An Efficient Approach for the Generation of Allen Relations

Kleanthi Georgala[1] and **Mohamed Ahmed Sherif** [2] and **Axel-Cyrille Ngonga Ngomo**[3]

Abstract. Event data is increasingly being represented according to the Linked Data principles. The need for large-scale machine learning on data represented in this format has thus led to the need for efficient approaches to compute RDF links between resources based on their temporal properties. Time-efficient approaches for computing links between RDF resources have been developed over the last years. However, dedicated approaches for linking resources based on temporal relations have been paid little attention to. In this paper, we address this research gap by presenting AEGLE, a novel approach for the efficient computation of links between events according to Allen's interval algebra. We study Allen's relations and show that we can reduce all thirteen relations to eight simpler relations. We then present an efficient algorithm with a complexity of $O(n \log n)$ for computing these eight relations. Our evaluation of the runtime of our algorithms shows that we outperform the state of the art by up to 4 orders of magnitude while maintaining a precision and a recall of 1.

1 INTRODUCTION

Over the past years, technological progress in hardware development and network infrastructures have led to the collection of large amounts of event data in scenarios as diverse as monitoring industrial plants [12], monitoring open SPARQL endpoints [22], implementing the Internet of Things (IoT) and Cloud Computing [13]. For example, the LSQ dataset [22] consists of more than 1.2 billion facts which describe more than 250 million query events on open SPARQL endpoints. The availability of such large collections of event data in Resource Description Framework (RDF)[4] format as well as the uptake of semantic technologies (in particular RDF) to represent machine events [20] has consequently led to the need for interlinking these events, especially to support structured machine learning [11] (e.g., predictive maintenance for machine data or discovering sequences of query patterns that a triple store is often faced with) over these datasets.

Given that the computation of links is the fourth principle of Linked Data,[5] a large number of frameworks have been developed to facilitate the computation of links between knowledge bases (see [14] for a survey). Still, to the best of our knowledge, only one approach has been developed for computing temporal links between events [23]. The approach presented in [23] is based on the Multi-Block algorithm [8] and employs multi-dimensional blocking to reduce the number of comparisons necessary to compute temporal relations. However, our evaluation suggests that this approach does not scale to larger number of events.

In this paper, we hence address the problem of computing temporal relations between events efficiently. To this end, we rely on Allen's Interval Algebra [1] as it encompasses all primitive temporal relations betwen events. Our approach, dubbed AEGLE (Allen's intErval alGebra for Link discovEry), relies on two insights: First, the 13 Allen relations can be reduced to 8 simpler relations that all compare exactly either the beginning or the end of an event with the beginning or end of another event. The second insight behind our approach is that given that time is ordered, we can reduce the problem of detecting such relations to the problem of finding matching entities in two sorted lists. As this problem has a complexity of $O(n \log n)$, our approach should scale well even for larger datasets. Importantly, our method achieves 100% precision and recall as it computes all temporal relations between events from a source set S and a target set T. The main contributions of our work are thus as follows:

- We show how the 13 Allen relations can be reduced to 8 atomic relations and how these 8 relations can be combined using set theory to reconstruct the 13 Allen relations.
- We provide an efficient approach to computing each of the 8 atomic relations aforementioned.
- We evaluate the runtime of our approach using real and synthetic data and show that we outperform the state of the art by up to 4 orders of magnitude.

The rest of this paper is organised as follows: Section 2 includes the basic notation and preliminaries behind link discovery (LD)[6] and Allen's Interval Algebra. Section 3 describes our approach by (1) defining the set of atomic relations we use to compute temporal relations and(2) showing how to derive more complex relations from the set of atomic relations derived previously. In Section 4, we present a systematic comparison of our approach with the state of the art. Finally, we give an overview of the existing related work and conclude with a brief summary of our work and future plans.

2 PRELIMINARIES

In this section, we present the concepts and notation that are necessary to understand the rest of the paper. First, we introduce the LD problem by providing a formal definition akin to that introduced in [16]. Thereafter, we provide the notation for the basic relations between intervals as introduced in [1].

[1] AKSW Research Group, University of Leipzig, Germany, email: georgala@informatik.uni-leipzig.de
[2] AKSW Research Group, University of Leipzig, Germany, email: sherif@informatik.uni-leipzig.de
[3] AKSW Research Group, University of Leipzig, Germany, email: ngonga@informatik.uni-leipzig.de
[4] `https://www.w3.org/RDF/`
[5] http://www.w3.org/DesignIssues/LinkedData.html

[6] We use the term "link discovery" to signify the computation of links of particular types between pairs of resources represented in RDF. Never do we use this term in the sense of mining links between nodes in a graph.

2.1 Link Discovery

Throughout this paper, we deal with facts represented in RDF. Each fact is a triple $(c, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$, where

1. c is the subject of the triple (i.e., what the triple is mainly about),
2. p is the predicate of the triple (i.e., the relation that the subject has with the object),
3. o is the object of the triple (that which is to be related to the subject through the predicate),
4. $\mathcal{R}$ is the set of all RDF resources, where each resource stands for a thing from the real world, e.g., an event,
5. $\mathcal{P} \subseteq \mathcal{R}$ is the set of all RDF properties, which are binary predicates,
6. $\mathcal{B}$ is the set of all RDF blank nodes, which basically model existential semantics and
7. $\mathcal{L}$ is the set of all literals, i.e., of all data types (e.g., time points).

An example of a fact would be (`:E1`, `:begin`, 0.1), which states that the event `:E1` begins at the point 0.1 in time.

We call a set of triples a knowledge base (KB). Given two sets of resources S and T from two (not necessarily distinct) KBs as well as a binary relation R, the main goal of LD is to discover the set $M = \{(s, t) \in S \times T : R(s, t)\}$. We call M a mapping. Naive approaches towards this goal are quadratic in complexity as they have to compare every $s \in S$ with every $t \in T$, which is clearly impracticable for large S and T. In this work, we thus consider the efficient computations of temporal relations between events. Hence, we assume that each of the resources in S and T considered in the subsequent portion of this paper describes an event v with a beginning time denoted $b(v)$ and an end time denoted $e(v)$. Note that we assume that $b(v) < e(v)$ throughout this work.

