

Kennesaw State University

DigitalCommons@Kennesaw State University

Master of Science in Computer Science Theses

Department of Computer Science

Fall 12-15-2021

Graph based management of temporal data

alex Fotso

Follow this and additional works at: https://digitalcommons.kennesaw.edu/cs_etd



Part of the [Data Storage Systems Commons](#)

Recommended Citation

Fotso, alex, "Graph based management of temporal data" (2021). *Master of Science in Computer Science Theses*. 50.

https://digitalcommons.kennesaw.edu/cs_etd/50

This Thesis is brought to you for free and open access by the Department of Computer Science at DigitalCommons@Kennesaw State University. It has been accepted for inclusion in Master of Science in Computer Science Theses by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact digitalcommons@kennesaw.edu.

Graph based management of temporal data

A Thesis presented to The Faculty of the
Computer Science Department

By

Alex Fotso

In Partial Fulfillment
of Requirements for the Degree
Master of Science, Computer Science

Kennesaw State University

Fall 2021

Graph based management of temporal data

Approved:

DocuSigned by:
Ramazan Aygun December 13, 2021
6A0A9AE64C45477

Dr. Ramazan Aygun - Advisor

DocuSigned by:
Rebecca Rutherford December 13, 2021
669713A768A4439

Dr. Rebecca Rutherford – Computer Science Chair

DocuSigned by:
Sumanth Yenduri December 13, 2021
D04458D098CF4E8

Dr. Sumanth Yenduri - Dean

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of this thesis which involves potential financial gain will not be allowed without written permission.

Alex Fotso

Notice to Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement

The author of this thesis is:

Alex Fotso

The director of this thesis is:

Dr. Ramazan Aygun

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

Graph based management of temporal data

An Abstract of
A Thesis Presented to
The Faculty of the Computer Science Department

By

Alex Fotso

Bachelor of Science in Electrical Engineering Technology, Kennesaw State University, 2016

In Partial Fulfillment
of Requirements for the Degree
Master of Science, Computer Science

Kennesaw State University

Fall 2021

Abstract

In recent decades, there has been a significant increase in the use of smart devices and sensors that led to high-volume temporal data generation. Temporal modeling and querying of this huge data have been essential for effective querying and retrieval. However, custom temporal models have the problem of generalizability, whereas the extended temporal models require users to adapt to new querying languages. In this thesis, we propose a method to improve the modeling and retrieval of temporal data using an existing graph database system (i.e., Neo4j) without extending with additional operators. Our work focuses on temporal data represented as intervals (event with a start and end time). We propose a novel way of storing temporal interval as cartesian points where the start time and the end time are stored as the x and y axis of the cartesian coordinate. We present how queries based on Allen's interval relationships can be representing using our model on a cartesian coordinate system by visualizing these queries. Temporal queries based on Allen's temporal intervals are then used to validate our model and compare with the traditional way of storing temporal intervals (i.e., as attributes of nodes). Our experimental results on a soccer graph database with around 4000 games show that the spatial representation of temporal interval can provide significant performance (up to 3.5 times speedup) gains compared to a traditional model.

Graph based management of temporal data

A Thesis Presented to The Faculty of the
Computer Science Department

By

Alex Fotso

In Partial Fulfillment
of Requirements for the Degree
Master of Science, Computer Science

Advisor: Dr. Ramazan Aygun

Kennesaw State University

Fall 2021

Acknowledgment

I would like to thank my mother, Denise Fotso, for supporting me through my education for all these years. I would like to thank my father, Nestor Fotso, for bringing me up and guiding me towards this amazing career. I would like to thank Dr. Aygun for being an excellent advisor and guidance. He always made himself available to help me out through my research and writing every time I had a problem. I would like to thank Dr. Donghyun Kim who gave me my first opportunity as Research assistant and guided me through my first steps in the research world. I would like to thank Dr. Jing (Selena) He for her course in advanced data structure and analysis. This course inspired me to dig more into database systems and motivated me through my research. I would like to thank my committee members: Dr. Ramazan Aygun, Dr. Yong Shi, Dr. Junggab Son for their support and advice. I would like to thank the Kennesaw writing center for taking their time to help me through the writing of my thesis. I would like to thank the computer science department from KSU for providing the financial and technical support to help me through this research.

Table of Contents

<i>Chapter 1 Introduction</i>	1
1.1 Motivation	2
1.2 Method	4
1.3 Organization	5
<i>Chapter 2 Related Work</i>	6
2.1 Interval Labeled Temporal Graphs	6
2.2 Duration Labeled Temporal Graphs	11
2.3 Snapshot Based Temporal Graphs	12
2.4 Summary	13
<i>Chapter 3 Approach</i>	14
3.1 Spatial Data Structure	14
3.2 Our Temporal Data Model	15
3.3 Spatial Point Comparison	18
3.4 Allen’s Interval Relationships	19
3.4.1 Before/After Relationship.....	21
3.4.2 During Relationship	22
3.4.3 Meets / Met By Relationship	24
3.4.4 Overlaps/Overlapped By Relationship	26
3.4.5 Starts/Started By	29
3.4.6 Finishes/Finished By Relationship	31
3.4.7 Equals Relationship.....	33
3.5 Other Queries	34
3.5.1 Previous/Next	34
3.5.2 Conditional Before/After	35
3.5.3 Conditional Time Instance	35
3.5.4 Conditional Temporal Graph Traversing	36
3.6 Summary	37
<i>Chapter 4 Experiments</i>	38
4.1 Dataset	40
4.2 Performance Comparison	44
4.2.1 Before/After Relationship.....	44
4.2.2 During Relationship	48
4.2.3 Meets / Met By Relationship	51
4.2.4 Overlaps/Overlapped By Relationship	54
4.2.5 Starts/Started By	57
4.2.6 Finishes/Finished By	59
4.2.7 Equals.....	61
4.3 Other Query Examples	63
4.3.1 Previous/Next	63

