Temporal Rule Mining for Knowledge Graph Completion

Master Thesis
Computing Science

Pascalle Doorn (6508308)

Utrecht University

Coordinators:
First supervisor: Mel Chekol
Second supervisor: Yannis Velegrakis

10 December 2024

Temporal Rule Mining for Knowledge Graph Completion

Pascalle Doorn

Abstract

This thesis introduces StaTeR (Static to Temporal Rule learner), the first dynamic symbolic rule learner designed to generate temporal rules from static rules. StaTeR combines symbolic reasoning with temporal logic to enable link and time prediction on temporal facts. The process begins with learning static rules using a symbolic rule learner. For each static rule, five different temporal confidences are computed: Naive, Naive Overlap, Same Interval, Intersection Over Union, and Temporal Alignment Coefficient. These temporal rules are evaluated for their effectiveness in link and time prediction tasks. Furthermore, this thesis generated refined datasets for temporal datasets Wikidata12k and YAGO11k. Both datasets were cleaned in such a way that identical static facts do not have any overlap in their time interval. Furthermore, temporal ranges from before the year 0 were removed, ensuring compatibility with temporal knowledge graph tools. The performance of the StaTeR model was compared against state-of-the-art models on the two datasets for link and time prediction. We show that for some quality measures, StaTeR achieves competitive results against the state-of-the-art-systems for link and time prediction tasks.

Contents

Chapter 1

Introduction

In a time where the Web has become a fundamental part of our daily lives, the reliance on its search engines has increased dramatically. With just a few taps on a keyboard, we can access an enormous amount of information on virtually anything. However, as our queries become more complex, we encounter a common obstacle: linguistic ambiguity. This refers to the situation where a word, phrase, sentence, or any piece of language can be interpreted in more than one way depending on the context. Consider the search term "apple red" for images in the search engine Google. Does this query refer to the color of the fruit, perhaps to a specific colored device from Apple Inc., or maybe a red version of Apple Inc.'s logo? Figure 1.1 shows some results images for the given search query. The Semantic Web aims to enrich the existing Web with a layer of machine-interpretable information, enabling computers to understand data semantically. Knowledge graphs are the realization of the Semantic Web as it structures information in a way that helps machines understand relationships between various entities. Unlike keyword matching search engines, knowledge graphs provide a more nuanced understanding of language and context. By storing information in a structured format, knowledge graphs can distinguish between ambiguous queries more effectively. In the context of our example search term, a knowledge graph could distinguish between the red apple fruit and the red-colored iPhone by associating each term with its respective category. This allows for a more accurate interpretation of user queries, enhancing the retrieval of relevant information.

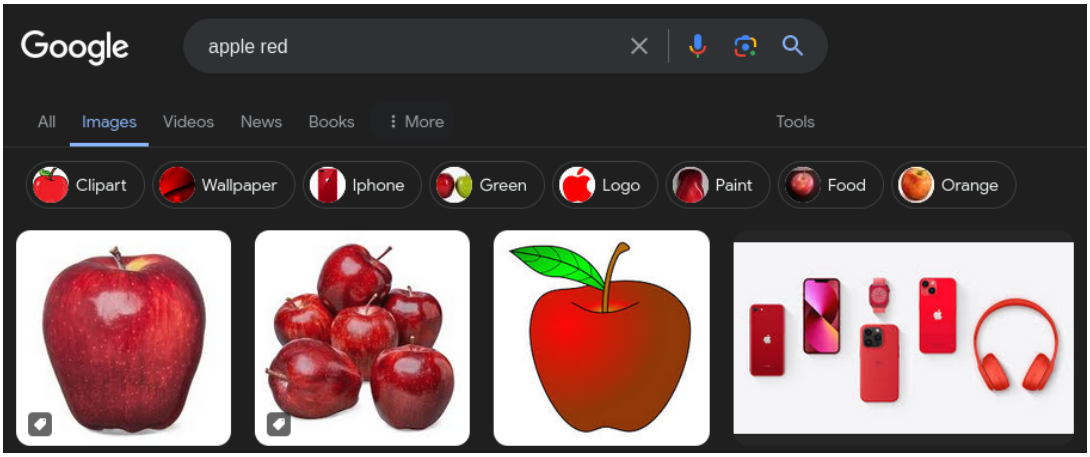

Figure 1.1: Google images results for the query "apple red", which refer to the fruit and red devices of the Apple Inc. company.

In essence, a knowledge graph is a set of facts representing relations between subject entities and object entities. Entities can take the form of many different kinds of "things"; physical entities like places, people, or products, or abstract entities such as nationality, gender, or emotions. For instance, a simple fact might state that "Harry Potter studies at Hogwarts". Such a relation can also be expressed using triples, comprising subject, relation, and object. Using the syntax of first-order logic, the given fact can be formulated as: *studiesAt(Harry Potter, Hogwarts)*. Knowledge graphs can contain up to millions of entities and billions of such facts. Notable example knowledge graphs include DBpedia [1], YAGO [2], and Freebase [3]. These knowledge graphs have been constructed by mining the web for information. For example, the DBpedia knowledge graph is constructed from the data corpus of the Wikipedia encyclopedia [4], which currently contains up to approximately 63 million articles describing entities [5]. The DBpedia knowledge graph focuses on transforming this enormous amount of information into structured knowledge using knowledge graphs. In this way, it facilitates complex queries upon the data, interlinking with other datasets accessible on the Web, and creating new innovative applications.

Knowledge graphs have grown so large that additional information can be inferred from existing data. For example, if the following two facts are known to be true:

"Hogwarts is a school in Scotland."
"Scotland is a country in the United Kingdom."

Then, the following fact can be inferred:

"Hogwarts is a school in the United Kingdom."

Furthermore, by analyzing these graphs, possible patterns and rules can be found in knowledge graphs that describe common correlations in the data. For example, a pattern could look something like this:

"If a couple is married, then they live in the same city."

Knowledge graph completion involves predicting these correct implicit facts within a knowledge graph. However, many real-world relations are time-dependent. Such relations are only correct for a certain time interval. To incorporate the temporal dimension of the relations, we introduce the temporal knowledge graph. In addition to the subject entity, relation, and object entity, a timestamp or time interval is added indicating when the relation is valid. Temporal knowledge graphs are valuable for capturing dynamic and evolving information over time, such as changes in relations between entities, the duration of events or activities, or the validity periods of statements. They find applications in various domains, including social network analysis [6] and event prediction [7]. The problem of inferring correct implicit facts in a temporal knowledge graph is called temporal knowledge graph completion.

Different approaches have been proposed for this problem. Generally, they can be classified into three main approaches; subsymbolic approaches, symbolic approaches, and neuro-symbolic approaches. Subsymbolic approaches, also known as embedding-based approaches, embed a given knowledge graph into a lower-dimensional vector space. Such approaches offer scalability, but often lack explainability. Unlike these subsymbolic black-box approaches, symbolic approaches rely on logic rule learning to reason about knowledge graphs. Frequent patterns in the data are converted into logic paths, which can be translated into logic rules. These logic rules provide high-level knowledge and human-

understandable explanations. Lastly, the neuro-symbolic approach combines the path-based and embedding-based strategies. Manually creating logic rules for large knowledge graphs is extremely difficult. For this reason, automatically learning logic rules is very important in the task of (temporal) knowledge graph completion.

In this thesis, we propose StaTeR (Static to Temporal Rule learner), a purely symbolic approach that transforms static rules into temporal rules for the temporal knowledge graph completion problem.

Chapter 2

Preliminaries

This Chapter lays the foundation for understanding (temporal) knowledge graphs and the process of knowledge graph completion. It introduces the different temporal relations and the formulation of logical rules. Finally, it explores methods for assessing the quality of these logical rules.

2.1 (Temporal) knowledge graph

A *knowledge graph* (KG) is a collection of facts representing relations between entities. A *temporal knowledge graph* (tKG) extends this by specifying the time interval or timestamp indicating the period during which the relations are valid. Let $\mathcal{G}$ be a temporal knowledge graph in the form $\mathcal{G} = (\mathcal{R}, \mathcal{E}, \mathcal{T})$, $\mathcal{R}$ denote the set of relations, $\mathcal{E}$ the set of entities, and $\mathcal{T}$ the set of time intervals. Each fact is structured as a quadruple, consisting of a *relation* or *predicate* $r \in \mathcal{R}$ between a *subject* entity $s \in \mathcal{E}$ and an *object* entity $o \in \mathcal{E}$ within a *time interval* $I \in \mathcal{T}$. This time interval I is defined by its *start time* t_s and *end time* t_e, denoted as $I = [t_s, t_e]$, indicating the time period during which the relation is valid in the temporal knowledge graph. For two different start and end times, we have $t_s < t_e$. However, if a relation holds for a timestamp, the start and end time can be considered equal, and we write $t_s = t_e$. Consequently, the general rule concludes $t_s \leq t_e$. Various equivalent representations of temporal facts exist, yet in this thesis, the fact is presented as $r(s, o, t_s, t_e)$. For example, a fact is written as *studiesAt(Harry Potter, Hogwarts*, 1991, 1997). See Figure 2.1 for an example temporal knowledge graph.

Each entity belongs to one or more *types* and each relation specifies a *domain* and *range* which describe the types of subject entity and object entity, respectively. For example, the relation *studiesAt* defines a relation where the domain consists of entities of the type *Person*, and the range contains entities of the type *School*.

The facts of a knowledge graph can be separated into two categories; *instance-level* facts and *ontological-level* facts. *Instance-level* facts describe knowledge about entities and their relations. For example:

livesIn(*Harry Potter, Godric's Hollow,* 1980, 1981)

This fact describes how the entities *Harry Potter* and *Godric's Hollow* are related by the relation *livesIn* during the time period [1980, 1981]. Furthermore, *instance-level* facts

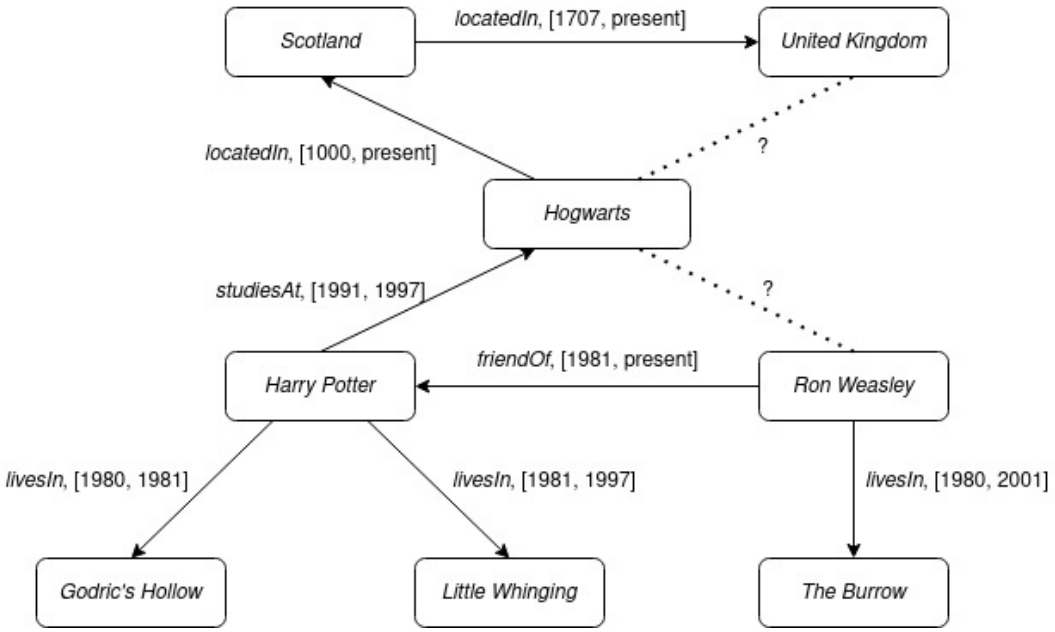

Figure 2.1: Example temporal knowledge graph with dotted missing links marked with a question mark.

also describe the types of entities, for example:

"*Hogwarts* is of type *School*."

In this fact, an entity is associated with a specific class. On the other hand, *ontological-level* facts describe knowledge at the schema level. They describe relations between classes and their properties. Below is an example of an ontological-level fact:

"The class *School* is a subclass of the class *Educational Institute*."

Such a fact encodes a hierarchical relationship between two classes. These two kinds of facts enable for reasoning within a knowledge graph. For example, combining the instance-level fact and the ontological-level fact from above, the following fact can be inferred:

"*Hogwarts* is of type *Educational Institute*."

It is important to note that in a temporal knowledge graph, similar facts cannot overlap in their temporal components. This ensures that each fact represents a unique piece of knowledge. For example, consider the following two facts:

livesIn(*Harry Potter, Little Whinging,* 1981, 1990)
livesIn(*Harry Potter, Little Whinging,* 1990, 1997)

This information would be the same as the fact:

livesIn(*Harry Potter, Little Whinging,* 1981, 1997)

If two facts with identical subject, predicate, and object overlap in their temporal component, it would be ambiguous to distinguish between which fact should apply within the time period of the overlap.

2.2 (Temporal) knowledge graph completion

The task of correctly constructing implicit facts in a knowledge graph is called *knowledge graph completion* (KGC). This thesis will operate under the *open-world assumption* (OWA); missing facts are not automatically considered false but rather unknown. On the other hand, the *closed-world assumption* (CWA) assumes that the knowledge graph contains all relevant information, implying that if a relation is absent, it is deemed to be false. Consider the following fact and test query:

"*Harry Potter* studies at *Hogwarts*."
"Does *Ron Weasley* study at *Hogwarts*?"

Under the CWA, the answer to the test query is "no" because there is no evidence confirming that *Ron Weasley* studies at *Hogwarts*. However, under the OWA, the answer to the test query is "unknown" because, based on the given information, it cannot be confirmed or denied that *Ron Weasley* studies at *Hogwarts*. Compared to static KGC, *temporal knowledge graph completion* (tKGC) presents more difficulties since the information changes over time. For example, we have the following two temporal facts and a test query:

studiesAt(*Harry Potter*, *Hogwarts*, 1991, 1994)
studiesAt(*Harry Potter*, *Hogwarts*, 1996, 1997)
"Does *Harry Potter* study at *Hogwarts*?"

The facts indicate that *Harry Potter* studied at *Hogwarts* during two distinct time periods, with a gap between them. The test query cannot be answered directly because it does not specify a year. Without temporal context, it is unclear whether the question refers to a specific year, or a range of years, of *Harry Potter*'s study status. This ambiguity highlights the complexities of reasoning over time in tKGC.

To evaluate the performance of a tKGC algorithm, a temporal knowledge graph $\mathcal{G} = (\mathcal{R}, \mathcal{E}, \mathcal{T})$ is used as an evaluation dataset. The temporal knowledge graph $\mathcal{G}$ is split into *training* graph $\mathcal{G}_t$, *validation* graph $\mathcal{G}_v$, and *evaluation* graph $\mathcal{G}_e$. Each graph contains its own sets of relations, entities, and time intervals. For instance, the training graph is defined as $\mathcal{G}_t = (\mathcal{R}_t, \mathcal{E}_t, \mathcal{T}_t)$, where $\mathcal{R}_t, \mathcal{E}_t$, and $\mathcal{I}_t$ represent the sets of relations, entities, and time intervals specific to the training graph. Similarly, the validation graph $\mathcal{G}_v = (\mathcal{R}_v, \mathcal{E}_v, \mathcal{T}_v)$ and the evaluation graph $\mathcal{G}_e = (\mathcal{R}_e, \mathcal{E}_e, \mathcal{T}_e)$ follow the same structure. Although the graphs are disjoint in terms of individual facts, the set of relations, entities, and time intervals in the validation and evaluation sets are subsets of those of the training set. This ensures that the algorithm is evaluated on familiar concepts while still being tested on new combinations. In formal terms, we have the following:

$$\mathcal{R}_v, \mathcal{R}_e \subseteq \mathcal{R}_t \qquad \mathcal{E}_v, \mathcal{E}_e \subseteq \mathcal{E}_t \qquad \mathcal{T}_v, \mathcal{T}_e \subseteq \mathcal{I}_t$$

Normally, the validation graph is used to tune the hyperparameters of a classifier. However, a rule-based approach typically uses a fixed hyperparameter setting. The datasets we have used divide the knowledge graph into training, validation, and evaluation graphs as many models require a validation set.

The algorithm uses the training graph to learn a set of rules that capture common patterns of the graph. Since the validation and evaluation graphs share the same vocabulary, predictions for the evaluation graph can be made using the learned rules of the training

graph. We call a fact in the evaluation graph a test query. Such a query can take five different forms. For example, consider the following fact:

studiesAt(*Harry Potter*, *Hogwarts*, 1991, 1997)

This fact produces the following five test queries:

1. *studiesAt*(?, *Hogwarts*, 1991, 1997)
2. *studiesAt*(*Harry Potter*, ?, 1991, 1997)
3. *studiesAt*(*Harry Potter*, *Hogwarts*, ?, 1997)
4. *studiesAt*(*Harry Potter*, *Hogwarts*, 1991, ?)
5. *studiesAt*(*Harry Potter*, *Hogwarts*, ?, ?)

Given a test query $r(s, ?, t_s, t_e)$, where the object entity o is asked for the relation r with the subject entity s in the time interval $[t_s, t_e]$, the correct object entity may change within the given time interval. *Link prediction* is a problem that aims to estimate the probability of a relation, or link, existing between two entities [8]. The problem of link prediction can be categorized into two categories; constructing missing relations between entities from the knowledge graph and predicting relations that may exist in the future based on trends and patterns. *Time prediction* is the task of predicting if and when an event will happen in the future [9]. For example, for the last test query, an algorithm aims to predict the likelihood of *Harry Potter* studying at *Hogwarts* and estimate the corresponding time period of this.

2.3 Temporal relations

A temporal relation between two timestamps, denoted t_x and t_y, can take three distinct forms; $t_x < t_y$ signifies that t_x is *before* t_y, $t_x = t_y$ indicates that t_x is *equal* to t_y, and $t_x > t_y$ denotes that t_x is *after* t_y. In contrast, when considering the temporal relation between two time intervals I_x and I_y, represented as $I_i = [s_i, e_i]$ where $s_i \leq e_i$, up to thirteen different temporal relations arise. This was introduced by Allen's interval algebra, a calculus for temporal reasoning developed by James F. Allen in 1983 [10]. Together with the three temporal relations introduced for two timestamps, all the temporal relations for two intervals, X and Y with respective start and end times X_s, Y_s, X_e, Y_e, are defined in Table 2.1.