2.2 Allen's Interval Algebra

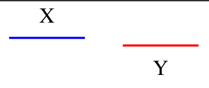
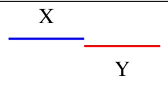
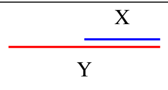
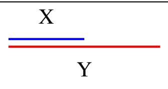
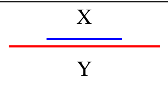
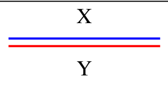
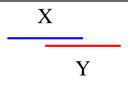
Allen's Interval Algebra [1] is a widely known time interval calculus, which provides a set of 13 "distinct, exhaustive, and qualitative" relations between time intervals [2]. Table 1 illustrates this set of relations and shows a set of six relations between two time intervals X and Y, their corresponding symbols along with the symbols of their inverse relation. The *equal* relation is symmetric.

2.3 Link Discovery between Events

An event can be modelled as a time interval because we assume that its description always includes a begin time property and an end time property, Thus, an event instance s can be described as pair of time points $(b(s), e(s))$, where $b(s) < e(s)$. Formally, computing the temporal relations between events can thus be reduced to computing the following mappings M:

- if $R = bf$, then $M = \{(s, t) \in S \times T : (b(s) < b(t)) \wedge (b(s) < e(t)) \wedge (e(s) < b(t)) \wedge (e(s) < e(t))\}$
- if $R = bfi$, then $M = \{(s, t) \in S \times T : (b(s) > b(t)) \wedge (b(s) > e(t)) \wedge (e(s) > b(t)) \wedge (e(s) > e(t))\}$
- if $R = m$, then $M = \{(s, t) \in S \times T : (b(s) < b(t)) \wedge (b(s) < e(t)) \wedge (e(s) = b(t)) \wedge (e(s) < e(t))\}$
- if $R = mi$, then $M = \{(s, t) \in S \times T : (b(s) > b(t)) \wedge (b(s) = e(t)) \wedge (e(s) > b(t)) \wedge (e(s) > e(t))\}$
- if $R = f$, then $M = \{(s, t) \in S \times T : (b(s) > b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) = e(t)\}$
- if $R = fi$, then $M = \{(s, t) \in S \times T : (b(s) < b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) = e(t)\}$

Table 1. Allen's Interval Algebra

Relation	Notation	Inverse	Illustration
X before Y	$bf(X, Y)$	$bfi(X, Y)$	
X meets Y	$m(X, Y)$	$mi(X, Y)$	
X finishes Y	$f(X, Y)$	$fi(X, Y)$	
X starts Y	$st(X, Y)$	$sti(X, Y)$	
X during Y	$d(X, Y)$	$di(X, Y)$	
X equal Y	$eq(X, Y)$	$eq(X, Y)$	
X overlaps with Y	$ov(X, Y)$	$ovi(X, Y)$	

- if $R = st$, then $M = \{(s, t) \in S \times T : (b(s) = b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) < e(t)\}$
- if $R = sti$, then $M = \{(s, t) \in S \times T : (b(s) = b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) > e(t)\}$
- if $R = d$, then $M = \{(s, t) \in S \times T : (b(s) > b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) < e(t)\}$
- if $R = di$, then $M = \{(s, t) \in S \times T : (b(s) < b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) > e(t)\}$
- if $R = eq$, then $M = \{(s, t) \in S \times T : (b(s) = b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) = e(t)\}$
- if $R = ov$, then $M = \{(s, t) \in S \times T : (b(s) < b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) < e(t)\}$
- if $R = ovi$, then $M = \{(s, t) \in S \times T : (b(s) > b(t)) \wedge (b(s) < e(t)) \wedge (e(s) > b(t)) \wedge e(s) > e(t)\}$

3 APPROACH

3.1 Overview

As we have now introduced the necessary notations and concepts behind LD and Allen's Interval Algebra, we can proceed to explain our approach for the rapid computation of temporal links between events in detail. The main goal of our approach is to compute all Allen's interval relations (as illustrated in Table 1) between two sets of atomic events efficiently. The main insight underlying this work is that we can reduce the computation of the 13 relations to the computation and combinations of a mere 8 atomic relations and thus reduce the overall computation time of Allen relations. We use this insight to devised means to compute all interval relations efficiently by reducing all of Allen's relations to re-usable atomic relations that can be computed efficiently. We then combine the results of these atomic relations to compute Allen's relations. While doing so, we ensure that we achieve 100% accuracy in retrieving all possible Allen relations between resources in the given sets of resources S and T.

3.2 AEGLE

The main idea behind our approach is to represent each relation of Table 1 as a Boolean combination of atomic relations. By computing each of the atomic relations only once and only if needed, we can decrease the overall runtime of the computation of a given set of Allen relations.

As described in Section 2.3, each atomic event s can be described using two time points $b(s)$ and $e(s)$. To compose the atomic interval relations, we define all possible binary relations between the begin and end points of two event resources $s = (b(s), e(s))$ and $t = (b(t), e(t))$ as follows:

- Atomic relations between $b(s)$ and $b(t)$:
  - $BB^1(s,t) \Leftrightarrow (b(s) < b(t))$
  - $BB^0(s,t) \Leftrightarrow (b(s) = b(t))$
  - $BB^{-1}(s,t) \Leftrightarrow (b(s) > b(t)) \Leftrightarrow \neg(BB^1(s,t) \vee BB^0(s,t))$
- Atomic relations between $b(s)$ and $e(t)$:
  - $BE^1(s,t) \Leftrightarrow (b(s) < e(t))$
  - $BE^0(s,t) \Leftrightarrow (b(s) = e(t))$
  - $BE^{-1}(s,t) \Leftrightarrow (b(s) > e(t)) \Leftrightarrow \neg(BE^1(s,t) \vee BE^0(s,t))$
- Atomic relations between $e(s)$ and $b(t)$:
  - $EB^1(s,t) \Leftrightarrow (e(s) < b(t))$
  - $EB^0(s,t) \Leftrightarrow (e(s) = b(t))$
  - $EB^{-1}(s,t) \Leftrightarrow (e(s) > b(t)) \Leftrightarrow \neg(EB^1(s,t) \vee EB^0(s,t))$
- Atomic relations between $e(s)$ and $e(t)$:
  - $EE^1(s,t) \Leftrightarrow (e(s) < e(t))$
  - $EE^0(s,t) \Leftrightarrow (e(s) = e(t))$
  - $EE^{-1}(s,t) \Leftrightarrow (e(s) > e(t)) \Leftrightarrow \neg(EE^1(s,t) \vee EE^0(s,t))$

Out of Table 1, we can derive how each of Allen's relations can be reduced to a Boolean combination of a subset of the relations above as follows:

- $bf(s,t) \Leftrightarrow BB^1(s,t) \wedge BE^1(s,t) \wedge EB^1(s,t) \wedge EE^1(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $e(s) < b(t) \Rightarrow b(s) < b(t)$,
  2. $e(s) < b(t) \Rightarrow b(s) < e(t)$ (by virtue of 1.) and
  3. $e(s) < b(t) \Rightarrow e(s) < e(t)$.