4.3.2 Conditional before/after	65
4.3.3 Conditional time instance.....	67
4.3.4 Conditional temporal graph traversing	70
4.4 Summary.....	72
<i>Chapter 5 Conclusion and Future work</i>	73
5.1 Future work	73
<i>References.....</i>	74

List of figures

Figure 1-1: Friendship intervals between 3 users on a social network	3
Figure 2-1: Picture taken from [12] showing their proposed temporal indexing	7
Figure 2-2: Sample query using model in [12]	7
Figure 2-3: Sample node from [13] and [4] proposal with time represented as interval (T_1 as start time and T_2 as end time).....	8
Figure 2-4: A temporal graph with varying types of nodes from [4].....	9
Figure 2-5: Query from [4] : finding all continuous paths of friends between Mary Smith Taylor and Peter Burton, in the interval [2018, 2020], with a minimum length of 2 and maximum length of 3.	9
Figure 3-1: Temporal representation of events in proposed model	15
Figure 3-2: Temporal representation of events in traditional way.....	16
Figure 3-3: Spatial representation of temporal events	16
Figure 3-4: Cartesian diagram showing areas representing various comparison operators.....	18
Figure 3-5: Diagram showing Allen's before/after relationship	21
Figure 3-6: Cartesian representation of before/after relationship	21
Figure 3-7: Diagram showing Allen's during relationship.....	22
Figure 3-8: Cartesian representation of during relationship	23
Figure 3-9: Diagram showing Allen's meets/met by relationship.....	24
Figure 3-10: Cartesian representation of meets/met by relationship	26
Figure 3-11: Diagram showing Allen's overlaps / overlapped by Relationship	26
Figure 3-12: Cartesian representation of overlaps/overlapped by relationship	27
Figure 3-13: Diagram showing Allen's start/started by relationship	29
Figure 3-14: Cartesian representation of starts/start by relationship	29
Figure 3-15: Diagram showing Allen's finishes/finished by relationship.....	31
Figure 3-16: Cartesian representation of finishes / finished by relationship	31
Figure 3-17: Diagram showing Allen's Equals relationship	33
Figure 4-1: Sample neo4j node with traditional temporal interval storage start and end time represented as two distinct properties	39
Figure 4-2: Sample neo4j node with the point property representing the temporal interval	39
Figure 4-3: Soccer dataset with traditional temporal interval representation	40
Figure 4-4: Soccer dataset with proposed model temporal interval representation	41
Figure 4-5: Mini dataset represented on a timeline. Red are Manchester's games and Blue are Chelsea's games.....	42
Figure 4-6: Graph representation of mini dataset in our model.....	43
Figure 4-7: Graph representation of mini dataset in traditional model.....	43
Figure 4-8: Results of before query (a) our model and (b) the traditional model	45
Figure 4-9: Results of after query (a) our model and (b) the traditional model.....	46
Figure 4-10: Our model(blue) vs traditional model query time comparison(red) for before relationship of Allen's interval (a) total time, (b) consumed time, and (c) available time	47
Figure 4-11: Results of during query (a) our model and (b) the traditional model	48
Figure 4-12: Our model(blue) vs traditional model query time comparison(red) for during relationship of Allen's interval (a) total time, (b) consumed time, and (c) available time	50
Figure 4-13: Results of meets query (a) our model and (b) the traditional model.....	51
Figure 4-14: Results of met by query (a) our model and (b) the traditional model	52

Figure 4-15: Our model(blue) vs traditional model query time comparison(red) for met by relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time	53
Figure 4-16: Results of overlaps query (a) our model and (b) the traditional model	54
Figure 4-17: Results of overlapped by query (a) our model and (b) the traditional model	55
Figure 4-18: Our model(blue) vs traditional model query time comparison(red) for overlaps relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time	56
Figure 4-19: Results of starts query (a) our model and (b) the traditional model	57
Figure 4-20: Our model(blue) vs traditional model query time comparison(red) for starts relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time	58
Figure 4-21: Results of finishes query (a) our model and (b) the traditional model	59
Figure 4-22: Our model(blue) vs traditional model query time comparison(red) for finishes relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time	60
Figure 4-23: Results of equals query (a) our model and (b) the traditional model.....	61
Figure 4-24: Our model(blue) vs traditional model query time comparison(red) for Equal relationship of Allen’s interval	62
Figure 4-25: Our model(blue) vs traditional model query time comparison(red) for previous queries	64
Figure 4-26: Our model(blue) vs traditional model query time comparison(red) for conditional after	66
Figure 4-27: Our model(blue) vs traditional model query time comparison(red) for conditional time instance	69
Figure 4-28: Our model(blue) vs traditional model query time comparison(red) for conditional temporal graph traversal	71

List of Tables

Table 3-1: Summary of Allen's relations	20
Table 3-2: Summary of our model query compared to that of the traditional mode for before/after relation	22
Table 3-3: Summary of our model query compared to that of the traditional model for during relation	23
Table 3-4: Summary of our model query compared to that of the traditional model for meets/met by relationship.....	25
Table 3-5: Summary of our model query compared to that of the traditional model for overlaps/overlapped by relationship	28
Table 3-6: Summary of our model query compared to that of the traditional model for starts/started by relationship.....	30
Table 3-7: Summary of our model query compared to that of the traditional model for finishes/finished by relationship	32
Table 3-8: Summary of our model query compared to that of the traditional model for equals relationship.....	33
Table 3-9: Our model compared to traditional model for previous queries	34
Table 3-10: Our model compared to traditional model for conditional after query	35
Table 3-11: Our model compared to traditional model for conditional time instance.....	36
Table 3-12: Our model compared to traditional model for conditional temporal graph traversing	36
Table 4-1: Mini dataset for temporal queries validation.....	42
Table 4-2: Before queries using cypher query based on our model and the traditional model	45
Table 4-3: After queries using Cypher query based on our model and the traditional mode	46
Table 4-4: During queries using cypher query based on our model and the traditional model	48
Table 4-5: Meets queries using cypher query based on our model and the traditional model	51
Table 4-6: Met by queries using cypher query based on our model and the traditional model.....	52
Table 4-7: Overlaps queries using Cypher query based on our model and the traditional model	54
Table 4-8: Overlapped by queries using cypher query based on our model and the traditional model	55
Table 4-9: Starts queries using cypher query based on our model and the traditional model	57
Table 4-10: Finishes queries using cypher query based on our model and the traditional model	59
Table 4-11: Equal queries using cypher query based on our model and the traditional model	61
Table 4-12: Our model compared to traditional model previous query example	63
Table 4-13: Our model compared to traditional model conditional after query example	65
Table 4-14: Our model compared to traditional model conditional time instance query example	68
Table 4-15: Our model compared to traditional model conditional temporal graph traversing query example	70