2.4 Logical rules

An *atom* is a fact that includes subject, predicate, and object. The subject and object can refer to either constants, which refer to entities, or variables. Lowercase letters are used for constants and uppercase letters for variables. An example atom is *friendOf*(X, Y), stating that variable X is a friend of variable Y. A *Horn clause* is a type of clause in logic that consists of at most one positive *atom*. When a Horn clause has exactly one positive *atom*, it is referred to as a *strict Horn clause*. A *Horn rule* consists of a singular *head atom* and a collection of *body atoms*. The disjunctive form of a clause is a logical expression that combines several atoms using the disjunction operator. By transforming the disjunctive form into an implication form using logic, rules can be explored. Below

Relation	Symbol	Symbol for inverse	Pictoral example	Logical interpretation
X *before* Y	$<$	$>$	XXX YYY	$X_e < Y_s$
X *equal* Y	$=$	$=$	XXX YYY	$X_s == Y_s \wedge X_e == Y_e$
X *meets* Y	m	mi	XXXYYY	$X_e == Y_s$
X *overlaps* Y	o	oi	XXX YYY	$X_s < Y_s < X_e \wedge X_e < Y_e$
X *during* Y	d	di	XXX YYYYY	$X_s > Y_s \wedge X_e < Y_e$
X *starts* Y	s	si	XXX YYYYY	$X_s == Y_s \wedge X_e < Y_e$
X *finishes* Y	f	fi	XXX YYYYY	$X_s > Y_s \wedge X_e == Y_e$

Table 2.1: Different types of temporal relations between two time intervals X and Y, with respective start and end times X_s, Y_s, X_e, Y_e. Each type of relation has a different type of inverse relation, except for the *equal* relation, summing up to thirteen different types of relations.

are representations of a logical rule that can be extracted from Figure 2.1. In disjunctive form, the rule is written as follows:

$\neg \mathit{studiesAt}(X, A) \vee \neg \mathit{friendOf}(A, Y) \vee \mathit{studiesAt}(X, Y)$

Alternatively, in the implication form, the same rule is expressed as:

$\mathit{studiesAt}(X, Y) \longleftarrow \mathit{friendOf}(X, A) \wedge \mathit{studiesAt}(A, Y)$

The rule above states that if X is a friend of A and A studies at Y, then it implies that X also studies at Y. The part left of the arrow is called the *head* of the rule, whereas all the atoms to the right of the arrow together are called the *body* of the rule. The *grounding of a rule* is the process in which all variables are replaced by constants. A possible grounding for the example Horn rule is, for example, given by:

$\mathit{studiesAt}(\mathit{Harry\ Potter}, \mathit{Hogwarts}) \longleftarrow$
$\mathit{friendOf}(\mathit{Harry\ Potter}, \mathit{Ron\ Weasley}) \wedge \mathit{studiesAt}(\mathit{Ron\ Weasley}, \mathit{Hogwarts})$

This translates to if *Harry Potter* is a friend of *Ron Weasley* and *Ron Weasley* studies at *Hogwarts*, then *Harry Potter* also studies at *Hogwarts*. When we define the rule as r, then this grounding could be expressed as $r\theta_{X=\mathit{Harry\ Potter}, A=\mathit{Hogwarts}, Y=\mathit{Ron\ Weasley}}$; In rule r, X is replaced by *Harry Potter*, A is replaced by *Hogwarts*, and Y is replaced by *Ron Weasley*.

In many rule-based systems and datasets, a consistent convention is followed when creating rules. When not grounded, the variable X is assigned to represent the subject of the head atom of the rule, while the variable Y is used to denote the object of the head atom. The body atoms will create a path from X to Y. Furthermore, when additional

variables are required in the body of the rule, they are introduced in alphabetical order. Thus, starting from A to as many variables as needed. In this thesis, this convention is adopted for consistency and to align with common practice in state-of-the-art rule-based systems and datasets.

Two atoms are *connected* if they share a variable. A rule is considered connected if every atom in the rule is connected to another atom in the rule. A variable is *closed* in a rule if it appears at least twice. A rule is considered closed if all the variables in the rule are closed. If a rule is closed and connected, then the rule does not contain unrelated atoms of variables. Consider the following two rules:

$\mathit{studiesAt}(X, Y) \longleftarrow \mathit{friendOf}(X, A) \wedge \mathit{studiesAt}(X, Y)$
$\mathit{studiesAt}(X, Y) \longleftarrow \mathit{studiesAt}(X, Y) \wedge \mathit{loves}(A, A)$

The first rule is connected, but not closed, since the variable A appears only once in the rule. The second rule is closed, but it is not connected because the atom *loves(A, A)* does not share the variable A with any other atom in the rule. Intuitively, a *closed path rule* is a closed rule where body atoms form a path from the subject of the head atom to the object of the head atom. Formally, a closed rule path of length n is represented in the form:

$$h(A_0, A_n) \longleftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i) \tag{1}$$

In the above rule, the symbol $\bigwedge$ represents the conjunction of all body atoms from $i = 1$ to $i = n$. The relations h and b_1 to b_n do not necessarily have to be different relations. For example, consider the following transitive rule where the same relation is used in each of the atoms:

$\mathit{locatedIn}(\mathit{Hogwarts}, \mathit{United\ Kingdom}) \longleftarrow$
$\mathit{locatedIn}(\mathit{Hogwarts}, \mathit{Scotland}) \wedge \mathit{locatedIn}(\mathit{Scotland}, \mathit{United\ Kingdom})$

2.5 Bottom-up rule learning

AnyBURL [11] generates candidate rules by exploring instance-level paths in knowledge graphs. It uses a *bottom-up* approach in rule learning that substitutes entities for variables in the paths, forming rule patterns. The algorithm identifies higher-level patterns in the knowledge graph from instance-level data. The algorithm distinguishes three different types of rules; binary rules ($\mathbf{B}$), unary rules ending with a *dangling atom* ($\mathbf{U}_d$), and unary rules ending with an atom that includes a *constant* ($\mathbf{U}_c$). A *dangling atom* refers to an atom that is not fully specified or lacks connections to other atoms, leaving it "dangling". The three types of rules are structured as follows.

$$\mathbf{B} : h(A_0, A_n) \longleftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i) \tag{2}$$

$$\mathbf{U}_d : h(A_0, c) \longleftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i) \tag{3}$$

$$\mathbf{U}_c : h(A_0, c) \longleftarrow \left(\bigwedge_{i=1}^{n-1} b_i(A_{i-1}, A_i) \right) \wedge b_n(A_{n-1}, c') \tag{4}$$

In the rules above, h and b_i refer to relations, uppercase letters refer to variables, and c or c' refer to constants. The head atom of the unary rules consists of a variable as the subject entity and a constant as the object entity. Since the object entity is constant, such an atom can be viewed as an unary predicate, hence the name. For example, the binary relation *studiesAt*$(X, \mathit{Hogwarts})$ could also be written as *studiesAtHogwarts*(X). The $\mathbf{B}$ and $\mathbf{U}_c$ rules are closed connected rules. Meanwhile, the $\mathbf{U}_d$ rules are not closed, since A_n appears only once in the rule.

Below, Rule 5 is an example of a $\mathbf{B}$ rule; X is a *Wizard* if A is a friend of X and A lives at *Hogwarts*. An example of a $\mathbf{U}_d$ rule is given by Rule 6; if X is born in *The Netherlands*, then X has nationality *Dutch*. Lastly, Rule 7 describes a common $\mathbf{U}_c$ rule; X is *female* if it is the biological parent of Z while Y is a *male* and also the biological parent of Y.

$$\mathit{occupation}(X, \mathit{Wizard}) \longleftarrow \mathit{friendOf}(X, A) \wedge \mathit{livesAt}(A, \mathit{Hogwarts}) \tag{5}$$

$$\mathit{nationality}(X, \mathit{Dutch}) \longleftarrow \mathit{bornIn}(X, \mathit{The\ Netherlands}) \tag{6}$$

$$\begin{gathered} \mathit{gender}(X, \mathit{female}) \longleftarrow \\ \mathit{biologicalParent}(X, A) \wedge \mathit{biologicalParent}(Y, A) \wedge \mathit{gender}(Y, \mathit{male}) \end{gathered} \tag{7}$$

Consider the knowledge graph in Figure 2.2. Using this knowledge graph, the process of creating rules can be illustrated, where the head of the rule is *occupation(Harry Potter, Wizard)*. To begin, the bottom rules of length n, starting from the head atom of the rule, are constructed. This is done by randomly traversing n edges in the graph, starting from *Harry Potter* or *Wizard*. This leads to the creation of paths of length $n+1$. These paths are color-coded in Figure 2.2 and written as rules below.

$$\begin{gathered} \mathit{occupation}(\mathit{Harry\ Potter}, \mathit{Wizard}) \longleftarrow \\ \mathit{friendOf}(\mathit{Harry\ Potter}, \mathit{Hagrid}) \wedge \mathit{livesAt}(\mathit{Hagrid}, \mathit{Hogwarts}) \end{gathered} \tag{8}$$

$$\begin{gathered} \mathit{occupation}(\mathit{Harry\ Potter}, \mathit{Wizard}) \longleftarrow \\ \mathit{childOf}(\mathit{Harry\ Potter}, \mathit{James\ Potter}) \wedge \mathit{occupation}(\mathit{James\ Potter}, \mathit{Wizard}) \end{gathered} \tag{9}$$

$$\begin{gathered} \mathit{occupation}(\mathit{Harry\ Potter}, \mathit{Wizard}) \longleftarrow \\ \mathit{friendOf}(\mathit{Harry\ Potter}, \mathit{Ron\ Weasley}) \wedge \mathit{occupation}(\mathit{Ron\ Weasley}, \mathit{Wizard}) \end{gathered} \tag{10}$$

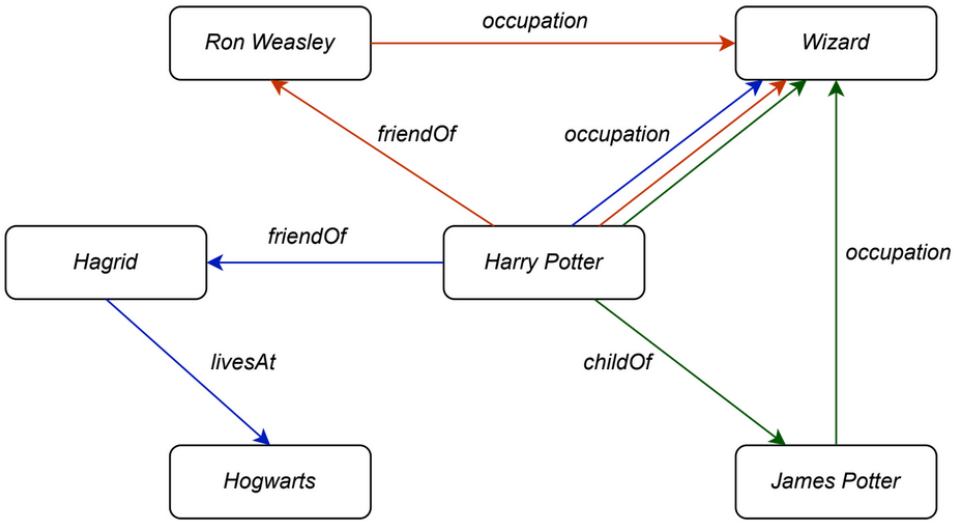

Figure 2.2: Example knowledge graph for explaining path rules. The blue path corresponds to Rule 8, green path to Rule 9, and the red path to Rule 10.

These rules can be generalized to the form of either **B** rules (2), $\mathbf{U}_d$ rules (3), or $\mathbf{U}_c$ rules (4). To generalize the rule from instance-level to higher-level, we apply one of the following two generalization operations.

1. Replace all occurrences of a constant with a fresh variable.
2. Drop one of the atoms of the body.

The generalization lattice of the blue path (Rule 8) in Figure 2.2 is displayed in Figure 2.3. In each step, one of the two generalization steps mentioned above is used. The generalization steps in Figure 2.3 are color-coded; orange arrows indicate generalization operations of type 1, and blue arrows indicate generalization operations of type 2. In the end, the instance-level blue path (Rule 8) is generalized to the higher-level rule (Rule 11).

$$occupation(X, Wizard) \longleftarrow friendOf(X, A) \wedge livesAt(A, B) \tag{11}$$

2.6 Quality measures of a rule

Once the rules are created, some kind of measures are needed to assess the quality of a learned rule. Consider the following rule:

$livesIn(X, Y) \longleftarrow worksIn(X, Y)$

This rule states that if X works in place Y, then X lives in place Y. However, this conclusion is not always true. Evaluating the quality of this rule involves determining the probability that X actually lives in Y given that X works in Y. Evaluation of logical rules is an important aspect of rule learning since it indicates the plausibility of the rules learned. In this Section, three statistical measures are explained; *support*, *confidence*, and *PCA confidence*.

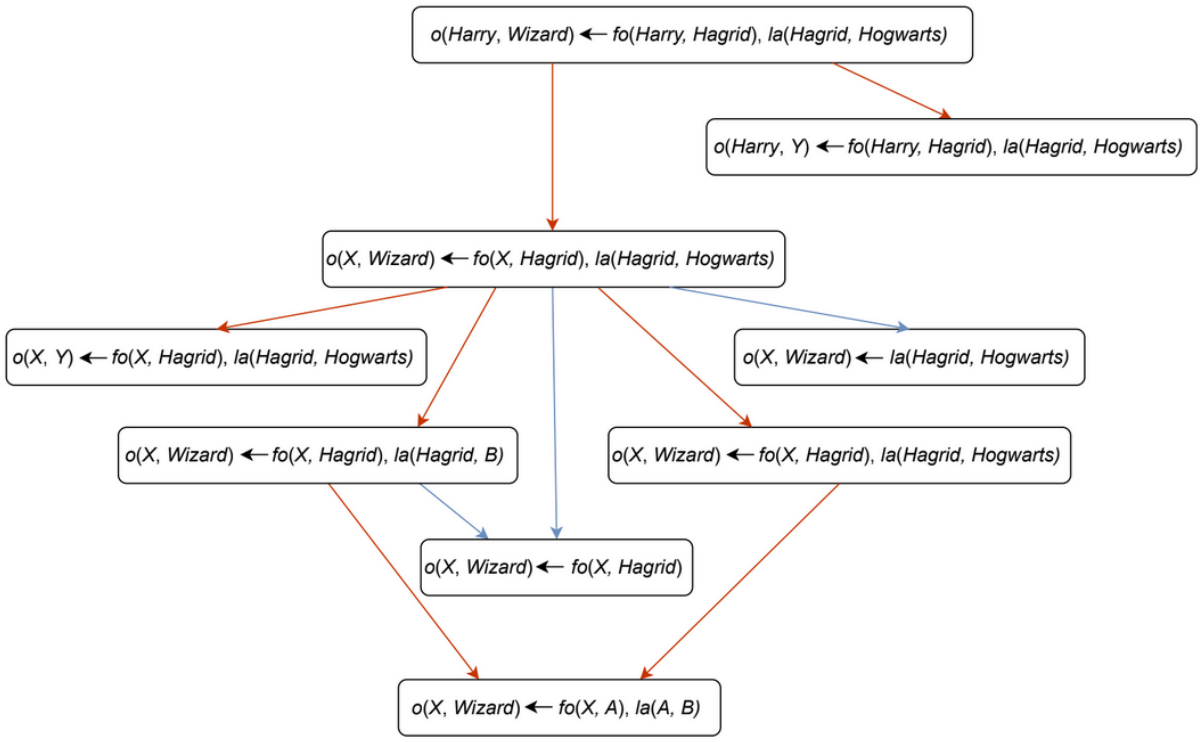

Figure 2.3: Generalization lattice for Rule 8. The relations are abbreviated as follows: o = *occupation*, fo = *friendOf*, and la = *livesAt*. Orange paths use the first type of generalization operation and blue paths use the second type of generalization operation. This results in the generalized Rule 11.

The *support* of a rule measures the frequency of the rule in the data. The support for rule r is defined as the number of instances where both the head and body of r are satisfied on the graph. Formally, it is defined as described below.

$$supp(r) = |x_H(r) \cap x_B(r)| \tag{2.1}$$

In this definition, $x_H(r)$ refers to all instances satisfying the head of the rule r, while $x_B(r)$ refers to those satisfying the rule's body. Thus, the support of a rule indicates how often a grounding of a rule is present in the knowledge graph. If the support for a rule is high, then the grounding of this rule is frequently observed.

The *confidence* of a rule refers to the ratio of instances where the head and body are satisfied to the instances where only the body is satisfied. The confidence estimates the probability of the head of the rule being true, given that the body is true. The high confidence of a rule indicates that the rule is often correct when applicable. Given the definition of $x_B(r)$ as before in Equation 2.1, the confidence of rule r is calculated in the following way:

$$conf(r) = \frac{supp(r)}{|x_B(r)|} \tag{2.2}$$

To demonstrate the calculations for support and confidence, consider the following rule r formed as:

$livesIn(X, Y) \longleftarrow worksIn(X, Y)$

In addition, consider the facts in Table 2.2, which is a small knowledge graph consisting of instances of the relations *worksIn* and *livesIn*.

$$worksIn(Harry\ Potter,\ London)$$
$$worksIn(Harry\ Potter,\ Little\ Whinging)$$
$$worksIn(Hagrid,\ Scotland)$$
$$livesIn(Harry\ Potter,\ London)$$
$$livesIn(Vernon\ Dursley,\ Little\ Whinging)$$

Table 2.2: Example knowledge graph containing five facts using the the six entities *Harry Potter*, *Hagrid*, *Vernon Dudley*, *London*, *Little Whinging*, and *Scotland*, and the two relations *livesIn* and *worksIn*.

The support of a rule corresponds to the number of groundings that satisfy both the head and the body of the rule. An example in this case is the following grounding of rule r: $r\theta_{X=Harry\ Potter,Y=London}$. This is because both $livesIn(Harry\ Potter,\ London)$ and $worksIn(Harry\ Potter,\ London)$ are satisfied. There are no other groundings for which both the head and the body are satisfied. Therefore, $supp(r) = 1$.

The confidence is calculated as the support divided by the number of satisfied body instances. There are three instances in which the body is satisfied: $worksIn(Hagrid,\ Scotland)$, $worksIn(Harry\ Potter,\ London)$, and $worksIn(Harry\ Potter,\ Little\ Whinging)$. Among these, only one instance is also satisfied in the head. Therefore, $conf(r) = \frac{1}{3}$.

