Collaborative Project

GeoKnow - Making the Web an Exploratory for Geospatial Knowledge

Project Number: 318159 **Start Date of Project:** 2012/12/01 **Duration:** 36 months

Deliverable 3.1.3
Evaluation of Spatial Interlinking

Dissemination Level	Public
Due Date of Deliverable	Month 30, 31/05/2015
Actual Submission Date	Month 30, 27/05/2015
Work Package	WP3, Spatial Knowledge Aggregation, Fusing and Quality Assessment
Task	T3.1
Type	Report
Approval Status	Approved
Version	1.0
Number of Pages	36
Filename	D3.1.3_Evaluation_of_spatial_interlinking.pdf

Abstract: The aim of this deliverable is to summarize the evaluation of geo-spatial link discovery carried out within GeoKnow. We thus provide evaluations of ORCHID pertaining to different metric implementations and hardware configurations. We evaluate for runtime, F-measure, precision and recall on manually validated benchmarks.

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

History

Version	Date	Reason	Revised by
0.0	08/05/2015	First draft created	Mohamed Ahmed Sherif
0.1	13/05/2015	First versions of sections 2–7	Mohamed Ahmed Sherif Mofeed Hassan
0.2	15/05/2015	First complete draft	Mohamed Ahmed Sherif Axel-Cyrille Ngonga Ngomo
0.3	20/05/2015	Review of the deliverable	Giorgos GIannopoulos
0.4	22/05/2015	Revised version	Mohamed Ahmed Sherif
1.0	26/05/2015	Final version	Mohamed Ahmed Sherif Axel-Cyrille Ngonga Ngomo

Author List

Organization	Name	Contact Information
INFAI	Mohamed Ahmed Sherif	sherif@informatik.uni-leipzig.de
INFAI	Axel-Cyrille Ngonga Ngomo	ngonga@informatik.uni-leipzig.de
INFAI	Mofeed Hassan	mounir@informatik.uni-leipzig.de

Executive Summary

The aim of this deliverable is to present the experiments carried out to evaluate spatial interlinking. We begin by presenting the evaluation of Orchid, our time-efficient and reduction-ratio-optimal algorithm for link discovery. It is first evaluated in combination with the Hausdorff distance. Thereafter, we evaluate other distances for sets of points thoroughly and show that the mean measure should be the measure of choice when preforming link discovery of geo-spatial data. Our subsequent experiments all pertain to the improvement of runtime. We begin by studying caching for link discovery. Here, we show that current caching approaches can be used for geo-spatial link discovery but that dedicated approaches for this purpose should be designed. We also look at parallel implementations of Orchid including load balancing. Finally, we look at different possible hardware configurations to run Orchid and show that local architectures are superior to remote clouds when it comes to link discovery.

Abbreviations and Acronyms

FIFO	First-In First-Out
FIFO2ndChance	First-In First-Out Second Chance
LD	Link Discovery
LFUDA	Least Frequently Used with Dynamic Aging
LGD	LinkedGeoData dataset
LIMES	Link Discovery Framework for Metric Spaces
LOD	Linked Open Data
LRU	Least Recently Used
MSE	Mean Squared Error
NER	Named Entity Recognition
PSO	Particle Swarm Optimization
RDF	Resource Description Framework
SLRU	Segmented Least Recently

Contents

1 Introduction **8**

2 Datasets **9**

3 ORCHID **10**

3.1 Experimental Setup . . . 10
3.2 Hausdorff Implementations . . . 10
3.3 Deduplication . . . 10
3.4 Scalability . . . 12
3.5 Comparison with other approaches . . . 12

4 Geo-Spatial Metrics **15**

4.1 Experimental Setup . . . 16
4.2 Scalability . . . 17
4.3 Robustness . . . 17
4.3.1 Robustness against Discrepancies in Granularity . . . 17
4.3.2 Robustness against Measurement Discrepancies . . . 18
4.3.3 Overall Robustness . . . 19
4.4 Scalability with ORCHID . . . 19
4.5 Experiment on Real Datasets . . . 20

5 Caching **22**

5.1 Caching Approaches . . . 22
5.1.1 FIFO . . . 23
5.1.2 FIFO2ndChance . . . 23
5.1.3 LRU . . . 23
5.1.4 LFU . . . 23
5.1.5 SLRU . . . 23
5.1.6 LFUDA . . . 23
5.2 Experiments and Results . . . 23
5.2.1 Experimental Setup . . . 24
5.2.2 Results . . . 24

6 Parallel ORCHID **28**

6.1 Experimental Setup 28
6.2 Orchid vs. Parallel Orchid 28

7 Hardware Assessment **32**
7.1 Experimental Setup 32
7.2 Performance Comparison 32
7.3 Scalability 34

8 Summary **35**

References **35**

List of Figures

1 Distribution of polygon sizes. . . . 9
2 Number of comparisons and runtimes on samples of the datasets. . . . 11
3 Number of comparisons and runtime of ORCHID. . . . 13
4 Comparison of runtime of SILK and ORCHID. . . . 14
5 Scalability evaluation on the Nuts dataset. . . . 17
6 Comparison of different point set distance measures against granularity discrepancies. . . . 18
7 Comparison of different point set distance measures against measurement discrepancies. . . . 19
8 Comparison of different point set distance measures against granularity and measurement discrepancies. . . . 20
9 Scalability evaluation with ORCHID. . . . 21
10 Number of cache hits for different distance thresholds (dataset size = 10^4 resources) 25
11 Runtimes of the different caching approaches for different distance thresholds (dataset size = 10^4 resources) 26
12 Cache hits for different cache sizes (dataset size = 10^5 resources) 27
13 Runtimes for different cache sizes (dataset size = 10^5 resources) 27
14 Runtime and MSE generated when applying ORCHID [5] vs. parallel implementations of ORCHID using naïve, greedy, pair based, PSO and DPSO load balancing algorithms against the three real datasets of *Nuts*, *DBpedia* and *LinkedGeoData* using 2, 4 and 8 threads 30
15 Runtime and MSE generated when applying parallel implementations of ORCHID using naïve, greedy, pair based, PSO and DPSO load balancing algorithms against the five synthetic datasets of sizes 1, 2 , ..., 5 million polygons using 2, 4 and 8 threads 31
16 Datasets used for evaluation 32
17 Comparison of runtimes for Experiment 1 33
18 Comparison of runtimes on DS4 34

List of Tables

1 Scalability results. The top section shows the results on DBpedia while the lower section shows the results on LinkedGeoData. . . . 12
2 Runtimes of the different caching approachesand varying distance thresholds (dataset size = 10^4 resources) . . . 25
3 Number of hits for different caching approaches and varying distance thresholds (dataset size = 10^4 resources). . . . 26
4 Runtimes (seconds) for different cache sizes(dataset size = 10^5 resources) . . . 26
5 Hit rates of different caching approaches for different cache sizes(dataset size = 10^5 resources) 27

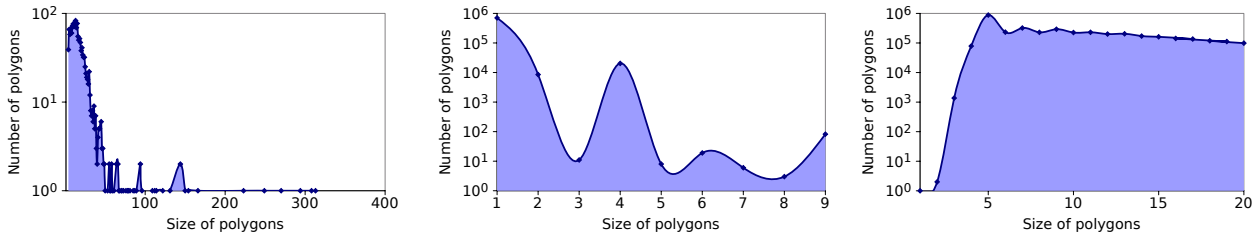
1 Introduction

The Linked Open Data Cloud has grown from a mere 12 datasets at its beginning to a compendium of at least 10 000 public RDF data sets.[1] In addition to the number of the datasets published growing steadily, we also witness the size of single datasets growing with each new edition. For example, *DBpedia* has grown from 103 million triples describing 1.95 million things (DBpedia 2.0) to 583 million triples describing 4.58 million things (DBpedia 2014) within 7 years. Other popular data sources such as *LinkedGeoData* has underwent even more drastic changes since its creation to reach now more than 30 billion triples. The growth of the Data Web engenders an increasing need for dedicated time-efficient Link Discovery (LD) approach. We have addressed this problem for geo-spatial datasets by developing the Orchid algorithm, which is now the cornerstone of the Limes framework when linking geo-spatial datasets is concerned. In this deliverable, we aim to describe the experiments carried out to evaluate geo-spatial LD (and especially Orchid) in different configurations. Note that parts of these results were also included in previous deliverables as we targeted to evaluate our approaches as well as possible from the beginning of the project on. We thus begin in section 3 by presenting the evaluation of Orchid– our time-efficient and reduction-ratio-optimal algorithm for link discovery. Orchid is still the fastest complete algorithm for link discovery on geo-spatial datasets. The algorithm is first evaluated in combination with the Hausdorff distance, a point-set distance that is used to approximate the distance between polygons. Thereafter, we evaluate other distances for sets of points thoroughly (e.g., the Fréchet distance, the mean distance, surjection-based approaches, etc.) and show that the mean measure should be the measure of choice when preforming link discovery for geo-spatial data (see section 4). Our subsequent experiments all pertain to the improvement of runtime. We begin by studying caching for link discovery (see section 5). Here, we show that current caching approaches can be used for geo-spatial link discovery but that dedicated approaches for this purpose should be designed. We also look at parallel implementations of Orchid including load balancing (see section 6). Finally, in section 7 we look at different possible hardware configurations to run Orchid and show that local architectures are superior to remote clouds when it comes to link discovery.

[1] http://lodstats.aksw.org

2 Datasets

We selected three publicly available datasets of different sizes for our experiments. The first dataset, *Nuts*, contains a detailed description of 1,461 specific European regions.[2] The second dataset, *DBpedia*, contains all 731,922 entries from DBpedia that possess a geometry entry.[3] Finally, the third dataset, LGD, contains all 3,836,119 geo-spatial objects from LinkedGeoData that are instances of the class `Way`.[4] An overview of the distribution of the polygon sizes in these datasets is given in Figure 1. In addition, we used a dataset that consists of all points which have the `wgs84:geometry` property[5] from DBpedia for the comparison with SILK.[6] The 732,224 entities in this dataset are single points on the surface of the planet.



(a) Nuts (b) DBpedia (c) LGD

Figure 1: Distribution of polygon sizes.

[2] We used version 0.9.1 as available at http://nuts.geovocab.org/data/.

[3] We used version 3.8 as available at http://dbpedia.org/Datasets.

[4] We used the RelevantWays dataset (version of April 26th, 2011) of LinkedGeoData as available at http://linkedgeodata.org/Datasets.

[5] `wgs84` stands for http://www.w3.org/2003/01/geo/wgs84_pos#.

[6] The dataset was extracted from the RelevantNodes dataset (version of April 26th, 2011) of DBpedia as available at http://linkedgeodata.org/Datasets.

3 ORCHID

The idea behind ORCHID [5] is to improve the runtime of algorithms for measuring geo-spatial distances by adapting an approach akin to divide-and-conquer. ORCHID assumes that it is given a distance measure (not necessarily a metric) m that abides by $m(s,t) \leq \theta \rightarrow \forall s_i \in s\ \exists t_j \in t : \delta(s_i, t_j) \leq \theta$.

The goal of the evaluation was to assess the performance of our approaches first introduced in a previous deliverable[7] with respect to their runtime and the number of computation of the orthodromic distance that they carried out. To achieve this goal, we first compare the bound, CS and $BC + CS$ implementations of Hausdorff distance on samples from three different datasets. Note that we refrained from using the whole datasets because the runtime of the naive approach would have been impracticable. In the second part of our evaluation, we study the combination of ORCHID and our Hausdorff implementations.

3.1 Experimental Setup

All experiments were carried out on a 32-core server running JDK 1.7 on Linux 10.04. The processors were 8 quadcore AMD Opteron 6128 clocked at 2.0 GHz. Unless stated otherwise, each experiment was assigned 10GB of memory and was ran 5 times. The time-out for experiments was set to 3 hours per iteration. The granularity parameter α was set to 1. In the following, we present the minimal runtime of each of the experiments.

