89.98</td>
</tr>
<tr>
<td>RF</td>
<td>Organization</td>
<td>89.45</td>
<td>92.44</td>
<td>65.60</td>
<td>90.53</td>
<td>87.93</td>
</tr>
<tr>
<td>RF</td>
<td>Person</td>
<td>97.02</td>
<td>98.25</td>
<td>86.61</td>
<td>89.95</td>
<td>95.91</td>
</tr>
<tr>
<td>MLP</td>
<td>Location</td>
<td>91.79</td>
<td>95.22</td>
<td>53.78</td>
<td>82.13</td>
<td>89.62</td>
</tr>
<tr>
<td>MLP</td>
<td>Organization</td>
<td>89.34</td>
<td>92.45</td>
<td>65.72</td>
<td>90.38</td>
<td>87.63</td>
</tr>
<tr>
<td>MLP</td>
<td>Person</td>
<td>97.07</td>
<td>98.04</td>
<td>86.94</td>
<td>90.14</td>
<td>95.73</td>
</tr>
<tr>
<td>ABM1</td>
<td>Location</td>
<td>91.75</td>
<td>95.10</td>
<td>55.11</td>
<td>81.19</td>
<td>89.90</td>
</tr>
<tr>
<td>ABM1</td>
<td>Organization</td>
<td>89.49</td>
<td>92.00</td>
<td>65.47</td>
<td>89.91</td>
<td>87.96</td>
</tr>
<tr>
<td>ABM1</td>
<td>Person</td>
<td>97.12</td>
<td>97.89</td>
<td>86.53</td>
<td>90.37</td>
<td>95.87</td>
</tr>
</tbody>
</table>

Overall, ensemble learning outperforms all included NER tools and the simple voting approach for all datasets with respect to f-score, which answers our first and second question. Here, it is worth mentioning that Stanford and Illinois are the best tools in our framework. The three best classifiers with respect to the averaged f-scores over our datasets for token-based evaluation are the Random Forest classifier with the highest value, closely followed by Multilayer Perceptron and AdaBoostM1 with J48 and for entity-based evaluation AdaBoostM1 with J48 with the highest value, closely followed by MLP and J48. We cannot observe a significant difference between these.

In Table 19 and Table 20, we depict the f-scores of these three classifiers at class level for our datasets. The statistically significant differences are marked in bold. Note that two out of three scores being marked bold for the same setting in a column means that the corresponding approaches are significantly better than the third one yet not significantly better than each other. In the token-based evaluation, the Multilayer Perceptron and Random Forest classifier surpass the AdaBoostM1 with J48 on the News* and Web datasets. On the News* dataset, MLP surpasses RF for Location but RF surpasses MLP for Person. On the Web dataset, RF is better than MLP for Location but not significantly different from one another for Person. Also, for the Organization class, no significant difference could be determined on both datasets. On the Reuters dataset, MLP and RF are better than ABM1 for Location and Organization, but do not differ one another. For the class Person, no significant difference could be determined for all three classifiers. On the News and All dataset, Random Forest is significantly best for Location. Random Forest and AdaBoostM1 with J48 surpass the Multilayer Perceptron for Organization but are not significantly different. For the class Person, ABM1 is significantly best on the News dataset and RF is best on the All dataset. The entity-level results also suggest shifts amongst the best systems depending on the datasets. Interestingly, MLP and ABM1 are the only two classes of algorithm that appear as top algorithms in both evaluation schemes.

Consequently, our results suggest that while the four approaches RF, MLP, ABM1 and J48 perform best over the datasets at hand, MLP and ABM1 are to be favored. Note that significant differences can be observed across the different datasets and that all four paradigms RF, MLP, ABM1 and J48 should be considered when applying ensemble learning to NER. This answers the last and most important question of this evaluation. Within FOX 2.2.2, we opted for using MLP as they scale better.
### Table 20: F-score of the best 3 classifiers on class level entity-based.