The confidence treats all facts that are not in the knowledge graph as negative examples. This follows the closed-world assumption (CWA), where missing facts are considered false. This contradicts the open-world assumption (OWA), which would classify facts that are not in the graph as unknown rather than false. To address this limitation, the AMIE+ approach [12] refines the confidence to assume the *partial completeness assumption* (PCA). The PCA assumes that if one fact about an entity is known in the knowledge graph, then all relevant facts are known. More clearly, if it is known that $livesIn(Hagrid,\ Scotland)$, then it assumes that all the places where *Hagrid* works are present in the knowledge graph. The PCA is calculated in the following way:

$$conf_{PCA}(r) = \frac{supp(r)}{|x_{B,PCA}(r)|} \tag{2.3}$$

Similarly to the calculation of the confidence, the denominator counts the number of instances where both the head and body of the rule are satisfied. In contrast to the calculation of the confidence, the numerator considers only body instances for which the knowledge graph contains information about at least one valid head instance, this is what $x_{B,PCA}(r)$ refers to. Going back to the example, the PCA confidence of the rule results in: $conf_{PCA}(r) = \frac{1}{2}$. This is because, there is still one correct grounding for the rule: $r\theta_{X=Harry\ Potter,Y=London}$. Furthermore, the prediction $livesIn(Harry\ Potter, Little\ Whinging)$ is counted for $|x_{B,PCA}|$ since we already know that *Harry Potter* lives in *London*. The prediction $livesIn(Hagrid,\ Scotland)$ is not counted since there is no information in the knowledge graph at all about the place where *Hagrid* lives.

Chapter 3

Related work

This Chapter lists various models of all different types addressing the (temporal) knowledge graph completion problem. Table 3.1 provides a clear overview of all the models discussed in this Chapter.

3.1 Subsymbolic approaches

Subsymbolic machine learning approaches, also known as embedding-based methods, embed a given knowledge graph into a lower-dimensional vector space. This approach involves creating vectors for entities and relations and evaluating the plausibility of a fact through a scoring function. These models offer scalability, making them suitable for handling large and sparse knowledge graphs. However, such models often lack explainability and interpretability. Embedding approaches for knowledge graph completion can be subdivided into *translation-based*, *tensor factorization*, and *neural network* based. Furthermore, all of these approaches can be used for a static or temporal knowledge graph.

3.1.1 Static subsymbolic approaches

One of the methods for embedding a knowledge graph is a *translation-based model*. TransE [13] is an example of such a model. In such models, relationships are represented as translations in the embedding space. This means that if a relation $r(s, o)$ holds, then the embedding of the object entity o should approximate the embedding of the subject entity s plus a vector that represents the relation r. This approach is parameter-efficient as it learns only one low-dimensional vector for each entity and relation.

Another type of subsymbolic approach is a *tensor factorization model* such as ComplEx [14]. In such models, a triple is denoted as $x_{ijk} = (s_i, p_k, o_j)$ and represents predicate p_k with subject entity s_i and object entity o_j. The binary variable y_{ijk} indicates the existence of the relation x_{ijk} in a binary tensor Y that is decomposed into a product of embedding matrices. Thus, for a given fact x_{ijk}, the score can be discovered as a multi-linear product between the embeddings of the vectors s, p, and o.

More recently, *deep neural network-based models* have become prominent, using neural networks to create embeddings. In practice, these models learn one embedding (a low-

dimensional vector) for each entity and one operator (a matrix) for each relation [15]. R-GCN [16] and ConvE [17] use graph convolutional network models for link prediction. Using such an approach enhances the model's ability to capture complex dependencies and patterns in the data.

3.1.2 Dynamic subsymbolic approaches

In the context of temporal knowledge graphs, other models are used to incorporate the extra dynamic time dimension. The translation-based model TransE [13] is extended to TTransE [18] embedding temporal knowledge graphs in a vector space using the timestamps in the scoring function.

An example of a temporal tensor factorization model is TNTComplEx [19] which builds on the static ComplEx [14] decomposition using another dimension in the tensor for timestamps. This decomposition provides embeddings for each timestamp. Both approaches share the principle of embedding the temporal knowledge graph into a low-dimensional vector space while keeping the temporal dynamics of the tKG intact using time-dependent embeddings. The algorithm TimePlex [20] is also based on ComplEx [14]. It works by embedding entities, relations, and time in a uniform, compatible space. The algorithm has components that allow it to learn temporal constraints as model weights. The model learns temporal patterns directly from the data instead of learning the rules provided.

Temporal subsymbolic neural network-based approaches include HyTE [21] and RE-NET [22]. The HyTE [21] approach projects entities and relations onto temporal hyperplanes to capture the temporal aspects of the data. RE-NET [22] models future events of knowledge graphs using recurrent neural networks.

3.2 Symbolic approaches

Symbolic approaches rely on logical representations and rules to reason about knowledge graphs. Facts are analyzed within the knowledge graph, and frequent patterns among entities and relations are identified, which are turned into logical rules. These generalized rules capture the relationships in the knowledge graph in a more abstract way. These generalized rules, also called the *ground rules*, are used to infer new or missing relations in the data. Formally, a *rule learner* derives such rules. A rule is deemed plausible if there are many instances where variables can be substituted by entities of the graph, resulting in known facts of the graph. The greater the number of such instances, the more plausible the rule. Logical rules enhance the explainability and trustworthiness of the solutions; for a given solution, we can explain why this solution (missing relation) could be present in the knowledge graph based on the logical rules. However, the creation of such rules is often challenging, especially for complex temporal knowledge graphs. For this reason, symbolic approaches suffer from scalability issues. The task of *rule learning* includes both discovering the rule structures and estimating their plausibility. Symbolic approaches can be divided into *ILP-based* and *path-based* approaches. This Section only provides static approaches, since there is no existing research that uses a purely symbolic approach for the temporal knowledge graph completion problem.

3.2.1 Static symbolic approaches

Mining logical rules from a dataset is the central task of Inductive Logic Programming (ILP). The structure of rules is learned by systematically exploring the rule space. This is one of the reasons why classical ILP systems like QuickFOIL [23] and ALEPH [24] are unable to mine logical rules from knowledge graphs; the current knowledge graphs have grown to a scale that classical ILP systems are unable to handle. Secondly, ILP systems usually require negative statements as counterexamples. Knowledge graphs typically do not contain negative statements. Furthermore, because of the OWA, absent facts cannot serve as negative statements. The AMIE+ rule learner [12] has been developed to apply the ILP-based approach to large knowledge graphs. It operates only on binary relations, resulting in a language bias that significantly reduces the search space in rule learning. AMIE+ will generate all closed and connected rules that meet the qualifications set in the configuration. Therefore, it can be characterized as a complete search. However, for very large datasets, AMIE+ may encounter difficulties in completing the task since it needs to construct all possible rules. Additionally, the approach uses plausibility metrics adapted from association rule mining to compensate for the absence of negative statements. AMIE+ assigns a confidence score under the PCA to each learned rule. If the learned rule meets some confidence threshold, the rule is selected as a candidate rule.

Other approaches tackle the problem of the lack of negative examples by generating them themselves. Such programs we referred to as *nonmonotonic logic programs*, where rules with negations are learned. Consider the knowledge graph in Figure 3.1. A prominent pattern in the knowledge graph could be "Married people live in the same place". This pattern could be cast to a correct Horn rule, which would look like the following:

$$livesIn(Y, Z) \longleftarrow married(X, Y) \wedge livesIn(X, Z)$$

This rule states that if persons X and Y are married, then they live in the same place. For example, the knowledge graph contains the fact that *James Potter* and *Lily Potter* are married and both live in *Godric's Hollow*. This rule can be used to infer the missing living place of *Harry Potter* based on the married relations he has with *Ginny Weasley* suggesting that he lives in *Godric's Hollow*. Furthermore, this rule can be used to signal incorrect facts in the knowledge graph. For instance, the living places of *Ron Weasley* and *Hermoine Granger* differ while they are married, which contradicts the created Horn rule. Horn rules that can be learned are limited to only positive relations in the body of the rule. This approach lacks the capacity to describe rules with exceptions, such as "Married people live in the same place unless one's profession is Auror.". This rule, which is no longer a Horn rule, can be captured in the following way:

$$livesIn(Y, Z) \longleftarrow married(X, Y) \wedge livesIn(X, Z) \wedge not\ Auror(Y)$$

This paper [25] uses a nonmonotonic logic program that produces *nonmonotonic rules* that capture exceptions by adding negative atoms into their bodies. This approach simplifies the search for negated atoms by converting binary relations into unary predicates. For instance, the binary relation $livesIn(Y, London)$ can be translated into a unary one $livesInLondon(Y)$. This improves the quality of the rules for predicting missing links. However, in this way, the knowledge graph only contains unary facts. Another paper [26] addresses this problem by directly handling binary relations, resulting in learning negative rules while maintaining the relational structure of the knowledge graphs.

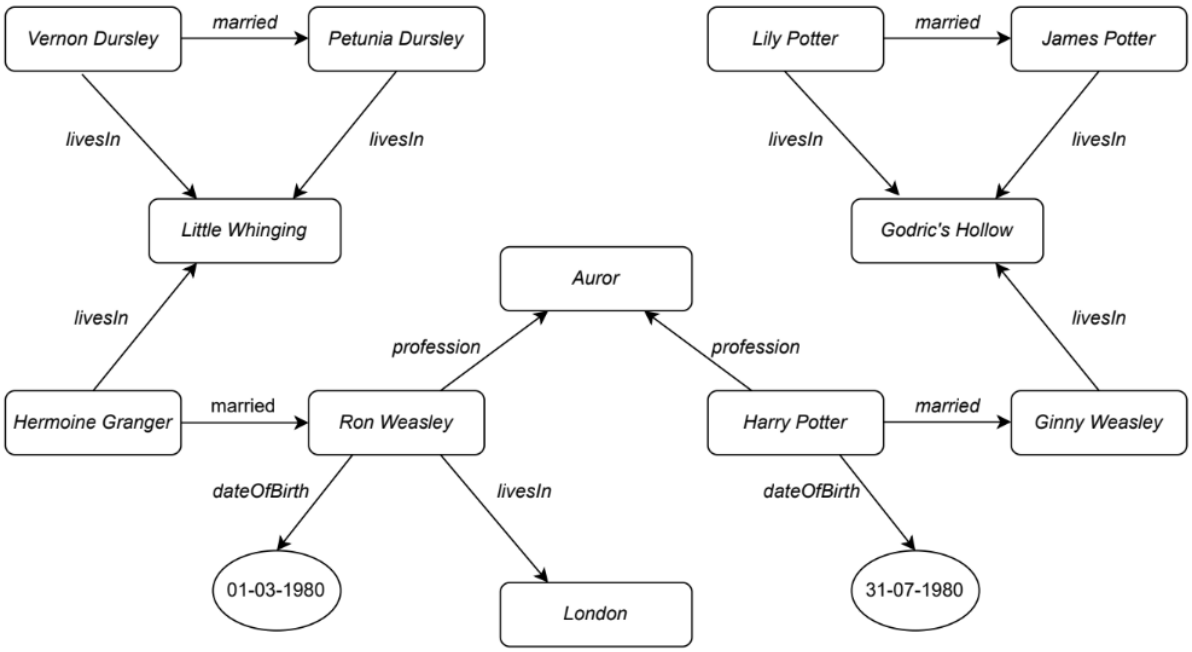

Figure 3.1: Example knowledge graph.

AnyBURL [11] generates candidate rules by exploring instance paths in knowledge graphs. It proposes a *bottom-up* approach that substitutes entities for variables in the paths forming rule patterns. Due to its "anytime" behavior, the algorithm can iteratively refine rules to improve their accuracy. This allows for more efficient rule learning in large-scale settings. This approach identifies higher-level patterns in the knowledge graph from instance-level, hence the name bottom-up approach. The algorithm involves random walks in the knowledge graph which are then used for the creation of rules. It instantiates a set of rule patterns directly, which results in only the subset of rules that are relevant. One of the key challenges in rule-based learning is determining which types of rules are supported. The set of supported rules is also known as the language bias, which dictates which patterns are captured and which ones are overlooked. In the case of AnyBURL, this rule learner supports three types of rules described in Section 2.5. Thus, while the AnyBURL search is not complete, its selective instantiation helps to identify the most important rules quickly at the start of the search.

Like AnyBURL [11], RuDiK [27] also uses a bottom-up approach. It is different from AnyBURL in the sense that it can discover both the *positive* and *negative* rules. Negative examples are generated in the following way: If the rule $p(s, o)$ does not occur in the knowledge graph, then it is considered a negative example if at least one of the triples of the form $p(s', o)$, $p(s, o')$, or $p'(s, o)$ will be present in the knowledge graph. Furthermore, the algorithm enables comparisons of the numerical and string values in the rules. For example, a negative rule that includes numerical value comparisons would look something like this:

$$\bot \longleftarrow \mathit{childOf}(X, Y) \wedge \mathit{dateOfBirth}(X, a) \wedge \mathit{dateOfBirth}(Y, b) \wedge a < b$$

This rule states that person X cannot be the child of person Y if Y was born after X. For example, using the knowledge graph of Figure 3.1, it can be inferred that *Ron Weasley* cannot be the child of *Harry Potter* since the birth date of *Harry Potter* is

after the birth date of *Ron Weasley*. The symbol $\perp$ in the head of the rule, often called *falsum*, is a logical constant that refers to the truth value *false*. Using such rules increases the expressive power of the rule language with respect to previous existing systems. This results in rule mining that is robust to existing errors and incompleteness in the knowledge graph. RuDiK has been deployed on various knowledge graphs, including DBPedia [1], Yago [2], and Freebase [3], which resulted in identifying new facts and real errors with an accuracy of 85% and 97%, respectively. This paper [28] builds upon the RuDik [27] algorithm. This study extends the algorithm by enabling more complex comparisons in the search for rules. Beyond simple equality, the enhanced algorithm can mine for rules with the comparisons $>, \geq, \leq, <$ between numerical values and dates. Furthermore, it supports the inequality symbol $\neq$ for relations between all literals. Moreover, the paper provides a detailed exploration of the negative rules including sophisticated techniques for recognizing contradictions within the data.

In addition to *instance-level* pathfinders, there are also rule learners that explore paths on *ontological-level* within knowledge graphs. In contrast to bottom-up approaches, these approaches are referred to as *top-down*. For example, the ScaLeKB [29] approach uses the Ontological Pathfinding (OP) algorithm [30] to mine rules from the enormous knowledge graphs. Like most rule learners, the algorithm of ScaLeKB uses language bias and assumes that the rules are connected and closed. In this way, the rule does not contain irrelevant atoms or variables. The OP algorithm searches paths over a domain-range schema graph. Furthermore, it can handle type hierarchy to divide entities into subtypes for refined classification. The algorithm is focused on scalability using a series of parallelization and optimization techniques. It constructs candidate rules in relational tables according to structural equivalence, partitions them into smaller inputs, eliminates non-functional rules, and runs the parallel-rule-mining algorithm for each partition to compute the scores. The typing of the entities and relations is used in a way that restricts the relations that can be combined to form candidate rules. This significantly reduces the search space of paths. However, ScaLeBK relies on type information as hard constraints for candidate rules. Each entity in the rule path must belong to a specific type. This constraint is too restrictive in practice.

RARL [31] is another one of those top-down ontological-level path rule learners. It splits the facts of a knowledge graph into two sets: the *TBox* and the *ABox*. The TBox refers to the set of terminological facts, the ontology of a knowledge graph. It contains relations like *subclassOf* and defines types for the *domain* and *range* of the relations. Furthermore, it is the structure of the ABox. The ABox contains the facts on instance-level, knowledge about entities, their relations, types, and properties. Consider Figure 3.2, a fact from the ABox would be that *Harry Potter* studies at *Hogwarts*. Using the TBox of this knowledge graph, we can state that *Harry Potter* is of type *Wizard* and of type *Person* since *Wizard* is a subclass of *Person*. To illustrate that an entity is an instance of a certain type, we use the relation *a*. Unlike ScaLeBK [29], which explores ontological paths as candidate rules and enforces hard constraints on them, RALR systematically samples paths based on the information of the TBox. The algorithm reduces the search space by computing the semantic relatedness between the relations in the atoms of the body and the relation of the head using the term frequency-inverse document frequency (TF-IDF) weighting factor. Given an input head of a rule, RARL generates the set of most semantically related candidate rule bodies. This method provides a more flexible way to learn typed rules in the environment of incomplete type information.

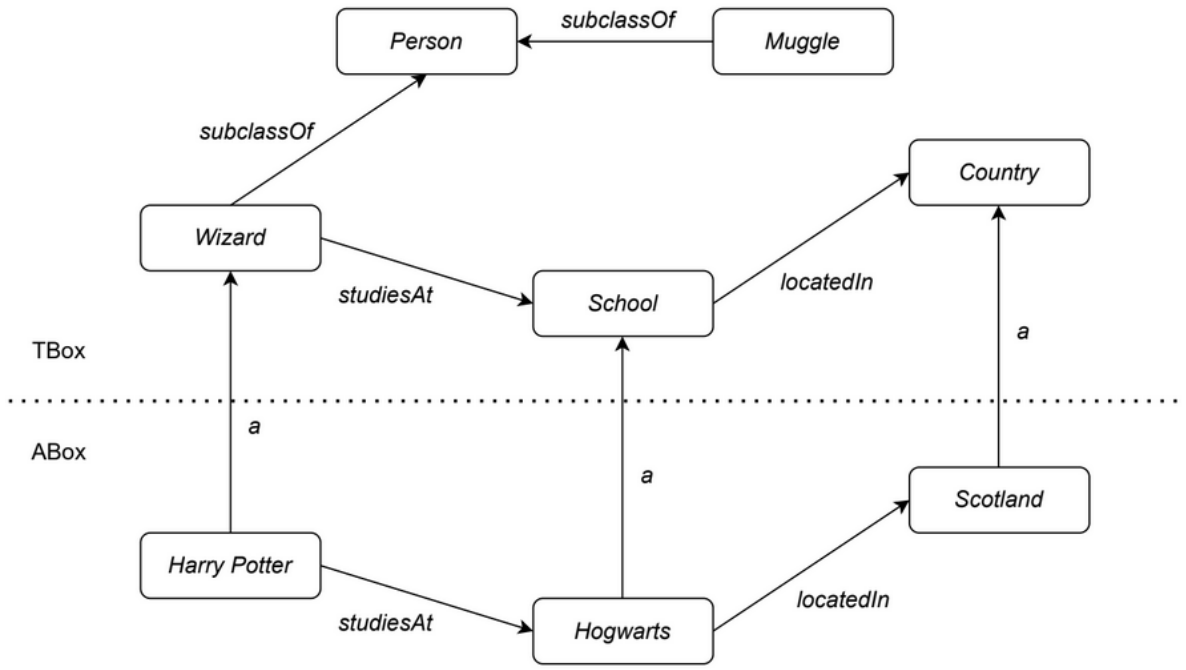

Figure 3.2: Example knowledge graph with separate TBox and ABox.