Chapter 1 Introduction

In recent decades, there has been a significant increase in the use of smart devices and sensors that led to high-volume temporal data generation. This huge data is generated automatically from sensor devices, such as internet of things (IOT) devices, that continuously generate time-based telemetric data, or it is inputted by billions of people to online social media platforms. Querying and analyzing data based on temporal information would be important to obtain robust and relevant results as seen in the following applications:

- a) When a user goes on YouTube and searches the highlights of a soccer game between Team A and Team B that took place yesterday, sometimes the search engine presents the game between those two teams that happened a year ago. The most recent game should have been listed first, but it did not because the data is not always returned in temporal order.
- b) Online social networks produce user data that can be used in an extended range of fields. This type of data can be used to identify potential civil unrest-oriented threats [1]. Marketing companies use it to promote their products. Content delivery networks, such as Netflix or YouTube, use online user generated data to improve the quality of the content proposed to the user and thereby improving user experience. Using outdated data for such situations will generate biased results, thereby providing inaccurate contents to the users.
- c) Khandpur et al. [2] propose a way of using social media for detecting various cyber-attacks, such as distributed denial of service, data breach, account hijacking and other types of attacks. In this application, working with data that is not in an ideal temporal order could result in security issues.

With such a wide range of temporal data applications, the need for an appropriate storage mechanism for temporal data arises. In literature, temporal data involves two widely accepted types of times: *valid time* and *transaction time*. Dyreson et al. [3] define valid time as a time when a fact is true in the real world. It corresponds to the time interval when an event happened. Transaction time is the time at which an object is stored and stays valid in the database [3]. Unlike valid time, transaction time can be associated to any object, not only events. This thesis will only focus on the valid time since this is the only temporal data time that is associated with events. Temporal models for temporal data with a valid time can be divided into three main categories:

duration labeled temporal graphs, snapshot based temporal graphs, and interval based temporal graphs [4]. This thesis will only focus on interval based temporal graphs.

Temporal intervals are attributes of events. In this thesis, we will represent events or temporal relationships as nodes in a graph database. The proposed model can be used to represent two main forms of temporal events: instantaneous events and interval events. *Instantaneous events* are events that happened at a specific time or an event with a start time equal to the end time. *Interval events* are events with a start time different from end time. In this thesis, we will propose a method to improve the modeling and retrieval of these forms of data.

1.1 Motivation

With the evolution of technology and computing systems, temporal data has become a present member in most real-world problems. Most of the time, users want to query temporal data as illustrated in the following examples:

1) Online social networks:

- Who were friends with Mary **on** February 10th, 2015?
- Find all friends of John **ranked** by earliest friendship dates
- Find the peoples who graduated from Kennesaw State University **after** 2018

2) Health care:

- Who were Dr Joseph's patients **between** 2019 and 2021?
- How has patient Mary's diseases evolved **since** her last visit in 2017?
- Which patient had symptoms that **started at the same time** as patient A's symptoms but **lasted longer**?

3) Natural phenomena:

- What climate changes happened **before** the tornado appeared?
- What was the temperature **during** the tornado?
- Which cyclones **ranked in order** of appearance happened **between** 2004 and 2010?

All these queries involve some type of temporal data organization, and to address such query needs, query languages on temporal data, such as TQUEL[5], TSQL2[6], SQL3[7], ATSQL[8], have been developed. Languages for temporal XML trees like TXpath[9] have also been proposed. In addition to these models, [10] gives a good review of other models that have been studied to represent temporal data in relational database; however, storing temporal data in a relational database is not always ideal for the following reasons:

- 1) *Performance issue*: Join operations are costly in relational databases as querying temporal relations may require many joins.
- 2) *Complexity of query language*: Use of new vocabulary for queries as in [11] leads to complex query languages.
- 3) *Modeling temporal data*: Trying to represent temporal data is not straightforward if the relationships are recursive or iterative. To illustrate this, consider the situation to keep track of the friendship start and end date between users in a social network.



Figure 1-1: Friendship intervals between 3 users on a social network

Figure 1-1 shows the friendship duration between three users on a social network. Friendship between User 1 and User 2 started on T_1 and ended on T_1' . Friendship between User 2 and User 3 started on T_2 and ended on T_2' . This is an example of temporal data with recursive relationships. For example, finding temporal relationship between User 3 and User 2, between User 2 and User 1, and so on has a recursive nature. Representing such relationships on a relational data model is not straightforward. Due to these reasons, graph databases have advantages over relational databases for some temporal data applications.

Graph databases are becoming increasingly popular for various kinds of applications such as social media or network data storage and analysis. Graph databases are built over property graphs models [4]. In a graph database, data objects are represented by nodes. Edges represent the relationship between the nodes. Both nodes and edges can have properties or attributes that describe their characteristics. With these features, graph databases can support storing temporal data. Models that we will see in more details in Chapter 2 have been developed to represent temporal intervals in graph databases ([4], [12], and [13]). The work in this study will leverage graph databases for temporal querying.

1.2 Method

We can categorize the approaches using temporal graphs in the literature as follows: i) custom temporal models and ii) extended temporal models. Custom temporal models have the limitation of generalizability for different application domains. Moreover, they require specialized index structures. On the other hand, extended temporal models utilize existing graph models but extend them with additional temporal query operators with the support of additional indexing structures. This would require the users to adapt new temporal querying languages. This thesis will take a different approach by not changing the existing graph models. Rather, our proposal is to model the temporal data in a way that it can work on an existing graph model as it is without any extensions. This enables the generalizability of our approach. Our intention is not to compare our method with custom or extended models. However, we will investigate and compare our model to the base component in these models. We call the base structures in these models as the traditional model. The traditional model keeps intervals (start and end times) as nodes attributes.

