Index Optimization of Extended Characteristic Set for Temporal RDF

Yan Wu[1,2], Yu Liu[1,2], Tianyi Zhou[1,2], QingZhe Guo[1], Feng Gao[1,2], Simon Gu[1,2]
[1]Department of Computer Science and Technology, Wuhan University of Science and Technology
[2]Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial
Wuhan, Hubei, China
eanw@wust.edu.cn

ABSTRACT

Extended Characteristic Set(ECS) is an efficient RDF(Resource Description Framework) index structure. By storing data as graphs in the index, ECS can quickly recognize candidate graph patterns. In order to meet the management needs of temporal RDF data on ECS, the typical strategy is to employ the RDF reification method. However, the storage and query performance of the method will decline sharply as the data volume increases. According to the above limitation, we introduce H-ECS, a new composite index that takes the properties of temporal RDF data into account. By splitting the property bitmaps of ECS and indexing the temporal interval of split parts with HINT^m(A Hierarchical Index for Intervals), H-ECS can enhance storage and retrieval performance for temporal RDF data. A key-value index is also used in H-ECS to speed up the query. Based on H-ECS, we propose a specific pattern matching algorithm for typical queries of temporal RDF and implement a storage and query engine for temporal RDF data named TECStore. The experimental results show that the storage and query performance of TECStore outperforms AxonDB and PeriodDB for both synthetic and real-world temporal RDF data.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; *Data management systems*; • **Computing methodologies** → Artificial intelligence.

KEYWORDS

Temporal RDF, Extended Characteristic Set, Hierarchical Interval Index, Temporal Query

1 INTRODUCTION

In traditional RDF models, the temporal dimension is usually ignored or represented as a property, which has limitations in time-sensitive domains, such as finance, healthcare, and social media. To address these limitations, temporal RDF has been proposed. Temporal RDF can better describe the timing-related properties of events, resources, and data.

There are two main approaches to temporal RDF representation. The traditional approach focuses on preserving the structure of classical RDF triples, such as RDF reification [1]. However, the storage and query performance of this approach decreases dramatically as the data volume increases. Therefore, recent work has favored a second approach, extending the standard triple format to a quadruple or even quintuple format. Zhou T *et al.* [2], WANG Yin-di [3], and Yan L [4] all adopt the second approach and then extend the RDF database based on bit matrices, composite tables, and triple tables, respectively, to support the management of temporal RDF.

However, all these strategies fail to consider utilizing the property characteristics of temporal RDF to improve storage and query efficiency. For this reason, we decided to use ECS [5] and HINT^m [6]. The ECS is an efficient RDF index structure that utilizes the inherent structure of triples. RDF databases can quickly recognize candidate graph patterns by storing data as graphs in the index and using ECS to match graph patterns. In addition, the ECS method avoids generating large intermediate results and reduces storage space consumption. HINT^m is a method for indexing time intervals based on hierarchies. It allows quick querying of time intervals and is straightforward to implement and expand. Additionally, it facilitates time interval operations across all Allen [7] relations. The contributions of this paper are mainly in three aspects:

1) Based on ECS and HINT^m, we proposed a temporal RDF oriented index H-ECS. The index takes full advantage of the efficient RDF triples matching of ECS and the fast execution of temporal queries with HINT.

2) Based on H-ECS, we proposed a pattern matching algorithm for temporal queries that supports thirteen Allen temporal relations.

3) We used the H-CS index and the matching algorithm to implement TECStore, a temporal RDF-oriented storage and query engine based on H-ECS, and evaluated the query and storage performance of TECStore by using a synthetic dataset and two real-world datasets. The experimental results show that TECStore can significantly outperform AxonDB and PeriodDB regarding storage and query performance.

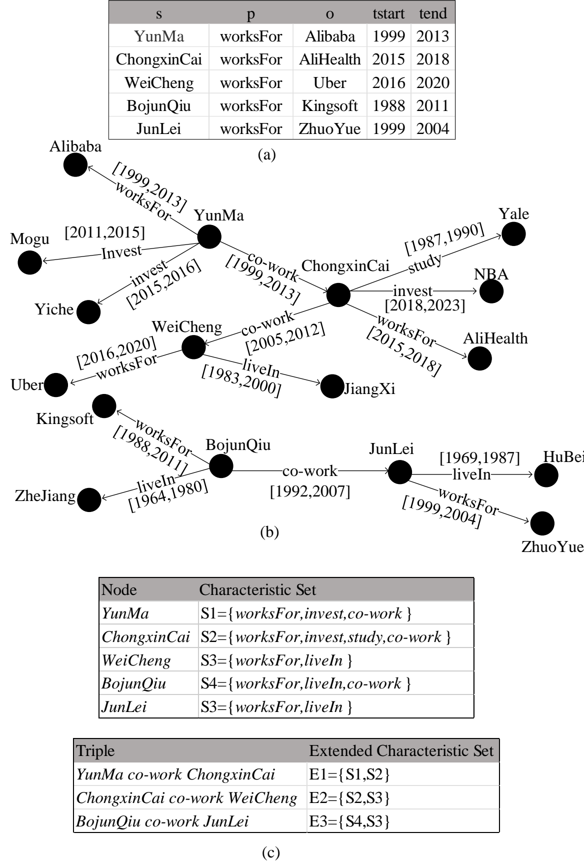
2 RELATED WORK

2.1 Temporal RDF

In addition to RDF reification, temporal RDF can be represented using quadruple [8][9][10] or quintuples [11][12][13]. Compared to quadruples, quintuples refine the time of temporal RDF with better scalability, greater expressiveness, and more efficient processing. Therefore, temporal RDF is expressed in quintuples in this paper. A quintuple is of the form: $(s, p, o) : [ts, te]$ or $(s, p, o, tstart, tend)$. (s, p, o) represents the triple part, $[ts, te]$ represents the time interval, and $(s, p, o) : [ts, te]$ means that the triple (s, p, o) is valid within the time interval $[ts, te]$. Several quintuples are illustrated in Fig. 1(a).