3.3 Neuro-symbolic approaches

Another group of rule learning approaches are the *neuro-symbolic* approaches which try to simultaneously learn both rule structures and parameters, resulting in a combined approach of path-based and embedding-based strategies. One of the first methods to introduce such a framework is Neural Logic Programming (Neural LP) [32]. For a broad discussion of neuro-symbolic approaches, we refer the reader to this paper [33].

3.3.1 Static neuro-symbolic approaches

While ScaLeKB [29] and RARL [31] use type information in rule search and evaluation, they do not learn types rules, only enforce types on them, which makes such hard constraints too restrictive. The TyRuLe [34] algorithm introduces a neuro-symbolic approach by incorporating the typed information of knowledge graphs into the rules through embeddings. This embedding-based rule learning method uses the type information of knowledge graphs to enhance rule learning without using hard type constraints. This enables TyRuLe to learn typed rules, even when the typed information is incomplete.

For enormous knowledge graphs, several studies have explored the idea of a scalable approach to rule learning. For example, the neurosymbolic approach RLvLR [35] is much faster and learns rules of much higher quality compared to AMIE+ [12]. It combines neural network models with rule learning strategies through knowledge graph embeddings, providing a scalable approach for learning first-order logical closed-path rules in large knowledge graphs. To achieve this, embedding models are used to prune the search space effectively, and a sampling algorithm limits the number of relevant entities for the embeddings to accomplish this. In that way, embeddings are computed only for those relevant to the target. Moreover, a scoring function is introduced that enables fast ranking of rules based on the embeddings of the entities, relations, and arguments.

These aspects contribute to the scalability of RLvLR in handling large knowledge graphs with more than 10 million facts. This research [36] also provides an embedding-based rule extraction approach that outperforms TransE [13] for the link prediction task on Freebase [3]. It shows that a formulation of a bilinear model can outperform the state-of-the-art. Upon analyzing the embeddings, logical rules from the knowledge graph are extracted.

Another example of a static neuro-symbolic approach is RNNLogic [37], which uses a neural network-based rule generator and reasoning predictor that work cooperatively. First, the reasoning predictor is updated to explore rules of reasoning. Then, a set of high-quality rules is selected. Lastly, the rule generator is updated with the selected rules. With the creation of rules by the generator allows for computation of the likelihood of answers conditioned on the rules and the existing knowledge.

3.3.2 Dynamic neuro-symbolic approaches

StreamLearner [38] presents an approach to learning temporal rules from knowledge graph streams which can be applied in link forecasting. It is created by extending RLvLR [35] to learn the temporal rules of a temporal knowledge graph using timestamps. A knowledge graph stream captures the relations in a temporal knowledge graph for a specified start time and end time. The stream is divided into a separate graph for each timestamp in the knowledge graph stream. Then, the algorithm extracts rules by doing static random walks on each of those graphs. As a consequence, all body atoms in the rules have the same timestamp.

TLogic [39] is a framework also based on the extraction of temporal logical rules by *non-increasing* temporal random walks. This model only operates on temporal data in the form of timestamps, just like StreamLearner [38]. A *non-increasing* temporal random walk is a walk where the timestamps of the walk remain constant or decrease over time. Thus, a walk is created by moving only backward in time along the edges or moving over edges with identical timestamps. Furthermore, these timestamps are fixed during training leading to the obstruction of learning temporal constraints. It aims to tackle the problem of link prediction.

TILP [40] proposes a neuro-symbolic framework for learning temporal logical rules on temporal knowledge graphs. It uses neurally generated symbolic representations of entity and temporal relationships. The relation between two time intervals $I = [t_s, t_e]$ and $I' = [t'_s, t'_e]$ can be either classified in one of three groups; *before* meaning that $t_e < t'_s$, *after* which is the inverse of before, or *touching* which are the other eleven types of relations according to Allen's interval algebra. TILP is a neural network-based framework that learns temporal rules based on constrained random walks without restrictions on temporal knowledge graphs.

Approach		Static	Dynamic
Subsymbolic	*Translation-based*	TransE [13]	TTransE [18]
	Tensor factorization based	ComplEx [14]	TNTComplEx [19] TimePlex [20]
	Neural network-based	Structure Embeddings [15] R-GCN [16] ConvE [17]	HyTE [21] RE-NET [22]
Symbolic	*ILP-based*	QuickFOIL [23] ALEPH [24] AMIE+ [12] Nonmonotonic RL [25] [26]	
	Path-based	AnyBURL [11] Rudik [27] [28] OP (ScaLeKB) [30] [29] RALR [31]	StaTeR
Neuro-symbolic		NeuralLP [32] TyRuLe [34] RLvLR [35] Embeddings and Learning [36] RNNLogic [37]	StreamLearner [38] TLogic [39] TILP [40]

Table 3.1: Categorization of the models described in Chapter 3. StaTeR is categorized in the symbolic path-based dynamic cell, which is the first of its kind.

Chapter 4

Problem description

This chapter provides the problem description and introduces the proposed system, StaTeR, a symbolic temporal rule learning algorithm designed to address these challenges.

Temporal knowledge graphs are often incomplete, limiting their effectiveness in tasks that rely time-sensitive knowledge. For this reason, temporal knowledge graph completion is studied, which focuses on inferring missing information in temporal knowledge graphs. A key task within knowledge graph completion is the task of temporal link and time prediction. This task involves predicting missing elements of a temporal fact of the form: *predicate*(*subject*, *object*, *start time*, *end time*). The tasks of link and time prediction are detailed in Section 5.7.

There exist various approaches for tKGC, including embedding-based models as described in Section 3.1 and symbolic reasoning techniques as described in Section 3.2. In this thesis, we focus on rule learning, specifically developing a purely symbolic temporal rule learning algorithm.

StaTeR starts with learning static logical rules using a symbolic rule learner. This rule learner is applied to a temporal dataset where the temporal component is removed and duplicate static facts are discarded. Then, the static rules are filtered based on a confidence threshold, and rules that do not meet this threshold are removed. The rules that are left are then extended with temporal components. For these temporal rules, the various temporal confidences, as described in Section 5.2, can be calculated. Finally, these temporal rules are applied to the link and time prediction tasks, producing ranked lists of entities or time intervals. For link prediction, the rankings are determined by the confidence values of the rules that predict them multiplied by either the aeIOU or the TAC factor calculated based on the instances of the time components of the rule. For time prediction, the intersection of the time components of the rule are used for the prediction and the confidence of the rules that predict them is used for the ranking of the rules. In case the rule only has one body atom, the time component of the body atom is used for the time prediction. In the final prediction, the highest ranked result represents the most likely prediction. The link and time prediction processes are discussed further in Section 5.7.

The pseudo-code for StaTeR is described in Algorithm 1.

Algorithm 1 StaTeR - Static to Temporal Rule learning

Input: Temporal knowledge train graph $\mathcal{G}_t$, temporal knowledge evaluation graph $\mathcal{G}_e$.
Parameters: Confidence threshold t_c, temporal confidence measure t_m.
Output: Temporal rules R_t with temporal confidences, ranked predictions.
1. Convert $\mathcal{G}_t$ into a static knowledge graph $\mathcal{G}'_t$ by stripping temporal components.
2. Apply a symbolic rule learner to $\mathcal{G}'_t$ to generate static logical rules R_s.
3. For each rule $r_s \in R_s$, filter out those with confidence values below threshold t_c.
4. For each rule $r_s \in R_s$, calculate the temporal confidence based on the chosen temporal confidence metric t_m and construct the temporal rule set R_t. Possible temporal confidence measures include:
   - *Naive* confidence: See Equation 2.2.
   - *Naive Overlap* confidence: See Equation 5.2.
   - *Same Interval* confidence: See Equation 5.3.
   - *Intersection Over Union* confidence: See Equation 5.4.
   - *Temporal Alignment Coefficient* confidence: See Equation 5.5.
5. Perform link and time predictions for each query $q \in \mathcal{G}_t$:
   - Link prediction: Predict the missing entity of q using R_t.
   - Time prediction: Predict the missing temporal interval of q using R_t.
6. Output for each query $q \in \mathcal{G}$:
   - For link prediction: The link predictions ranked on the confidence values.
   - For time prediction: A predicted time interval.

Chapter 5

Methodology

This Chapter explains the methodology of the thesis, developing the purely symbolic temporal rule learning algorithm StaTeR. First, the process of transforming static logical rules into temporal logical rules is described. Next, we outline how these temporal logical rules are evaluated using different quality measures of a rule. Furthermore, we present the datasets that have been used, along with an overview of their structure and characteristics. Lastly, it is discussed how the temporal logical rules are used for link and time prediction, incorporating all the concepts and methods introduced earlier.

5.1 Temporal logical rules

To turn static logical rules into temporal logical rules, we extend the definition of Rule 1 to define a temporal closed rule path of length n. This temporal rule incorporates temporal information, which takes the following form:

$$h(A_0, A_n, I_n) \longleftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i, I_i) \bigwedge_{i=1}^{n-1} \left(\bigwedge_{j=i+1}^{n} TR_{i-1,j}(I_{i-1}, I_j) \right) \tag{12}$$

In the temporal rule above, I represents the temporal component of each atom, which can be either a timestamp of the time interval. For time intervals, $I = [t_s, t_e]$ where t_s is the start time and t_e is the end time. Similarly to the static closed path rule represented in Rule 1, this rule connects the subject and object of the head through the body atoms. Furthermore, temporal relations (denoted as TR) are created between all temporal components of the rule, including the head. By creating a double for loop mechanism, the rule connects all the temporal components of the rule. Below are examples of temporal rules of different lengths, illustrating the structure.

For length 1:

$$livesIn(X, Y, I_1) \longleftarrow studiesIn(X, Y, I_0) \wedge TR(I_0, I_1)$$

For length 2:

$$studiesIn(X, Y, I_2) \longleftarrow$$
$$friendOf(X, A, I_0) \wedge studiesIn(A, Y, I_1)$$
$$\wedge TR(I_0, I_1) \wedge TR(I_0, I_2) \wedge TR(I_1, I_2)$$

For length 3:

$$studiesIn(X, Y, I_3) \longleftarrow$$
$$friendOf(X, A, I_0) \wedge friendOf(A, B, I_1) \wedge studiesIn(B, Y, I_2)$$
$$\wedge TR(I_0, I_1) \wedge TR(I_0, I_2) \wedge TR(I_0, I_3)$$
$$\wedge TR(I_1, I_2) \wedge TR(I_1, I_3) \wedge TR(I_2, I_3)$$

The pattern continues for increasing length by creating more body atoms and temporal relations as required.

The *TR* atom defines the relation between two time components of the rule. Depending on whether timestamps of time intervals are used, the temporal relations can differ. For timestamps, possible relations are *before*, *after*, and *equal*. For time intervals, there are thirteen temporal relations defined by Allen's Algebra [10]. All these temporal relations are discussed more deeply in Section 2.3.

However, using these specific temporal relations results in very specific rules, especially for longer rules. Although this increased their precision, it also makes them computationally expensive to evaluate and apply. As illustrated above, as the length of the rule increases, the number of temporal relation atoms grows significantly. The growth pattern can be described as:

$$\binom{n}{2}$$

This function computes the number of unique pairs that can be formed from n items, where the order of selection does not matter. This is exactly what we need to combine all the time components of the rule. This function simplifies to:

$$\binom{n}{2} = \frac{n^2 - n}{2}$$

In this simplified function, it is observed that for small n, the growth appears to be linear. However, as n becomes larger, the term n^2 dominates the function leading to quadratic growth. Therefore, it is computationally expensive and challenging to find groundings for such rules with many body atoms.

5.2 Quality measures of a temporal rule

Now that the static logical rules have been converted into temporal logical rules, some new rule quality measures for a rule have to be introduced. The newly added time dimension complicates rule evaluation, as the quality measures described in Section 2.6 do not incorporate the time component. Using the static support and confidence calculations

produces incorrect results. For example, consider the small temporal knowledge graph in Table 5.1 consisting of instances of the relations *worksIn* and *livesIn*.

$$\begin{aligned}
&\textit{worksIn}(\textit{Harry Potter}, \textit{London}, 1998, 2002)\\
&\textit{worksIn}(\textit{Harry Potter}, \textit{London}, 2005, 2006)\\
&\textit{livesIn}(\textit{Harry Potter}, \textit{London}, 1998, 2002)\\
&\textit{livesIn}(\textit{Harry Potter}, \textit{London}, 2005, 2008)\\
&\textit{livesIn}(\textit{Harry Potter}, \textit{London}, 2010, 2012)\\
&\textit{livesIn}(\textit{Harry Potter}, \textit{London}, 2015, 2020)
\end{aligned}$$

Table 5.1: Example knowledge graph containing six facts using the the two entities *Harry Potter*, and *London*, and the two relations *livesIn* and *worksIn*.

Consider the following rule:

$$\textit{livesIn}(X, Y, I_1) \longleftarrow \textit{worksIn}(X, Y, I_0), \textit{TR}(I_0, I_1) \tag{13}$$

The support of a rule is defined as the number of instances in which both the head and body of a rule are satisfied, as stated in Equation 2.1. In the context of our example, the first fact (*worksIn*(*Harry Potter*, *London*, 1998, 2002)) would match with all facts with the relation *livesIn*, and vice versa for the second fact. This would result in a support of 4 + 4 = 8. Furthermore, the confidence of a rule is defined by dividing the support by the number of satisfied body instances, as stated in Equation 2.2. The number of satisfied body instances is 2. This would mean that the confidence is equal to $\frac{8}{2} = 4$. This is incorrect since confidence is another way to describe probability, and the probability of something can never be higher than 1. This initial definition of confidence calculated on a temporal knowledge graph, will be referred to as the *Naive temporal confidence*. The error in this calculation arises from matching body groundings with head groundings while their temporal information does not align or are unrelated. Without considering temporal relations, the support and therefore the confidence overestimates the rule's probability. Therefore, a correct redefinition of temporal confidence for a rule r, $\textit{temporalConf}(r)$, should satisfy the confidence inequality:

$$0 \leq \textit{temporalConf}(r) \leq 1 \tag{5.1}$$

The next Subsections will introduce different quality measures of a temporal rule.

5.2.1 Naive Overlap confidence

The original confidence calculation is given by Equation 2.2. To adapt this equation to work for temporal rules, the calculation of the support has to be adjusted. Specifically, the calculation for the support must be redefined to incorporate the temporal components correctly. The original support calculation, given by Equation 2.1, does not account for the overlap of the temporal components. As described in Section 5.2, it is insufficient to count rule instances that do not have any alignment of the temporal components. To

address this limitation, we introduce the *Naive Overlap confidence* that modifies the support calculation to incorporate only instances with overlap of their temporal components. The Naive Overlap confidence is computed by the following calculation:

$$\mathit{NaiveOverlapConf}(r) = \frac{|\{(I_i, I_j) | I_i, I_j \in \mathcal{I}(r), I_i \bigcap I_j \neq \emptyset, i \neq j\}|}{|x_B(r)|} \tag{5.2}$$

In this formula, $\mathcal{I}(r)$ represents the set of all temporal components related to a particular instance of the rule r. Furthermore, the pair (I_i, I_j) denotes all pairwise combinations of those temporal components in $\mathcal{I}$. In this way, only the support is taken into account of instances that have overlap of the temporal components of the rule for every pair of temporal components. Calculating the Naive Overlap confidence for rule r with the knowledge graph of Table 5.1, results in:

$$\mathit{NaiveOverlapConf}(r) = \frac{|\{((1998, 2002), (1998, 2002)), ((2005, 2006), (2005, 2008))\}|}{2} = 1$$

However, this approach is called the "Naive" Overlap confidence because this formula still has a significant limitation, since it can result in confidence values higher than 1 in certain situations. This is demonstrated in the following example temporal knowledge graph:

worksIn(*Harry Potter*, *London*, 1998, 2002)
livesIn(*Harry Potter*, *London*, 1998, 1999)
livesIn(*Harry Potter*, *London*, 2001, 2001)

Table 5.2: Example knowledge graph containing three facts using the the two entities *Harry Potter*, and *London*, and the two relations *livesIn* and *worksIn*.

Using Rule 13 as r, the calculation would result in:

$$\mathit{NaiveOverlapConf}(r) = \frac{|\{((1998, 2002), (1998, 1999)), ((1998, 2002), (2001, 2002))\}|}{1} = 2$$

A confidence value of greater than 1 is calculated because the temporal intervals of both the first and the second fact have some overlap with the temporal interval of the third fact. This aligns with the definition of Naive confidence. Despite the intention of temporal alignment, the Naive Overlap confidence can compute values greater than 1, which is incorrect for the use of confidence. Such examples highlight the need for further refinement to correctly constrain the temporal confidence calculation and ensure meaningful results. Thus, the temporal inequality of Equation 5.1 is not satisfied for Naive Overlap confidence.

5.2.2 Same Interval confidence

The *Same Interval confidence* introduces a stricter constraint to calculate the temporal confidence. In this definition, we require that every temporal component of the body

matches the temporal component of the head. In other words, every temporal component of the rule instance is the same. This is given by the following formula:

$$sameIntervalConf(r) = \frac{|\{I_0, ..., I_n | I_0 = I_1 = ... = I_n\}|}{|x_B(r)|} \tag{5.3}$$

Here, $\{I_0, ..., I_n\}$ represents the set of temporal components of a rule instance. The denominator only counts the rule instances in which all temporal components are identical. This new definition of temporal confidence ensures that the inequality of Equation 5.1 is satisfied. This is due to the definition of a temporal knowledge graph; Similar static facts cannot have overlap in their temporal component, as explained in Section 2.1.