3.2 Hausdorff Implementations

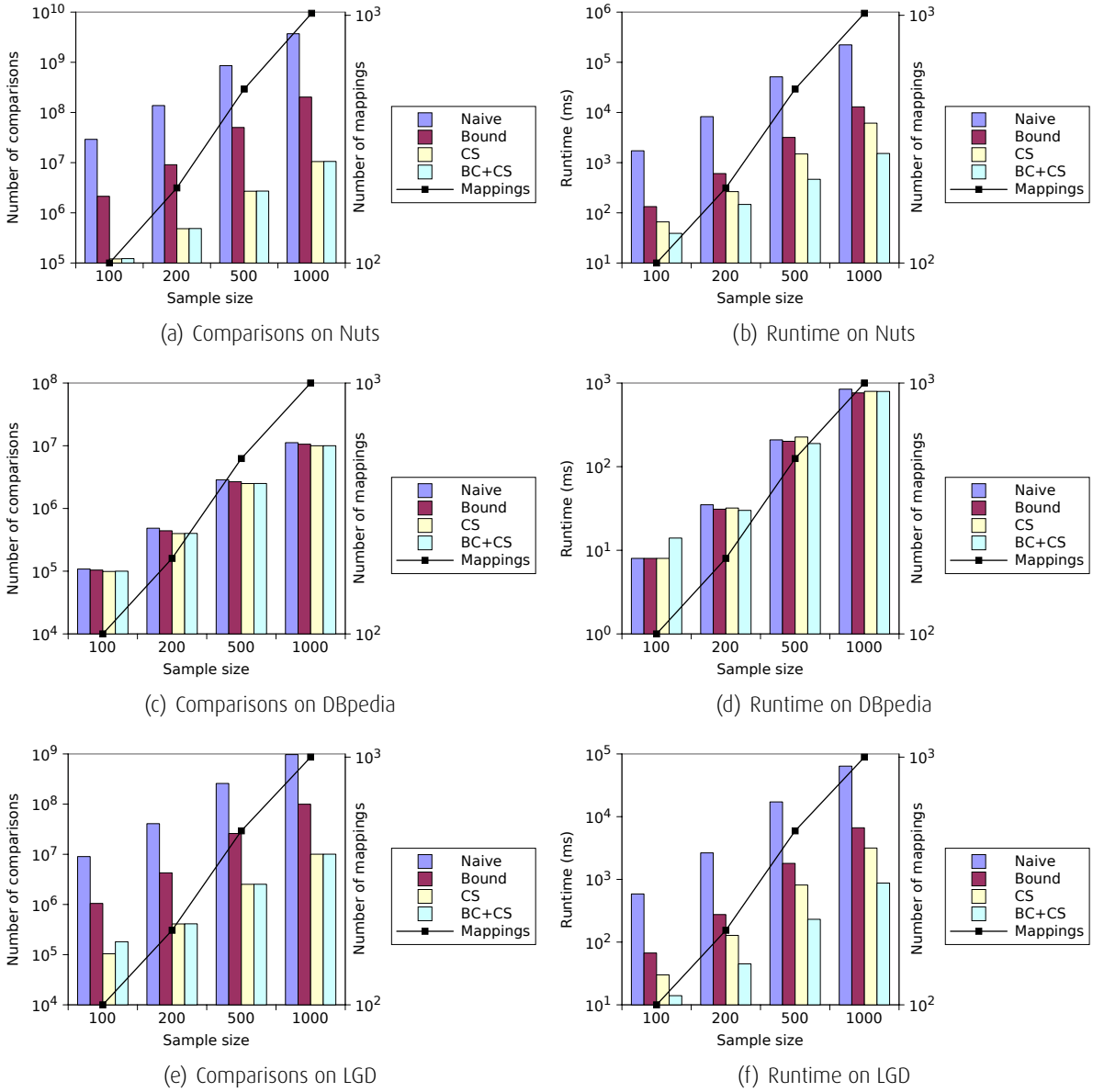
In the first part of our evaluation, we measured the runtimes achieved by the three different implementation of the Hausdorff distances (i.e. bound, CS and $BC + CS$) on random samples of the Nuts, DBpedia and LGD data sets. We used three different thresholds for our experiments, i.e., $100\,m$, $0.5\,km$ and $1\,km$. In Figure 2, we present the results achieved with a threshold of $100\,m$. The results of the same experiments for $0.5\,km$ and $1\,km$ did not provide us with significantly different insights. As expected the runtime of all three approaches increases quadratically with the size of the sample. There is only a slight variation in the number of comparisons carried by the three approaches on the DBpedia dataset, this is simply due to most polygons in the dataset having only a small number of nodes as shown in Figure 1. With respect to runtime, there is no significant difference between the different approaches on DBpedia. This is an important result as it suggests that we can always use the CS or $BC + CS$ approaches even when the complexity of the polygons in the datasets is unknown.

On the two other datasets, the difference between the approaches with respect to both the number of comparisons and the runtime can be seen clearly. Here, the bound implementation requires an order of magnitude less comparisons than the naive approach while the indexed implementations need two orders of magnitude less comparisons. The runtimes achieved by the approaches reflect the observations made on the comparisons. In particular, the bound approach is an order of magnitude faster than the naive approach. Moreover, the $BC + CS$ approach outperforms the bound approach by approximately one further order of magnitude. Note that up to approximately 1.07% of the comparisons carried out by $BC + CS$ are the result of the indexing step.

3.3 Deduplication

In our second series of experiments, we deduplicated the three datasets at hand by using four different thresholds between $100\,m$ and $2\,km$. We compared the combination of ORCHID ($\alpha = 1$) and of all different imple-

[7] For more details see deliverable 3.1.1 "Development of First Prototype for Spatially Interlinking Data Sets"



(a) Comparisons on Nuts

(b) Runtime on Nuts

(c) Comparisons on DBpedia

(d) Runtime on DBpedia

(e) Comparisons on LGD

(f) Runtime on LGD

Figure 2: Number of comparisons and runtimes on samples of the datasets.

mentations of the Hausdorff distance. The rationale behind this experiment was to measure whether the bound and indexed implementations were of any use even within the smaller sub-problems generated by ORCHID. The results achieved show that using these implementations can indeed lead to significant improvements in both runtime and comparisons (see Figure 3). In particular, the indexed distance profits from the fact that it can discard a large number of computations that would lead to distance below and above the distance threshold. Thus, the indexed approach requires over than two orders of magnitude less computations than the bound and naive versions on the Nuts dataset. Given the small size of the index generated for Nuts, the indexed approach is also two orders of magnitude faster across all the thresholds. On the LGD dataset, only the experiment of the indexed approach is terminated because it exceeds the iteration time-out of 3 hours. Due to the topology of the DBpedia data, the runtimes on DBpedia are comparable for all approaches. Here, it is important to note that for smaller thresholds, the indexed approach still requires close to an order of magnitude less comparisons than the naive approach.

3.4 Scalability

We were also interested in knowing how our approach performs with growing dataset sizes. We thus ran ORCHID in combination with BC with randomly selected slices of LinkedGeoData and DBpedia and computed the runtime against the size of the data slices. The similarity threshold was set to $0.1\,km$ as in the previous experiment. The results on DBpedia and LinkedGeoData are shown in Table 1. We omitted Nuts because it is too small for scalability experiments. The runtimes and number of comparisons on DBpedia suggest that the approach behaves in a quasi-linear fashion on low-dimensional and sparsely distributed data. Note that the number of mappings being relatively larger than the number of computations on this dataset is simply due to the fact that items with the same URI are found in both source and target and thus do not require any comparisons for deduplication. This is more rarely the case in the LinkedGeoData dataset. The runtimes on LinkedGeoData yet suggest that both the number of computations and the runtime required of our approach grow sub-linearly with the number of mappings to be computed when the number of points per polygon grows. This can be attributed to the fact that our approach makes effective use of existing data to discard computations and reduce the ratio of number of computations to mappings with growing data size. Thus, our approach promises to scale well to even larger data sets.

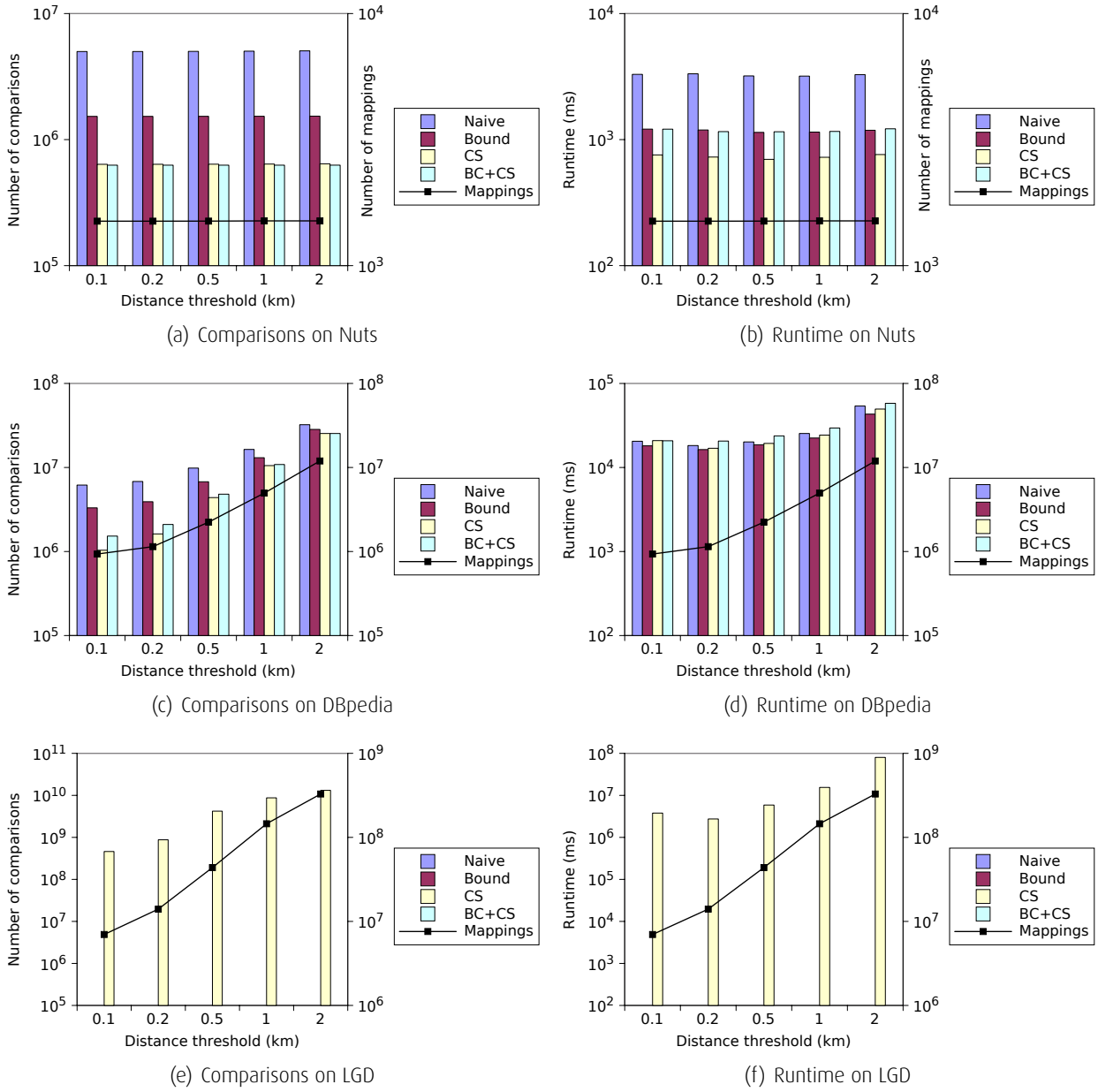
Table 1: Scalability results. The top section shows the results on DBpedia while the lower section shows the results on LinkedGeoData.