Temporal intervals are attributes of events. In this thesis, we represent events or temporal relationships as nodes in a graph database. We propose a model to represent the temporal property of time interval events. In our model, the interval is defined as a cartesian point (x, y) with x representing the start time and y representing the end time. This representation enables the conversion of temporal queries into spatial queries. Such spatial representation of temporal data can be achieved with the use of an attribute graph. Attribute based graphs have become widely popular in various literatures [14], [15], and this forms the basis of most graph database that are found in the literature. Neo4j which we are going to use to represent and analyze our model is a graph database based on this framework. Even though it is not the focus of this thesis, this storage system can be used to answer the temporal query examples shown in Section 1.1. We should note that our temporal querying is not equivalent to comparing each axis with some conditions as it would be for the traditional model. Axis-based comparisons would not benefit temporal querying, hence temporal queries are represented on a spatial domain.

In this thesis, we first describe our temporal storage model. We then use Allen's interval relationships (a set of all possible relationships between two temporal intervals) [16] as a guide and show how our proposed model can be used to generate temporal query and organize temporal data. Given that multiple research studies ([17],[18],[19],[20],[21]) involving temporal data have

used a more traditional way (more details are provided in Chapter 2) of storing temporal intervals, we performed a performance analysis and comparison between our proposed model and the traditional model on a dataset containing more than 4000 entries. This dataset contains premier league games (England championship games) from 2011 to 2021 for 21 premier league teams. Each game is considered as an event with the start and end time of the game being the temporal interval of the event. The games are stored as nodes in the graph. In addition, we provide additional temporal query types using our model. The query examples involve the following query categories: i) previous/next, ii) conditional before/after, iii) conditional time instance, and iv) conditional temporal graph traversing

In summary, the purpose of this research is to study (i) how graph databases can be used to store temporal data such that graph models can be used without extensions and (ii) show how temporal queries based on Allen's intervals can be performed using graph models without the overhead encountered in relational databases.

1.3 Organization

The rest of the thesis has the following organization. Chapter 2 presents the related work. It provides the relevant temporal models pertaining to temporal interval graphs that have been found in literature. It also covers the work that has been done on other temporal data graphs (duration labeled and snapshot based temporal graphs).

Chapter 3 describes the details of our temporal data storage model. This chapter shows how a cartesian plane can be utilized for query structures based on all the 7 Allen's interval relationships and their inverses. Note that Allen's interval relationships are chosen as the basis to evaluate our model.

In Chapter 4 we evaluate our model for different types of queries and provide computational evaluation using our model. We provide how sample queries for each Allen's interval relationship is represented using our model and then provide computational performance comparisons.

Chapter 5 concludes our work and presents an overview of our future work that this proposed model leaves open to be explored.

Chapter 2 Related Work

In this chapter, we provide an overview of work done on storing and querying temporal data in graph databases. The two main methods that have generally been used for the representation of temporal data as a graph are *versioning* and *attribute-based graph models* [4]. Our method is an attribute-based approach. An attribute graph is a graph where both nodes and edges have properties. For example, Neo4j is a graph database based on attribute graphs. Debrouvier et al.[4] categorize temporal data models into three groups: *duration labeled temporal graphs*, *snapshot based temporal graphs*, and *interval based temporal graphs*. The duration labeled and interval based temporal graphs both fall under the attribute-based graph model while the snapshot based temporal graph falls under the versioning model.

2.1 Interval Labeled Temporal Graphs

An interval labeled graph can be defined as $G = (N, E)$ where N represents the set of nodes and E represents the set of edges. In this type of graph, any of the edges or nodes could have temporal attributes represented as an interval (start and end time). The work and model proposed in this thesis that will be seen in more details in Chapter 3 falls under this temporal data model category.

Different models have been developed under this category. Cattuto et al. [12] try to find a way for modeling the storage and retrieval of time varying social network data. In their proposed data model, the nodes represent the individuals in the social network and the edges represent the instances. They propose a temporal model where the temporal data is represented as frames with a frame being defined as the finest unit of temporal aggregation. Each frame is a node, and there is a main frame called the RUN frame. The run frame is connected to every other frame using a RUN_FRAME relation(edge). The authors propose a temporal indexing structure (Figure 2-1) and perform their sample queries from this temporal indexing. The indexing is structured in a way that each frame has a timeline node, and the timeline node is linked to a year node which is then linked to the month node. This relationship keeps going all the way to a finer time granularity.

While this model can be useful for events with a continuously changing time intervals, it has some limitations that makes it unsuitable for events that have fixed time intervals. 1) This temporal indexing increases the complexity of the query structure. Consider this sample query from [12]: **Get all time frames of run "HT2009", recorded between 9:00-13:00 of July 1st**

2009, ordered by timestamp. Figure 2-2 shows the query proposed by [12] to answer this question.