  Hence $bf(s,t) = EB^1(s,t)$.
- $bfi(s,t) \Leftrightarrow BB^{-1}(s,t) \wedge BE^{-1}(s,t) \wedge EB^{-1}(s,t) \wedge EE^{-1}(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) > e(t) \Rightarrow b(s) > b(t)$,
  2. $b(s) > e(t) \Rightarrow e(s) > b(t)$ (by virtue of 1.) and
  3. $b(s) > e(t) \Rightarrow e(s) > e(t)$.

  Hence $bfi(s,t) = BE^{-1} = \neg(BE^1(s,t) \vee BE^0(s,t))$.
- $m(s,t) \Leftrightarrow BB^1(s,t) \wedge BE^1(s,t) \wedge EB^0(s,t) \wedge EE^1(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $e(s) = b(t) \Rightarrow b(s) < b(t)$,
  2. $e(s) = b(t) \Rightarrow e(s) < e(t)$ and
  3. $e(s) = b(t) \Rightarrow b(s) < e(t)$ (by virtue of 1.).

  Hence $m(s,t) = EB^0(s,t)$.
- $mi(s,t) \Leftrightarrow BB^{-1}(s,t) \wedge BE^0(s,t) \wedge EB^{-1}(s,t) \wedge EE^{-1}(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) = e(t) \Rightarrow b(s) > b(t)$,
  2. $b(s) = e(t) \Rightarrow e(s) > e(t)$ and
  3. $b(s) = e(t) \Rightarrow e(s) > b(t)$ (by virtue of 1.).

  Hence $mi(s,t) = BE^0(s,t)$.
- $f(s,t) \Leftrightarrow BB^{-1}(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^0(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) > b(t) \Rightarrow e(s) > b(t)$ and
  2. $e(s) = e(t) \Rightarrow b(s) < e(t)$

  Hence $f(s,t) = \{EE^0(s,t) \wedge BB^{-1}(s,t)\} = \{EE^0(s,t) \wedge \neg(BB^0(s,t) \vee BB^1(s,t))\}$.
- $fi(s,t) \Leftrightarrow BB^1(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^0(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) < b(t) \Rightarrow b(s) < e(t)$ and
  2. $e(s) = e(t) \Rightarrow e(s) > b(t)$

  Hence $fi(s,t) = \{BB^1(s,t) \wedge EE^0(s,t)\}$.
- $st(s,t) \Leftrightarrow BB^0(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^1(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) = b(t) \Rightarrow b(s) < e(t)$ and
  2. $e(s) < e(t) \Rightarrow e(s) > b(t)$

  Hence $st(s,t) = \{BB^0(s,t) \wedge EE^1(s,t)\}$.
- $sti(s,t) \Leftrightarrow BB^0(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^{-1}(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) = b(t) \Rightarrow e(s) > b(t)$ and
  2. $b(s) = b(t) \Rightarrow b(s) < e(t)$

  Hence $sti(s,t) = \{BB^0(s,t) \wedge EE^{-1}(s,t)\} = \{BB^0(s,t) \wedge \neg(EE^0(s,t) \vee EE^1(s,t))\}$.
- $d(s,t) \Leftrightarrow BB^{-1}(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^1(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $b(s) > b(t) \Rightarrow e(s) > b(t)$ and
  2. $e(s) < e(t) \Rightarrow b(s) < e(t)$

  Hence $d(s,t) = \{EE^1(s,t) \wedge BB^{-1}(s,t)\} = \{EE^1(s,t) \wedge \neg(BB^0(s,t) \vee BB^1(s,t))\}$.
- $di(s,t) \Leftrightarrow BB^1(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^{-1}(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get
  1. $e(s) > e(t) \Rightarrow e(s) > b(t)$ and
  2. $b(s) < b(t) \Rightarrow b(s) < e(t)$

  Hence $di(s,t) = \{BB^1(s,t) \wedge EE^{-1}(s,t)\} = \{BB^1(s,t) \wedge \neg(EE^0(s,t) \vee EE^1(s,t))\}$.
- $eq(s,t) \Leftrightarrow BB^0(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^0(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get

1. $e(s) = e(t) \Rightarrow e(s) > b(t)$ and
2. $b(s) = b(t) \Rightarrow b(s) < e(t)$

Hence $eq(s,t) = \{BB^0(s,t) \wedge EE^0(s,t)\}$.

- $ov(s,t) \Leftrightarrow BB^1(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^1(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get

  1. $b(s) < b(t) \Rightarrow b(s) < e(t)$.

  Hence $ov(s,t) = \{BB^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^1(s,t)\} = \{(BB^1(s,t) \wedge EE^1(s,t)) \wedge \neg(EB^0(s,t) \vee EB^1(s,t))\}$.
- $ovi(s,t) \Leftrightarrow BB^{-1}(s,t) \wedge BE^1(s,t) \wedge EB^{-1}(s,t) \wedge EE^{-1}(s,t)$. Now given that $b(s) < e(s)$ and $b(t) < e(t)$ and by virtue of the transitivity of $<$, we get

  1. $e(s) > e(t) \Rightarrow e(s) > b(t)$.

  Hence $ovi(s,t) = \{BB^{-1}(s,t) \wedge BE^1(s,t) \wedge EE^{-1}(s,t)\} = \{(BE^1(s,t) \wedge \neg(BB^0(s,t) \vee BB^1(s,t))) \wedge \neg(EE^0(s,t) \vee EE^1(s,t))\}$.

Clearly, we hence only need to compute the 8 atomic relations $EB^0, EB^1, EE^0, EE^1, BB^0, BB^1, BE^0$ and BE^1 to be able to generate all of Allen's relations. In the following, we explicate our approach to computing these 8 relations efficiently.