<table>
<thead>
<tr>
<th></th>
<th>Class</th>
<th>News</th>
<th>News*</th>
<th>Web</th>
<th>Reuters</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM1</td>
<td>Location</td>
<td>91.26</td>
<td>95.71</td>
<td>58.21</td>
<td>78.99</td>
<td>90.05</td>
</tr>
<tr>
<td>ABM1</td>
<td>Organization</td>
<td>85.19</td>
<td>85.87</td>
<td>50.66</td>
<td>80.45</td>
<td>85.43</td>
</tr>
<tr>
<td>ABM1</td>
<td>Person</td>
<td>95.91</td>
<td>95.81</td>
<td>77.63</td>
<td>93.02</td>
<td>96.21</td>
</tr>
<tr>
<td>MLP</td>
<td>Location</td>
<td>91.14</td>
<td>95.35</td>
<td>56.72</td>
<td>76.32</td>
<td>89.63</td>
</tr>
<tr>
<td>MLP</td>
<td>Organization</td>
<td>85.17</td>
<td>87.30</td>
<td>52.29</td>
<td>78.74</td>
<td>85.38</td>
</tr>
<tr>
<td>MLP</td>
<td>Person</td>
<td>95.79</td>
<td>96.61</td>
<td>81.09</td>
<td>90.88</td>
<td>95.83</td>
</tr>
<tr>
<td>J48</td>
<td>Location</td>
<td>91.27</td>
<td>95.71</td>
<td>56.53</td>
<td>78.99</td>
<td>90.08</td>
</tr>
<tr>
<td>J48</td>
<td>Organization</td>
<td>85.18</td>
<td>85.87</td>
<td>50.56</td>
<td>80.49</td>
<td>85.44</td>
</tr>
<tr>
<td>J48</td>
<td>Person</td>
<td>95.91</td>
<td>95.81</td>
<td>77.10</td>
<td>92.36</td>
<td>96.23</td>
</tr>
</tbody>
</table>

### Table 21: Runtime evaluation of FOX.

<table>
<thead>
<tr>
<th></th>
<th>News</th>
<th>News*</th>
<th>Web</th>
<th>Reuters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOX 2.2.2</td>
<td>16h22m</td>
<td>7m41s</td>
<td>9m28s</td>
<td>2m27s</td>
</tr>
<tr>
<td>FOX 1.0</td>
<td>&gt;24h</td>
<td>11m35s</td>
<td>24m53s</td>
<td>9m22s</td>
</tr>
</tbody>
</table>

### 4.4.2.6 Runtime Evaluation

In addition to improving the accuracy of FOX, we also improved its runtime. To this end, we implemented a controller that allows several FOX instances to run in parallel and dispatches requests to the next free instances. Moreover, we extended FOX as to enable it to run only one of the algorithms contained in its backend (we dub this mode FOX light\[32\]). We evaluated FOX 2.2.2 against FOX 1.0 on the benchmark datasets used in the previous section. As shown in Table 21, we achieve a speedup of up 76.5%, which suggest that FOX is easily amenable to scaling horizontally.

\[32\]See demo at [http://fox.aksw.org](http://fox.aksw.org).
Figure 27: *News* and *News* dataset.
Figure 28: **Web** and **Reuters** dataset.
Figure 29: All dataset.
5 Summary

This deliverable is primarily concerned with the scalability extensions to GeoLift and its major components. We explored several architectures for parallel processing and showed that local parallel processing is the best compromise between data size and scalability. We thus implemented parallel versions of the algorithms underlying GeoLift, LIMES and FOX. We also look at planning and load balancing to improve the runtime of these algorithms and were able to show that these approaches can indeed improve the performance of the approaches at hand. Within WP3, we will analyze the performance of these novel developments on real data and update the algorithms and implementations as needed. The results of these updates and evaluations will be presented in D.3.1.3.
References