5.2.3 Intersection Over Union confidence

In this redefinition of temporal confidence, the *Intersection Over Union* (IOU) *confidence* is introduced to address the limitation of binary overlap measures. Instead of checking wether temporal components match exactly, this approach quantifies the proportion of overlap relative to the total union of temporal components. The IOU confidence is given by the following formula:

$$IOUConf(r) = \frac{\sum_{(I_i, I_j) \in \mathcal{I}(r), I_i \bigcap I_j \neq \emptyset} \frac{I_i \bigcap I_j}{I_i \bigcup I_j}}{|x_B(r)| \times \binom{|\mathcal{I}(r)|}{2}} \tag{5.4}$$

In this definition, $x_B(r)$ refers to all instances satisfying the body of the rule. Furthermore, $I_i \bigcap I_j$ represents the intersection of the two intervals, the time where the intervals overlap, and $I_i \bigcup I_j$ refers to the union of the two intervals, the time covered by both intervals. Therefore, the summation in the numerator calculates the Intersection Over Union ratios for each pair of temporal components that have nonempty intersection. In the denominator, $\binom{|\mathcal{I}(r)|}{2}$ corresponds to the number of unique pairs in $\mathcal{I}(r)$. In other words, it corresponds to the number of temporal relations in rule r. Note that $\binom{|\mathcal{I}(r)|}{2}$ is only used if $|\mathcal{I}(r)| > 1$, otherwise it is treated as 1. This is because $\binom{1}{2}$ is undefined since it is impossible to pick two items out of a group of one. Calculating the IOU confidence of Rule 13 for the temporal knowledge graph of Table 5.2, results in the following:

$$IOUConf(r) = \frac{\frac{2}{5} + \frac{1}{5}}{2 \times 1} = 0.3$$

The correction in the denominator with the value $\binom{|\mathcal{I}(r)|}{2}$ is needed because the numerator can be maximum the number of temporal relations, while the body coverage of the rule instance has a minimum of one. To illustrate this, consider the following example temporal knowledge graph:

studiesAt(*Harry Potter*, *Hogwarts*, 1991, 1997)

friendOf(*Harry Potter*, *Ron Weasley*, 1991, 1997)

studiesAt(*Ron Weasley*, *Hogwarts*, 1991, 1997)

Table 5.3: Example knowledge graph containing three facts using the the three entities *Harry Potter*, *Hogwarts*, and *Ron Weasley*, and the two relations *studiesAt* and *friendOf*.

Furthermore, consider the rule r constructed as:

$studiesAt(X, Y, I_2) \longleftarrow$
$friendOf(X, A, I_0) \wedge studiesAt(A, Y, I_1) \wedge TR(I_0, I_1) \wedge TR(I_0, I_2) \wedge TR(I_1, I_2)$

When calculating the IOU confidence without correcting for the number of temporal relations, the following is calculated:

$$IOUConf(r) = \frac{\frac{8}{8} + \frac{8}{8} + \frac{8}{8}}{1} = 3$$

Thus, without correcting for the number of temporal relations of a rule, the formula would give an overestimation of the numerator, which could result in a confidence value greater than 1. The IOU confidence formula with correction for the number of temporal relations of a rule given in Equation 5.4, satisfies the temporal inequality of Equation 5.1.

5.2.4 Temporal Alignment Coefficient confidence

The *Temporal Alignment Coefficient* (TAC) *confidence* focuses on the start and end times of temporal intervals. It quantifies the alignment based on the absolute temporal differences between their start and end times. Where the IOU confidence is more focused on the direct measure of overlap between two time intervals, the TAC confidence captures the alignment by focusing on the relative distances of start and end times. The TAC confidence is particularly useful to assess the temporal closeness between two time intervals, even if the time intervals do not overlap. The TAC confidence is constructed as follows:

$$TACConf(r) = \frac{1}{|x_B(r)| \times \binom{|\mathcal{I}(r)|}{2}} \sum_{(I_i, I_j) \in \mathcal{I}(r)} \left(0.5 \times \left(\frac{1}{1 + |s_i - s_j|} + \frac{1}{1 + |e_i - e_j|}\right)\right) \tag{5.5}$$

In this definition, s_x and e_x refer to the start and end times of the time interval I_x. Again, with the restraint that $\binom{|\mathcal{I}(r)|}{2}$ is only used if $|\mathcal{I}(r)| > 1$, otherwise it is treated as 1. When calculating the TAC confidence for Rule 13 for the temporal knowledge graph of Table 5.2, the following is calculated:

$$TACConf(r) = \frac{1}{2 \times 1}\left(0.5\left(1 + \frac{1}{4}\right) + 0.5\left(\frac{1}{4} + \frac{1}{2}\right)\right) = 0.5$$

If all time intervals of the rule are equal, then all pairwise combinations of time intervals yield equal temporal differences. In such cases, the summation results in the number of unique pairs in $\mathcal{I}(r)$, thus $\binom{|\mathcal{I}(r)|}{2}$. The TAC confidence is normalized by correcting for the number of unique pairs of time intervals of the rule. In this way, the temporal inequality of Equation 5.1 is satisfied for TAC confidence.

5.3 Data collection

5.3.1 Data statistics

To calculate the temporal confidences of the rules, the rules themselves must first be obtained. However, for the creation of rules, access to a dataset is required. The Wikidata [41] and YAGO [42] knowledge graphs incorporate the time intervals on a subset of

facts. The real-world temporal knowledge graph dataset Wikidata12k is extracted from Wikidata explained by [18]. YAGO11k is also a real-world temporal knowledge graph generated from YAGO proposed in [43].

	Wikidata12k	**YAGO11k**
# entities	12,554	10,623
# relations	24	10
# train facts	32,497	16,408
# valid facts	4,062	2,050
# test facts	4,062	2,051

Table 5.4: Dataset statistics of datasets Wikidata12k and YAGO11k.

Table 5.4 provides a summary of the statistics of the Wikidata12k and the YAGO11k dataset. Both datasets are divided into training, validation, and test sets that contain entities and relations. The entities and relations in the datasets have been represented by unique identifiers (IDs) rather than their descriptive names. The meaning of these IDs is provided in the *entity2id* and *relation2id* files, where each ID is mapped to its corresponding entity or relation of the dataset. For example, the following mappings are in the Wikidata12k dataset:

Q202735 2454
P1376 0

Here, the entity ID 2454 in the dataset refers to the Wikidata entity Q202735. Visiting the website `https://www.wikidata.org/wiki/Q202735` reveals that this entity refers to "Billy Bob Thornton", an American actor, filmmaker, and musician. The other mapping is a relational mapping where the relation ID 0 corresponds to the property P1376. This refers to the relation "capital of" as can be seen on the website `https://www.wikidata.org/wiki/Property:P1376`. For the Yago11 dataset, a similar approach is used. However, in these mappings, the IDs refer directly to the actual name of the entity or relation. For example:

Jane_Bryan 3605 1918-06-11 2009-04-08
wasBornIn 0

As can be seen above, for the YAGO11k dataset, entity mappings also include temporal information. For people, for example, the start and end times refer to birth and death dates respectively. It should be noted that for the YAGO11k dataset, the end dates are mostly undefined, and some entities do not have start or end dates.

For both datasets, the temporal data has been simplified to just the year, where the specific day and month information have been removed.

5.3.2 Data cleaning

Upon running the different calculations for the temporal confidence, irregular results were observed. For this thesis, the static rules of AnyBURL were used for the experiments. The following static rule r was created by AnyBURL for the Wikidata12k dataset:

$14(4028, Y) \longleftarrow 14(4851, Y)$

Using the mappings provided in the dataset and referencing Wikidata, it translates to:

"If *Édouard Courtial* held the position Y,
then *Christian Bataille* held the position Y."

Converting this static rule to its temporal equivalent results in the following:

$14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$

For this rule, the head and body instances from the dataset are displayed in Table 5.5.

Head Instances				**Body Instances**			
4028	14	3498	1997-2001	4851	14	3498	2003-2015
4028	14	3431	1988-1993	4851	14	3431	2002-2007
4028	14	3431	1993-1997	4851	14	3431	2007-2011
4028	14	3431	1997-2002	4851	14	3431	2012-2012
4028	14	3431	2002-2007	4851	14	3431	2012-2017
4028	14	3431	2007-2012				
4028	14	3431	2012-2017				

Table 5.5: Instances for the rule $14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$. Head instances are displayed on the left and body instances on the right.

From this, two groundings for Y in r are identified; 3498 and 3431. The ID 3498 refers to the entity *Mayor of a place in France* and ID 3431 corresponds to the entity *Member of the French National Assembly*.

As described in Section 2.1, overlapping of temporal components in facts with identical subject, predicate, and object create ambiguity. In the set of facts above, some static identical static facts have overlap in their time interval. For example, in the year 2012 the relations $14(4028, 3431)$ and $14(4851, 3431)$ both occur twice. Furthermore, if the relation $14(4851, 3431)$ is true from 2002 to 2007 and from 2007 to 2011, it is equivalent to stating that the relation is true from 2002 to 2011. Removing such redundancies in the body, lowers the number of body instances of a rule, and thereby increases the confidence for every temporal approach.

Moreover, temporal confidence calculations for the Naive confidence as well as the IOU confidence for datasets with such overlap of temporal time intervals leads to overestimating the temporal confidence. Both calculations for the rule $14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$ using the instances from Table 5.5 are shown below, where in each time interval pairing, the body time interval is listed first.

$$NaiveConf(r) = \frac{\left|\begin{array}{l}\{((2002,2007),(1997,2002)),((2002,2007),(2002,2007)),\\ ((2002,2007),(2007,2012)),((2007,2011),(2002,2007)),\\ ((2007,2011),(2007,2012)),((2012,2012),(2007,2012)),\\ ((2012,2012),(2012,2017)),((2012,2017),(2007,2012)),\\ ((2012,2017),(2012,2017))\}\end{array}\right|}{5} = 1.8$$

$$IOUConf(r) = \frac{(\frac{1}{11} + \frac{6}{6} + \frac{1}{11} + \frac{1}{10} + \frac{5}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{11} + \frac{6}{6})}{5 \times 1} \approx 0.71$$

In these calculations, the Naive confidence is calculated as 1.8 and the IOU confidence is calculated as approximately 0.71. In Table 5.6 is the reduced set of facts of instances of the rule, where the overlapping time intervals of identical static facts are merged.

Head Instances				**Body Instances**			
4028	14	3498	1997-2001	4851	14	3498	2003-2015
4028	14	3431	1988-2017	4851	14	3431	2002-2011
				4851	14	3431	2012-2017

Table 5.6: Instances for the rule $14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$ with merged intervals. Head instances are displayed on the left and body instances on the right.

Below are calculations of the Naive confidence and IOU confidence of rule $14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$ with the instances of Table 5.6.

$$NaiveConf(r) = \frac{|\{((2002,2011),(1988,2017)),((2012,2017),(1988,2017))\}|}{3} \approx 0.67$$

$$IOUConf(r) = \frac{(\frac{10}{30} + \frac{6}{30})}{3 \times 1} \approx 0.21$$

As can be seen in the calculations above, the cleaning of overlapping temporal intervals in the dataset has a significant impact on the values of the temporal confidence calculations. Considering the rule $14(4028, Y, I_0) \longleftarrow 14(4851, Y, I_1), TR(I_0, I_1)$, the Naive temporal confidence dropped from 1.8 to 0.67, and the IOU confidence also decreased from 0.71 to 0.21. Calculating the Naive confidence and IOU confidence of r using the uncleaned dataset results in overestimation caused by duplicated time intervals. Table 5.7 lists the updated number of the train, valid, and test sets after merging the overlapping time intervals of identical static facts for the Wikidata12k dataset.

	Wikidata12k		YAGO11k	
# entities	12,554		10,623	
# relations	24		10	
# train facts	32,497	31,713	16,408	16,398
# valid facts	4,062	4,042	2,050	2,050
# test facts	4,062	4,043	2,051	2,051

Table 5.7: Dataset statistics of datasets Wikidata12k and YAGO11k after merging overlapping time intervals of identical static facts and removing facts with starting times before the year 0. The left column of each dataset column represents the original number of facts and the right column represents the number of facts after cleaning.

The YAGO11k dataset did not contain overlapping time intervals for identical static facts. However, there are still 10 facts filtered from the train set during cleaning. This is because those facts have start or end times before the year 0. These facts need to be filtered out due to a limitation of the rdflib library, which cannot handle dates earlier than the year 0. The rdflib library is used for constructing the temporal knowledge graph of the dataset. Furthermore, creating the temporal knowledge graph requires both start and end times for all facts. However, not all facts have a start time, end time, or both. To solve this, the earliest and latest dates in the dataset are found and used to substitute any missing start time or end time respectively. More on the creation of the temporal knowledge graph is explained Section 5.4.

The initial static rules are learned by a symbolic rule learner, which requires unique static training facts as input. Therefore, a separate file containing only the unique static facts was created for the rule learner to generate rules from. The specific number of unique training facts used is presented in Table 5.8.

	Wikidata12k	YAGO11k
# temporal train facts	32,497	16,408
# unique static train facts	27,774	16,408

Table 5.8: Number of unique training set facts of datasets Wikidata12k and YAGO11k.

5.4 Temporal knowledge graph creation

An RDF (Resource Description Framework) graph is a standard model for representing structured information about resources on the web, such as temporal knowledge graphs. A general function is designed to construct temporal knowledge graphs for the temporal data of both the datasets Wikidata12k and YAGO11k. The input consists of quintuples that specify subject, predicate, object, and their start and end times. The function supports both datasets by dynamically switching URI namespaces and predicate mappings based on the input for the dataset name. For the creation of temporal knowledge graphs a method called "reification" is needed since both start and end times describe a triple.

Reification is a way of treating a statement itself as an object. It is implemented by using a unique URI for each statement, linking the subject, predicate, and object of the triple through the properties rdf:subject, rdf:predicate, and rdf:object. Lastly, temporal data is added by using the time:hasStart and time:hasEnd predicates, with stores dates as literals obeying to the XML Schema date datatype.

The original data from the Wikidata12k dataset and its corresponding RDF statement for the temporal knowledge graph are provided below.

10924 22 10926 2012-##-## 2014-##-##

⟨urn:uuid:{ID}⟩	rdf:object	⟨http://wikidata.org/entity/10926⟩;
	rdf:predicate	⟨http://wikidata.org/prop/direct/22⟩;
	rdf:subject	⟨http://wikidata.org/entity/10924⟩;
	time:hasEnd	"2014-01-01"^xsd:date;
	time:hasStart	"2012-01-01"^xsd:date.

5.5 Filter static logical rules

Upon creation of the knowledge graph of the training set, static rules are generated using a symbolic rule learner. These static rules serve as the foundation to derive temporal rules by reintroducing the temporal dimensions and calculating the various temporal confidences. The static rules are initially evaluated based on their confidence values. If a static rule has low confidence, the corresponding temporal rule will never perform better. Temporal confidence is bounded by the confidence of the corresponding static rule. This is because temporal rules represent subsets of the static rule's coverage, with added temporal constraints. Therefore, for a static rule r_s and its corresponding temporal rule r_t, the following inequality is introduced:

$$0 \leq \mathit{temporalConfidence}(r_t) \leq \mathit{staticConfidence}(r_s) \tag{5.6}$$

To address this, a confidence threshold is introduced which filters out those static rules which have a confidence value of below the confidence threshold. For example, consider a confidence threshold of 0.1. Any static rule with a confidence score less than 0.1 is discarded as it is considered weak. Temporal rules derived from such static rules cannot achieve better results considering Equation 5.6. This reduces the number of rules to process, and thereby saving computational resources. At the same time, accuracy is preserved since these rules were weak to begin with.

5.6 Calculate temporal confidences

After filtering the static rules whose confidence does not exceed the confidence threshold, the next step is to calculate the temporal confidences for each rule. This thesis considers the five temporal confidence quality measures described in Section 5.2. To compute these temporal confidences, a SPARQL query is formulated for each quality measure. SPARQL

(SPARQL Protocol and RDF Language) is a semantic query language designed to retrieve data stored in RDF format. It enables efficient querying of (temporal) knowledge graphs by using patterns to extract information. Each SPARQL query is created to compute one of the temporal support quality measures by accessing the temporal aspects of the temporal knowledge graph. Once the temporal support is determined, the temporal body coverage is queried, and the confidence is calculated by dividing the support by the body coverage. The exact SPARQL queries along with an example query for each quality measure can be found in Appendix A.

5.7 Prediction

Once the confidence values of the temporal rules have been computed for each quality measure, the effectiveness of the temporal rules can be evaluated through link and time prediction. To assess the predictive power of the rules, a test set consisting of facts is used. For each fact in the test set, the subject, object, or temporal component is removed to create an incomplete fact. The goal is to infer the missing element of the incomplete fact using the temporal rules. Predicting a missing subject or object is referred to as *link prediction*, while predicting a missing temporal component is known as *time prediction*.

For each prediction task, a ranked list of potential candidates for the missing element of the incomplete fact is generated. For link prediction, this ranking is ordered based on the confidence values associated by the rules used to generate the candidates multiplied by either the aeIOU or the TAC factor of the time components of the rule. For time prediction, the confidences of the rules that predict the time are used for the ranking of the rules. If two predictions have the same confidence value, then the confidence values of next rule that predicts each of those candidates is compared. This process continues until all candidates are ranked. In this thesis, the top 10 predictions for each prediction task are saved. By comparing the ranked list with the actual values, the accuracy of the rules can be assessed. The predictions can be evaluated in many ways, such as the quality measures *hits@k* and *Mean Reciprocal Rank* (MRR) for link prediction, and *affinity enhanced IOU* (aeIOU) and *Temporal Alignment Coefficient* (TAC) for time prediction.

5.7.1 Link prediction

Link prediction focuses on predicting either the subject or object from an incomplete test fact. For example, this is a test fact from the test set:

10924 22 10926 2012 2014

For predicting the object, the true object 10926 is removed from the triple resulting in an incomplete fact. Rules are searched for heads of the form $22(10924, Y)$, since they predict an object for the incomplete fact. Furthermore, rules with heads of the form $22(X, 10926)$ are also identified. These rules are grounded using the object of the test fact, since they then also predict an object for the incomplete fact. For example, a rule such as $22(X, 10926) \longleftarrow 22(X, 10143)$ is grounded by substituting $\theta_{X=10924}$ resulting in $22(10924, 10926) \longleftarrow 22(10924, 10143)$. This process abstracts the order of the variables, considering flipped-variable rules without explicitly enumerating them.