3.3 Algorithm

Algorithm 1: AEGLE

```
Input: source $S$, target $T$, set of Allen relations $\mathcal{AIR}$
Output: Set of mappings $\mathcal{M}$
1  $\mathcal{M} \longleftarrow \emptyset$
2  $\mathcal{A} \longleftarrow \emptyset$
3  foreach $rel \in \mathcal{AIR}$ do
4      $requiredRelations \longleftarrow getAtmRelations(rel)$
5      $atomics(rel) \longleftarrow \emptyset$
6      foreach $atomicRel \in requiredRelations$ do
7          if $\mathcal{A}$ does not contain $atomicRel$ then
8              $a \longleftarrow computeAtmRelation(atomicRel, S, T)$
9              $\mathcal{A}.put(atomicRel, a)$
10         $atomics(rel).put(atomicRel, \mathcal{A}.get(atomicRel))$
11     $M \longleftarrow computeRelation(atomics(rel))$
12     $\mathcal{M}.add(M)$
13 Return $\mathcal{M}$
```

Given a set $\mathcal{AIR}$ of Allen relations that are to be computed, the basic idea behind our approach is to begin by detecting the subset of the 8 atomic relations that needs to be computed and to compute each of these relations exactly once. Algorithm 1 describes how the idea was implemented. Our approach, AEGLE, takes two sets of events, S and T, and the set $\mathcal{AIR}$ as input. The algorithm returns a set of mappings $\mathcal{M}$, of which each corresponds to exactly one of the relations in $\mathcal{AIR}$.

We begin by initialising the final set of mappings $\mathcal{M}$ in line 1 and the map $\mathcal{A}$ in line 2. $\mathcal{A}$ includes the labels of atomic relations as keys and their corresponding mapping as values. During the first step of our algorithm, for each $rel \in \mathcal{AIR}$, AEGLE retrieves the set of required atomic relations in line 4 by calling the function $getAtmRelations(rel)$. This function is responsible for retrieving the set of the labels of the atomic relations that are required to compute rel based on the rules defined in Section 3.2. For each required $atomicRel$ of the current rel, the algorithm checks if the mapping is already computed (line 7). If not, it invokes the function *computeAtmRelation* to compute the appropriate atomic relations in line 8 and places the resulting mapping along with the atomic relation label in $\mathcal{A}$. Then, it retrieves the mapping from $\mathcal{A}$ and places it in the $atomics(rel)$ map needed to compute the mapping of rel. Each Allen's relation described in $\mathcal{AIR}$ constructs its own $atomics(rel)$ map that has the labels of the requisite atomic relations as keys and their corresponding mappings as values. Finally, the algorithm computes the mapping M of rel by calling the function *computeRelation* (line 11) and adds the resulting set of links in $\mathcal{M}$ (line 12).

The time-critical portion of the execution lies in the computation of the atomic relations. The idea underlying our approach to computing these relations is that one can reduce their computation to the problem of finding pairs of matching elements in two sorted lists. For example, to compute BB^0, one needs to (1) sort the list of elements of S and T according to the time at which they began (guaranteed time complexity: $O(|S|\log|S|)$ resp. $O(|T|\log|T|)$), (2) search for the elements of the smaller set in the larger set ($O(\min(|S|,|T|)\log(\max(|S|,|T|)))$. This leads to an overall complexity of $O(n\log n)$. The complexity is the same for the computation of all relations.

Algorithm 2: $computeAtmRelations(atomicRel, S, T)$ for $atomicRel = BB^0$

```
Output: mapping of $BB^0$ $AM$
1  $AM \longleftarrow \emptyset$
2  $sources \longleftarrow orderByDate(S,$ beginDate$)$
3  $targets \longleftarrow orderByDate(T,$ beginDate$)$
4  $AM \longleftarrow mapEvents(sources, targets,$ concurrent$)$
5  Return $\mathcal{AM}$
```

Algorithm 3: $computeAtmRelations(atomicRel, S, T)$ for $atomicRel = EE^1$

```
Output: mapping of $EE^1$ $AM$
1  $AM \longleftarrow \emptyset$
2  $sources \longleftarrow orderByDate(S,$ endDate$)$
3  $targets \longleftarrow orderByDate(T,$ endDate$)$
4  $AM \longleftarrow mapEvents(sources, targets,$ predecessor$)$
5  Return $\mathcal{AM}$
```

Algorithm 4: $orderByDate(S, dateType)$

```
Output: $O$
1  foreach $s \in S$ do
2      $timeStamp \longleftarrow s.getDate(dateType)$
3      $tempO \longleftarrow \emptyset$
4      if $O$ contains $timeStamp$ then
5          $tempO \longleftarrow O.get(timeStamp)$
6      $tempO \longleftarrow tempO \cup s$
7      $O.put(timeStamp, tempO)$
8  Return $O$
```

To illustrate the main procedure of Algorithm 1 (lines 3- 12), consider $\mathcal{AIR} = \{st, sti\}$ as example. The other relations are com-

Algorithm 5: $mapEvents(sources, targets, eventType)$

```
Output: mapped events $Events$
1  $Events \longleftarrow \emptyset$
2  foreach $sourceTimeStamp \in sources$ do
3      if $eventType == concurrent$ then
4          $tempT \longleftarrow targets.get(sourceTimeStamp)$
5      else
6          $tempT \longleftarrow targets.getHigher(sourceTimeStamp)$
7      if $tempT != \emptyset$ then
8          foreach $s \in sources.get(sourceTimeStamp)$ do
9              $Events.put(s, tempT)$
10 Return $Events$
```

Algorithm 6: $computeRelation(atomics)$ for st

```
Output: mapping $M$
1  $M \longleftarrow \emptyset$
2  foreach $s \in atomics.get(BB^0)$ do
3      $M1 \longleftarrow atomics.get(BB^0).get(s)$
4      $tempEE1 \longleftarrow atomics.get(EE^1)$
5      if $tempEE1$ contains $s$ then
6          $M2 \longleftarrow tempEE1.get(s)$
7          $M.put(s, M1 \cap M2)$
8  Return $M$
```

puted analogously. In line 8 of Algorithm 1, AEGLE calls *computeAtmRelations* in order to generate the mappings for the required atomic relations for rel, where $rel = st$ and $requiredRelations = BB^0, EE^1$. Since $\mathcal{A}$ is empty and the condition in line 7 holds, Algorithm 1 will call the function *computeAtmRelation* for BB^0 and then for EE^1.