2.2 Extended Characteristic Set

Thomas Neumann and Guido Moerkotte [14] proposed the Characteristic Set (CS), which can identify node types based on the set of predicates with this node as subject. Meanwhile, Marios



s	p	o	tstart	tend
YunMa	worksFor	Alibaba	1999	2013
ChongxinCai	worksFor	AliHealth	2015	2018
WeiCheng	worksFor	Uber	2016	2020
BojunQiu	worksFor	Kingsoft	1988	2011
JunLei	worksFor	ZhuoYue	1999	2004

(a)

(b)

Node	Characteristic Set
YunMa	S1={*worksFor,invest,co-work* }
ChongxinCai	S2={*worksFor,invest,study,co-work* }
WeiCheng	S3={*worksFor,liveIn* }
BojunQiu	S4={*worksFor,liveIn,co-work* }
JunLei	S3={*worksFor,liveIn* }

Triple	Extended Characteristic Set
YunMa co-work ChongxinCai	E1={S1,S2}
ChongxinCai co-work WeiCheng	E2={S2,S3}
BojunQiu co-work JunLei	E3={S4,S3}

(c)

Figure 1: A example of quintuple(a). A temporal RDF graph (b), its CS and ECS (c).

Meimaris [5] proposed the ECS based on the CS and offered an RDF storage engine, AxonDB, based on the ECS. When two CSs have predicates between their nodes, these two CSs can be constituted as an ECS. In this configuration, the CS of the subject node of the predicate is the subject CS, and the CS of the object node of the predicate is the object CS.

Fig. 1(b) and Fig. 1(c) demonstrate the relationship between CS/ECS and triples. The CS of the node *YunMa* is *(worksFor, invest, co-work)*, and the CS of *ChongxinCai* is *(worksFor, invest, study, co-work)*, and so on for other nodes. At the same time, node *YunMa* and node *ChongxinCai* appear in the triple *(YunMa, co-work, Chongxin-Cai)* as subject and object, respectively, so the combination of their CSs form an ECS. In this ECS, the CS of *YunMa* is the subjectCS, and the CS of *ChongxinCai* is the objectCS.

2.3 Interval index

Nowadays, researchers have proposed several strategies for interval indexing by using different data structures and retrieval algorithms.

The 1D-grid[15] is a simple interval data structure in terms of interval indexing. It divides a time axis into several disjoint partitions, and each time interval is assigned to all partitions that overlap with it. If the collection contains many long intervals, the index size may grow large due to excessive replication, which increases the number of duplicate results to be eliminated.

Instead, Andreas Behrend *et al.* [16] proposes Period Index. In Period Index, the whole dataset is divided into many buckets according to a certain length from the minimum to the maximum time interval. Each bucket is further divided into cells of different levels. Period Index can quickly locate the target bucket by other algorithms, such as binary search, quickly reducing unnecessary comparisons. However, in complex Allen temporal queries, the Period Index must traverse and compare the interval in different cells in the target bucket, which incurs a significant time overhead.

On the other hand, George Christodoulou [6] proposes a hierarchical partitioned interval index structure called HINT, which creates a hierarchical structure where each level is subdivided into multiple partitions. Moreover, the time intervals in the dataset will be assigned to different partitions at different levels based on a specific allocation algorithm. HINT^m proposes some optimizations based on HINT. It uses replication and interval segmentation strategies to avoid duplicate results and reduce the number of interval matches. In addition, HINT^m efficiently supports temporal interval retrieval for temporal relations of Allen [7]. Experiments show that HINT^m outperforms other classical interval indexes, such as Interval Tree [17] and Timeline Index [18], in several aspects: throughput, query speed, space consumption, and update efficiency.

3 OPTIMIZATION OF TEMPORAL INDEXING FOR EXTENED CHARACTERISTIC SET

3.1 Definitions

Definition 1 (Temporal graph). *Temporal graph* $\boldsymbol{G = T \cup Q}$, *where* $\boldsymbol{T}$ *is a finite set of triples and* $\boldsymbol{Q}$ *is a finite set of quintuples.*

The Dataset D of *Pre-processing* in Fig. 2 is the temporal graph of Fig. 1(b).

Definition 2 (CS). *Let* $\boldsymbol{G = T \cup Q}$ *be a temporal graph and s be a subject in G. Then the CS of s is defined as* $\boldsymbol{CS(s) = \{p | \exists o : (s, p, o) \in T \vee (s, p, o, tstart, tend) \in Q\}}$ *and the set of all CS for the G is defined as*
$\boldsymbol{CS(G) = \{CS(s) | \exists p, o : (s, p, o) \in T \vee (s, p, o, tstart, tend) \in Q\}}$.

The CS of node *YunMa* in Fig. 1(b) is *{worksFor, invest, co-work}* because these are the only three predicates in the quintuple with *YunMa* as the subject. The $CS(D)$ of the Dataset D in the *Pre-processing* of Fig. 2 contains only S1, S2, S3, and S4 as illustrated in Fig. 1(c).