Sample Size	od computations	Runtime (ms)	Mappings
10^5	34,959	2,936	103,428
2×10^5	97,798	5,783	215,096
4×10^5	341,986	10,423	459,681
7.3×10^5	1,035,222	20,727	932,848
10^5	5,703,683	42,437	77,003
2×10^5	11,734,609	57,935	159,878
4×10^5	24,844,435	153,174	342,477
8×10^5	55,212,459	411,248	777,826
16×10^5	131,405,064	819,636	1,902,803

3.5 Comparison with other approaches

SILK[8] [3] is of the few other LD framework which implements the orthodromic distance. To the best of our knowledge, no other LD framework implements the Hausdorff distance. Thus, we compare ORCHID in combination with the naive implementation of the Hausdorff distance to SILK on all 732,224 points from DBpedia that contain longitude and latitude information. The results of four different distance thresholds are shown in Figure 4. Our results clearly show that ORCHID outperforms SILK by more than one order of magnitude in all settings.

[8] Throughout our experiments, we used SILK 2.5.3.



(a) Comparisons on Nuts

(b) Runtime on Nuts

(c) Comparisons on DBpedia

(d) Runtime on DBpedia

(e) Comparisons on LGD

(f) Runtime on LGD

Figure 3: Number of comparisons and runtime of ORCHID.

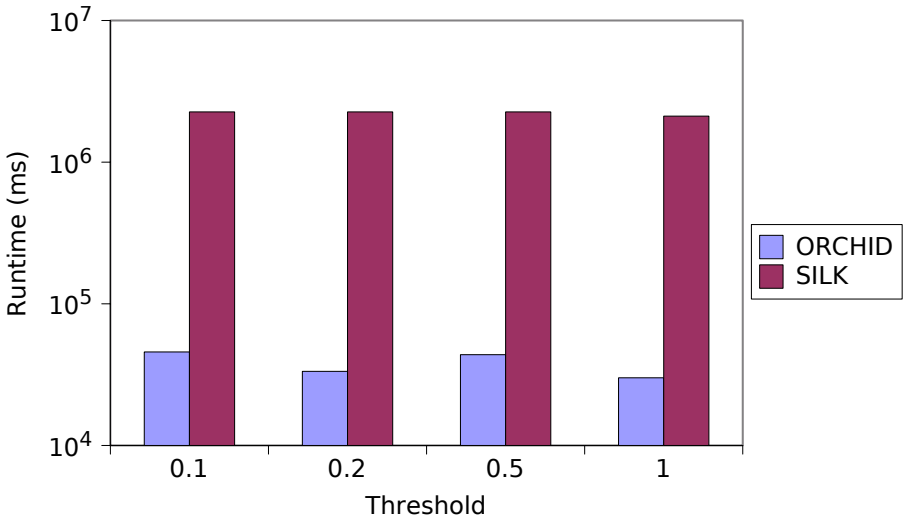


Figure 4: Comparison of runtime of SILK and ORCHID.

4 Geo-Spatial Metrics

Geo-spatial resources are commonly described by means of vector geometry.[9] Each vector description can be modelled as a set of points. We will write $S = (s_1, \ldots, s_n)$ to denote that the vector description of the resource S comprises the points $s_1, \ldots, s_n$. A point s_i on the surface of the planet is fully described by two values: its latitude $lat(s_i) = \varphi_i$ and its longitude $lon(s_i) = \lambda_i$. We will denote points s_i as pairs (φ_i, λ_i). Then, the distance between two points s_1 and s_2 can be computed by using the *orthodromic distance*

$$\delta(s_1, s_2) = R \cos^{-1}\left(\sin(\varphi_1)\sin(\varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\cos(\lambda_2 - \lambda_1)\right),$$

where $R = 6371km$ is the planet's radius.[10] Computing the distance between sets of points is yet a more difficult endeavour. Over the last years, several measures have been developed to achieve this task. Most of these approaches regard vector descriptions as ordered set of points. In the following, we present an evaluation of several metrics for linking geospatial datasest first presented in a previous deliverable[11]. The input for the distances consists of two point sets $S = (s_1, \ldots, s_n)$ and $T = (t_1, \ldots, t_m)$, where n resp. m stands for the number of distinct points in the description of S resp. T. W.l.o.g, we assume $n \geq m$. In the following, we evaluate the ten metrics:

1. Mean Metric
$$D_{mean}(S,T) = \delta\left(\frac{\sum_{s_i \in S} s_i}{n}, \frac{\sum_{t_j \in T} t_j}{m}\right). \tag{1}$$
2. Max Metric
$$D_{max}(S,T) = \max_{s_i \in S, t_j \in T} \delta(s_i, t_j). \tag{2}$$
3. Min Metric
$$D_{min}(S,T) = \min_{s_i \in S, t_j \in T} \delta(s_i, t_j). \tag{3}$$
4. Average Metric
$$D_{avg}(S,T) = \frac{1}{nm} \sum_{s_i \in S, t_j \in T} \delta(s_i, t_j). \tag{4}$$
5. Sum of Minimums Metric
$$D_{som}(S,T) = \frac{1}{2}\Big(\sum_{s_i \in S} \min_{t_j \in T} \delta(s_i, t_j) + \sum_{t_i \in T} \min_{s_j \in S} \delta(t_i, s_j)\Big). \tag{5}$$
6. Surjection Metric
$$D_s(S,T) = \min_{\eta} \sum_{(e1,e2) \in \eta} \delta(e_1, e_2), \tag{6}$$
where η is the surjection from the larger of the point sets S and T to the smaller.
7. Fair Surjection Metric
$$D_{fs}(S,T) = \min_{\eta'} \sum_{(e1,e2) \in \eta'} \delta(e_1, e_2), \tag{7}$$
where η' is the evenly mapped surjection from the larger of the sets S and T to the smaller.

[9] Most commonly encoded in the WKT format, see http://www.opengeospatial.org/standards/sfa.
[10] We assume the planet to be a perfect sphere.
[11] Deliverable 3.4.1 "Metrics for Linked Geospatial Information"

8. Link Metric

$$D_{link}(S,T) = \min_{R} \sum_{(s_i,t_j)\in R} \delta(s_i,t_j), \tag{8}$$

where minimum is computed from all relations R, where R is a linking between S and T satisfying the previous two conditions.

9. Hausdorff Metric

$$D_{hd}(S,T) = \max_{s_i\in S}\left\{\min_{t_j\in T}\left\{\delta(s_i,t_j)\right\}\right\}. \tag{9}$$

10. Fréchet Metric

$$D_{fr}(S,T) = \inf_{\substack{\alpha[0,1]\to[s_1,s_n]\\ \beta[0,1]\to[t_1,t_n]}}\left\{\sup_{t\in[0,1]}\left\{\delta(f(\alpha(t)) - g(\beta(t)))\right\}\right\}, \tag{10}$$

where $f : [s_1, s_n] \to V$ and $g : [t_1, t_n] \to V$. α, β range over continuous and increasing functions with $\alpha(0) = s_1$, $\alpha(1) = s_n$, $\beta(0) = t_1$ and $\beta(1) = t_n$ only.

The goal of our evaluation was to answer the folowing research questions:

Q_1: Which of the existing measures is the most time-efficient measure?

Q_2: Which measure generates mappings with a high precision, recall, or F-measure?

Q_3: How well do the measures perform when the datasets have different granularities?

Q_4: How sensitive are the measures to measurement discrepancies?

Q_5: How robust are the measures when both types of discrepancy occur?

4.1 Experimental Setup

To the best of our knowledge, there is no gold standard benchmark geographic dataset that can be used to evaluate the robustness of geo-spatial distance measures. We thus adapted the benchmark generation approach proposed by [2] to geo-spatial distance measures. We implemented two modifiers, which both take a polygon s and a threshold as input and return a polygon. The *granularity modifier* M_g regards the threshold $\gamma \in [0, 1]$ as the probability that a point of s will be in the output polygon. To ensure that an empty polygon is never generated, the modifier always includes the first point of s into its output. For all other points s_i, a random number r between 0 and 1 is generated. If $r \leq \gamma$, then s_i is added to the output of the modifier. Else, s_i is discarded. The *measurement error modifier* M_e emulates measurement errors across datasets. To this end, it alters the latitude and longitude of each the points of s by at most the threshold μ. Consequently, the new coordinates of a point s_i are located within a square of size 2μ with s_i at the center. We used a sample of 200 points from each dataset for our discrepancy experiments.

To measure how well each of the distances performed w.r.t. to the modifiers, we first created a reference mapping $M = \{(s, s) \in S\}$ when given a set of input resources S. Then, we applied the modifier to all the elements of S to generate a target dataset T. We then measured the distance between each of the point sets in the set T and the resources in S. For each element of S we stored the closest point $t \in T$ in a mapping M'. We now computed the precision, recall and F-measure achieved within the experiment by comparing the pairs in M' with those in M.

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 8 GB of memory and was ran 5 times.

4.2 Scalability

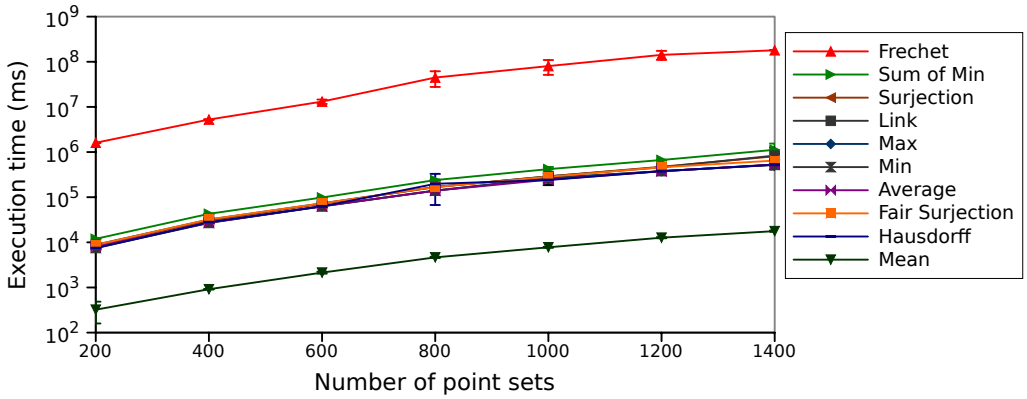


Figure 5: Scalability evaluation on the Nuts dataset.

To quantify how well the measures scale, we measured the runtime of the measures on fragments of growing size of each of the input datasets. This experiment emulates a naive deduplication on datasets of various sizes. The results achieved on Nuts are shown in Figure 5. We chose to show Nuts because it is the smallest and most fine-granular of our datasets. Thus, the measures achieved here represent an upper bound for the runtime behaviour of the different approaches. D_{mean} is clearly the most time-efficient approach. This was to be expected as its algorithmic complexity is linear. While most of the other measures are similar in their efficiency, the Fréchet distance sticks out as the slowest to run. Overall, it is at least two orders of magnitude slower than the other measures. These results give a clear answer to question Q_1, which pertains to the time-efficiency of the measures at hand: D_{mean} is clearly the fastest.

4.3 Robustness

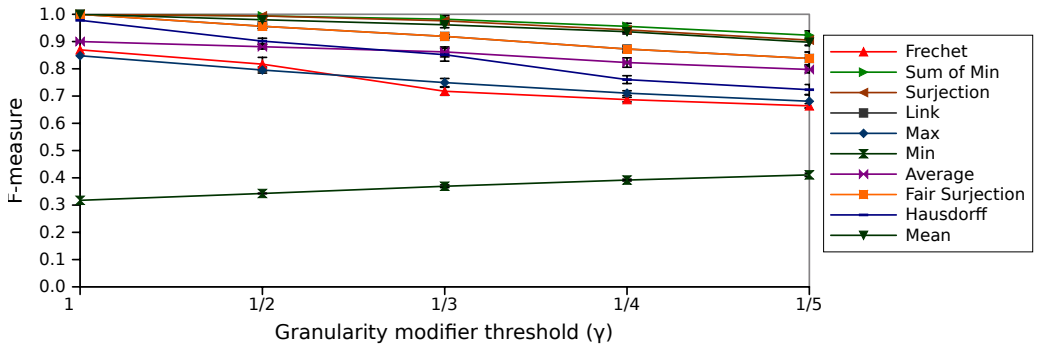
We carried out three types of evaluations to measure the robustness of the measures at hand. First, we measured their robustness against discrepancies in granularity. Then, we measured their robustness against measurement discrepancies. Finally, we combined discrepancies in measurement and granularity and evaluated all our measures against these. We chose to show only a portion of our results for the sake of space. All results can be found at `http://limes.sf.net`.