After generating the list of applicable rules for predicting the object, each rule is evaluated against the training set. For each rule, it is checked whether its body exists in the training

graph. If a match is found, the rule's prediction is saved along with the confidence value of the rule. Additionally, the confidence value is adjusted by the factor TAC or the factor aeIOU. This adjustment favors rules whose instances closely align with the time interval of the test fact. The TAC factor quantifies the degree of alignment between two intervals and is defined as:

$$TAC(I_i, I_j) = 0.5 \times \left(\frac{1}{1 + |s_i - s_j|} + \frac{1}{1 + |e_i - e_j|} \right) \tag{5.7}$$

In this definition, s_x and e_x refer to the start and end times of the time interval I_x. The aeIOU factor, proposed by [44], for two time intervals is defined as:

$$aeIOU(I_i, I_j) = \frac{\max\{1, I_i \bigcap I_j\}}{I_i \bigcup I_j} \tag{5.8}$$

In this definition, $I_i \bigcap I_j$ represents the intersection of the two intervals, the time where the intervals overlap, and $I_i \bigcup I_j$ refers to the union of the two intervals, the time covered by both intervals. This equation differs from the original IOU formula for intervals by ensuring that, even when there is no overlap between the intervals, the smallest granularity of time in the dataset is used for the intersection. In the case of the datasets Wikidata12k and YAGO11k, this would be 1, representing 1 year. At first glance, it may be counterintuitive that if there is no overlap, the numerator is still set to one. However, this approach favors predictions that are close but have no overlap. For example, consider the true interval [2010, 2012] and the predictions [2008, 2009] and [1990, 1994]. The aeIOUs the predictions would be $\frac{1}{5}$ and $\frac{1}{33}$ respectively, reflecting the relative closeness of the first prediction compared to the second. The term *Intersection Over Union* can be slightly misleading since the prediction of [2005, 2005] for true value [2006, 2006] results in $\frac{1}{2}$ while they do not intersect at all.

The confidences of the temporal rules are then adjusted by multiplying it with either the TAC factor or the aeIOU factor. In case the rule has multiple body atoms, the factors are calculated as the sum of factors between all pairwise combinations of temporal intervals corrected by the number of pairwise combinations. Finally, the list of predictions is ordered based on the new confidence values of the rules that predict it.

To assess the predictions for link prediction, the metrics hits@k and MRR are used. Hits@k is a metric used to evaluate the top k predictions. It is the fraction of times the true predicted value appears in the top k ranked predictions out of all test queries. Formally, hits@k is calculated as:

$$hits@k = \frac{\textit{Count of test queries with the true value among the top k predictions}}{\textit{Total number of test queries}} \tag{5.9}$$

MRR is a metric that evaluates the rank at which the correct prediction appears in the ranked list of predictions. For each prediction task, the reciprocal rank is calculated. This is the inverse of the rank at which the true value appears in the predictions list. The MRR is the average of the reciprocal ranks over all test queries. Formally, MRR is calculated as:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i} \tag{5.10}$$

In this definition, N denotes the total number of test queries, and $rank_i$ represents the rank of the correct prediction for the i-th query. The two link prediction metrics are calculated for the following example. Consider the test query $22(10924, Y)$ where the true value is 10926. Given the following ranked predictions:

$predictions = [100, 200, 300, 10926, 400]$

Using the definitions of the metrics hits@k and MRR, the following is calculated:

$hits@1 = \frac{0}{1} = 0 \qquad hits@5 = \frac{1}{1} = 1 \qquad MRR = \frac{1}{4}$

5.7.2 Time prediction

In the context of time prediction, the structure of the rule's head is known. For each test query, the time component is removed. The remaining triple is used to search for grounded rules with matching heads. These rules are ordered based on the confidence value of the rules. For each rule, the body is checked against the training set. If the body is contained in the training set and consists of only a single body atom, the time component of that atom is directly used as the prediction. If this body atom has multiple instances over time, one is picked at random. If the body contains multiple body atoms, the overlap of intervals is used as the prediction. If multiple overlapping intervals are found, the first one is used. If there exists no overlap, the earliest and latest times are used for the prediction of the start and end times, respectively. For example, consider the test fact where the time component is removed, resulting in the following:

10924 22 10926 2012 2014
10924 22 10926 *startTime* *endTime*

The following rule and facts from the training set are found:

$22(10924, 10926) \longleftarrow 22(10924, 10143)$
10924 22 10143 2015 2015
10924 22 10143 2015 2020

In this example, the predicted start and end time are both 2015. To evaluate this prediction, the metrics aeIOU and TAC, introduced in Subsection 5.7.1, are computed as follows:

$$aeIOU((2015, 2015), (2012, 2014)) = \frac{\max\{1, (2015,2015) \bigcap (2012,2014)\}}{(2015,2015) \bigcup (2012,2014)} = 0.25$$
$$TAC((2015, 2015), (2012, 2014)) = 0.5 \times \left(\frac{1}{1+|2015-2012|} + \frac{1}{1+|2015-2014|}\right) = 0.375$$

Chapter 6

Experiments

This Chapter covers the experiments and their results for temporal confidence calculations, link predictions, and time predictions.

6.1 Setup

The experiments for temporal confidence calculation and temporal link and time prediction were conducted using static rules learned by the AnyBURL framework. AnyBURL produced rules learned after different time windows: 10 seconds, 50 seconds, and 100 seconds. The rules were learned using the default parameters of AnyBURL in addition to WORKER_THREADS = 6. Once the rules were generated, temporal confidence values were calculated for each temporal quality measure. Then, both link and time predictions were made using the temporal rules. For the evaluation of the predictions, hits@k and MRR were used for link prediction, while aeIOU and TAC were applied for time prediction.

The repository of the project can be found at `https://github.com/Pascalle5935/StaTeR`. The experiments were implemented in Python. The RDFlib library was used to manipulate data into RDF format, while Blazegraph was employed for maintaining the knowledge graph. The experiments were conducted on datasets Wikidata12k and YAGO11k. The system used for the experiments was a laptop equipped with an Inter Core i17-10750H processor, which features 6 physical cores with one thread per core. The system operates on an x86_64 architecture.

All temporal confidence calculations were produced from a single run of the AnyBURL program. This is due to the amount of experiments that can be done for the three files of AnyBURL per run; temporal confidence calculations for five different quality measures for different thresholds and the link and time predictions for these temporal rules. Each run of AnyBURL produces a unique set of static rules and a varying number of rules for the same data input. As a result, the produced static rules can differ between runs. However, we assume that the best rules for the dataset will be found in each run of AnyBURL, with only lower-confidence rules potentially being inconsistent depending on the specific rule. The confidence threshold for the AnyBURL program is set to 0.0001. The thresholds used in these experiments are set to 0.15 or 0.2. The datasets used in this experiment are the cleaned Wikidata12k and YAGO11k described in Section 5.3.2.

Rule	Quality measure	Naive	Naive overlap	Same interval	IOU	TAC
$22(X, 9938)$	**BC**	17				
$\longleftarrow$	**S**	4	0	0	0	0.705
$22(X, 10125)$	**TC**	0.235	0	0	0	0.041
$22(10282, Y)$	**BC**	10				
$\longleftarrow$	**S**	2	0	0	0	0.093
$22(11563, Y)$	**TC**	0.2	0	0	0	0.009
$22(11905, Y)$	**BC**	6				
$\longleftarrow$	**S**	3	1	0	0.333	0.892
$22(11780, Y)$	**TC**	0.5	0.162	0	0.056	0.149
$15(5272, Y)$	**BC**	2				
$\longleftarrow$	**S**	2	2	2	2	2
$15(5254, Y)$	**TC**	1	1	1	1	1
$5(X, Y) \longleftarrow$	**BC**	77				
$5(X, A), 5(B, A),$	**S**	33	33	6	28.254	0.705
$5(Y, B)$	**TC**	0.429	0.429	0.078	0.367	0.041

Table 6.1: Qualitative analysis of temporal confidence calculations on 5 randomly chosen rules. Body coverage is abbreviated as "BC", support as "S", and temporal confidence as "TC". Temporal confidence listed in this Table are from threshold = 0.15, static AnyBURL rules learned after 10 seconds, for dataset Wikidata12k.

6.2 Results

The following Subsections present the results for the temporal confidence calculations, link and time prediction. The run times for all experiments are listed in Appendix C. Furthermore, all results of the experiments can be found in the source code directory.

6.2.1 Temporal confidence calculations

As described in Section 5.5, temporal rules derived from static rules can not achieve better confidence values. Therefore, a confidence threshold is used to filter the static rules that were considered weak in the first place. The results of a qualitative analysis of five randomly selected rules are presented in Table 6.1. These results provide insights into the different temporal confidence calculations for static rules learned after 10 seconds for the dataset Wikidata12k. For each rule, three metrics, *body coverage*, *support*, and *confidence*, are evaluated across the various temporal confidence quality measures *Naive*, *Naive Overlap*, *Same Interval*, *IOU*, *TAC*.

Interestingly, the quality measures Naive Overlap, Same Interval, and IOU, are consistently 0 for the first two rules. This indicates that the instances of these rules do not have any overlap in their time intervals. In contrast, the Naive and TAC confidences show non-zero values, 0.235 and 0.041 respectively. This is because these quality measures are not influenced by whether or not time intervals overlap. As a result, the Naive

and TAC confidences can never be exactly 0. If this were the case, it would imply that the original support was 0, meaning there are no instances where both the head and body are satisfied. Thus, AnyBURL will never produce such a static rule in the first place.

Another observation is that the confidences of the fourth rule are 1 across all quality measures. This suggests that, for both instances of this rule, the head and body time intervals are exactly equal. The third rule is an example for which the confidence decreases as the quality measures become stricter. For the Naive quality measure, the confidence is 0.5, whereas for the Naive Overlap quality measure, it drops significantly to 0.162. This suggest that a large portion of the instances do not have any overlap. Furthermore, the Same Interval confidence is 0, indicating that no instances have perfectly aligned time intervals. The lower IOU confidence compared to the TAC confidence implies that the overlap is small while the start and end times are relatively close.

For the final rule, there are 77 body instances. This is, compared to the other numbers for body coverage, a lot. It is explained by the fact that the final rule is a very general rule consisting of only variables. Among these, there are 33 instances that all have temporal overlap. Furthermore, 6 of those have the exact same time interval, which is rare since the rule has four body atoms. Thus, there are 6 instances for which all the time intervals of the rule are equal. The IOU confidence and TAC confidence show that while there is a significant temporal overlap, the relatively closeness of the start and end times of the time intervals is not as close. The relatively high IOU confidence can be explained by the presence of large intervals. For example, in rule instances with head interval of [1000, 2000] and a body interval of [1100,1900] would result in high IOU confidence, while the TAC confidence is low due to the large absolute gap between the start and end times.

Similar qualitative analyses on random rules for the YAGO11k dataset and other scenarios for the Wikidata12k dataset can be done. The results of all scenarios considered in this experiment of the temporal confidence calculations for both datasets can be found in the "confidence_calculations" folder in the code repository.

Before calculating temporal confidences, rules are filtered based on the applied threshold. Table 6.2 shows the amount of rules after filtering for the different thresholds used. It is evident that AnyBURL generates significantly more rules for Wikidata12k compared to YAGO11k, which is expected due to its larger training set. The unique static training set of Wikidata12k consists of 27,774 facts, while there are 16,408 unique facts for the YAGO11k training set. Additionally, the table shows that the number of rules removed from the Wikidata12k dataset is much higher than for the YAGO11k dataset. For example, for rules learned after 100 seconds for threshold 0.15, the proportion of filtered rules for YAGO11k is approximately $\frac{(19{,}808-19{,}779)}{19{,}808} \approx 0.001\%$ while for the same scenario for Wikidata12k it $\frac{(174{,}377-155{,}761)}{174{,}377} \approx 0.107\%$. This suggests that YAGO11k tends to produce more rules of higher confidence as fewer rules are filtered based on the threshold.

Table C.1 of the Appendix presents the run times for the creation of temporal rules for AnyBURL rules learned after different intervals, using various quality measures and thresholds, for the datasets Wikidata12k and YAGO11k. These run times are directly related to the number of generated filtered rules. As shown in the table, the run time for rules learned after 100 seconds is consistently longer than for rules learned after 50 of 10 seconds. More temporal confidences have to be calculated, which takes more computational time. Furthermore, the table shows that the run times for calculating

Dataset	Rules after x sec	Threshold		
		0.15	**0.2**	**0.0**
Wikidata12k	**10**	19,063	17,508	33,583
	50	128,026	123,669	146,634
	100	155,761	151,390	174,377
YAGO11k	**10**	4,273	3,727	4,286
	50	17,521	15,479	17,550
	100	19,779	17,663	19,808

Table 6.2: Number of static rules learned by AnyBURL for different thresholds for datasets Wikidata12k and YAGO11k.

IOU and TAC confidences are consistently higher for almost every scenario. This can be explained by the computational complexity of these quality measures. Unlike simpler quality measures such as "Same Interval", which check for perfect temporal alignment only, IOU and TAC calculations involve more sophisticated operations. IOU calculates the intersection over union, while TAC calculates temporal alignment coefficient, both of which generally demand more processing time.

The run times for calculating temporal confidences emphasize the trade-off between the number of rules generated, the proportion of rules filtered based on the threshold, and the choice of quality measure, all of which influence computational effort and run time for temporal confidence calculations.

6.2.2 Link prediction

Now that all temporal confidences have been calculated for different scenarios of thresholds, quality measures, datasets, and rule set learned after x seconds, link prediction for these rules with temporal confidences can be performed. The results of the link prediction experiments on the datasets for temporal rules learned after 10 seconds for both thresholds 0.15 and 0.2 provide insight into the performance of the different quality measures. The link prediction results for the Wikidata12k dataset are shown in Table 6.3 and for the YAGO11k dataset in Table 6.4.

Immediately noticeable is that the best-performing quality measure across all columns for both datasets is the Naive measure. However, the TAC quality measure shows some competitive results. For Wikidata12k, the TAC measure outperforms the Naive Overlap measure in all scenarios, indicating that incorporating temporal alignment leads to better predictions compared to relying on the overlap. However, for the YAGO11k dataset, the Naive Overlap quality measure outperforms the TAC quality measure in all scenarios. The differences between the values of all evaluating measures is not significant, highlighting the relative effectiveness of these measures can vary for different datasets. What is evident is that the Same Interval measure performs the lowest overall for both datasets. This is because this is the strictest measure. The performances of IOU and TAC are comparable for the YAGO11k dataset, while TAC definitely outperforms IOU for Wikidata12k. This

T_c	Quality measure	Hits@1		Hits@10		MRR	
		aeIOU	TAC	aeIOU	TAC	aeIOU	TAC
0.15	**N**	**0.1349**	**0.1360**	**0.3384**	**0.3415**	**0.1919**	**0.1948**
	NO	0.1153	0.1177	0.2459	0.2481	0.1559	0.1587
	SI	0.0791	0.0800	0.1441	0.1440	0.1010	0.1019
	IOU	0.0984	0.0997	0.1838	0.1853	0.1264	0.1283
	TAC	0.1227	0.1248	0.2462	0.2482	0.1620	0.1644
0.2	**N**	0.1231	0.1244	0.3304	0.3227	0.1783	0.1806
	NO	0.0966	0.0984	0.1945	0.1966	0.1281	0.1304
	SI	0.0663	0.0670	0.1170	0.1172	0.0838	0.0845
	IOU	0.0768	0.0788	0.1394	0.1407	0.0979	0.0998
	TAC	0.1020	0.1043	0.2000	0.2001	0.1336	0.1354

Table 6.3: Link prediction evaluation for various quality measures and thresholds for the temporal rules for dataset Wikidata12k learned after 10 seconds. In bold is the best ranked quality measure for that column. For readability, we use the abbreviations T_c = threshold, N = Naive, NO = Naive Overlap, SI = Same Interval, IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.

means that there is more overlap in YAGO11k compared to Wikidata12k.

In terms of the threshold, the threshold set at 0.15 outperforms all results compared to the threshold of 0.2 for both datasets. This is what is to be expected, because with a lower threshold, there are more rules to be considered. The rule set of threshold 0.2 is a subset of the rule set of threshold 0.15. Therefore, a lower threshold will produce the same or better results compared to a higher threshold. For this reason, the link prediction experiments for rules learned after 50 and 100 seconds have only been performed for threshold set at 0.15 for both datasets, since the results of threshold 0.2 are redundant. Furthermore, comparing the aeIOU factor with the TAC factor, it can be seen that the TAC factor almost always outperforms the aeIOU factor. Only in three instances is this not the case; Same Interval for threshold 0.15, Naive for threshold 0.2, and Same Interval for threshold 0.2 all for the dataset Wikidata12k. However, the differences in performance between aeIOU and TAC are minimal, often just fractions of a percentage point. Therefore, it cannot be claimed that the TAC factor outperforms the aeIOU factor.

T_c	Quality measure	Hits@1		Hits@10		MRR	
		aeIOU	TAC	aeIOU	TAC	aeIOU	TAC
0.15	**N**	**0.0822**	**0.0843**	**0.1814**	**0.1819**	**0.1114**	**0.1127**
	NO	0.0812	0.0839	0.1672	0.1663	0.1074	0.1090
	SI	0.0524	0.0524	0.0612	0.0612	0.0554	0.0557
	IOU	0.0739	0.0744	0.1190	0.1185	0.0873	0.0879
	TAC	0.0783	0.0783	0.1304	0.1307	0.0938	0.0943
0.2	**N**	0.0792	0.0822	0.1687	0.1702	0.1065	0.1080
	NO	0.0795	0.0834	0.1521	0.1546	0.1023	0.1052
	SI	0.0510	0.0512	0.0568	0.0566	0.0530	0.0533
	IOU	0.0707	0.0719	0.1034	0.1036	0.0805	0.0814
	TAC	0.0729	0.0734	0.1068	0.1068	0.0829	0.0834

Table 6.4: Link prediction evaluation for various quality measures and thresholds for the temporal rules for dataset YAGO11k learned after 10 seconds. In bold is the best ranked quality measure for that column. For abbreviations, see Table 6.3.

The evaluation results for more link prediction experiments for different rule sets and quality measures for the Wikidata12k dataset are presented in Table B.1 and Table B.2 present it for the YAGO11k dataset. It can be noticed that the evaluation measures of rules learned after 50 and 100 are comparable. The best results for all columns are predicted using the rules found after 50 seconds for Wikidata12k. For YAGO11k, the rules learned after 50 seconds and 100 seconds share the top evaluation performances. For both datasets and all scenarios, Same Interval performs the worst, while Naive performs the best. This is what we would expect since the Same Interval confidence is calculated in the strictest way and the Naive confidence the least strictest way. For these tables, most of the time the TAC factor outperforms the aeIOU fact, but marginally.