Definition 3 (ECS). *Let* $\boldsymbol{G = T \cup Q}$ *be a temporal graph and* $\boldsymbol{t = (s, p, o)(resp.t = (s, p, o, tstart, tend))}$ *be a tuple in* $\boldsymbol{T(resp.Q)}$. *Then the extended characteristic set* $\boldsymbol{ECS(t)}$ *of* $\boldsymbol{t}$ *is defined as*
$\boldsymbol{ECS(t) = \{CS(s), CS(o)\}}$. *The set of all ECS in G is:* $\boldsymbol{ECS(G) = \{ECS(t) | t \in T \cup Q\}}$.

The ECS of the quintuple *(YunMa co-work ChongxinCai 1999 2013)* of Dataset D in *Pre-processing* in Fig. 2 is E1 of Fig. 1(c) because the CS of the subject and object of this quintuple are S1 and S2, respectively. The set of all ECS of Dataset D contains only E1, E2 and E3 of Fig. 1(c).

Definition 4 (ECSLinks). *Let $G = T \cup Q$ be a temporal graph, $getS(t)$ be a function to get the subject of t, and $getO(t)$ be a function to get the object of t. Then ECSLinks of G is:*
$ECSLinks = \{ECS(i) \rightarrow \{ECS(m)\} | i, m \in T \cup Q \wedge CS(getO(i)) = CS(getS(m))\}$.

Fig. 2 shows the ECSLinks for Dataset D in the *Pre-procession* in the *CS/ECS Index Structure*. The reason is that among all the ECSs of Dataset D, only E1 and E2 satisfy that the object CS of the former ECS is the subject CS of the latter ECS.

Definition 5 (property bitmaps). *Let $G = T \cup Q$ be a temporal graph, $PSet$ be the set of ids of predicates in G, and $isExi(i, Sj)$ be the function that determines whether a predicate with id i in CS Sj. If it exists, it returns 1, otherwise it returns 0. The property bitset of CS Sj is a one-dimensional array defined as $PS(Sj) = (isExi(i, Sj), isExi(i+1, Sj), ..., isExi(i+n, Sj))$, $i, i+1, ..., i+n \in PSet$. The property bitmaps of G is:*

$$PB(G) = \begin{bmatrix} PS(Sj_1) \\ PS(Sj_2) \\ \vdots \\ PS(Sj_n) \end{bmatrix},$$

$Sj_1, Sj_2, ..., Sj_n \in CS(G)$.

In the property bitmaps of the *CS/ECS Index Structure* in Fig. 2, the property bitset of CS S1 is {1,1,1,0,0} because S1 contains only three predicates: worksFor, invest, and co-work. Moreover, the property bitmaps of Dataset D is the stacking of bitsets of all CSs and is a two-dimensional array.

Definition 6 (CSIndex). *Let $G = T \cup Q$ be a temporal graph and $CSid(Si)$ be a function that gets the id of CS Si. Then the CSIndex of G is defined as $CSIndex(G) = \{CSid(Si) \rightarrow PS(Si) | Si \in CS(G)\}$.*

In Fig. 2, the CSIndex of Dataset D in Pre-processing is shown in the CS/ECS Index Structure.

Definition 7 (P-SO-T table). *Let $G = T \cup Q$ be a temporal graph, $PSet$ be the set of ids of predicates in G, p be any of the predicates in $PSet$, and $GTable$ be a quintuple table consisting of the subject id, predicate id, object id, start time and end time of G together. Then the $P-SO-T$ $table$ of p is defined as*

$$P-SO-T\ table\ p = \begin{bmatrix} (sid1, oid1, tstart1, tend1) \\ (sid2, oid2, tstart2, tend2) \\ \vdots \\ (sidn, oidn, tstartn, tendn) \end{bmatrix},$$

$(sid1, p, oid1, tstart1, tend1), (sid2, p, oid2, tstart2, tend2), ..., (sidn, p, oidn, tstartn, tendn) \in GTable$. *The $P-SO-T$ $table$ of G is defined as*

$$P-SO-T\ table = \begin{bmatrix} P-SO-T\ table\ p_1 \\ P-SO-T\ table\ p_2 \\ \vdots \\ P-SO-T\ table\ p_n \end{bmatrix},$$