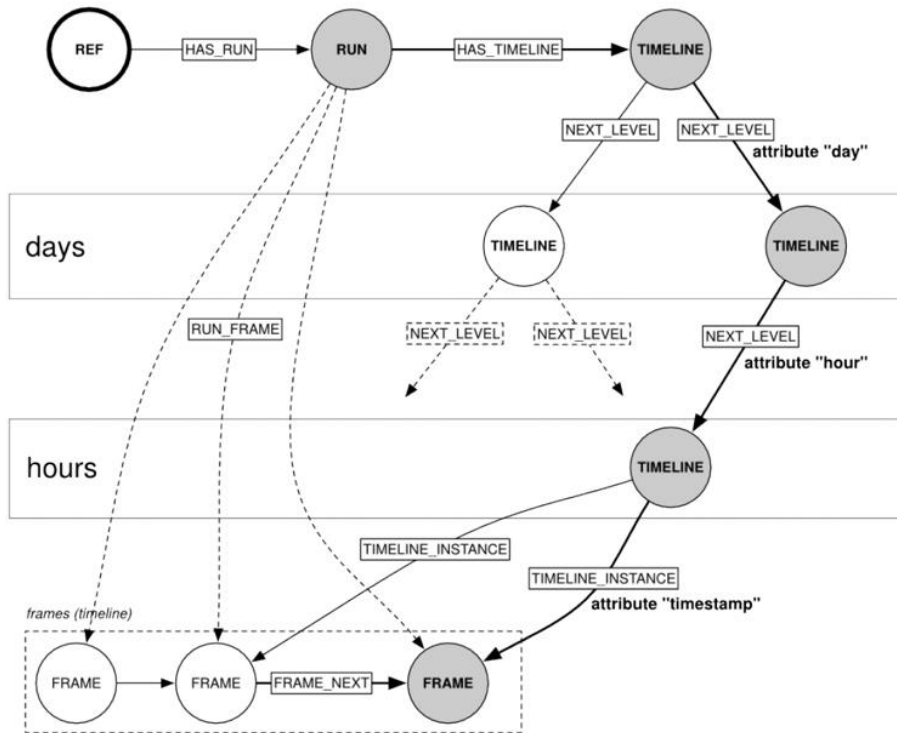


Figure 2-1: Picture taken from [12] showing their proposed temporal indexing

```

START root = node(root_node_id)
MATCH root-[:HAS_RUN]->run-[:HAS_TIMELINE]->t1,
      t1-[y:NEXT_LEVEL]->()-[m:NEXT_LEVEL]->month,
      month-[d:NEXT_LEVEL]->[h:NEXT_LEVEL]->hour,
      hour-[:TIMELINE_INSTANCE]->frame
WHERE run.name="HT2009" and y.year=2009 and m.month=7
      and d.day=1 and h.hour>=9 and h.hour<13
RETURN frame ORDER BY frame.timestamp

```

Figure 2-2: Sample query using model in [12]

In Figure 2-2, a complex match pattern needs to be provided before the “WHERE” clause to match the frame time granularity. Such query requires the user to be knowledgeable of the indexing structure and complicates querying. Ideally, querying language should not require the users to know the internal organization of temporal data. The second problem with this approach as mentioned in their paper is the performance bottleneck. The frame represents a temporal interval

and all the nodes that are linked to this temporal interval has a relationship going from the frame to the node. Some of these frames end up having multiple relationships (about 20,000), making query execution time grow to the point where the query execution becomes impractical. Hence, temporal data models that do not have such query complexity are needed.

As an alternative method, Campos et al. [13] and Debrouvier et al. [4] both use a simple model to represent interval based temporal data. [4] is an extension of work by Campos et al. [13] in which the temporal interval is represented the same way, but Debrouvier et al. provide a broader set of algorithms and operators used for temporal queries. An attribute graph is used, and the temporal property is created as an interval attribute (Figure 2-3), which is then attached to any temporal node or relationship.

Node properties:



- Label: "given label"

- **Interval:** $[T_1, T_2]$

Figure 2-3: Sample node from [13] and [4] proposal with time represented as interval (T_1 as start time and T_2 as end time)

In [4], a temporal graph example is provided as in Figure 2-4. In this graph, temporal attributes appear as a part of relationships. Moreover, temporal attributes are attached to people to indicate when they were born and died. However, building a temporal index on a graph like this is challenging. The authors consider continuous, pairwise continuous, and consecutive path queries. This is required to build an index based on Neo4j nodes including properties, the start and end nodes, the nodes in the path, and the time interval of the continuous path for continuous paths. This would require editing of the index structure as new nodes are added. Figure 2-5 shows the query for finding all continuous paths of friends between Mary Smith Taylor and Peter Burton, in the interval [2018, 2020], with a minimum length of 2 and maximum length of 3. To increase the query retrieval performance, explicit index structure is needed.