The best results of StaTeR for each link prediction evaluation measure for both datasets is compared to other models, presented in Table 6.5. Neuro-symbolic approach TILP [40] outperforms all other models across both datasets. This indicates that TILP is the most effective model for link prediction in both the Wikidata12k and YAGO11k datasets. However, since this is a neuro-symbolic approach, the predictions are not explainable. The predictions of StaTeR are explainable since we can derive which rule predicted a certain prediction. The approached TLogic [39] and TimePlex [20] consistently show lower performance compared to TILP and StaTeR. Only hits@10 for TLogic shows some competitive results. Because of the greater size of the Wikidata12k dataset, the link prediction results are consistently higher compared to the YAGO11k dataset, regardless of the model.

For the StaTeR model, we see that the results of Same Interval are the lowest for all evaluation measures. This is because of the strictness of the measure. Furthermore, the non-strict measures Naive and TAC outperform the others because the intervals do not

Model	Wikidata12k			YAGO11k		
	Hits@1	**Hits@10**	**MRR**	**Hits@1**	**Hits@10**	**MRR**
TimePlex	0.0651	0.1580	0.0976	0.0341	0.1146	0.0639
TLogic	0.0934	0.3201	0.1508	0.0621	0.1804	0.1107
TILP	**0.1623**	**0.4412**	**0.2657**	**0.1194**	**0.3686**	**0.2069**
	Wikidata12k clean			**YAGO11k clean**		
StaTeR (N-aeIOU)	0.1399	0.3645	0.2029	0.0856	0.2072	0.1182
StaTeR (N-TAC)	0.1405	0.3687	0.2047	0.0882	0.2058	0.1188
StaTeR (NO-aeIOU)	0.1271	0.2947	0.1782	0.0822	0.1931	0.1127
StaTeR (NO-TAC)	0.1323	0.2998	0.1833	0.0865	0.1955	0.1166
StaTeR (SI-aeIOU)	0.0952	0.1775	0.1227	0.0612	0.0770	0.0664
StaTeR (SI-TAC)	0.0977	0.1766	0.1246	0.0629	0.0773	0.0678
StaTeR (IOU-aeIOU)	0.1159	0.2284	0.1527	0.0814	0.1543	0.1018
StaTeR (IOU-TAC)	0.1179	0.2304	0.1555	0.0841	0.1524	0.1034
StaTeR (TAC-aeIOU)	0.1349	0.2995	0.1853	0.0831	0.1638	0.1063
StaTeR (TAC-TAC)	0.1397	0.3015	0.1897	0.0863	0.1650	0.1084

Table 6.5: Link prediction performance of StaTeR compared to other models for datasets Wikidata12k and YAGO11k. In bold is the best ranked quality measure for that column. For readability, we use the abbreviations T_c = threshold, N = Naive, NO = Naive Overlap, SI = Same Interval, IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient, aeIOU = affinity enhanced Intersection Over Union.

need to have overlap. For the comparison of Naive Overlap with IOU, it can be seen that Naive Overlap consistently outperforms IOU. While Naive Overlap simply checks for overlap, the IOU corrects the confidence value with the ratio of the intersection. That Naive Overlap outperforms IOU seems counterintuitive, as one might expect a higher IOU since this measure favors rules with more overlap.

The results of all scenarios considered in this experiment for the link prediction task for both datasets can be found in the "link_predictions" folder in the source code directory.

6.2.3 Time prediction

The performance for time prediction on the datasets Wikidata12k and YAGO11k of StaTeR is presented in Table 6.6. The time prediction performance of StaTeR is compared to the models TimePlex [20] and TEILP [45], a model inspired by TILP [40] that predicts time intervals. Both approaches TimePlex and TEILP outperform every quality measure of StaTeR for the Wikidata12k dataset. However, the IOU and TAC quality measures of StaTeR outperform both TimePlex and TEILP for the YAGO11k dataset. Both IOU

Model	Wikidata12k		YAGO11k	
	aeIOU	**TAC**	**aeIOU**	**TAC**
TimePlex	0.2636	0.3054	0.2003	0.2253
TEILP (rule-split)	**0.3285**	**0.3153**	0.2977	0.2877
StaTeR (N)	0.1620	0.1845	0.1897	0.2074
StaTeR (NO)	0.1822	0.1946	0.2634	0.2833
StaTeR (SI)	0.1999	0.1916	0.2798	0.3019
StaTeR (IOU)	0.1887	0.1819	**0.3143**	**0.3238**
StaTeR (TAC)	0.1861	0.2020	0.3002	0.3136

Table 6.6: Time prediction performance of StaTeR compared to other models for datasets Wikidata12k and YAGO11k. In bold is the best ranked quality measure for that column. For readability, we use the abbreviations T_c = threshold, N = Naive, NO = Naive Overlap, SI = Same Interval, IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient, aeIOU = affinity enhanced Intersection Over Union.

and TAC reorder the temporal rules based on their quality measure. For IOU, that is the intersection over union ratio, which is the most effective on the YAGO11k dataset. The TAC quality measure defines the temporal alignment coefficient between time intervals, which prove to be more effective on the YAGO11k dataset as opposed to approaches of TimePlex and TEILP.

For the Wikidata12k dataset, the IOU quality measure of StaTeR performs the worst using the TAC evaluation measure. This is notable since this quality measure outperforms the state-of-the-art models for the YAGO11k dataset. This highlights the data dependency for both quality measures IUO and TAC of StaTeR. Finally, run times of these experiments are presented in C.2 and the results of all scenarios considered in this experiment for the time prediction task for both datasets can be found in the "time_predictions" folder in the source code directory.

Chapter 7

Conclusion

StaTeR (Static to Temporal Rule learner) is the first dynamic symbolic rule learner which generates temporal rules from its static variant. Initially, static rules are learned by a symbolic rule learner. Then, five different temporal confidences are computed, and these temporal rules are then used for link and time prediction. The key contributions of this thesis are listed below in small paragraphs.

The process of cleaning the temporal dataset Wikidata12k has resulted in the creation of a refined dataset that eliminates overlapping intervals for identical static facts. Specifically, overlapping time intervals for identical static facts have been merged, ensuring that each unique statical fact can be true in non-overlapping intervals. This adjustment addresses ambiguities that arise from overlapping temporal intervals, which introduced challenges in accurate temporal confidence calculations. Similarly, the dataset YAGO11k was altered in such a way that invalid temporal ranges, such as start and end times occurring before the year 0, are removed. This step was essential to ensure compatibility with the tools used, such as the rdflib library, which cannot handle dates earlier than this year. The cleaned datasets provide more accurate temporal information, making them more usable and reliable for studying temporal rule reasoning.

Furthermore, several temporal quality measures for evaluating rules were introduced. By redefining the temporal confidence based on static rules, the temporal aspect of temporal knowledge is incorporated. Five distinct temporal confidences were explored: Naive, Naive Overlap, Same Interval, Intersection Over Union (IOU), and Temporal Alignment Coefficient (TAC). The Naive confidence measure is unbounded as it can match rule bodies to heads without temporal overlap. Therefore, Naive Overlap is introduced limiting matches to instances with overlapping intervals. However, this quality measure can still yield confidence values exceeding 1. The stricter Same Interval confidence requires prefect alignment of intervals of the rule instances. This quality measure yields many confidence values of 0, but indicates good rule quality when satisfied. The IOU confidence measures the ratio of the intersection to the union of the time intervals. Lastly, TAC does not require overlapping intervals, but captures the temporal alignment of the start and end dates. The provided quality measures differ in strictness and applicability resulting in a broad evaluation.

The link prediction experiments for various rule sets and quality measures provide a detailed assessment of temporal rules for the Wikidata12k and YAGO11k temporal datasets.

For both datasets, the results show that the Naive measure consistently outperforms the others. This is because this is the non-strictest measure, which counts all instances of head and body, even when there is no temporal alignment. The TAC measure shows competitive results comparing it to Naive Overlap. The Same Interval measure, being the strictest, performs worst overall. When comparing the StaTeR model with state-of-the-art models, it is clear that the neuro-symbolic approach of TILP is superior for link prediction on these datasets. Comparing StaTeR to the approaches TimePlex and TLogic, it can be seen that StaTeR outperforms both approaches for all evaluation measures. Furthermore, StaTeR has the advantage that due to its symbolic nature, its predictions are explainable.

The time prediction performance of StaTeR on datasets Wikidata12k and YAGO11k was assessed. The results show that IOU and TAC outperformed the other quality measures for YAGO11k, while IOU performed the worst for Wikidata12k. This highlights the data dependency of these quality measures considering time intervals.

Chapter 8

Discussion

This thesis has explored the task of learning temporal rules based on static rules using rule based learning. The output is a system called StaTeR (Static to Temporal Rule learner), the first symbolic rule learner which turns static rules into temporal rules for link and time prediction. The process involves generating static rules from the training set of a temporal dataset (with the temporal component removed) using a symbolic rule learner. Temporal confidences are then computed for these static rules, and the resulting temporalized rules are applied to link and time prediction.

An alternative approach to this problem is via sampling. In this method, a subset of rules is randomly selected. For this subset, the temporal confidences are calculated and an alpha factor is generated. This alpha factor is then applied to convert all static confidences to temporal confidences. Future research may approach the problem for finding this alpha. What is known of this alpha is that it is bound between 0 and 1, ensuring that the temporal confidence cannot exceed the static confidence of the same rule. This is because, temporal support requires the head to match the body only when their temporal components overlap. Without overlap, the prediction is useless and leads to overestimating the support which potentially generates confidences greater than 1.

Currently, the time predictions are limited to only predict existing start and end dates. A potential direction for future research is to design an algorithm that identifies trends and uses them to predict future events. Furthermore, future research could extend StaTeR to incorporate the PCA confidence in temporal rules.

Bibliography

[1] Sóren Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *The Semantic Web*. Springer Berlin Heidelberg, 2007, pp. 722–735. ISBN: 978-3-540-76298-0.

[2] Fabian Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Large Ontology from Wikipedia and WordNet". In: *The Semantic Web*. Springer Berlin Heidelberg, 2007, pp. 203–216. ISBN: 978-3-540-76298-0.

[3] Kurt Bollacker et al. "Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2008, pp. 1247–1250. ISBN: 9781605581026. DOI: 10.1145/1376616.1376746. URL: https://doi.org/10.1145/1376616.1376746.

[4] *Wikipedia*. 2001. URL: https://www.wikipedia.org.

[5] *List of Wikipedias*. May 2024. URL: https://meta.wikimedia.org/wiki/List_of_Wikipedias.

[6] Lekui Zhou et al. "Dynamic Network Embedding by Modeling Triadic Closure Process". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11257. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11257.

[7] Sam De Winter et al. "Combining Temporal Aspects of Dynamic Networks with Node2Vec for a more Efficient Dynamic Link Prediction". In: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (2018), pp. 1234–1241. URL: https://api.semanticscholar.org/CorpusID:53081185.

[8] Linyuan Lu and Zhou Tao. "Link prediction in complex networks: A survey". In: *Physica A: Statistical Mechanics and its Applications* 390.6 (2011), pp. 1150–1170. ISSN: 0378-4371. DOI: 10.1016/j.physa.2010.11.027. URL: http://dx.doi.org/10.1016/j.physa.2010.11.027.

[9] Seyed Mehran Kazemi et al. *Representation Learning for Dynamic Graphs: A Survey*. 2020. arXiv: 1905.11485.

[10] James F. Allen. "Maintaining knowledge about temporal intervals". In: *Commun. ACM* 26.11 (Jan. 1983), pp. 832–843. ISSN: 0001-0782. DOI: 10.1145/182.358434. URL: https://doi.org/10.1145/182.358434.

[11] Christian Meilicke et al. "Anytime bottom-up rule learning for large-scale knowledge graph completion". In: *The VLDB Journal* 33.1 (June 2023), pp. 131–161. ISSN: 1066-8888. DOI: 10.1007/s00778-023-00800-5. URL: https://doi.org/10.1007/s00778-023-00800-5.

[12] Luis Galárraga et al. "Fast rule mining in ontological knowledge bases with AMIE+". In: *The VLDB Journal* 24.6 (Jan. 2015), pp. 707–730. ISSN: 1066-8888. DOI: 10.

1007/s00778-015-0394-1. URL: https://doi.org/10.1007/s00778-015-0394-1.

[13] Antoine Bordes et al. "Translating embeddings for modeling multi-relational data". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Curran Associates Inc., 2013, pp. 2787–2795.

[14] Theo Trouillon et al. "Complex Embeddings for Simple Link Prediction". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. PMLR, June 2016, pp. 2071–2080. URL: https://proceedings.mlr.press/v48/trouillon16.html.

[15] Antoine Bordes et al. "Learning Structured Embeddings of Knowledge Bases". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 25.1 (Aug. 2011), pp. 301–306. DOI: 10.1609/aaai.v25i1.7917. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7917.

[16] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

[17] Tim Dettmers et al. "Convolutional 2D knowledge graph embeddings". In: New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[18] Julien Leblay and Melisachew Wudage Chekol. "Deriving Validity Time in Knowledge Graph". In: *Companion Proceedings of the The Web Conference 2018* (2018). URL: https://api.semanticscholar.org/CorpusID:13846713.

[19] Timothee Lacroix, Guillaume Obozinski, and Nicolas Usunier. *Tensor Decompositions for temporal knowledge base completion*. 2020. eprint: 2004.04926.

[20] Prachi Jain et al. "Temporal Knowledge Base Completion: New Algorithms and Evaluation Protocols". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 3733–3747. DOI: 10.18653/v1/2020.emnlp-main.305. URL: https://aclanthology.org/2020.emnlp-main.305.

[21] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. "HyTE: Hyperplane-based Temporally aware Knowledge Graph Embedding". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2001–2011. DOI: 10.18653/v1/D18-1225. URL: https://aclanthology.org/D18-1225.

[22] Woojeong Jin et al. "Recurrent Event Network: Autoregressive Structure Inferenceover Temporal Knowledge Graphs". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Association for Computational Linguistics, Nov. 2020, pp. 6669–6683. DOI: 10.18653/v1/2020.emnlp-main.541. URL: https://aclanthology.org/2020.emnlp-main.541.

[23] Qiang Zeng, Jignesh M. Patel, and David Page. "QuickFOIL: scalable inductive logic programming". In: *Proc. VLDB Endow.* 8.3 (Nov. 2014), pp. 197–208. ISSN: 2150-8097. DOI: 10.14778/2735508.2735510. URL: https://doi.org/10.14778/2735508.2735510.

[24] Ashwin Srinivasas. *The Aleph Manual*. 1999. URL: https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph_toc.html.

[25] Mohamed H. Gad-Elrab et al. “Exception-Enriched Rule Learning from Knowledge Graphs”. In: *The Semantic Web – ISWC 2016*. Springer International Publishing, 2016, pp. 234–251.

[26] Hai Dang Tran et al. “Towards Nonmonotonic Relational Learning from Knowledge Graphs”. In: *Inductive Logic Programming*. Ed. by James Cussens and Alessandra Russo. Springer International Publishing, 2017, pp. 94–107.

[27] Paolo Papotti et al. “RuDiK: Rule discovery in knowledge bases”. In: *VLDB 2018, 44th International Conference on Very Large Data Bases*. Ed. by ACM. Rio de Janeiro, 2018.

[28] Naser Ahmadi et al. “Mining Expressive Rules in Knowledge Graphs”. In: 12.2 (May 2020). ISSN: 1936-1955. DOI: 10.1145/3371315. URL: https://doi.org/10.1145/3371315.

[29] Yang Chen, Daisy Zhe Wang, and Sean Goldberg. “ScaLeKB: scalable learning and inference over large knowledge bases”. In: *The VLDB Journal* 25.6 (Dec. 2016), pp. 893–918. ISSN: 1066-8888. DOI: 10.1007/s00778-016-0444-3. URL: https://doi.org/10.1007/s00778-016-0444-3.

[30] Yang Chen et al. “Ontological Pathfinding”. In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 835–846. ISBN: 9781450335317. DOI: 10.1145/2882903.2882954. URL: https://doi.org/10.1145/2882903.2882954.

[31] Giuseppe Pirro. “Relatedness and TBox-Driven Rule Learning in Large Knowledge Bases”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03 (Apr. 2020), pp. 2975–2982. DOI: 10.1609/aaai.v34i03.5690. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5690.

[32] Fan Yang, Zhilin Yang, and William W. Cohen. *Differentiable Learning of Logical Rules for Knowledge Base Reasoning*. 2017. arXiv: 1702.08367.

[33] Lauren Nicole DeLong, Ramon Fernandez Mir, and Jacques D. Fleuriot. *Neurosymbolic AI for Reasoning over Knowledge Graphs: A Survey*. 2024. arXiv: 2302.07200.

[34] Hong Wu et al. “Learning Typed Rules over Knowledge Graphs”. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*. Aug. 2022, pp. 494–503. DOI: 10.24963/kr.2022/51. URL: https://doi.org/10.24963/kr.2022/51.

[35] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. “Scalable rule learning via learning representation”. In: Stockholm, Sweden: AAAI Press, 2018, pp. 2149–2155. ISBN: 9780999241127.

[36] Bishan Yang et al. *Embedding Entities and Relations for Learning and Inference in Knowledge Bases*. 2015. arXiv: 1412.6575.

[37] Meng Qu et al. *RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs*. 2021. eprint: 2010.04029.

[38] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. “Learning Temporal Rules from Knowledge Graph Streams.” In: *AAAI Spring symposium: combining machine learning with knowledge engineering*. 2019.

[39] Yushan Liu et al. *TLogic: Temporal Logical Rules for Explainable Link Forecasting on Temporal Knowledge Graphs*. 2022. eprint: 2112.08025.

[40] Siheng Xiong et al. “TILP: Differentiable Learning of Temporal Logical Rules on Knowledge Graphs”. In: *The Eleventh International Conference on Learning Representations*. 2023.

[41] Fredo Erxleben et al. “Introducing Wikidata to the Linked Data Web”. In: *The Semantic Web – ISWC 2014*. Ed. by Peter Mika et al. Springer International Publishing, 2014, pp. 50–65.