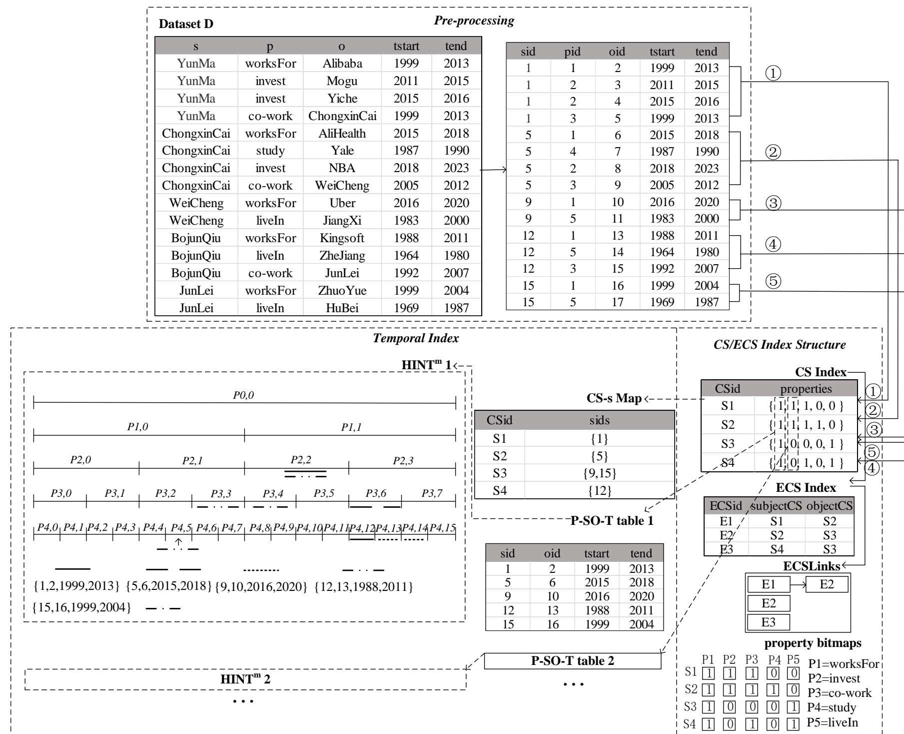


Figure 2: The H-ECS index construction process in TECStore. Dataset D is the quintuple dataset corresponding to the temporal RDF graph in Fig. 1(b).

$\boldsymbol{p_1, p_2, ..., p_n \in PSet}$.

In Fig. 2, the P-SO-T table for the predicate worksFor is P-SO-T table 1 since worksFor has a pid of 1. Inside is a table of unique identifiers for the quintuples of ignored predicates with worksFor as the predicate.

Definition 8 (CS-s Map). *Let* $\boldsymbol{G = T \cup Q}$ *be a temporal graph,* $\boldsymbol{CSids}$ *be a set of ids of CSs in* $\boldsymbol{CS(G)}$*,* $\boldsymbol{s}$ *be any subject in* $\boldsymbol{G}$*,* $\boldsymbol{sid}$ *be the id of* $\boldsymbol{s}$*,* $\boldsymbol{getCSid(Si)}$ *be a function that gets the CS id of CS* $\boldsymbol{Si}$*, and* $\boldsymbol{SIDs}$ *be a set of ids of nodes with any CS in* $\boldsymbol{CS(G)}$*. Then the* $\boldsymbol{CS{-}s\ Map}$ *of G is defined as* $\boldsymbol{CS{-}s\ Map = \{CSid \rightarrow \{sid\} | CSid \in CSids \land sid \in SIDs \land getCSid(CS(s)) = CSid\}}$.

In Fig. 2, the CS-s Map of the *Pre-processing* Dataset D is shown in the CS-s Map of the *Temporal Index*. In which the CS with CSid S3 corresponds to sids 9 and 15, and the subjects of the corresponding quintuple are *WeiCheng* and *JunLei*. The reason is that the CS of S3 is *{worksFor,liveIn}*, and only node *WeiCheng* and *JunLei* have the same CS among all the nodes in D.

3.2 H-ECS index construction

The H-ECS index contains three parts: the triple part, the temporal part, and the composite part. The role of the triple part is to find the ECS nodes that match the ECS matching condition, while the role of the temporal part is to select the ECS nodes that match the HINT^m matching condition from those ECS nodes. Finally, the composite part connects the triple and temporal parts, which can quickly determine the subject nodes corresponding to a CS and speed up the matching process.

The H-ECS index is created in several steps. First, a CS/ECS index is created for the triple part of the temporal graph, and that CS/ECS index is the triple part of the H-ECS. Subsequently, the property bitmaps in the CSIndex are split per property bit basis, and multiple HINT^m indexes are created for the set of tuples associated with the split property bits. These HINT^m indexes become the temporal part of H-ECS. Finally, the H-ECS index creates the $CS-sMap$, a key-value index storing the mapping relations between CSs and subject nodes, to minimize unnecessary matches. This key-value index becomes the composite part of H-ECS.

As shown in the *Pre-processing* section of Fig. 2, each entity and predicate assignd unique identifiers using sequential numbering starting from 1. The dataset D will be stored using a table containing five columns. Each row will sequentially store sid, pid, oid, tstart, and tend. The correspondence between the identifier of each element and the original entity will be saved in the data dictionary for the last result generation.

As shown in the *CS/ECS Index Structure* section of Fig. 2, we created the CS/ECS index and made it the triple part of the H-ECS. First, a unique identifier is assigned to the different CSs in the dataset, and their predicate sets are stored in the property bitmaps form of Fig. 2. Then, the correspondence between CS identifiers and property bitmaps is stored in the key-value index CSIndex. Similarly, the ECSIndex will hold the unique identifier of each ECS and the correspondence between ECSs. Finally, the ECSLinks defined in Definition 4 is stored.

After the construction of the triple part of the H-ECS, the construction of the temporal part of the H-ECS is started. As shown in the *Temporal Index* section in Fig. 2, we split the property bitmaps of CSIndex by property bits. The number of split blocks is equal to the number of different predicates in the dataset D. After partitioning, different blocks represent predicates, and the block number corresponds to the pid value. We then store the tuples corresponding to each block in the corresponding $P-SO-T\ table$ and construct a HINT^m index for each $P-SO-T\ table$. These HINT^m indexes become the temporal part of the H-ECS. In the process of constructing the HINT^m index, we first determine the partition range based on the minimum time and maximum time in the $P-SO-T\ table$. Then, each record in the $P-SO-T\ table$ is assigned to the corresponding interval by the HINT^m assignment algorithm with time as the entry parameter. The *Temporal Index* section in Fig. 2 shows the result of assigning the items in $P-SO-T\ table$ with pid 1 according to the HINT^m interval assignment algorithm. HINT^m is logically a hierarchical structure, and we have implemented it at the physical level using a key-value structure. The advantage of using the key-value structure is that the items in $P-SO-T\ table$ corresponding to each partition can be found quickly during the query process.