4.3.1 Robustness against Discrepancies in Granularity

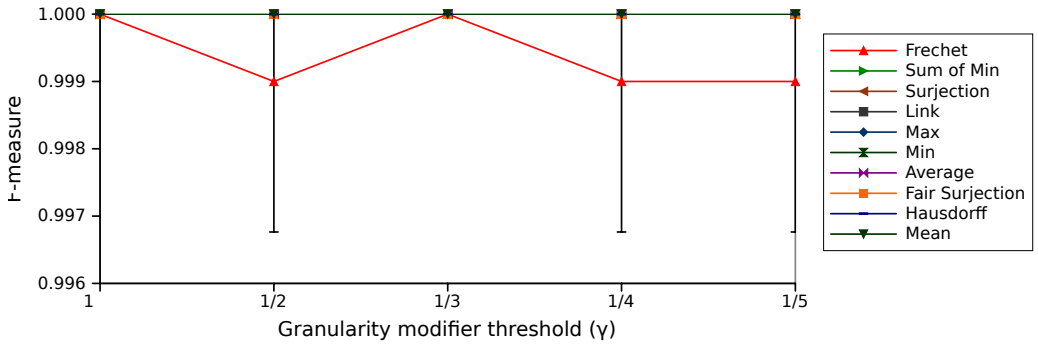
We measured the effect of changes in granularity on the measures at hand by using the five granularity thresholds 1, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$ and $\frac{1}{5}$. Note that the threshold of 1 means that no change was actually carried out in the dataset. This setting allows us to answer Q_2, which pertains to the measures that are most adequate for deduplication. On Nuts (see Figure 6(a)), our results suggest that D_{min} is the least robust of the measures w.r.t. the F-measure. In addition to being the least time-efficient measure, Fréchet is also not robust against changes in granularity. The best performing measure w.r.t. to its F-measure is the *sum of minimums*, followed closely by the surjection and mean measures. On the DBpedia and LGD datasets, all measures apart from the Fréchet distance perform in a similar fashion (see Figure 6(b)). This is yet simply due the sample of the

dataset containing point sets that were located far apart from each other. Thus, the answer to question Q_3 on the effect of discrepancies in granularity is that while the *sum of mins* is the least sensitive to changes in granularity. However, note that sum of mins is closely followed by the mean measure.

The answer to Q_2 can be derived from the evaluation with the granularity threshold set to 1. Here, mean, fair surjection, surjection, sum of mins and link perform best. Thus, mean should be used because it is more time-efficient.



(a) Nuts

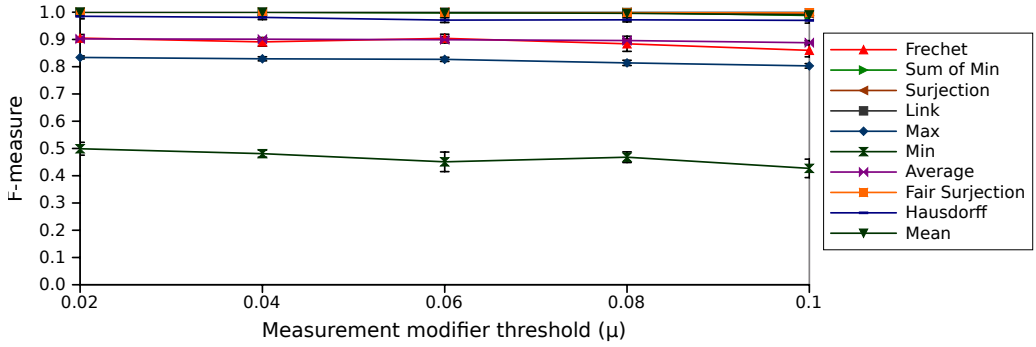


(b) LinkedGeoData

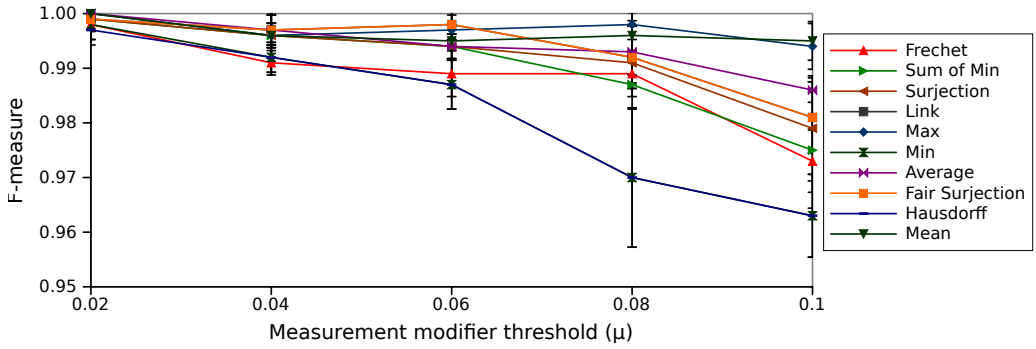
Figure 6: Comparison of different point set distance measures against granularity discrepancies.

4.3.2 Robustness against Measurement Discrepancies

The evaluation of the robustness of the measures at hand against discrepancies in measurement are shown in Figure 7. Interestingly, the results differ across the different datasets. On the Nuts data, where the regions are described with high granularity, five of the measures (mean, fair surjection, link, sum of mins and surjection) perform well. On LGD, the number of points pro resources is considerably smaller. Moreover, the resources are partly far from each other. Here, the Hausdorff distance is the poorest while max and mean perform comparably well. Finally, on the DBpedia dataset, all measures apart from Fréchet are comparable. Our results thus suggest that the answer to Q_4 is as follows: The mean distance is the distance of choice when computing links between geo-spatial datasets which contain measurement errors, especially if the resources described have a high geographical density or the difference in granularity is significant.



(a) Nuts



(b) LinkedGeoData

Figure 7: Comparison of different point set distance measures against measurement discrepancies.

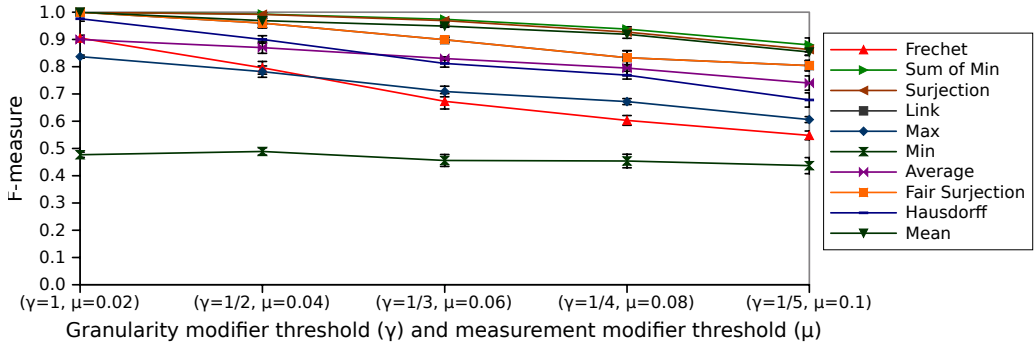
4.3.3 Overall Robustness

We emulated the differences across various real geographic datasets by combining the granularity and the measurement modifiers. Given a dataset S, we generated a modified dataset S' using the granularity modifier. The modified dataset was used as input for a measurement modifier, which generated our final dataset T. The results of our experiments are shown in Figure 8. Again, the results vary across the different datasets. While mean performs well on Nuts and LGD (Figure 8(a)), it is surjection that outperforms all the other measures on DBpedia Figure 8(b). This surprising result is due to the measurement errors having only a small effect on our DBpedia sample. Thus, after applying the granularity modifier, the surjection value is rarely affected.

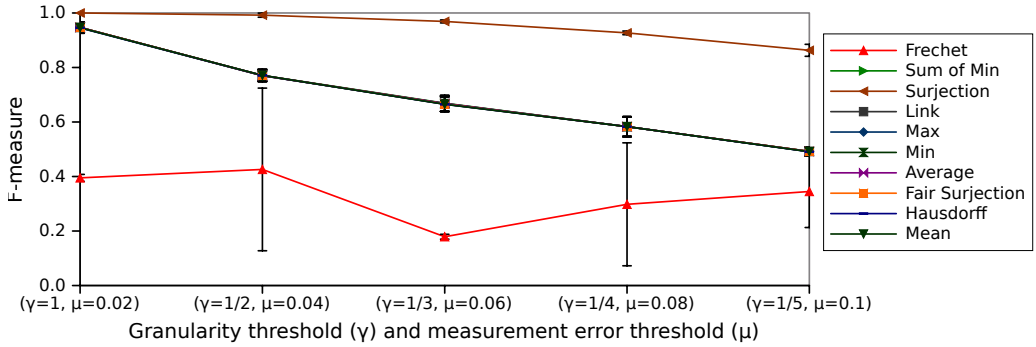
Overall, our results suggest that the following answer to Q_5: In most cases, using the mean distance leads to high F-measures. Moreover, mean presents the advantage of being an order of magnitude faster than the other approaches. Still, the surjection measure should also be considered when comparing different datasets as it can significantly outperform the mean measure.

4.4 Scalability with Orchid

We aimed to know how far the runtime of measures such as mean, surjection and sum of mins can be reduced so as to ensure that these measures can be used on large datasets. We thus combined these measures with the Orchid approach presented in [5]. The idea behind Orchid is to improve the runtime of algorithms for measuring geo-spatial distances by adapting an approach akin to divide-and-conquer. Orchid assumes that it is given a distance measure (not necessarily a metric) m that abides by $m(s,t) \leq \theta \rightarrow \forall s_i \in s\ \exists t_j \in t : \delta(s_i, t_j) \leq \theta$. This condition is obviously not satisfied by all measures considered herein, including min and mean. However,



(a) Nuts



(b) DBpedia

Figure 8: Comparison of different point set distance measures against granularity and measurement discrepancies.

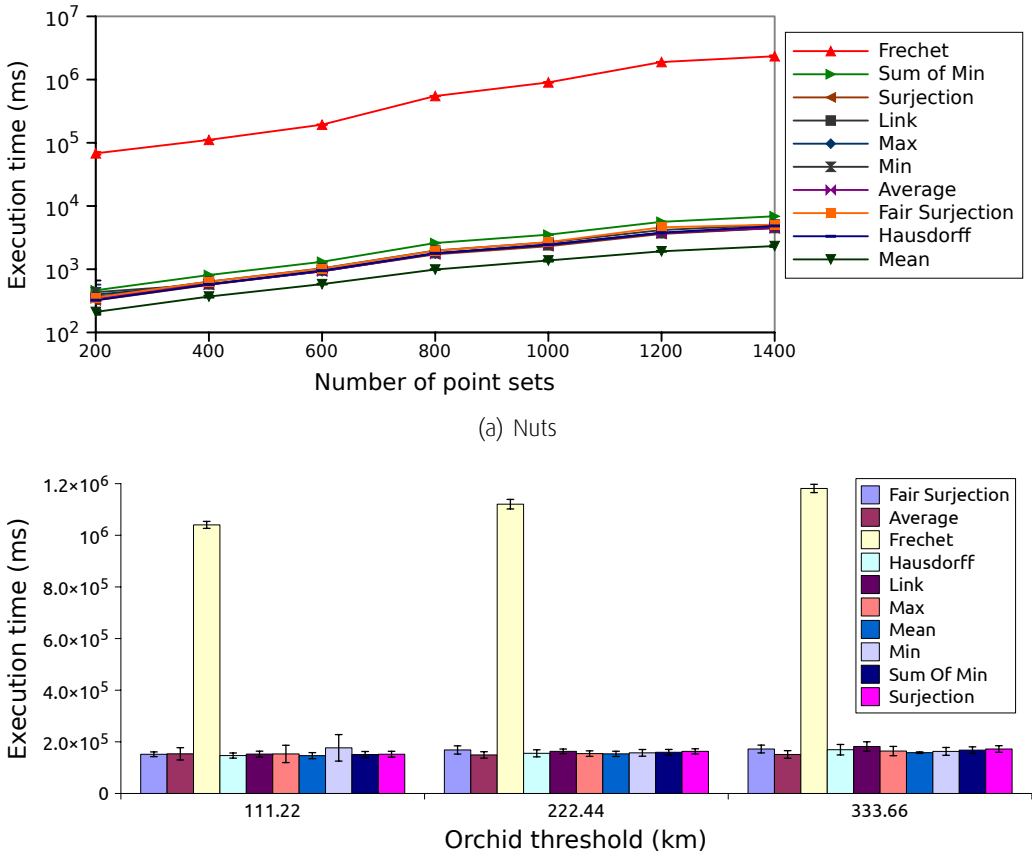
dedicated extensions of Orchid can be developed for these measures. Overall, Orchid begins by partitioning the surface of the planet. The points in a given partition are then only compared with points in partitions that abide by the distance threshold underlying the computation.