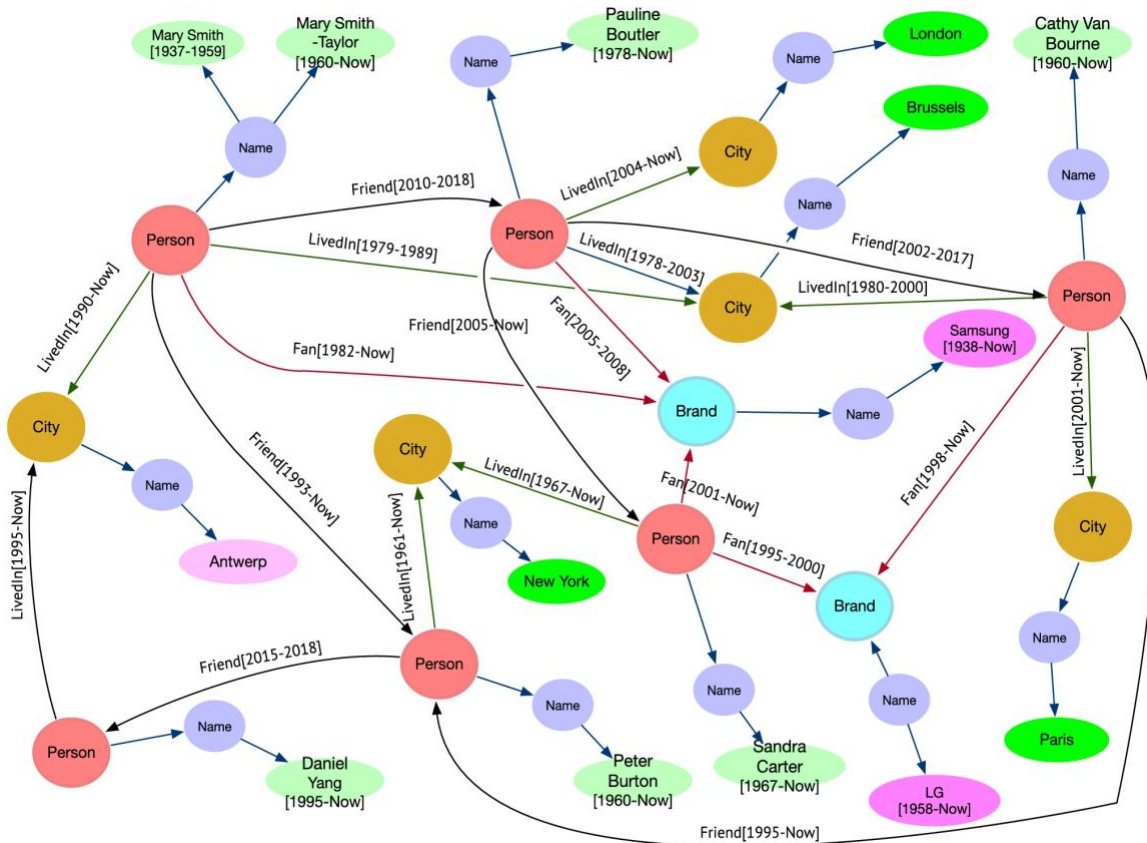


Figure 2-4: A temporal graph with varying types of nodes from [4]

```

SELECT paths
MATCH (p1:Person), (p2:Person),
paths = cPath((p1) - [:Friend*2..3] -> (p2),
              '2018', '2020')
WHERE p1.Name = 'Mary Smith-Taylor'
      and p2.Name = 'Peter Burton'

```

Figure 2-5: Query from [4] : finding all continuous paths of friends between Mary Smith Taylor and Peter Burton, in the interval [2018, 2020], with a minimum length of 2 and maximum length of 3.

For temporal intervals representation and querying, the authors developed a library with a set of procedures to incorporate as plugins in Neo4j servers. These procedures can then be used as operators to perform temporal queries. Let's assume a user wants to retrieve a set of events that happened between date T_1 to T_2 ; this model will use the keyword **BETWEEN** ' T_1 ' and ' T_2 '.

The BETWEEN operator used in this situation is not part of the native Neo4j temporal operators; it is a procedure developed by [13] and [4] in their temporal library. While Neo4j is enhanced with additional operators, the users get to learn a new set of keywords to build such queries. Additional indexing structures should be built to enhance query execution times. A temporal query representation using existing query language would be desired. Furthermore, none of those two methods show how this library compares to a more traditional way of temporal data representation (a way in which the start time and the end time are represented as two separate node attributes in the graph).

Liu et al.[17], Memarzadeh et al.[18], Yu.[19] , Zheng et al.[20] and Durand et al. [21] using temporal graph databases tried to solve various science problems: **keyword searching on graph database, tracking diseases progression, representation of natural phenomena, spatio-temporal data modeling** and **backlog and interval timestamp**, respectively. They all represent the temporal properties of their events using a traditional approach where the start time and end time of the event are individual graph attributes.

[17] represents their data as a directed graph with each node and edge being annotated with the time interval during which they are valid. The queries are formalized with the following structure: $\langle Q \rangle ::= \langle \text{KEYWORD} \rangle + \langle \text{PRED} \rangle * \langle \text{RF} \rangle *$ where KEYWORD represents one or more words included in the search, or it could also correspond to the labels of the data node. PRED represents a time predicate: time intervals during which the search results exist. It could be preceding, meets, overlaps, etc. The RF involves the ranking or order the user wants to give to the query results. Their paper considers various forms of searches that could be performed and proposes an algorithm on how to handle it.

[18] proposes a way to track diseases evolution over time using graph databases. Their data is organized in sequence where the whole graph shows the order in which the diseases evolved for each patient. Here, the time is represented as a property of the graph node, and each node represents a state of the diseases.

[19] builds a model to represent the dynamics of natural phenomenon as spatiotemporal events. The data model proposed consists of three entities: spatiotemporal (ST) objects, ST relationships, and ST events. An ST object represents an object that evolves with space and time. An ST relationship represents the relationship between consecutive time intervals, and an ST event represents the life cycle of a specific event from start to end. They do this by using a graphical

approach where spatio temporal events are represented as node and spatio-temporal relationships are represented as edges. The relation/edge indicates the changes that happens to the object from time t to time $t+1$.

Temporal Geographical Information Structures (TGIS) is a model developed by [20]. The authors realized that it is hard to convert temporal problem domain model to relational model. This leads to complex queries and low expansibility. To solve this problem, they come up with a spatio-temporal model based on graph databases.

[21] tries to find a suitable way to represent and model network data with graph databases given that there is a continuous evolution of this data with time. They design a model that supports temporal analysis with property graph databases, using a single-graph model limited to structural changes. To appropriately represent their model, they consider a group of devices, and management services which track the devices interconnection where an event represents the addition and removal of a device from the network. In their model, they represent timestamps intervals as node properties. They also introduce a global indexing and graph as an index to handle the backlogs. In the global indexing approach, indexes are linked with the creation and deletion time of items which can be used at run time to efficiently reconstruct order sets. By using the graph as index, they can come up with a backlog table indicating how the various time intervals are linked with each other.

2.2 Duration Labeled Temporal Graphs

[4] defined a duration labeled temporal graph as $G_d = (V, E)$ being a temporal graph where V is the set of vertices, and E is the set of edges. Each edge is represented as $(u, v, t, \lambda) \in E$ and represents a temporal relationship between vertex u and v starting at time t with a duration λ . Such graphs are studied by Wu et al. [22] to represent duration between vertices and to compute the shortest path or fastest path algorithms. In this temporal graph, the edges are labeled with the duration of the relationship between the two nodes. Based on this definition, the authors of [22] extended their work [23], [24], [25].

This form of temporal data representation is ideal for handling and solving path queries since the interval is summarized as duration and can be used for easier traversal/shortest path computations. The summarization of temporal intervals to duration labels comes with some

important data being omitted, so it is not ideal to solve a query such as finding the people who graduated from Kennesaw State University after 2018.