In addition, to reduce unnecessary temporal matching, another key-value index, $CS-s\ Map$, is incorporated as the composite part of H-ECS. According to the definition of CS in Definition 2, there may be several different nodes with the same CS in an RDF dataset. In order to improve the speed of temporal matching of property bits of CS, we constructs a key-value index $CS-s\ Map$ to save the mapping relationship between CS and the set of nodes with the same CS.

4 TEMPORAL QUERY

4.1 Temporal Query Syntax

James F Allen has proposed thirteen temporal relations in [7]. These relations include the seven relations shown in Fig. 3(a) and the inverse of the first six (the temporal positions before and after the

Temporal Relation	Constraint
[Ts,Te] BEFORE [T´s,T´e]	Te<T´s
[Ts,Te] MEETS [T´s,T´e]	Te=T´s
[Ts,Te] FINISHES [T´s,T´e]	Ts>T´s and Te=T´e
[Ts,Te] STARTS [T´s,T´e]	Ts=T´s and Te<T´e
[Ts,Te] DURING [T´s,T´e]	Ts>T´s and Te<T´e
[Ts,Te] OVERLAP [T´s,T´e]	Ts<T´s<Te and Te<T´e
[Ts,Te] EQUALS [T´s,T´e]	Ts=T´s and Te=T´e

(a)

iSPARQL EXAMPLE

```
SELECT ?s WHERE{
(?s worksFor Alibaba): ts1, te1.
(?s invest ?o): ts2, te2.
}FILTER(
BEFORE(?ts1,?te1,2015,2018)
&&
DURING(?ts2, ?te2,2014,2017)
)
```

(b)

Figure 3: thirteen temporal relations and the corresponding constraints(a) and a iSPARQL example(b)

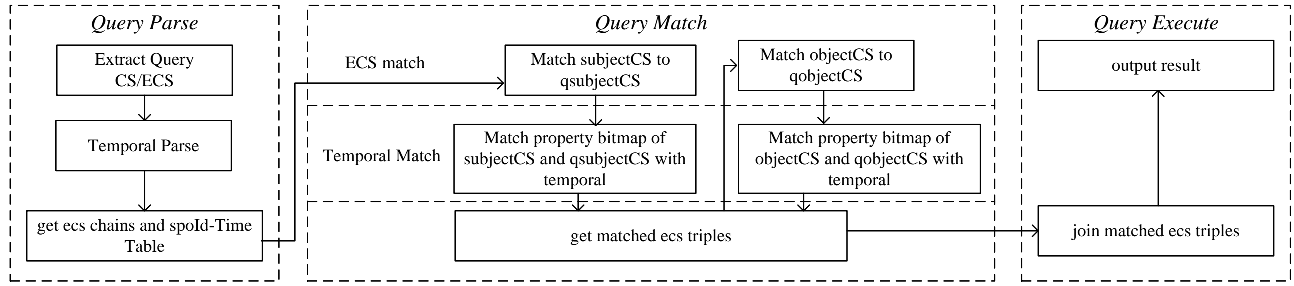


Figure 4: TECStore query process

relations are swapped). Ts, Te, and T´s, T´e represent the start time and end time of two time periods, respectively.

To fully express the temporal relationships of query statements, TECStore used iSPARQL [11] as the temporal query syntax. Fig. 3(b) shows an example of finding a person who worked at Alibaba before 2015 and had investments between 2014 and 2017.

4.2 Query processing

As shown in Fig. 4, the query processing consists of three steps: *Query Parse*, *Query Match*, and *Query Execute*. The CS/ECS in the basic graph schema, the ECSLinks of these ECSs, and the temporal query conditions are first resolved in the *Query Parse* step. For clarity of the later description, the ECSLinks of the query graph are referred to as ECS chains. Then, in the *Query Match* step, the matching ECSLinks of the ECS chains are found using the H-ECS index. Finally, these matching ECSLinks are merged and joined in the *Query Execute* step to generate the final result.

In the *Query Parse* step, the CS and ECS in the basic graph schema of the query statement and the ECS chains of these ECSs are first extracted according to the CS/ECS definitions in Definition 2 and Definition 3. Subsequently, the CSIndex and ECSIndex of the basic graph schema in the query statement are constructed. Unlike the CSIndex of the dataset, each CS in a query statement has a unique identifier, even if they have the same properties. This approach improves query speed by eliminating the need to consider cases where multiple nodes have the same CSid. Finally, temporal query conditions is extracted for each query pattern, including the CS id to which the query pattern belongs, the ids of the subject, predicate, and object of the non-variable, constraint time interval, and the keywords. In the TECStore, this information is stored in a 7-column two-dimensional array called $spoId - Time\ Table$. Each row of this two-dimensional array represents a temporal query condition for a query pattern. The order of the columns is CS id, subject id, predicate id, object id, start time of the constraint time, end time of the constraint time, and id of the keyword, in that order.

In the *Query Match* step, we utilized the triple part of the H-ECS index to match the ECS nodes in the ECS chain and the ECS nodes in the ECSLink in the order of the ECS nodes. The matching of the ECS includes the property bitmaps matching of the subject CS and the object CS, and we utilized the temporal part of the H-ECS index to perform the temporal matching after the matching of each property bitmap is finished. This process also utilizes the composite part of the H-ECS index to interrupt unnecessary matching in time to improve the matching speed.