We used the default settings of the implementation provided in the LIMES framework and the distance threshold of $0.02°$(2.2km). Figure 9(a) shows the runtime results achieved on the same datasets as Figure 5. Clearly, the runtimes of the approaches can be decreased by up to an order of magnitude. Therewith, Orchid allows most measures (except for Fréchet) to scale in a manner comparable to that of the mean measure. Therewith, the measures can now be used on the whole of the datasets at hand. For example, all distance measures except for the Fréchet distance require less than five minutes to run on the whole of the DBpedia dataset (see Figure 9(b)).

Overall, we can conclude that all measures apart from the Fréchet distance are amenable to being used for link discovery. While *mean performs best overall, surjection-based and minimum-based measures are good candidates* to use if mean returns unsatisfactory results. The Fréchet distance on the other hand seems inadequate for link discovery. This can yet be due to the point set approach chosen in this paper. An analysis of the Fréchet distance on the description of resources as polygons remains as future work.

4.5 Experiment on Real Datasets

We were interested in knowing whether the mean function performs well on real data. Validating link discovery results on geo-spatial data is difficult due to the lack of reference datasets. We thus measured the increase in precision and recall achieved by using geo-spatial information by sampling 100 links from the results of real



(a) Nuts

(b) DBpedia

Figure 9: Scalability evaluation with ORCHID.

link discovery tasks and evaluating these links manually. The links were evaluated by the authors who reached an agreement of 100%.

In the first experiment, we computed links between cities in DBpedia and LinkedGeoData by comparing solely their labels by means of an exact match string similarity. No geo-spatial similarity metric was used, leading to cities being linked if they have exactly the same name. Overall only 74% of the links in our sample were correct. The remaining 26% differed in country or even continent. We can assume that a recall of 1 would be achieved by using this approach as a particular city will most probably have the same name across different geo-spatial datasets. Thus, in the best case, linking geo-spatial resources in DBpedia to LinkedGeoData would only lead to an F-measure of 0.85.

In our second experiment, we extended the specification described above by linking two cities if their names were exact matches (which was used in the first experiment) and the mean distance function between their geometry representation returned a value under 100km. In our sample, we achieved a perfect accuracy and thus an F-measure of 1. While this experiment is small, it clearly demonstrates the importance of using geo-spatial information for linking geo-spatial resources. Moreoveor, it suggests that the mean distance is indeed reliable on real data. More experiments yet need to be carried out to ensure that the empirical results we got in this experiment are not just a mere artifact in the data. We will achieve this goal by creating a benchmark for geo-spatial link discovery in future work.

5 Caching

Most time-efficient approaches for link discovery rely on reducing the number of comparisons of s and t by grouping elements of the source set to $S_i \subseteq S$ and elements of the target set to subsets $T_j \subseteq T$ and only comparing certain S_i with certain T_j. For example, two strings have a distance less or equal to 1 w.r.t. to the edit distance if they share at least one letter. If we assume that the resources in S and T are described by their labels and that δ is the edit distance on labels, then we can group the elements of S by the letters contained in their labels. In this case, S_i would be the subset of S such that the label of each of the resources in S_i contains the ith letter of the alphabet. If we define T_j similarly, then we would not need to compare S_i with T_j if $i \neq j$, leading to several comparisons not having to be carried out at all.

The insight behind the use of caching for link discovery is that even when the sets S and T do not fit in memory, single elements of S and T do. Hence, given an element s of S, the data necessary to find all $t \in T$ such that $\delta(s,t) \leq \theta$ can be loaded in memory as required. Elements of t or even whole subsets T_i of T that are commonly used during computations should be cached so as to be read from memory during computations instead of being loaded from the hard drive, which is obviously more time-consuming.

We implemented these insights as follows (see Algorithm 1): Let A be a time-efficient algorithm and $A(s) \subseteq T$ be the set of all elements of T that are to be compared with s according the to algorithm A. We iterate over all $s \in S$ and call the function `load`$(A(s))$. This function encapsulates the cache and loads the portions of $A(s)$ that can be found in memory (i.e., in the cache) directly from the memory. The portions that cannot be found in the cache are loaded from external memory (e.g., the hard drive) sequentially and sent to the cache as well as to the `compare` method, which checks each of the loaded t for whether $\sigma(s,t) \geq \theta$ holds.

Algorithm 1: Basic caching-based approach to link discovery

```
Data: Source S, target T, distance measure δ, distance threshold θ
Result: Set M ⊆ S × T
M = ∅;
for s ∈ S do
    A = load(s);
    for t ∈ A do
        if compare(s, t) == true then
            M = M ∪ {(s, t)}
    return M;
```

5.1 Caching Approaches

Caching strategies have several characteristics and can be classified by these [6]. These characteristics are the time since the last reference to an element in the cache (recency), the number of requests to an element in the cache (frequency), the size of an element in the cache (size), the cost to fetch an element (cost), the time since the last modification (modification), the time when an element gets stale and can be evicted from the cache (expiration) [6].

We choose as simple strategies First-In First-Out (FIFO) and First-In First-Out Second Chance (FIFO2ndChance); as a recency-based strategy Least Recently Used (LRU); as a frequency-based strategy Least Frequently Used (LFU); as a recency/frequency-based strategy Segmented Least Recently Used (SLRU) and as a function-based strategy Least Frequently Used with Dynamic Aging (LFUDA).

5.1.1 FIFO

FIFO is a simple strategy. Once the cache is full, the cache element that has been longest in the cache is removed before the insertion of a new element. It is based on the idea of first-in-first-out (FIFO) lists [7].

5.1.2 FIFO2ndChance

FIFO2ndChance is a modified FIFO strategy in the way that a cache element that has been longest in the cache and was referenced in the past (i.e., used in a previous computation) is removed but inserted again only once so that it gets a second chance. An unreferenced element that has been longest in the cache is removed.

5.1.3 LRU

LRU is based on the locality of reference and thus tries to predict future accesses to cache elements from previous accesses. The idea is to evict a cache element that led to the oldest hit in the cache. One of the main drawback of this approach is that the cache is not scan-resistant. Still, this is one of the most commonly used approaches [1].

5.1.4 LFU

LFU is based on the count of the number of accesses to entries in the cache. The cache evicts the entries with the smallest frequency count when necessary. This approach is scan-resistant but does not make use of the locality of reference. The main drawbacks of this approach is that elements that were accessed often in the past and thus having a high count of the number of accesses can remain in the cache even when they are never requested in the future.

5.1.5 SLRU

SLRU extends LRU by splitting the cache into an unprotected (US) and a protected segment (PS), while the former is used for new cache elements the later is reserved for popular elements. Both segments are LRU strategies but only elements in the US are evicted. A new elements is inserted into the US and on an access to this element it is moved to the PS. Elements from the PS are moved back to the US as the most recently used element when the PS gets full.

5.1.6 LFUDA

LFUDA avoids the cache pollution drawback of LFU with a dynamic aging effect of the cached elements. It calculates a key value K for each element i in the cache with $K_i = F_i + L$ where F_i is the count of the number of accesses of i and L is the running age factor of the cache. L starts at 0 and is updated for each evicted element e to its key value (i.e., $L = K_e$). The strategy evicts the element with the smallest key value from the cache.

5.2 Experiments and Results

The goal behind our experiments was to determine whether current state-of-the art caching algorithms can be used for link discovery. To this end, we assessed the performance of different caching approaches on real data w.r.t. to the runtime they required and the numbers of hits they were able to achieve. In the following,

we begin by presenting the experimental setup used for our experiments. Thereafter, we present and discuss our results.

5.2.1 Experimental Setup

We used the ORCHID [5] algorithm to compute the data segmentations. We used this algorithm for two reasons: First, it is reduction-ratio-optimal and does not tend to overgenerate data segmentations. Moreover, ORCHID can deal with geo-spatial data. This is important because the (to the best of our knowledge) currently largest data set on the Linked Open Data Cloud, i.e., LinkedGeoData, is a geo-spatial data set. Hence, the bias caused by unnecessary comparisons could be minimized while the size of the datasets used in our experiments could be maximized.

We used the following approaches during our evaluation: FIFO, FIFO2ndChance, LRU, SLRU, LFU and LFUDA. For all approaches the evict size was set to be one. Variant cache sizes were used including 10, 100, 1K, 10K and 100K. The evaluation was carried out in two phases. In the first phase, the size of data is 10^4 resources. Different distance thresholds of 0, 0.1, 0.3 and 0.5 km were used. Increasing the distance threshold results in the up rise of the number of compared polygons. This imposes more polygons to be cached and more computation time. The cache size was assigned to 10^3 for all caching approaches in the first phase. We selected the best three approaches for the second phase, which was a scalability evaluation. Here, we measured the number of hits and runtime of the approaches. The promoted approaches were opposed to different cache sizes measuring and comparing their run times and revealing the best performed approach. Cache sizes were 10^1, 10^2, 10^3, 10^4 and 10^5. In this phase the data size was increased to be 10^5 resources. The evaluation was carried out on a laptop running an Intel CoreTM i7 Quad Core 2.80GHz processor using 8G RAM.

5.2.2 Results

In this section we present the results produced based on the aforementioned exponential setup. Figures 10 and 11 show the results of the first series on a sample of LinkedGeoData containing 10^4 resources, i.e., 10^4 polygons. The number of cache hits for each of the caching approaches w.r.t. different distance thresholds is presented in Figure 10. It is noticeable that LFU achieves the lowest number of cache hits. SLRU also shows lower number of cache hits compared to the rest of the approaches that are almost close in the results. In Figure 11, the runtimes of the different caching approach is depicted and it suggests that LRU, SLRU and FIFO have the lowest run times relative to different distance thresholds. It is clear that the lower number of hits the caching approach provides the longer time it takes for fetching or replacing the targeted data. In FIFO fetching and evicting data are performed in time complexity O(1). For LRU and SLRU approaches, their implementations tend to avoid time-consuming updates for cache entries. Given that all approaches achieve similar numbers of cache hits the run time turns to be the effective factor in selecting the approaches for the next phase. Tables 2 and 3 give detailed insight on achieved results in this phase. The presented runtime values in Table 2 are in seconds for the sake of readability.

The results of the second phase for different cache sizes are illustrated in Figure 12 and Figure 13. Note that we used a larger dataset of size 10^5 for this series of experiments. In contrast to the results of previous phase, the runtimes for LRU, SLRU and FIFO approaches are quite similar as shown in Figure 12. This similarity is due previously mentioned reasons in first phase. Figure 13 shows the superiority of LRU and FIFO in number of hits in accordance with cache sizes. Detailed results are presented in Tables 4 and 5.