For BB^0, Algorithm 2 describes the necessary steps to compute the mapping of BB^0. To begin with, Algorithm 2 invokes the function $orderByDate$ for the source S and the target T datasets, to order both complex event resources using the property *beginDate*. Algorithm 4 illustrates the procedure of ordering a complex event S given the value of a property $dateType$, in this case *beginDate*. The main idea of this function is to assert each atomic event $s \in S$ to the appropriate time-bucket, given its $dateType$ value. $orderByDate$ returns a map that has the unique $dateType$ values of the input KB S as keys and the set of events that correspond to each $dateType$ as values. Once $sources$ and $targets$ are retrieved (lines 2, 3 resp. of Algorithm 2), *computeAtmRelations* calls the function *mapEvents* using the label $concurrent$, that is responsible for matching each source event s with the set of target events with the same $b(s)$. In the function *mapEvents* (Algorithm 5), for each source event s that belongs to a time-bucket with time-stamp $sourceTimeStamp$, the algorithm retrieves the appropriate subset of target events that have the same time-stamp (line 4), if any (line 7). Then, it constructs a mapping between each s and the matching set of target events (line 9). Finally, the mapping is returned to Algorithm 1 and it is placed in $\mathcal{A}$ in line 9.

To continue, Algorithm 1 calls again *computeAtmRelations* since the mapping of EE^1 is not contained as well in $\mathcal{A}$, following the procedure described in Algorithm 3. For EE^1, *computeAtmRelations* is going to order S and T by invoking the *orderByDate* function that is going to order the event sources using the *endDate* property. Once both $sources$ and $targets$ are retrieved (lines 2, 3 resp. of Algorithm 3), *mapEvents* will be called with $eventType = predecessor$, in order to match each source $s \in S$ with the target events that were terminated after the source event s ended (line 6 of Algorithm 5). Finally, the mapping is returned to the main algorithm and it is placed in $\mathcal{A}$ in line 9.

Once both mappings of BB^0 and EE^1 are retrieved and placed in $atomics(rel)$, Algorithm 1 calls *computeRelation* for st. Algorithm 6 illustrates the procedure of computing st. For each source event s, the algorithm retrieves the set of targets with the same $b(s)$ from the $atomics$ set (line 3). Then, Algorithm 6 checks if there exists a set of targets with *endDate* higher than $e(s)$ (line 5). If the condition holds, then *computeRelation* retrieves the aforementioned set of targets (line 6) and based on the equation in Section 3.2, it computes the intersection between the two sub-sets of targets. The procedure is performed for each source instance and the final mapping M is returned in Algorithm 1 and placed in $\mathcal{M}$.

Then, the AEGLE proceeds into computing the sti relation, following the steps described above. However, since $sti(s,t) = \{BB^0(s,t) \land \neg(EE^0(s,t) \lor EE^1(s,t))\}$, the algorithm will only have to compute EE^0 and retrieve the mappings for BB^0 and EE^1.

4 EVALUATION

The aim of our evaluation was to address the following questions:

- Q_1: Does the reduction of Allen relations to 8 atomic relations influence the overall runtime of the approach?
- Q_2: How does AEGLE perform when compared with the state of the art in terms of time efficiency?

To the best of our knowledge, only one other link discovery framework implements an approach for the discovery of temporal relations. In [23], the blocking approach underlying SILK was extended to deal with spatio-temporal data. We thus compared our approach with the SILK LD framework.

4.1 Experimental Setup

We evaluated our approach on two different sets of datasets (see Table 2 for their characteristics):

- The first set of datasets (*3KMachines, 30KMachines, 300KMachines*) was created by generating synthetic event data using information obtained from real logs generated by production machinery. To this end, we retrieved 30,000 events from production machines which covered a full day of event generation.[7] Then, we computed the probability that an event began or ended at any given point in time. Finally, we constructed our synthetic datasets by generating a fixed number of events that maintained the probability of an event beginning or ending at a particular point in time.
- The second set of datasets (*3KQueries, 30KQueries, 300KQueries*) was obtained by collecting real event data from query logs of triple stores exposed on the Web. The data was retrieved from the SPARQL endpoint of the *LSQ* project [22].[8] For each dataset, we performed a *SPARQL* query against the *LSQ* endpoint and obtained a set of events from a set of consecutive days.

[7] The source of the events cannot be disclosed due to legal reasons.
[8] More information can be found at `http://aksw.github.io/LSQ/`

As evaluation measure, we computed the *runtime* of each of the atomic relations, the *time* required by our implementation to perform the $computeRelation$ for each Allen Relation (Algorithm 6) and the *total runtime* required for computing all 13 relations. For SILK, we measured the time it required to compute each of the Allen relations.[9]

Table 2. Characteristics of data sets. Size stands for the number of events contained in the dataset.

Log Type	Dataset name	Size	Unique $b(s)$	Unique $e(s)$
	3KMachines	3,154	960	960
Machinery	*30KMachines*	28,869	960	960
	300KMachines	288,690	960	960
	3KQueries	3,888	3,636	3,638
Query	*30KQueries*	30,635	3,070	3,070
	300KQueries	303,991	184	184

We set the value of SILK's block size to 1 ms.[10] Each temporal relation implemented in SILK was given a maximum runtime of 6 hours. We will use the symbol *NA* to signify that a run did not terminate within 6 hours. For the sake of comparison, we also implemented a naive *baseline* for the eq relation. This naive implementation performs an exhaustive comparison of the events of S and T to compute eq. For each experiment, we linked each data source with itself, i.e., we set $S = T$. All experiments for all implementations were carried out on the same 20-core Linux Server running *OpenJDK* 64-Bit Server 1.8.0_74 on Ubuntu 14.04.4 LTS on Intel(R) Xeon(R) CPU E5-2650 v3 processors clocked at 2.30GHz. Each experiment was ran on exactly one core using 64 GB of RAM. We implemented AEGLE using Java 1.8.0_60 and the sorting algorithm described in $orderByDate$ (Algorithm 4) was performed using the *MergeSort* algorithm [9] as implemented in Java 1.8.0_60 with a guaranteed time complexity $O(n \log n)$.

4.2 Results

To address Q_1, we computed the execution runtime of all 8 atomic relations as described in Section 3.2. Table 4 shows the runtimes of the atomic relations as well as the total runtime required to run the full set of atomic relations. For our largest dataset *300KQueries*, our approach needs only 84.83 s to compute all atomic relations. The maximum required runtime is achieved on the *300KMachines* dataset, where our algorithm needs approximately 7 min. As expected, the atomic relations which rely on equality (i.e., BB^0, BE^0, EB^0, EE^0) require less time than the rest of the atomic relations.

Table 3. Total runtime of Allen Relation for all datasets for AEGLE and SILK. All runtimes are presented in seconds.