Algorithm 1 shows the subjectCS matching process in detail, and the objectCS matching process is the same as the subjectCS. Line 2 utilizes the composite part of the H-ECS index to obtain the ids of nodes whose CS is jsubjectCS. Then, in line 3, the triple part of H-ECS is used to find the property bitset of jsubjectCS. Lines 4 to 8 find the query pattern corresponding to all predicates of qsubjectCS. Lines 9 to 14 are the property bitmaps and temporal matching of the query pattern, where line 13 utilizes the temporal part of the H-ECS index. The temporalMatching function in line 13 uses the HINT^m matching algorithm. It first finds a matching interval based on the input keywords and then equivalently matches the constant subject and object of the P-SO-T data within the interval.

Algorithm 1 subjectCS Match

Input: jECS: current matching ECS in ECSLink,
qECS: current matching ECS in ECS chain,
CS-sMap, CSIndex, spoId-TimeTable.
Output: sids:A result set of sid

1: $jsubjectCS \leftarrow jECS.getSubjectCS()$
$qsubjectCS \leftarrow qECS.getSubjectCS()$
2: $qCSItems \leftarrow newList()$
$sids \leftarrow CS - sMap.get(jsubjectCS)$
3: $propertyBitSet \leftarrow CSIndex.get(jsubjectCS)$
4: **for** $i = 0$ to $spoId - TimeTable.size$ **do**
5: **if** $spoId - TimeTable[i][0] == qsubjectCS$ **then**
6: $qCSItems.add(spoId - TimeTable[i])$
7: **end if**
8: **end for**
9: **for** $j = 0$ to $qCSItems.size$ **do**
10: $pid \leftarrow qCSItems.get(j)[2]$
11: **if** $propertyBitSet.get(pid) == 0$ or $sids.size == 0$ **then**
12: **return** $null$
13: **end if**
14: $sids \leftarrow temporalMatching(qCSItems.get(j))$
15: **end for**
16: **return** $sids$

After match, the quintuple with the sid in the sids as the subject is collected. Moreover, with jECS as the key, the collection of triple part of these quintuples as the value is stored in the key-value index

$ecsTriples - Map$, which will provide a source of results for the *Query Execute* step.

In the *Query Execute* step, merge joins are performed on the triples in $ecsTriplesMap$. The ECSLinks matched by each ECS chain are calculated in the *Query Match* step. The tuples of the ECS nodes inside each ECSLink are merge-joined first, and then hash-joins are performed on the common properties of the different ECSLinks to obtain the final query result. The merge join algorithm used by TECStore is similar to AxonDB.

5 EVALUATION

To validate the effectiveness of the strategy suggested in this work, the performances of TECStore, AxonDB, and PeriodDB are compared in terms of storage space, index building speed, and query speed. Among them, AxonDB is a classic RDF database based on ECS. It adopts the RDF reification temporal data model that can be directly applied to the ECS index. PeriodDB is a database based on AxonDB with Period Index [16]. It adopts the same temporal data model of quintuple as TECStore and adjusts the index construction and query algorithm for the characteristics of the Period Index to support temporal query. TECStore, AxonDB, and PeriodDB are all implemented in Java 1.8. All experiments were conducted on an Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00 GHz server running CentOS 7.7 with 128GB RAM.

The experimental datasets include synthetic and real-world datasets. The synthetic dataset uses the LUBM [19] dataset, which is a synthetic dataset constructed using the LUBM data generator. The real-world datasets include YAGO2 [20], a Wikipedia-based spatiotemporal real-world dataset, and FIN, a real-world dataset of equity transactions in the financial sector. The information of different datasets is shown in Table 1, where time-ratio indicates the proportion of tuples with time to the total tuples, max-path indicates the maximum number of connected triples(The object of the former triple is the subject of the latter triple), and predicate-num refers to the number of different predicates.

Table 1: Dataset characteristics

source	name	triples	time-ratio	max-path	predicate-num
LUBM	lubm-10M	10312342	100%	9	8
	lubm-20M	21233245	100%	9	8
FIN	fin-30W	363216	25%	5	5
	fin-1M	1089604	27%	5	5
YAGO2	yago2-3M	3268944	50%	3	6
	yago2-5M	5085024	50%	3	6

5.1 Storage Space

As shown in Fig. 5, TECStore occupies less storage space than the other two systems in all experimental datasets. In the more time-intensive YAGO2 and LUBM datasets, TECStore occupies 71% to 75% less storage space than AxonDB and 51% to 67% less storage space than PeriodDB.

There are several reasons for the experimental results. First, TECStore and PeriodDB reduce the complexity of temporal RDF data at the level of the underlying storage model. At the same time, the RDF reification method used by AxonDB increases the number of nodes and relationships in the dataset, as it takes five triples to represent a quintet containing time using reification, which results in AxonDB taking up more space than the other two systems. Second, the additional space taken up by the buckets and cells of the Period Index used by PeriodDB increases with time, causing PeriodDB to take up more space than TECStore.