Overall, FIFO and LRU seem to be the best of the caching approaches presented in this paper both in terms of runtime and cache hits. However, a careful study of the results they achieve makes clear that their relative hit rates still lie below 50%. This suggests that while current caching approaches do have the potential to reduce the runtime of link discovery approaches, dedicated approaches for link discovery could improve the

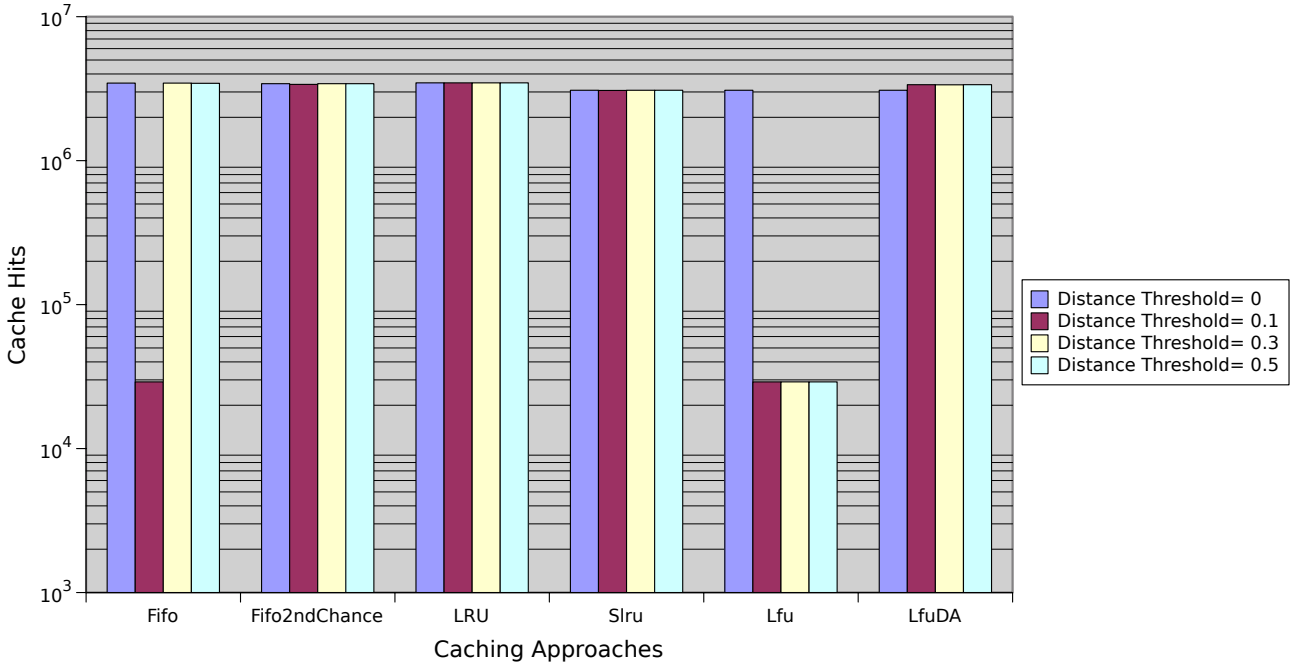


Figure 10: Number of cache hits for different distance thresholds (dataset size = 10^4 resources)

Table 2: Runtimes of the different caching approachesand varying distance thresholds (dataset size = 10^4 resources)

CacheType	Distance Threshold= 0	Distance Threshold= 0.1	Distance Threshold= 0.3	Distance Threshold= 0.5
FIFO	278.4	97098.9	347.7	372.8
FIFO2ndChance	9120.6	40228.7	11285.2	11242.9
LRU	321.3	386.5	423.7	440.3
SLRU	343.8	388.1	437	435.8
Lfu	3903.2	98271.1	100202	35161.7
LfuDA	343.8	42185.1	42216.3	33240.6

quality of caching. The study thus suggests that dedicated approaches for link discovery should be investigated in futute work to ensure the development of scalable link discovery approaches able to deal with Big Data.

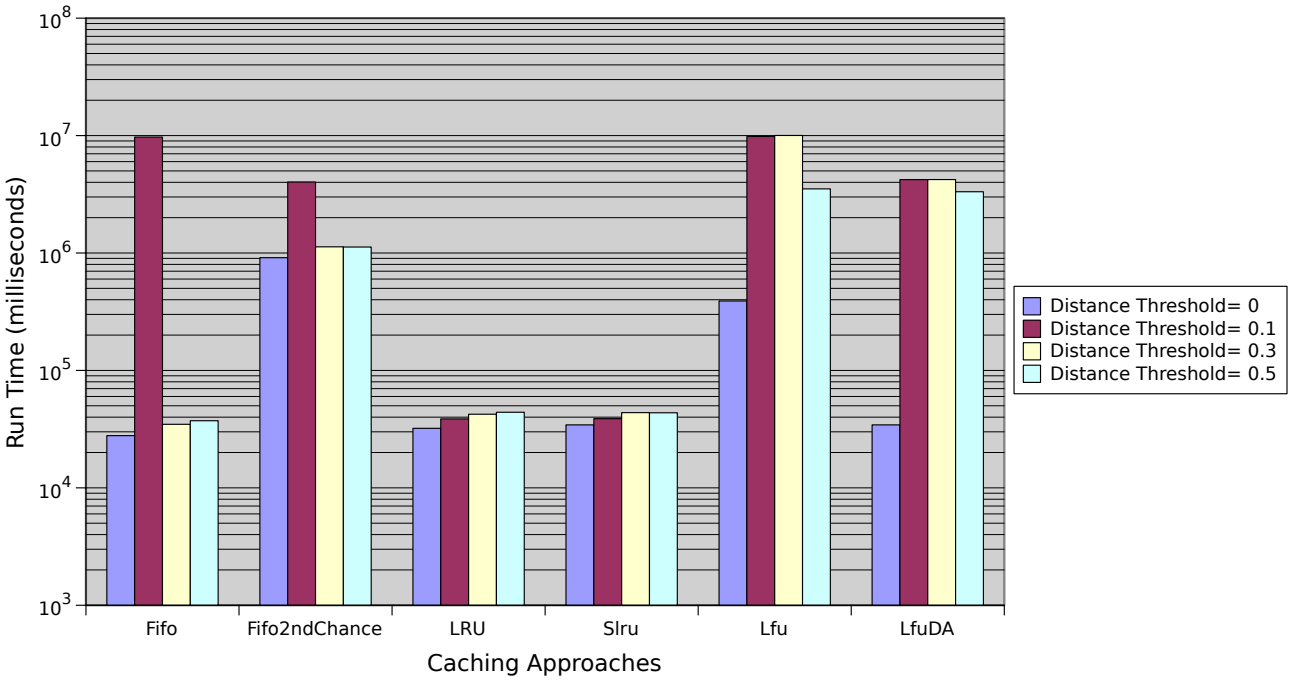


Figure 11: Runtimes of the different caching approaches for different distance thresholds (dataset size = 10^4 resources)

Table 3: Number of hits for different caching approaches and varying distance thresholds (dataset size = 10^4 resources).

CacheType	Distance Threshold= 0	Distance Threshold= 0.1	Distance Threshold= 0.3	Distance Threshold= 0.5
FIFO	3456400	29052	3455933	3445072
FIFO2ndChance	3424230	3386093	3424230	3424270
LRU	3469912	3469912	3469404	3469871
SLRU	3082611	3073706	3082620	3082503
Lfu	3082611	29052	29059	29052
LfuDA	3082611	3369919	3363075	3369851

Table 4: Runtimes (seconds) for different cache sizes(dataset size = 10^5 resources)

CacheType	Cache Size= 10^1	Cache Size= 10^2	Cache Size= 10^3	Cache Size= 10^4	Cache Size= 10^5
FIFO	721.7	678.8	669.9	610.8	618.4
LRU	695.7	708.9	700.8	653.6	704.3
SLRU	714	907.4	658.7	694.2	700.6

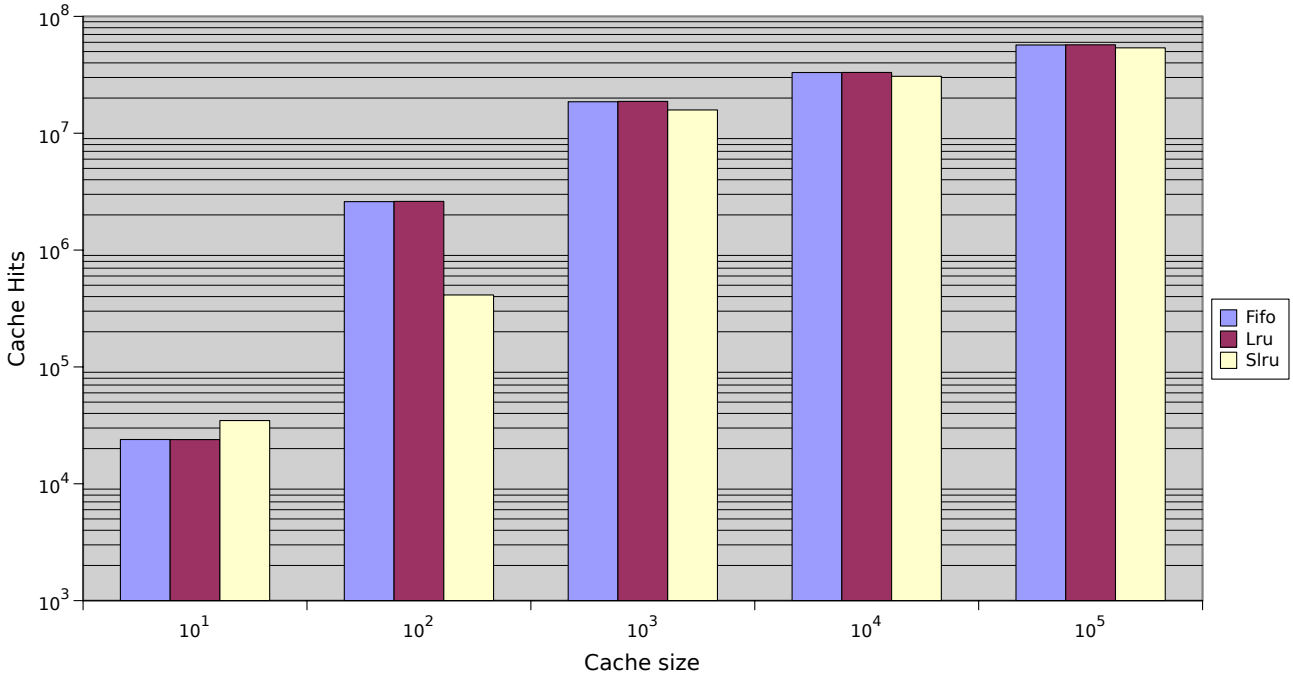


Figure 12: Cache hits for different cache sizes (dataset size = 10^5 resources)

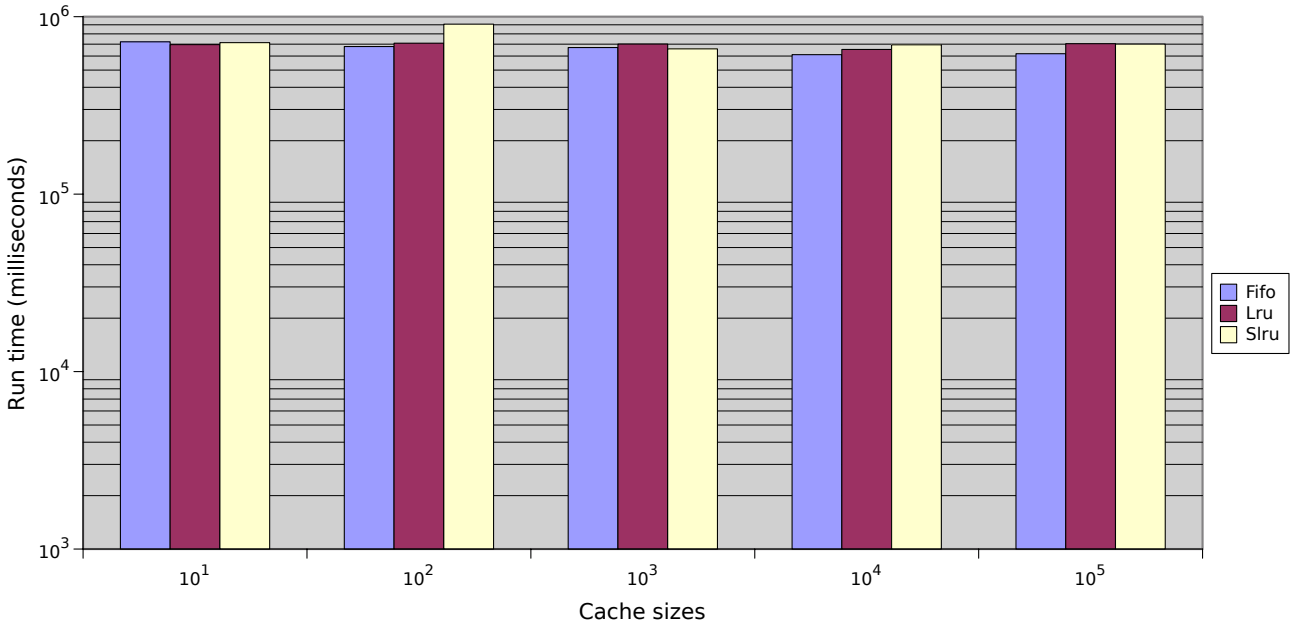


Figure 13: Runtimes for different cache sizes (dataset size = 10^5 resources)

Table 5: Hit rates of different caching approaches for different cache sizes(dataset size = 10^5 resources)

CacheType	Cache Size= 10^1	Cache Size= 10^2	Cache Size= 10^3	Cache Size= 10^4	Cache Size= 10^5
FIFO	23927	2597652	18594739	33080982	56912799
LRU	23891	2610343	18726491	33130161	57118089
SLRU	34756	412958	15822935	30696531	53798204

6 Parallel ORCHID

In the previous deliverable[12], we introduced five load balancing algorithms for parallel link discovery. The *naïve load balancer* divides all tasks between all processors based on their index and regardless of their complexities. The *greedy load balancer* sorts 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. The *pair-based load balancer* assigns processors tasks in pairs of the form (most complex, least complex). The *PSO load balancer* uses the particle swarm optimization technique to find the most homogeneous loads for processors. Finally, the *DPSO load balancer* implements a deterministic version of the PSO load balancer.