Log Type	Dataset Name	Total Runtime		
		AEGLE	AEGLE *	SILK
	3KMachines	11.26	5.51	294.00
Machine	*30KMachines*	1,016.21	437.79	29,846.00
	300KMachines	189,442.16	78,416.61	*NA*
	3KQueries	26.94	17.91	541.00
Query	*30KQueries*	988.78	463.27	33,502.00
	300KQueries	211,996.88	86,884.98	*NA*

[9] To measure this time, we contacted the author of [23], who informed us that measuring the duration of the "Match Task" was the way to measure the runtime of his approach.

[10] We contacted the authors of SILK's temporal relation extension and were informed that this setting should return the best results.

Another interesting observation derived from Table 4 is the relation between the size of the data, the number of the unique $b(s)$ and $e(s)$ among the event sources and the execution runtime of each relation. In the *Machines* datasets, the distribution of beginning and end times is equal among the different sizes of data. As expected by virtue of the complexity of our approach, the total runtimes grow in accordance with $O(n \log n)$ with the increase of the data. From *Query* datasets, we notice that the number of unique $b(s)$ and $e(s)$ has a significant impact on the runtime of our approach. For example, even though *300KQueries* includes 10 times more data than *30KQueries*, *30KQueries* has a significantly higher number of unique $b(s)$ and $e(s)$ than *300KQueries*. Hence, AEGLE requires 15 secs less for *300KQueries* than for the *30KQueries* dataset. The benefits of our implementation can be noticed clearly when comparing AEGLE with the *baseline* (see Table 6). For the eq relation, we see that AEGLE is 470 times faster than the brute-force approach. We can thus answer Q_1 by stating that (1) both the number of unique events and the distribution of events across time have a significant influence on the overall runtime and (2) our approach improves the overall runtime of the computation of Allen relations significantly.

Tables 3, 5 and 6 provide us with the insights necessary to answer Q_2. They show clearly that AEGLE outperforms SILK on all datasets in terms of time efficiency while achieving 100% precision and recall, i.e., while computing all the links that can be found. Therefore, our idea proves to be beneficial and time-efficient for the task of linking temporal data of various sizes.

In more detail, AEGLE requires $211,996.88\ s$ to run the complete computation of Allen relations on our largest dataset (*300KQueries*), whereas SILK is unable to produce full results for any of the relations withing the time frame of $280,800\ s$ (3.25 days). *30KQueries* is the largest dataset for which SILK was able to produce links for the given time limit. Here, we observe that AEGLE is more than 33 times faster than SILK. Furthermore, Table 5 suggests that the most costly operations are carried out for the inverse relations. However, by relying on the semantics of Allen relations, we can refrain from computing inverse relations and have them inferred by any forward or backward chaining system. The results under AEGLE* in Table 3 show that overall, the total runtime for computing the seven Allen relations bf, m, f, st, d, eq and ov amounts to less than half of AEGLE's runtime.

To conclude our answer for Q_2, we studied what would happen if we computed each of the Allen relation individually, i.e., we ran 13 experiments where we set $\mathcal{AIR}$ to contain exactly one of the Allen relations. We used this setting to allow for a fine-granular comparison of our runtimes with SILK's. The results of this experiment are shown in Table 6. Overall, we outperform SILK clearly even when computing each of the Allen relations on its own. This suggests that our core implementation for the computation of atomic relations is superior to the generic blocking scheme followed by SILK. This is especially clear when looking at the results on large datasets in more detail. For *30KQueries* for example, SILK needs 2,473 seconds while AEGLE only needs 0.45. The answer to Q_2 is hence that AEGLE outperforms the state of the art in all our experimental settings. Note that the total runtime of a relation is increased by the number of atomic relations involved in its computation when computed using AEGLE. As a result, AEGLE needs more time for the ovi relation (which is derived by combining 5 atomic relations) than for eq (2 atomic relations).

Table 4. Execution runtime of all 8 atomic relations for all datasets. All runtimes are presented in seconds.

Log Type	Dataset Name	BB^0	BB^1	BE^0	BE^1	EB^0	EB^1	EE^0	EE^1	Total runtime of atomic relations
Machine	*3KMachines*	0.02	0.41	0.02	0.41	0.02	0.41	0.02	0.42	1.73
	30KMachines	0.19	5.55	0.19	5.51	0.19	5.48	0.18	5.49	22.78
	300KMachines	2.70	95.55	2.14	92.26	3.39	115.66	2.13	94.4	408.23
Query	*3KQueries*	0.03	2.93	0.03	3.04	0.02	2.89	0.03	2.90	11.87
	30KQueries	0.19	24.5	0.19	26.28	0.21	23.85	0.19	23.80	99.22
	300KQueries	2.52	12.11	1.98	12.57	3.89	25.41	1.93	24.42	84.83

Table 5. Execution runtime of the 13 Allen Relations for all datasets for Aegle and Silk and *baseline*. The runtimes reported for Aegle are the times required to perform the set operations necessary to compute each relation. The overall runtimes (i.e., computation of required sets plus times for set operations) are presented in Table 6. All runtimes are presented in seconds.

			Machine			Query	
Relation	Approach	*3KMachines*	*30KMachines*	*300Machines*	*3KQueries*	*30KQueries*	*300KQueries*
bf	Aegle	0.00	0.00	0.05	0.00	0.00	0.03
	Silk	22.00	2,511.00	*NA*	43.00	2,794.00	*NA*
bfi	Aegle	1.52	127.37	27,103.19	2.37	127.37	32,023.10
	Silk	24.00	2,547.00	*NA*	42.00	2,961.00	*NA*
m	Aegle	0.00	0.00	0.03	0.00	0.00	0.00
	Silk	23.00	2,219.00	*NA*	41.00	2,466.00	*NA*
mi	Aegle	0.00	0.00	0.03	0.00	0.00	0.00
	Silk	23.00	2,290.00	*NA*	44.00	2,584.00	*NA*
f	Aegle	0.73	77.88	13,775.31	1.18	70.53	16,280.24
	Silk	23.00	2,306.00	*NA*	41.00	2,531.00	*NA*
fi	Aegle	0.42	47.07	7,837.04	0.62	40.04	8,600.89
	Silk	23.00	2,305.00	*NA*	43.00	2,535.00	*NA*
st	Aegle	0.21	29.48	4,849.29	0.34	22.70	5,796.87
	Silk	21.00	2,166.00	*NA*	40.00	2,613.00	*NA*
sti	Aegle	0.74	76.14	14,063.02	1.19	69.69	16,270.20
	Silk	21.00	2,226.00	*NA*	43.00	2,533.00	*NA*
d	Aegle	1.14	125.20	24,094.20	1.84	107.60	26,213.64
	Silk	24.00	2,363.00	*NA*	41.00	2,546.00	*NA*
di	Aegle	1.20	125.04	24,083.00	1.83	108.50	26,149.58
	Silk	23.00	2,293.00	*NA*	41.00	2,476.00	*NA*
eq	Aegle	0.01	0.40	45.01	0.00	0.06	344.10
	Silk	23.00	2,250.00	*NA*	41.00	2,473.00	*NA*
	baseline	2.05	171.10	23,436.30	3.15	196.09	31,452.54
ov	Aegle	1.70	182.04	35,244.48	2.68	163.16	38,165.31
	Silk	22.00	2,181.00	*NA*	39.00	2,487.00	*NA*
ovi	Aegle	1.87	202.80	37,939.27	3.02	179.90	42,068.13
	Silk	22.00	2,189.00	*NA*	42.00	2,503.00	*NA*