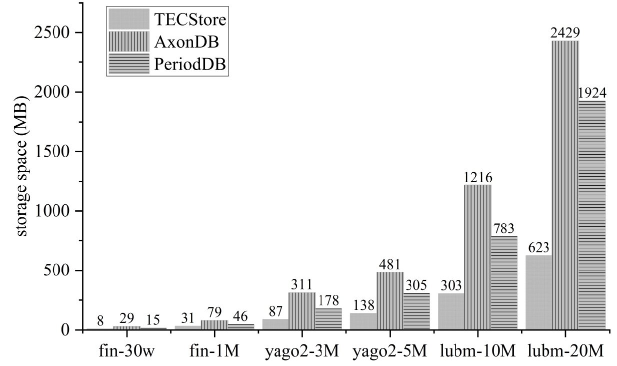


Figure 5: storage space comparison

5.2 Index Building Time

In this paper, the speedup ratio is employed as a metric to explain the time-consumption comparison of each system. As shown in Equation (1), T_f is the time consumed by AxonDB or PeriodDB, and T_t is the time consumed by TECStore.

$$S_t = T_f / T_t \tag{1}$$

In the case of TECStore, the index building time refers to the creation of the H-ECS index presented in this paper. For AxonDB, the index building time refers to the building time of its ECS index. Finally, for PeriodDB, the index building time includes the combined time of building the ECS and Period indexes. The experimental results in Fig. 6 show that TECStore builds indexes 4 to 28 times faster than AxonDB and 1.6 to 8.5 times faster than PeriodDB.

The main reason is that the H-ECS index used by TECStore does not have too many loops or nested loop operations. In contrast, RDF reification leads to only five different predicates in the dataset, resulting in only two CSs in the entire dataset. AxonDB also consequently spends much time looking for object-subject joins when extracting ECSs. In addition, PeriodDB needs to divide the entire dataset into several buckets using traversal and then traverse the entire dataset again to assign the time intervals to the cells of different buckets in the index building process. In the process of assigning, it is necessary to traverse the buckets again. Multiple traversals of the entire dataset and the nested loops cause the index building time of PeriodDB to be higher than the one of TECStore.

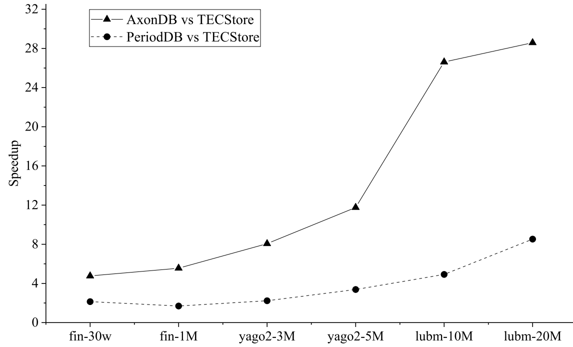


Figure 6: index building speed comparison

5.3 Query Performance

To evaluate the performance of TECStore's temporal queries, three types of queries are defined in this paper[1]:

1. The simple query, which contains only one triple pattern;
2. The path query, which consists of multiple connected triple patterns, where the object of the previous triple is equal to the subject of the subsequent triple;
3. The star query, which consists of more than two triple patterns with the same subject.

For TECStore and PeriodDB, all queries in this paper are written using iSPARQL statements. In addition, for AxonDB, query statements are converted to RDF reification form and temporal relations will be filtered using the FILTER keyword.

To more accurately evaluate the query efficiency of TECStore, all three types of queries are evaluated on different scales for each dataset, and the geometric mean of all the results is the final query results. For simple queries, we evaluated every query with temporal relationships. For path queries, we evaluated all queries with the number of triples ranging from 2 to the max-path in the evaluated dataset, with time constraints added to each triple. For star queries, on the other hand, we evaluated all queries with the number of triples ranging from 2 to the predicate-num in the evaluated dataset, with half of the triples adding a time limit.

The speedup ratio in Equation (1) is used to represent the elapsed time ratio of the other systems to TECStore. As shown in Fig. 7, the query efficiency of TECStore is tens of times higher than AxonDB and more than 100 times higher in lubm-20M datasets. There is also up to 26 times improvement compared to PeriodDB. The first reason is that the triple part of the H-ECS index used by TECStore can quickly exclude a large number of non-target tuples based on the structural features of the tuples. Then, the temporal part of the H-ECS can pick out tuples that meet the conditions of the temporal query from the non-excluded tuples. Moreover, the composite part of the H-ECS can accelerate the matching process. On the other hand, AxonDB adopts the RDF reification approach, which makes the whole dataset have at most two CSs: (hasSubject,hasPredicate,hasObject) and (hasSubject,hasPredicate,hasObjec t ,hasStartTime, hasEndTime). This makes many nodes have the same CS, and CS matching cannot filter out many nodes at once. As a result, exponential time consumption arises from the need to compare each potential node in the matching process. Secondly, the Period Index used by PeriodDB can only query intersecting intervals in temporal query, resulting in querying other different temporal relations requiring further traversal judgment. These indirect operations will generate much time overhead, so the query efficiency will be lower than TECStore.

In addition, we also observe that the query efficiency advantage of TECStore becomes increasingly evident as the amount of data increases. The reason is that the H-ECS index proposed in this paper not only fully utilizes the ability of ECS to answer complex queries quickly and the ability of HINT^m to execute temporal queries quickly but also reduces unnecessary pattern matching to improve query speed. These advantages will become more and more obvious as the complexity of the query increases.