The aim of our evaluation was to measure whether the PSO and DPSO can improve the runtime taken by traditional load balancing approaches (i.e. naïve, greedy and pair-based). To this end, we measured the run time for each of the five load balancing algorithms for both synthetic and real data. In the following, we begin by presenting the data that we used. Thereafter, we present our results on these different datasets.

6.1 Experimental Setup

We performed controlled experiments on five synthetic geographic datasets[13] and the aforementioned three real datasets. The synthetic datasets is generated in increasing the number of polygons, starting from 1 million polygons up to 5 million polygons by step of 1 million. Based on a *Gaussian* random variable, we varied the synthetic dataset polygons' sizes from one to ten points. Also, the (latitude, longitude) coordinates of each point are generated akin with the *Gaussian* distribution.

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 8 quad-core *Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz* with 8192 KB cache. Unless stated otherwise, each experiment was assigned 20 GB of memory. Because of the random nature of the PSO approach we ran it 5 times in each experiment and provided the mean of the five runs results. The approaches presented herein were implemented in the LIMES framework.[14] All results can are available at the project web site.[15]

6.2 ORCHID vs. Parallel ORCHID

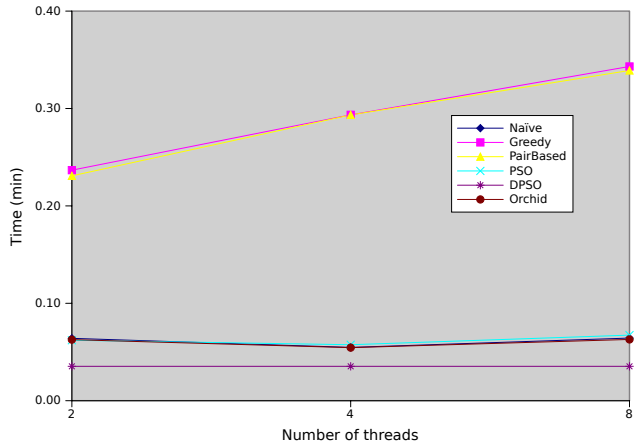
We evaluated the *speed up* gained by using parallel implementations of ORCHID algorithm. We ran a set of experiments on the three selected real datasets (Nuts, DBpedia and LinkedGeoData). In the first set of experiments, we ran one experiment for each of the three datasets at hand using the normal implementation of ORCHID [5]. In the rest of the experiments, for each dataset, we evaluated the parallel implementations of ORCHID using the aforementioned five load balanced approaches. To evaluate the *speed up* gained from increasing the number of parallel processing units, we reran each of the parallel experiments with 2, 4 and 8 of threads. Figure 14 shows the runtime results along with the mean squared error (MSE) results of the experiments.

We performed this set of experiments with two goals in mind: First, we wanted to measure the run time taken by each algorithm when applied to different datasets. Our second aim was to qualify the quality of the data distribution carried out by each of the implemented algorithm using MSE. To this end, we ran two sets of experiments. In the first set of experiments, we used the aforementioned three datasets of *Nuts*, *DBpedia*

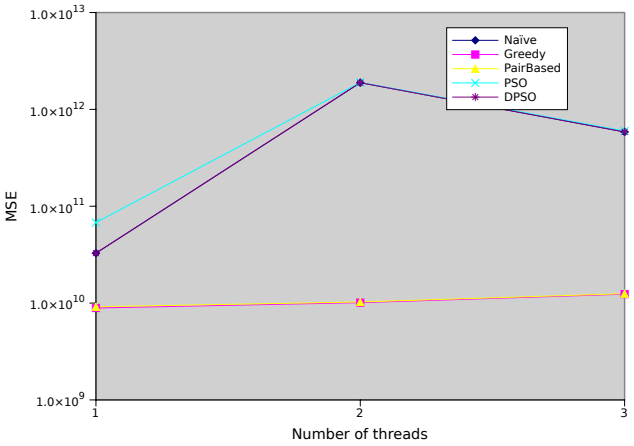
[12] Deliverable 3.1.2 "System to Integrate Large-Scale Data Sets Based on Implicit Spatial Relations"
[13] All synthetic dataset are available at https://github.com/AKSW/LIMES/tree/master/evaluationsResults/lb4ld
[14] http://limes.sf.net
[15] https://github.com/AKSW/LIMES/tree/master/evaluationsResults/lb4ld

and *LinkedGeoData*. The result of this set of experiments are presented in Figure 14. In the second set of experiments, we ran our five load balancing algorithms against a set of five synthetic randomly generated datasets (see subsection 6.1 for details). The results are presented in Figure 15.

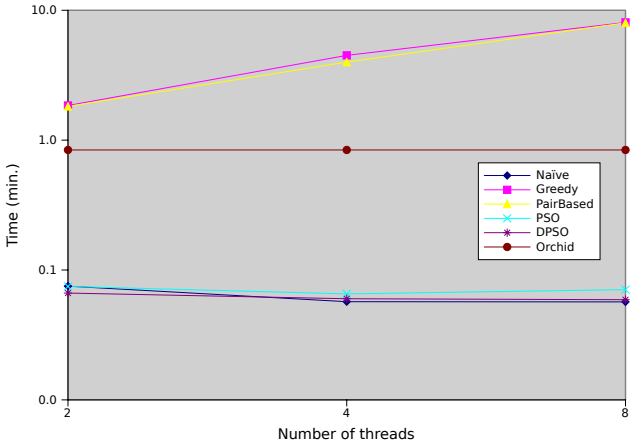
Our results suggest that DPSO and PSO outperform the naive approach in most cases. This can be seen most clearly in Figure 15 (note the log scale). DPSO is to be preferred over PSO as it is deterministic and is thus the default implementation of load balancing currently implemented in LIMES. Still, the improvements suggest that preserving the integrity of the hypercubes generated by ORCHID still leads to a high difference in load across the processors as shown by our MSE results. An interesting research avenue would thus be to study approaches which do not preserve this integrity while guaranteeing result completeness. This will be the core of our future work.



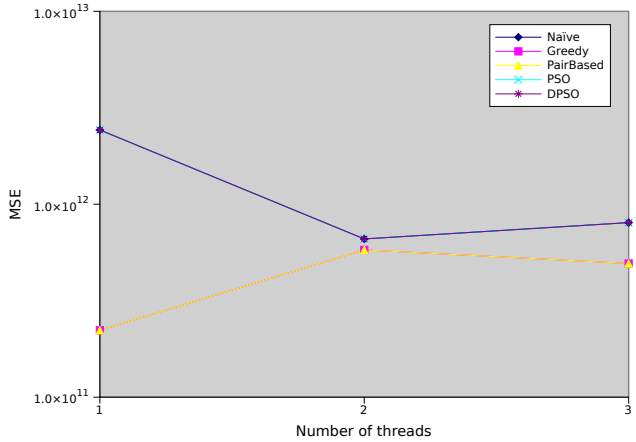
(a) Nuts runtime



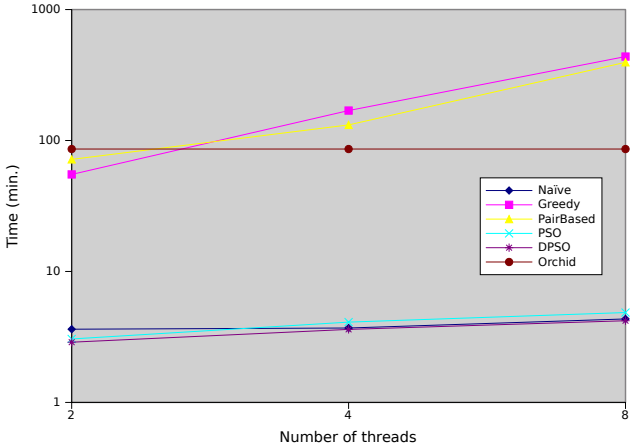
(b) Nuts MSE



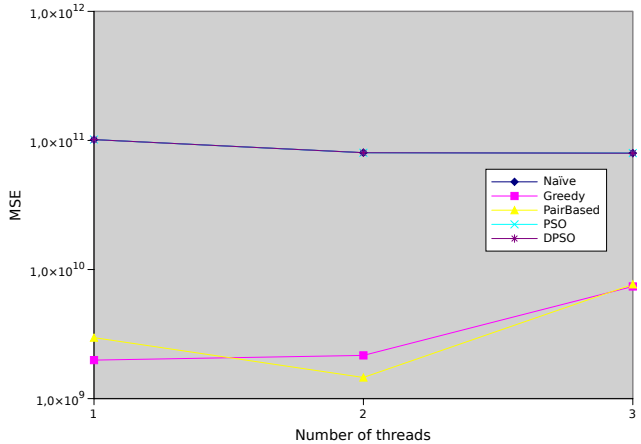
(c) DBpedia runtime



(d) DBpedia MSE

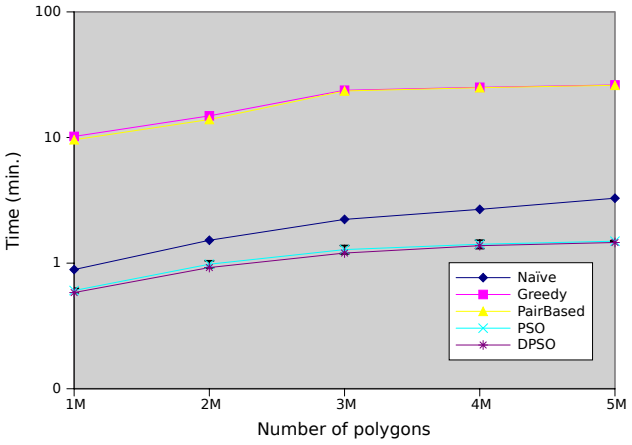


(e) LinkedGeoData runtime

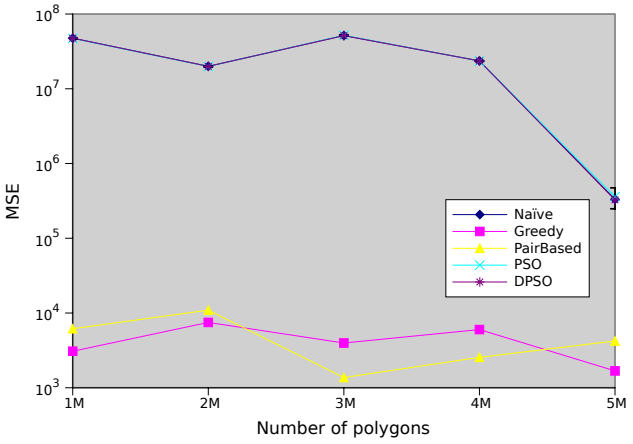


(f) LinkedGeoData MSE

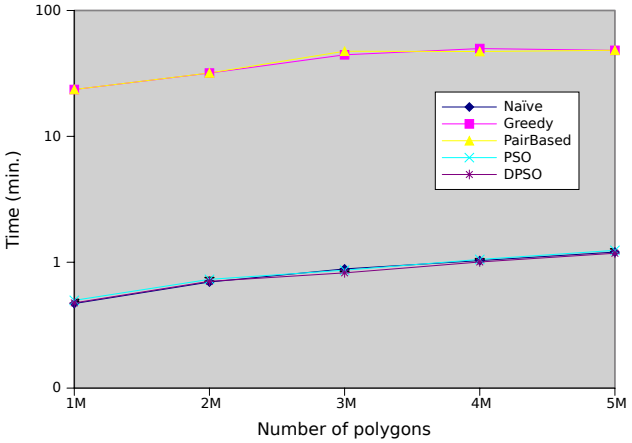
Figure 14: Runtime and MSE generated when applying Orchid [5] vs. parallel implementations of Orchid using naïve, greedy, pair based, PSO and DPSO load balancing algorithms against the three real datasets of *Nuts*, *DBpedia* and *LinkedGeoData* using 2, 4 and 8 threads