2.3 Snapshot Based Temporal Graphs

[4] defines this type of temporal graph as $G [t_i, t_j]$ in a time interval $[t_i, t_j]$ where $\{G_{t_i}, G_{t_{i+1}}, \dots, G_{t_j}\}$ is a sequence of graph snapshots. Multiple temporal graph models have been proposed in this category with most of them using a versioning method. For example, Khurana et al. [26] proposed a way to efficiently query historical data. They focus on querying the state of a network at a particular point in time (snapshot). They store the current graph and other versions of the graph representing the state of the data at various points in time. Their work helps to keep track of the evolution of data through time.

Throughout the literature we also see other models for snapshot retrieval. Copy and log are two snapshot retrieval approach models proposed by Salzberg and Tsotras [27]. The copy approach involves the storage of the database snapshot at each transaction state with the primary advantage of fast retrieval. The log approach involves recording all changes in the database annotated by time. They go further and propose a mix of the two approaches (copy+log) where a subset of the snapshots is explicitly stored. Other methods such as external index tree or segment tree [28] are also found in the literature for this same purpose. Snapshot retrieval model is not the only area where snapshot based temporal graphs have been used.

Huo et al. [29] propose a way to efficiently compute the shortest path on evolving social graphs. They extend the Dijkstra's algorithm to compute the shortest path for a time-point or time interval with the goal of efficiently querying the temporal shortest path problem within the social graph evolving history.

Algorithms for the traversal of snapshot based temporal models have also been proposed. Huang et al. [30] propose a temporal version of breadth-first search(BFS) / depth first search algorithms. A temporal graph consists of multiple snapshots where each snapshot is a non-temporal graph, so a naïve traversal approach will involve applying BFS/DFS on each of the snapshots; however, this will not be realistic since the number of snapshots in a temporal graph could be relatively large. Chrono graph is a system design proposed by Byun et al. [31] to manage and traverse temporal graph. [31] handles the limitations of [30] by converting time instant events(instantaneous events) used in [30] to time periods (interval events). In this thesis, our focus

is not on snapshot databases. Mate et al. [32] and Wongsuphasawat et al. [33] also did some work on temporal database with a focus on finding friendly ways for users to generate temporal queries.

2.4 Summary

Among the three categories of temporal graphs, interval labeled temporal graphs are closest to the proposed method in this thesis. The existing database models can be extended to support temporal data storage and querying. As the temporal data storage becomes specific, specialized indexing structures need to be built. To efficiently benefit from these extensions, new temporal querying operators are introduced. As the domain of temporal databases continue evolve, this debilitates traditional users adapting to new querying paradigms.

Our goal is to support temporal querying with existing models without extending them. This may require adapting the traditional operators of existing models for the purpose of temporal querying. However, the data should be stored in a specific way so that these inherent operators can be used without changing them. Hence, we are not trying to extend any model, but we are aiming at modeling temporal data to leverage from existing operators. In the literature, temporal information is either attached to nodes or edges for graph databases. When temporal information is attached to edges, the queries turn into path searching or pattern matching, which is not the goal here. We will compare our proposed model's performance with traditional representation where the temporal information is attached to the nodes, but without extending them with additional keywords for fair comparison.

Another problem we see in the literature is the limited indexing. Indexing is directly tied to the data, and indexing structure is heavily dependent on the data. However, we should assume that events happen at any time. Rather than data specific or model dependent explicit index structures customized according to the existing models, implicit indexing could be desired based for the temporal information.

Chapter 3 Approach

In this chapter, we explain our proposed model and show how temporal queries can be executed using our model. Since the proposed model leverages spatial data structure and benefit from spatial indexing, we start with the relevance of spatial representation. Then, we explain our temporal model where an interval is represented as a point on the cartesian coordinate system. This section also explains the mapping of time on an axis. After explaining Allen's intervals, we present how queries based on Allen's interval relationships can be expressed using our model. The major advantage of our model is that it can use an existing graph database system without any extension while benefiting from all inherent optimizations.

3.1 Spatial Data Structure

Our model for temporal data storage involves the conversion of temporal interval data to spatial data. Samet [34] defines spatial data as spatial objects made of points, lines, regions, rectangles, surfaces, volumes, and data of higher dimensions. Spatial data storage involves multidimensional key search. Multidimensional key search which is different from single key search where records could be stored and searched using only a single property such as name or ID number. Unlike single key search data structures, spatial data structures are designed in a way to achieve high performance results when it comes to multidimensional range queries such as searching for all cities within a given distance of a specific point[35]. Multi-dimensional range queries are the defining features of a spatial application.

Point data is the spatial data of interest in this thesis. Multidimensional point data could be represented in a variety of ways such as K-d tree which is preferable when dealing with dimensions greater than 3 [34] or point quadtree. Most of the point data representations available are some variants of the bucket method. The bucket method is based on data structures that are based on spatial occupancy. Spatial occupancy involves decomposing the space from which the data is drawn into regions called buckets. Examples of such bucketing methods are PR quadtree (P for Point and R for Region), R-tree and R^+ tree. R-trees are used for organizing collections of arbitrary spatial objects.

3.2 Our Temporal Data Model

This thesis aims at finding a way to organize and retrieve temporal data using a graph database. Temporal properties are represented as a cartesian point with the start time being the x-axis and the end time being the y-axis. This cartesian point will be used to perform queries on the nodes.

A temporal graph is defined as $G = (N, E)$ where N denotes the set of temporal nodes and E represents temporal ordering of semantically related intervals. A node is defined as $n = (u, \text{Interval}) \in N$ where u represents the label of the node and Interval represents the temporal property of the node.

In our model, the interval is defined as a cartesian point (x, y) with x representing the start time and y representing the end time as shown in Figure 3-1. This cartesian representation is the key difference with traditional models which have the interval represented as two individual node attributes (Figure 3-2).