[42] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. “YAGO: a core of semantic knowledge”. In: *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 697–706. ISBN: 978-1-59593-654-7. DOI: 10.1145/1242572.1242667. URL: http://doi.acm.org/10.1145/1242572.1242667.

[43] Farzaneh Mahdisoltani, Joanna Asia Biega, and Fabian M. Suchanek. “YAGO3: A Knowledge Base from Multilingual Wikipedias”. In: *Conference on Innovative Data Systems Research*. 2015. URL: https://api.semanticscholar.org/CorpusID:6611164.

[44] Prachi Jain et al. “Temporal Knowledge Base Completion: New Algorithms and Evaluation Protocols”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 3733–3747. DOI: 10.18653/v1/2020.emnlp-main.305. URL: https://aclanthology.org/2020.emnlp-main.305.

[45] Siheng Xiong et al. *TEILP: Time Prediction over Knowledge Graphs via Logical Reasoning*. 2024. arXiv: 2312.15816. URL: https://arxiv.org/abs/2312.15816.

Appendix A

SPARQL Queries

This appendix presents SPARQL queries and corresponding examples for the Wikidata12k dataset. The following rule is used as a running example throughout this Appendix:

$20(X, 8508) \longleftarrow 20(X, 9362)$

A.1 Body Coverage

Python function for computing body coverage

```
def compute_temporal_body_coverage(body: List[Triple], dataset: str) ->
                                    int:
    query = f"{get_prefixes(dataset)}\nSELECT (COUNT(*) AS ?count)
                                      WHERE {{\n"
    for atom in body:
        query += f"""
        ?a1 rdf:subject {convert_to_sql_query_part(atom.get_subject())}
                                                ;
            rdf:predicate p:{atom.get_predicate()} ;
            rdf:object {convert_to_sql_query_part(atom.get_object())} .
        """
    query += "}"

    return execute_sparql_count_query(query)
```

SPARQL query for body coverage

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (COUNT(*) AS ?count) WHERE {
  ?a0 rdf:subject ?x ;
      rdf:predicate p:20 ;
      rdf:object e:9362 .
}
```

A.2 Naive Support

Python function for computing Naive support

```
def compute_temporal_support_naive(head: Triple, body: List[Triple],
                                   dataset: str) -> int:
    all_triples = [head] + body
    triple_statements = []

    # generate SPARQL statements for each triple
    for i, triple in enumerate(all_triples):
        alias = f"?a{i}"
        triple_statements.append(
            f"""
            {alias} rdf:subject {convert_to_sql_query_part(triple.
                                                get_subject())} ;
                  rdf:predicate p:{triple.get_predicate()} ;
                  rdf:object {convert_to_sql_query_part(triple.
                                                    get_object())} .
            """
        )

    # build the complete SPARQL query
    query = f"""
    {get_prefixes(dataset)}

    SELECT (COUNT(DISTINCT *) AS ?count) WHERE {{
        {' '.join(triple_statements)}
    }}
    """

    return execute_sparql_count_query(query)
```

SPARQL query for Naive support

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (COUNT(*) AS ?count) WHERE {
  ?a1 rdf:subject ?x ;
      rdf:predicate p:20 ;
      rdf:object e:8508 .
  ?a2 rdf:subject ?x ;
      rdf:predicate p:20 ;
      rdf:object e:9362 .
}
```

A.3 Naive Overlap support

Python function for computing Naive Overlap support

```
def compute_temporal_support_naive_overlap(head: Triple, body: List[
                                    Triple], dataset: str) -> int:
    all_triples = [head] + body
    triple_statements = []
    temporal_constraints = []

    # generate SPARQL statements for each triple
    for i, triple in enumerate(all_triples):
        alias = f"?tr{i}"
        triple_statements.append(
            f"""
            {alias} rdf:subject {convert_to_sql_query_part(triple.
                                                get_subject())} ;
                  rdf:predicate p:{triple.get_predicate()} ;
                  rdf:object {convert_to_sql_query_part(triple.
                                                    get_object())} ;
                  time:hasStart ?s{i} ;
                  time:hasEnd ?e{i} .
            """
        )

    # generate pairwise temporal constraints
    n = len(all_triples)
    for i in range(n):
        for j in range(i + 1, n):
            temporal_constraints.append(f"(?s{i} <= ?e{j} && ?s{j} <= ?
                                            e{i})")

    # combine the temporal constraints
    temporal_filter = "FILTER (" + " && ".join(temporal_constraints) +
                                        ")"

    # build the complete SPARQL query
    query = f"""
    {get_prefixes(dataset)}

    SELECT (COUNT(DISTINCT *) AS ?count) WHERE {{
        {' '.join(triple_statements)}
        {temporal_filter}
    }}
    """

    return execute_sparql_count_query(query)
```

SPARQL query for Naive Overlap support

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT (COUNT(DISTINCT *) AS ?count) WHERE {
    ?tr0 rdf:subject ?x ;
         rdf:predicate p:20 ;
         rdf:object e:8508 ;
         time:hasStart ?s0 ;
         time:hasEnd ?e0 .
    ?tr1 rdf:subject ?x ;
         rdf:predicate p:20 ;
         rdf:object e:9362 ;
         time:hasStart ?s1 ;
         time:hasEnd ?e1 .

    FILTER ((?s0 <= ?e1 && ?s1 <= ?e0))
}
```

A.4 Same Interval support

Python function for computing Same Interval support

```
def compute_temporal_support_same_interval(head: Triple, body: Triple,
                                           dataset: str) -> int:
    all_triples = [head] + body
    triple_statements = []

    # generate SPARQL statements for each triple
    for i, triple in enumerate(all_triples):
        alias = f"?a{i}"
        triple_statements.append(
            f"""
            {alias} rdf:subject {convert_to_sql_query_part(triple.
                                                get_subject())} ;
                  rdf:predicate p:{triple.get_predicate()} ;
                  rdf:object {convert_to_sql_query_part(triple.
                                                get_object())} ;
                  time:hasStart ?s ;
                  time:hasEnd ?e .
            """
        )

    # build the complete SPARQL query
    query = f"""
    {get_prefixes(dataset)}

    SELECT (COUNT(DISTINCT *) AS ?count) WHERE {{
        {' '.join(triple_statements)}
    }}
    """

    return execute_sparql_count_query(query)
```

SPARQL query for Same Interval support

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT (COUNT(DISTINCT *) AS ?count) WHERE {
    ?a0 rdf:subject ?x ;
        rdf:predicate p:20 ;
        rdf:object e:8508 ;
        time:hasStart ?s ;
        time:hasEnd ?e .
    ?a1 rdf:subject ?x ;
        rdf:predicate p:20 ;
        rdf:object e:9362 ;
        time:hasStart ?s ;
        time:hasEnd ?e .
}
```

A.5 Intersection Over Union support

Python function for computing Intersection Over Union support

```
def compute_temporal_support_intersection_over_union(head, body: List[
                                    Triple], dataset: str) -> float:
    all_triples = [head] + body

    # generate SPARQL queries to get the time intervals for the head
                                        and body instances
    triple_statements = []
    for i, triple in enumerate(all_triples):
        alias = f"?a{i}"
        triple_statements.append(
            f"""
            {alias} rdf:subject {convert_to_sql_query_part(triple.
                                                    get_subject())} ;
                  rdf:predicate p:{triple.get_predicate()} ;
                  rdf:object {convert_to_sql_query_part(triple.
                                                    get_object())} ;
                  time:hasStart ?s{i} ;
                  time:hasEnd ?e{i} .
            """
        )

    # build the complete SPARQL query
    query = f"""
    {get_prefixes(dataset)}

    SELECT {' '.join([f'?s{i} ?e{i}' for i in range(len(all_triples))])
                                        } WHERE {{
        {' '.join(triple_statements)}
    }}
    """

    results = execute_sparql_query(query)
    if not results:
        return 0.0

    temporal_support = 0.0
    # generate all pair combinations
    pairs = list(combinations(range(len(all_triples)), 2))

    overlap_sum = 0
    for result in results:
        for i, j in pairs:
            # get the start and end times for the current pair of
                                                    instances
            s1, e1 = extract_year_from_date(result[f"s{i}"]['value']),
                                                    extract_year_from_date(
                                                    result[f"e{i}"]['value'
                                                    ])
            s2, e2 = extract_year_from_date(result[f"s{j}"]['value']),
                                                    extract_year_from_date(
                                                    result[f"e{j}"]['value'
                                                    ])
```

```
            # calculate the union
            latest_end = max(e1, e2)
            earliest_start = min(s1, s2)
            union = latest_end - earliest_start + 1

            # calculate the overlap
            beginning_overlap = max(s1, s2)
            ending_overlap = min(e1, e2)
            overlap = max(0, ending_overlap - beginning_overlap + 1)

            # calculate the overlap fraction
            overlap_fraction = overlap / union if union > 0 else 0
            overlap_sum += overlap_fraction

        # normalize by the number of pairs
        temporal_support = float(overlap_sum / len(pairs))

    return temporal_support
```

SPARQL query for Intersection Over Union support

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT ?s0 ?e0 ?s1 ?e1 WHERE {
    ?a0 rdf:subject ?x ;
          rdf:predicate p:20 ;
          rdf:object e:8508 ;
          time:hasStart ?s0 ;
          time:hasEnd ?e0 .

    ?a1 rdf:subject ?x ;
          rdf:predicate p:20 ;
          rdf:object e:9362 ;
          time:hasStart ?s1 ;
          time:hasEnd ?e1 .
}
```

A.6 Temporal Alignment Coefficient support

Python function for computing Temporal Alignment Coefficient support

```
def compute_temporal_temporal_alignment_coefficient(head, body: List[
                                    Triple], dataset: str) -> float:
    all_triples = [head] + body

    # generate SPARQL queries to get the time intervals for the head
                                        and body instances
    triple_statements = []
    for i, triple in enumerate(all_triples):
        alias = f"?a{i}"
        triple_statements.append(
            f"""
            {alias} rdf:subject {convert_to_sql_query_part(triple.
                                                get_subject())} ;
                  rdf:predicate p:{triple.get_predicate()} ;
                  rdf:object {convert_to_sql_query_part(triple.
                                                    get_object())} ;
                  time:hasStart ?s{i} ;
                  time:hasEnd ?e{i} .
            """
        )

    # build the complete SPARQL query
    query = f"""
    {get_prefixes(dataset)}

    SELECT {' '.join([f'?s{i} ?e{i}' for i in range(len(all_triples))])
                                        } WHERE {{
        {' '.join(triple_statements)}
    }}
    """
    print(query)
    results = execute_sparql_query(query)
    if not results:
        return 0.0

    # generate all pair combinations
    pairs = list(combinations(range(len(all_triples)), 2))

    tac_sum = 0
    for result in results:
        for i, j in pairs:
            # get the start and end times for the current pair of
                                                    instances
            s1, e1 = extract_year_from_date(result[f"s{i}"]['value']),
                                                    extract_year_from_date(
                                                    result[f"e{i}"]['value'
                                                    ])
            s2, e2 = extract_year_from_date(result[f"s{j}"]['value']),
                                                    extract_year_from_date(
                                                    result[f"e{j}"]['value'
                                                    ])
```

```
            # calculate the TAC contribution for the pair (I_i, I_j)
            start_diff = abs(s1 - s2)
            end_diff = abs(e1 - e2)

            # calculate the TAC for this pair
            tac = 0.5 * (1 / (1 + start_diff) + 1 / (1 + end_diff))
            tac_sum += tac

    return tac_sum
```

SPARQL query for Temporal Alignment Coefficient support

```
PREFIX p: <http://wikidata.org/prop/direct/>
PREFIX e: <http://wikidata.org/entity/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT ?s0 ?e0 ?s1 ?e1 WHERE {
    ?a0 rdf:subject ?x ;
        rdf:predicate p:20 ;
        rdf:object e:8508 ;
        time:hasStart ?s0 ;
        time:hasEnd ?e0 .

    ?a1 rdf:subject ?x ;
        rdf:predicate p:20 ;
        rdf:object e:9362 ;
        time:hasStart ?s1 ;
        time:hasEnd ?e1 .
}
```

Appendix B

Link prediction results

Rules after x sec	Quality measure	Hits@1		Hits@10		MRR	
		aeIOU	**TAC**	**aeIOU**	**TAC**	**aeIOU**	**TAC**
10	N	0.1349	0.1360	0.3384	0.3415	0.1919	0.1948
	NO	0.1153	0.1177	0.2459	0.2481	0.1559	0.1587
	SI	0.0791	0.0800	0.1441	0.1440	0.1010	0.1019
	IOU	0.0984	0.0997	0.1838	0.1853	0.1264	0.1283
	TAC	0.1227	0.1248	0.2462	0.2482	0.1620	0.1644
50	N	**0.1399**	**0.1405**	**0.3645**	**0.3687**	**0.2029**	**0.2047**
	NO	0.1271	0.1323	0.2947	0.2998	0.1782	0.1833
	SI	0.0951	0.0973	0.1767	0.1766	0.1225	0.1242
	IOU	0.1156	0.1176	0.2279	0.2299	0.1523	0.1551
	TAC	0.1349	0.1396	0.2995	0.3015	0.1853	0.1897
100	N	0.1393	0.1399	0.3637	0.3682	0.2022	0.2045
	NO	0.1266	0.1323	0.2943	0.2992	0.1780	0.1833
	SI	0.0952	0.0977	0.1775	0.1773	0.1227	0.1246
	IOU	0.1159	0.1179	0.2284	0.2304	0.1527	0.1555
	TAC	0.1349	0.1397	0.2988	0.3006	0.1851	0.1894

Table B.1: Link prediction evaluation for StaTeR for various quality measures and threshold = 0.15 for the temporal rules for dataset Wikidata12k. In bold are the best ranked systems for that column. For readability, we use the abbreviations T_c = threshold, N = Naive, NO = Naive Overlap, SI = Same Interval, IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.

Rules after x sec	Quality measure	Hits@1		Hits@10		MRR	
		aeIOU	TAC	aeIOU	TAC	aeIOU	TAC
10	N	0.0822	0.0843	0.1814	0.1819	0.1114	0.1127
	NO	0.0812	0.0839	0.1672	0.1663	0.1074	0.1090
	SI	0.0524	0.0524	0.0612	0.0612	0.0554	0.0557
	IOU	0.0739	0.0744	0.1190	0.1185	0.0873	0.0879
	TAC	0.0783	0.0783	0.1304	0.1307	0.0938	0.0943
50	N	0.0853	0.0880	**0.2072**	**0.2058**	0.1181	**0.1188**
	NO	0.0817	0.0865	0.1928	0.1955	0.1123	0.1164
	SI	0.0612	0.0627	0.0770	0.0773	0.0663	0.0677
	IOU	0.0814	0.0841	0.1541	0.1524	0.1018	0.1034
	TAC	0.0831	0.0863	0.1638	0.1648	0.1063	0.1084
100	N	**0.0856**	**0.0882**	**0.2072**	0.2048	**0.1182**	**0.1188**
	NO	0.0822	0.0837	0.1931	0.1945	0.1127	0.1166
	SI	0.0612	0.0629	0.0770	0.0773	0.0664	0.0678
	IOU	0.0804	0.0834	0.1534	0.1524	0.1014	0.1029
	TAC	0.0817	0.0853	0.1636	0.1650	0.1058	0.1080

Table B.2: Link prediction evaluation for StaTeR for various quality measures and threshold = 0.15 for the temporal rules for dataset YAGO11k. In bold are the best ranked systems for that column. For readability, we use the abbreviations T_c = threshold, N = Naive, NO = Naive Overlap, SI = Same Interval, IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.

Appendix C

Experiment run times

Rules after x sec	Quality measure	Wikidata12		YAGO11k	
		Threshold		Threshold	
		0.15	**0.2**	**0.15**	**0.2**
10	**Naive**	99.56	85.9	20.11	17.29
	Naive Overlap	99.76	88.62	20.56	17.61
	Same Interval	93.41	86.39	20.26	17.49
	IOU	120.33	105.83	21.75	18.46
	TAC	120.93	102.16	21.77	18.63
50	**Naive**	602.44	579.52	80.05	71.54
	Naive Overlap	630.04	618.10	83.15	73.93
	Same Interval	614.61	597.81	81.91	71.89
	IOU	630.13	629.64	82.71	74.06
	TAC	635.60	624.88	82.49	73.27
100	**Naive**	737.74	713.41	90.78	80.58
	Naive Overlap	764.28	749.18	93.22	82.90
	Same Interval	739.03	717.87	91.88	83.38
	IOU	770.26	734.11	94.86	83.32
	TAC	763.32	745.81	95.14	82.76

Table C.1: Temporal confidence calculation run times in seconds for StaTeR for different rule sets, quality measures and thresholds, for the datasets Wikidata12k and YAGO11k. For readability, we use the abbreviations IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.

Rules after x sec	Quality measure	Wikidata12		YAGO11k	
		Threshold		Threshold	
		0.15	**0.2**	**0.15**	**0.2**
10	**Naive**	265.30	215.88	71.02	53.94
	Naive Overlap	245.29	190.39	45.49	35.31
	Same Interval	125.92	128.70	2.25	1.22
	IOU	219.98	151.15	33.05	26.84
	TAC	256.90	174.90	35.22	26.24
50	**Naive**	1039.31	-	163.00	-
	Naive Overlap	470.00	-	96.78	-
	Same Interval	222.22	-	3.88	-
	IOU	389.76	-	55.43	-
	TAC	494.31	-	63.05	-
100	**Naive**	1062.69	-	164.27	-
	Naive Overlap	491.55	-	103.56	-
	Same Interval	249.59	-	4.28	-
	IOU	397.90	-	57.30	-
	TAC	469.22	-	64.77	-

Table C.2: Link prediction run times in seconds for StaTeR for different rule sets, quality measures and thresholds, for the datasets Wikidata12k and YAGO11k. Experiments for 50 and 100 seconds with a threshold of 0.2 were not conducted since link prediction with a threshold of 0.15 always produces the same or better for thresholds greater than 0.15, making the results redundant. For readability, we use the abbreviations IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.

Model	Wikidata12k	YAGO11k
Naive	1026.61	44.63
Naive Overlap	191.98	37.34
Same Interval	19.64	9.04
IOU	324.75	71.79
TAC	1067.21	72.60

Table C.3: Time prediction run times in seconds for StaTeR for different quality measures and threshold of 0.15 for the datasets Wikidata12k and YAGO11k. For readability, we use the abbreviations IOU = Intersection Over Union, TAC = Temporal Alignment Coefficient.