[1]The queries for TECStore: https://github.com/EanWo/TECStore

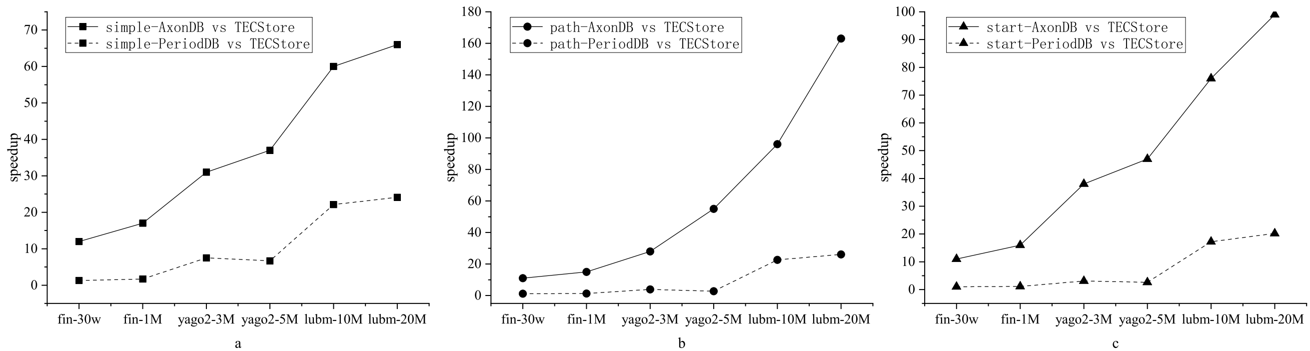


Figure 7: speedup ratios of AxonDB(solid line) and PeriodDB(dashed line) for three query types: simple(a), path(b), and star(c) on different datasets

6 CONCLUSIONS

In order to realize efficient management of temporal RDF data, the new composite H-ECS index based on an efficient RDF index ECS and an advanced time interval index is proposed in this paper. Also, based on H-ECS, we proposed a specific pattern matching algorithm for typical queries on temporal RDF. In addition, a temporal RDF data storage and query engine called TECStore is implemented, equipped with H-ECS indexing and the matching algorithm proposed in this paper. Compared to an ECS-based AxonDB database, our architecture reduces storage space by 71% to 75% and improves query and index building speeds by tens of times. Storage space, index building speed and query speed are significantly improved compared to PeriodDB, a database combining AxonDB and Period Index.

Currently, H-ECS indexing is only supported to run on a single machine, and its performance in handling big data still needs to be improved. We plan to investigate further the distributed extension of the H-ECS index in future work. Meanwhile, the H-ECS index only supports two-pair relationships based on ordinary graphs, simplifying the complexity of data relationships. At the same time, hypergraphs can more accurately describe the relationships between multiple associated entities. Therefore, we will also investigate the use of the H-ECS index to manage hypergraph data in our future work.

ACKNOWLEDGMENT

This work is supported by the Science and Technology Innovation 2030 - "New Generation Artificial Intelligence" (2020AAA0108501), the National Natural Science Foundation of China (Grant No. 622610 23) and Innovation and Entrepreneurship Training Projects for Students in Hubei Province(Grant No.S202310488123).

REFERENCES

[1] 2004. Rdf Reification. https://www.w3.org/wiki/RdfReification
[2] Tianyi Zhou, Yu Liu, Hang Zhang, Feng Gao, and Xiaolong Zhang. 2022. Optimization of Bit Matrix Index for Temporal RDF. In *China Conference on Knowledge Graph and Semantic Computing*. Springer, 95–107.
[3] ZHANG Zhe-qing WANG Yin-di and YAN Li. 2021. Indexing Bi-temporal RDF Model. *Computer Science* 48, 4 (4 2021), 63–69.
[4] Li Yan, Ping Zhao, and Zongmin Ma. 2019. Indexing temporal RDF graph. *Computing* 101 (2019), 1457–1488.
[5] Marios Meimaris, George Papastefanatos, Nikos Mamoulis, and Ioannis Anagnostopoulos. 2017. Extended characteristic sets: graph indexing for SPARQL query optimization. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 497–508.
[6] George Christodoulou, Panagiotis Bouros, and Nikos Mamoulis. 2023. HINT: a hierarchical interval index for Allen relationships. *The VLDB Journal* (2023), 1–28.
[7] James F Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11 (1983), 832–843.
[8] Gutierrez, Claudio, Hurtado, A. Carlos, Vaisman, and Alejandro. 2007. Introducing Time into RDF. *IEEE Transactions on Knowledge & Data Engineering* (2007).
[9] Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. 2013. Representation and querying of valid time of triples in linked geospatial data. In *The Semantic Web: Semantics and Big Data: 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings 10*. Springer, 259–274.
[10] Jakob Huber. 2014. Temporal reasoning for RDF (S): A Markov logic based approach. (2014).
[11] Melisachew Wudage Chekol, Giuseppe Pirrò, and Heiner Stuckenschmidt. 2019. Fast interval joins for temporal SPARQL queries. In *Companion Proceedings of The 2019 World Wide Web Conference*. 1148–1154.
[12] Melisachew Chekol, Giuseppe Pirrò, Joerg Schoenfisch, and Heiner Stuckenschmidt. 2017. Marrying uncertainty and time in knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.
[13] Boris Motik. 2012. Representing and querying validity time in RDF and OWL: A logic-based approach. *Journal of Web Semantics* 12 (2012), 3–21.
[14] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 984–994.
[15] J-P Dittrich and Bernhard Seeger. 2000. Data redundancy and duplicate detection in spatial join processing. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE, 535–546.
[16] Andreas Behrend, Anton Dignös, Johann Gamper, Philip Schmiegelt, Hannes Voigt, Matthias Rottmann, and Karsten Kahl. 2019. Period index: A learned 2d hash index for range and duration queries. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*. 100–109.
[17] H. Edelsbrunner. 1980. *Dynamic Rectangle Intersection Searching*. Inst. https://books.google.com/books?id=OGmkPgAACAAJ
[18] Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter Michael Fischer, Donald Kossmann, Franz Färber, and Norman May. 2013. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 1173–1184.
[19] Yuanbo, Guo, , , Zhengxiang, Pan, , , Jeff, and Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics Science Services & Agents on the World Wide Web* (2005).
[20] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Elsevier* (2013).