5 RELATED WORK

Over the past few years, the problem of scalable and time-efficient Link Discovery has been addressed by several approaches and frameworks such as *LIMES*[16], Silk [26], *KnoFuss* [18] and *Zhishi.links* [19]. These tools incorporate declarative approaches towards LD, with Silk and *KnoFuss* using blocking techniques to identify links between KBs so as to avoid unnecessary comparisons between resources. *LIMES* reduces the time-complexity of the LD procedure by combining techniques such as *PPJoin+* [27] and HR^3 [15] with set theoretical operators and planning algorithms [17]. *LIMES* provides both theoretical and practical guarantees of completeness and efficiency. A review comprising further LD approaches can be found in [14].

Up until now, only Silk provides temporal LD for RDF datasets by incorporating the recently published work of Smeros et al.[23]. The authors of this paper used MultiBlock to develop an approach for the efficient computation of temporal links. As shown in Section 4.2, Aegle is able to outperform this approach by 4 orders of magnitude.

In the field of stream reasoning and CEP on Linked Data, there has been a notable amount of research over querying temporal data. For example, *Continuous SPARQL* (*C-SPARQL*) [4] provides a syntactic and semantic extension of SPARQL to query RDF temporal data by defining a time window for processing events. *C-SPARQL* is able to incrementally re-materialise the input data, using partial static background knowledge. The novel idea behind *C-SPARQL* is the author's contribution to add an *expiration date* to each RDF triple in order to support fast deletion of events that are no longer valid. However, the use of time window frames for linking events prohibits the opportunity of linking previous events with current or future events.

Table 6. Execution runtime of all Allen Relations if computed individually. All runtimes are presented in seconds.

		Machine			Query		
Relation	Approach	*3KMachines*	*30KMachines*	*300Machines*	*3KQueries*	*30KQueries*	*300KQueries*
bf	AEGLE	0.41	5.48	115.71	2.89	23.86	25.44
	SILK	22.00	2,511.00	*NA*	43.00	2,794.00	*NA*
bfi	AEGLE	1.95	133.42	27,197.59	5.58	153.84	32,037.64
	SILK	24.00	2,547.00	*NA*	42.00	2,961.00	*NA*
m	AEGLE	0.02	0.19	3.42	0.02	0.21	3.89
	SILK	23.00	2,219.00	*NA*	41.00	2,466.00	*NA*
mi	AEGLE	0.02	0.20	2.17	0.03	0.19	1.98
	SILK	23.00	2,290.00	*NA*	44.00	2,584.00	*NA*
f	AEGLE	1.20	84.18	13,875.70	4.20	95.67	16,296.80
	SILK	23.00	2,306.00	*NA*	41.00	2,531.00	*NA*
fi	AEGLE	0.85	53.74	7,934.73	3.57	64.78	8,614.93
	SILK	23.00	2,305.00	*NA*	43.00	2,535.00	*NA*
st	AEGLE	0.66	35.23	4,946.39	3.29	46.70	5,823.81
	SILK	21.00	2,166.00	*NA*	40.00	2,613.00	*NA*
sti	AEGLE	1.20	83.71	14,162.25	4.13	94.39	16,299.07
	SILK	21.00	2,226.00	*NA*	43.00	2,533.00	*NA*
d	AEGLE	2.10	138.55	24,286.85	7.69	156.87	26,252.70
	SILK	24.00	2,363.00	*NA*	41.00	2,546.00	*NA*
di	AEGLE	2.15	138.47	24,275.08	7.67	157.75	26,188.04
	SILK	23.00	2,293.00	*NA*	41.00	2,476.00	*NA*
eq	AEGLE	0.05	0.79	49.84	0.05	0.45	348.51
	SILK	23.00	2,250.00	*NA*	41.00	2,473.00	*NA*
	baseline	2.05	171.10	23,436.30	3.15	196.09	31,452.54
ov	AEGLE	2.96	199.73	35,553.48	11.42	236.87	38,231.15
	SILK	22.00	2,181.00	*NA*	39.00	2,487.00	*NA*
ovi	AEGLE	3.16	222.27	38,226.32	11.97	257.59	42,121.68
	SILK	22.00	2,189.00	*NA*	42.00	2,503.00	*NA*

Similarly, *Streaming SPARQL* [5] provides an extension of semantics and algebraic functions of *SPARQL* that translates queries into logical algebra plans.

ETALIS is an open-source engine that is able to detect and report changes over events in near real time, by combining both static and streaming knowledge. It incorporates the *ETALIS Language for Events* (*ELE*) and *Event Processing SPARQL* (*EP-SPARQL*) [3]. The core of *ETALIS* is implemented in Prolog and incorporates the fundamentals of logic programming: an event is modeled by *ELE*, using logic facts and Prolog-style rules. In addition, *EP-SPARQL* was used to assist real-time complex event detection. In contrast to *C-SPARQL*, using this framework, the user is able to define time windows in the past.

A novel approach in the area of query processing over Linked Stream Data is *C-QUELS* [10]. In this work, the authors proposed a white-box approach for querying stream data efficiently. To this end, they define and use techniques such as query optimisation, caching and indexing. Similarly, *INSTANS* [21] (which is based on the Rete-algorithm) is able to process streams of RDF data and cache the data after the processing is over. Moreover, *INSTANS* is the only approach that supports the simultaneous processing of *SPARQL* queries, where the immediate results of a query can be used from other queries once stored. Additionally, another *SPARQL* query extension language is described in [25], where the authors proposed τ-*SPARQL* that combined with an index structure for temporal intervals achieves better runtime performance.