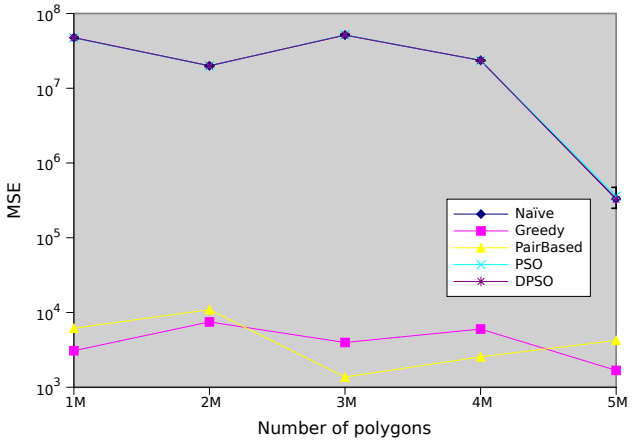
(a) 2 threads runtime



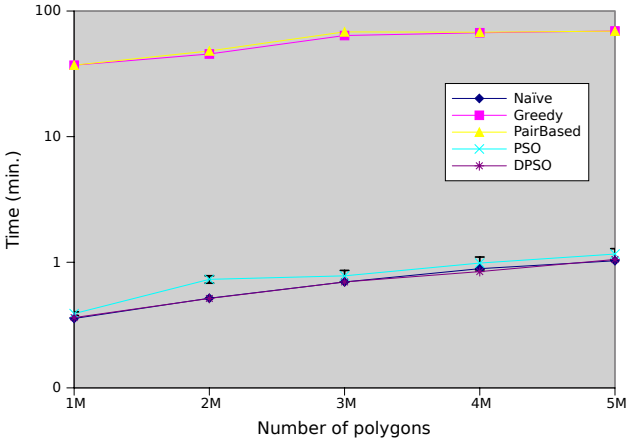
(b) 2 threads MSE



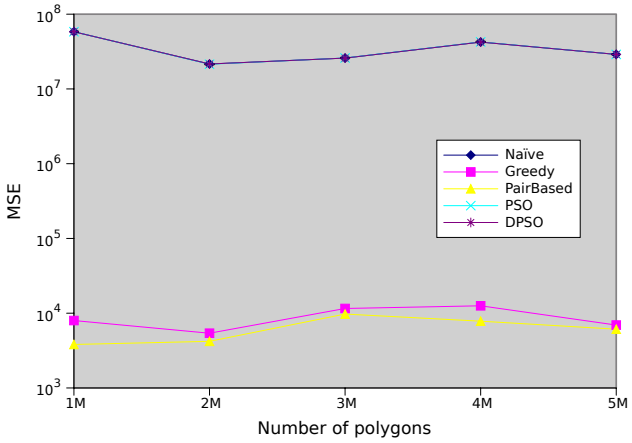
(c) 4 threads runtime



(d) 4 threads MSE



(e) 8 threads runtime



(f) 8 threads MSE

Figure 15: Runtime and MSE generated when applying parallel implementations of Orchid using naïve, greedy, pair based, PSO and DPSO load balancing algorithms against the five synthetic datasets of sizes 1, 2 , ..., 5 million polygons using 2, 4 and 8 threads

7 Hardware Assessment

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 $\mathcal{HR}^3$ presented in the previous deliverable[16] 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.

Dataset	Source	Size	Features
DS_1	DBPedia	25,781	min/medium/max elevation
DS_2	DBPedia	475,000	latitude, longitude
DS_3	Linked Geo Data	500,000	latitude, longitude
DS_4	Linked Geo Data	6,000,000	latitude, longitude

Figure 16: Datasets used for evaluation

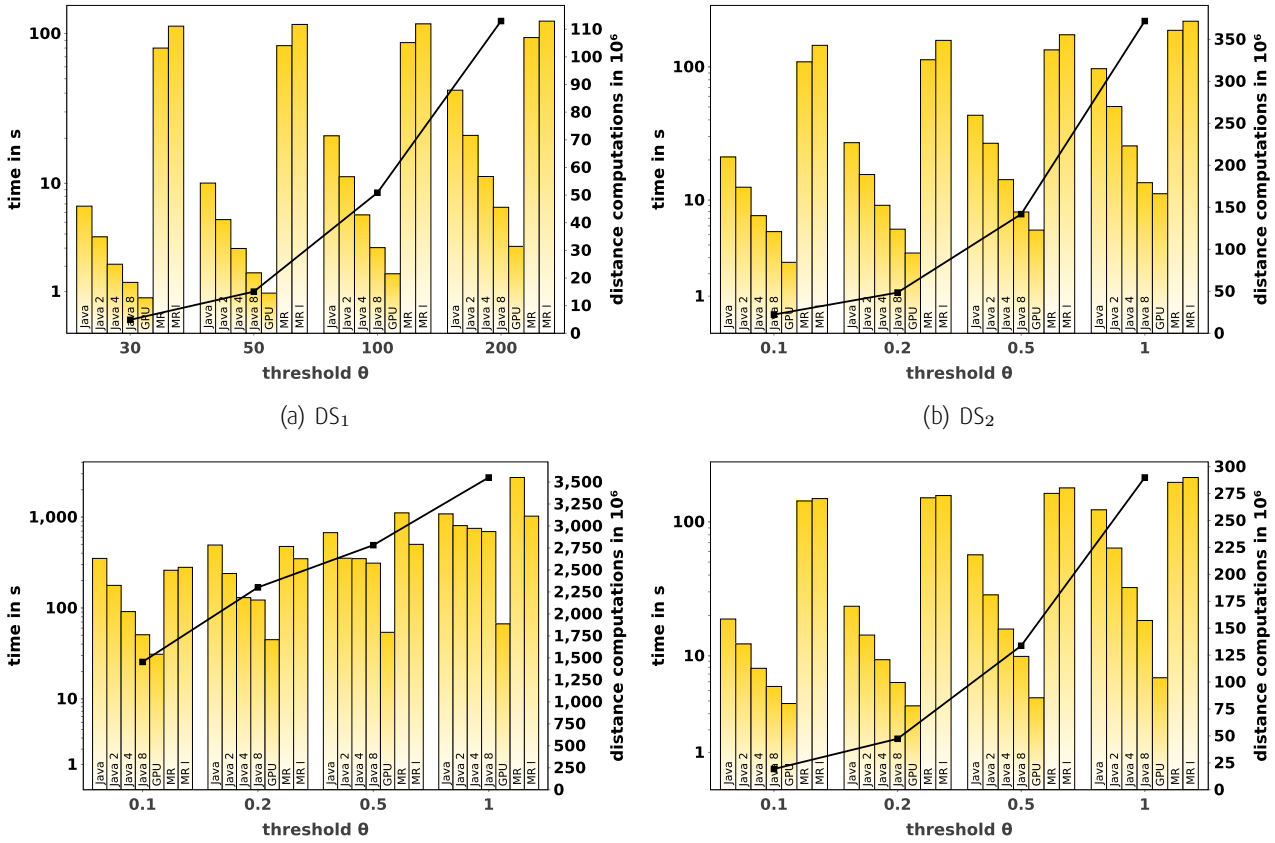
7.1 Experimental Setup

We utilized the four datasets of different sizes shown in Figure 16. 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 JDK 1.7 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* [4] 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).

7.2 Performance Comparison

The results of our performance comparison are shown in Figure 17. While the parallel implementation of $\mathcal{HR}^3$ 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 $1°$. This impressively demonstrates the need for load balancing in all parallel link discovery tasks on skewed data. This need is further supported by the results achieved by MR and MR_l on DS_3. Here, MR_l clearly outperforms MR

[16] Deliverable 3.1.2 "System to Integrate Large-Scale Data Sets Based on Implicit Spatial Relations"



(a) DS_1 (b) DS_2

(c) DS_3 (d) DS_2–DS_3

Figure 17: Comparison of runtimes for Experiment 1

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 suggests 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, $\theta = 1°$) 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.

7.3 Scalability

The strengths of the cloud are revealed in the second series of experiments we performed (see Figure 18). 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.

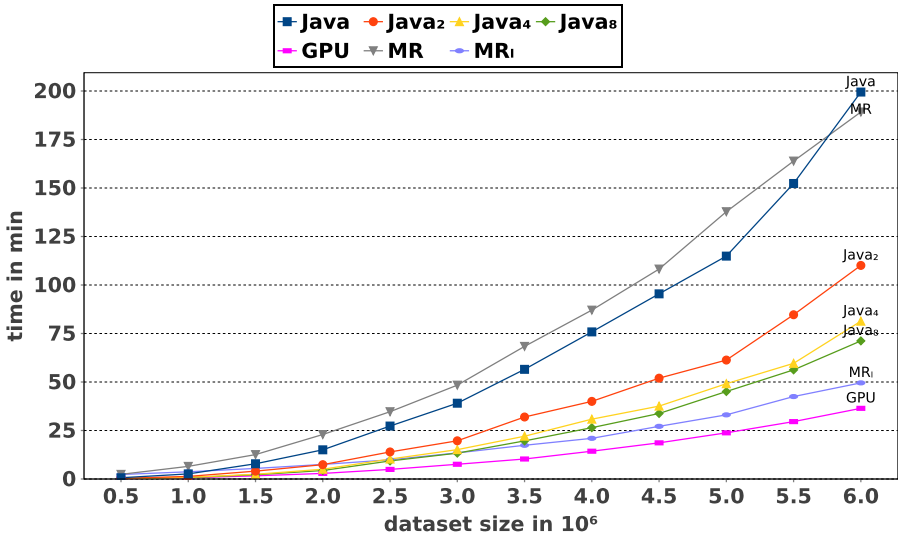


Figure 18: Comparison of runtimes on DS4

8 Summary

In this deliverable, we presented a thorough evaluation of geo-spatial link discovery pertaining to diverse aspects such as runtime, hardware configuration and caching. Our results suggest that we were able to push the state of the art in geo-spatial link discovery significantly with the developments carried in this project. With ORCHID, we create the (to the best of our knowledge) only reduction-ratio-optimal link discovery algorithm specialised on orthodromic spaces, to which geo-spatial data belong. We showed that it outperforms the state of the art and used it as basis for our further experiments. Our results pertaining to hardware show that local hardware is to be preferred when dealing with geo-spatial data. Our experiments of caching show that this alley is still open for research as dedicated approaches for caching could significantly improve the runtime of link discovery approaches for large amounts of geo-spatial data. Finally, our results on point-set measures show that using the mean measure does lead to the best results in most cases, thus allowing us to scale even better due to its linear complexity. In future works, we will explore caching as well load-balancing further as new algorithms for these NP-complete problems have the potential to lead to new algorithms in considerable number of disciplines.

References

[1] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for web proxy caches. *SIGMETRICS Performance Evaluation Review*, 27(4):3–11, 2000.

[2] A. Ferrara, S. Montanelli, J. Noessner, and H. Stuckenschmidt. Benchmarking Matching Applications on the Semantic Web. In *The Semantic Web: Research and Applications*, 2011.

[3] Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.

[4] Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient Deduplication with Hadoop. *PVLDB*, 5(12):1878–1881, 2012.

[5] Axel-Cyrille Ngonga Ngomo. Orchid - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *Proceedings of ISWC 2013*, 2013.

[6] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, December 2003.

[7] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating systems - design and implementation (3. ed.)*. Pearson Education, 2006.