Most of these approaches focus on extending the semantics and functions of *SPARQL*. To the best of our knowledge, the only *SPARQL* extension that incorporates Allen's Interval Algebra is *T-SPARQL* [7]. *T-SPARQL* is a temporal extension of *SPARQL* using the multi-temporal RDF database model of [6], using similar design characteristics as *TSQL2* [24]. To query an event KB, *T-SPARQL* enhances the `FILTER` field in order to identify links between monodimensional temporal data. *T-SPARQL* utilises operators that explicitly define the bf, eq, ov, m and di relations.

6 CONCLUSIONS AND FUTURE WORK

With the use of RDF to represent an ever-growing amount of event data (e.g., for predictive maintenance of industrial machinery) comes the need to compute temporal relations between events. We presented an approach based on the reduction of Allen relations to 8 atomic relations that can be computed efficiently. We showed that by using simple sorting, we can reduce the complexity of computing any of these relations to $O(n \log n)$. Our experiments showed that our approach outperforms the state of the art, which is based on multi-dimensional blocking. In future work, we will extend the scalability of our approach by providing dedicated solutions for load balancing within a parallel execution setting. Moreover, we will study the incremental computation of temporal links on streams of data.

REFERENCES

[1] James F. Allen, 'Maintaining Knowledge About Temporal Intervals', *Commun. ACM*, **26**(11), 832–843, (November 1983).

[2] Thomas A Alspaugh, 'Software Support for Calculations in Allen's Interval Algebra', (UCI-ISR-05-2), (February 2005).

[3] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic, 'EP-SPARQL: a unified language for event processing and stream reasoning', in *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pp. 635–644, New York, NY, USA, (2011). ACM.

[4] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus, 'Incremental Reasoning on Streams and Rich Background Knowledge', in *Proceedings of the 7th International Conference on The Semantic Web: Research and Applications - Volume Part I*, ESWC'10, pp. 1–15, Berlin, Heidelberg, (2010). Springer-Verlag.

[5] Andre Bolles, Marco Grawunder, and Jonas Jacobi, 'Streaming SPARQL - Extending SPARQL to Process Data Streams', in *Proceedings of the 5th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC'08, pp. 448–462, Berlin, Heidelberg, (2008). Springer-Verlag.

[6] Fabio Grandi, 'Multi-temporal RDF Ontology Versioning', in *International Workshop on Ontology Dynamics*. CEUR-WS, (2009).

[7] Fabio Grandi, 'T-SPARQL: A TSQL2-like Temporal Query Language for RDF', in *ADBIS (Local Proceedings)*, eds., Mirjana Ivanovic, Bernhard Thalheim, Barbara Catania, and Zoran Budimac, volume 639 of *CEUR Workshop Proceedings*, pp. 21–30. CEUR-WS.org, (2010).

[8] Robert Isele, Anja Jentzsch, and Christian Bizer, 'Efficient Multidimensional Blocking for Link Discovery without losing Recall', in *Proceedings of the 14th International Workshop on the Web and Databases 2011, WebDB 2011, Athens, Greece, June 12, 2011*, (2011).

[9] Donald Knuth, *The Art of Computer Programming*, Addison-Wesley, United States, 1968.

[10] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth, 'A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data', in *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pp. 370–388, Berlin, Heidelberg, (2011). Springer-Verlag.

[11] Jens Lehmann and Pascal Hitzler, 'Concept learning in description logics using refinement operators', *Machine Learning*, **78**(1), 203–250, (2009).

[12] M. Loskyll, J. Schlick, S. Hodek, L. Ollinger, T. Gerber, and B. Prvu, 'Semantic service discovery and orchestration for manufacturing processes', in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pp. 1–8, (Sept 2011).

[13] Nijat Mehdiyev, Julian Krumeich, David Enke, Dirk Werth, and Peter Loos, 'Determination of Rule Patterns in Complex Event Processing Using Machine Learning Techniques', in *Procedia Computer Science. Complex Adaptive Systems (CAS-15), July 2-4, San Jose, CA, USA*, volume 61, pp. 395–401. Elsevier, (2015).

[14] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm, 'A survey of current Link Discovery frameworks', *Semantic Web*, (Preprint), 1–18, (2015).

[15] Axel-Cyrille Ngonga Ngomo, 'Link Discovery with Guaranteed Reduction Ratio in Affine Spaces with Minkowski Measures', in *International Semantic Web Conference (1)*, pp. 378–393, (2012).

[16] Axel-Cyrille Ngonga Ngomo, 'On link discovery using a hybrid approach', *Journal on Data Semantics*, **1**(4), 203 – 217, (December 2012).

[17] Axel-Cyrille Ngonga Ngomo, 'HELIOS - Execution Optimization for Link Discovery', in *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pp. 17–32. Springer, (2014).

[18] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta, 'Unsupervised learning of link discovery configuration', in *9th Extended Semantic Web Conference (ESWC 2012)*, (2012).

[19] Xing Niu, Shu Rong, Yunlong Zhang, and Haofen Wang, 'Zhishi.links results for oaei 2011', in *OM*, (2011).

[20] Mikko Rinne, Eva Blomqvist, Robin Keskisärkkä, and Esko Nuutila, 'Event processing in RDF', in *Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns-Volume 1188*, pp. 52–64. CEUR-WS. org, (2013).

[21] Mikko Rinne, Esko Nuutila, and Seppo Törmä, 'INSTANS: High-Performance Event Processing with Standard RDF and SPARQL', in *Proceedings of the International Semantic Web Conference (ISWC) 2012 Posters & Demonstrations Track, Boston, USA, November 11-15, 2012*, (2012).

[22] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo, *LSQ: The Linked SPARQL Queries Dataset*, 261–269, Springer International Publishing, Cham, 2015.

[23] Panayiotis Smeros and Manolis Koubarakis, 'Discovering Spatial and Temporal Links among RDF Data', in *Proceedings of the 25th World Wide Web Conference Workshop*, (2016).

[24] Michael D. Soo and Richard T. Snodgrass, 'Temporal Data Types', in *The TSQL2 Temporal Query Language*, 119–148, (1995).

[25] Jonas Tappolet and Abraham Bernstein, 'Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL', in *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pp. 308–322, Berlin, Heidelberg, (2009). Springer-Verlag.

[26] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov, 'Discovering and Maintaining Links on the Web of Data', in *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pp. 650–665, Berlin, Heidelberg, (2009). Springer-Verlag.

[27] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu, 'Efficient similarity joins for near duplicate detection', in *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pp. 131–140, New York, NY, USA, (2008). ACM.