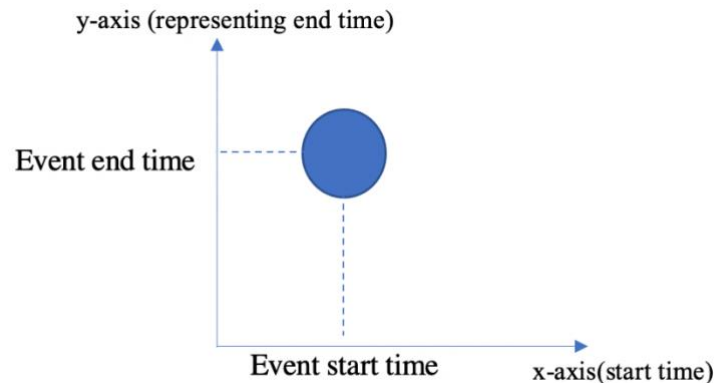


Figure 3-1: Temporal representation of events in proposed model

Node attributes:


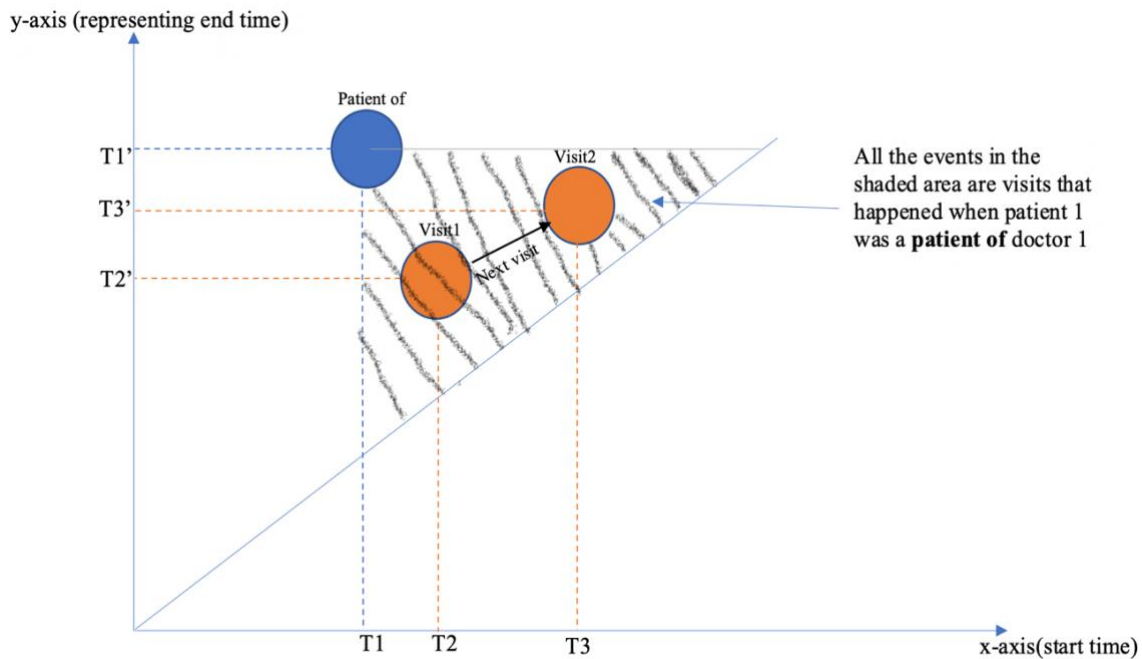
-  - Label: "given label"
- **StartTime: dateTime (T_1)**
- **EndTime: dateTime (T_2)**

Figure 3-2: Temporal representation of events in traditional way

To appropriately demonstrate our model, let us consider the following example: Patient1 was a patient of doctor1 during the time interval from T_1 to T_1' . In our model, the relationship between patient 1 and the doctor 1 is a node and is denoted as (patientOf, [T_1 , T_1']). During this time interval, patient 1 made two visits to doctor1 both denoted as (visit1, [T_2 , T_2']) and (visit2, [T_3 , T_3']). The graphical structure of this information is shown in Figure 3-3.

**Figure 3-3: Spatial representation of temporal events**

With the cartesian point system in place, the major issue is to represent time as a number to be plotted on a cartesian coordinate. Let us consider the example below to illustrate how this can be done.

John went to the hospital on July 28th, 2021, at 8:30am and he came out on August 2nd, 2021, at 8:15pm. To represent this in our model, we are going to concatenate all the numbers together to form an integer and use the time as the floating part of the number. To distinguish between am and pm time, the 12 hours o'clock convention will be transformed to the military time chart. The format to represent time is **yyyymmdd.hhmm** (4-digit year followed by 2-digit months followed by 2-digit days followed by decimal point followed by 2-digit hours followed by 2-digit minutes). Therefore,

July 28th, 2021, at 8:30am in cartesian numbers will be 20210728.0830 and
August 2nd, 2021, at 8:15pm in cartesian numbers will be 2021802.2015.

Converting the temporal data to spatial data comes with some characteristics that give a better temporal ordering performance: i) By converting the temporal data to spatial data, we can make use of the multidimensional storage capabilities provided by spatial storage systems. ii) Storing the data as a cartesian point implies storing the start and end times as an integer (or real numbers) which is different from a datetime object that stores the date as a string. From [36], it can be inferred that Integer comparison has a better performance compared to string comparison. Things could be different now with the recent advancement in technologies; however, this characteristic is part of the reasons why we believe that storing data as a point will be beneficial for temporal queries.

When converting the temporal interval to cartesian point, the following features are to be considered. i) It is important to follow the standard temporal notation year/month/day when doing the concatenation. Any different order could result in the end time number generated being smaller than the start time number generated, which will cause some future search queries to fail. ii) It is important to keep the day and month as two digits. This will ensure that the end time is not smaller than the start time. Having the time converted to cartesian point simplifies our temporal queries. It allows intervals to be represented as points to be plotted effectively and compared easily. We will demonstrate how our model can be used to perform querying based on temporal interval for each of Allen's interval relationships.

3.3 Spatial Point Comparison

Developing temporal queries based on spatial points will require points comparison, so understanding the comparison operators between spatial points is important to effectively generate temporal queries with our model.

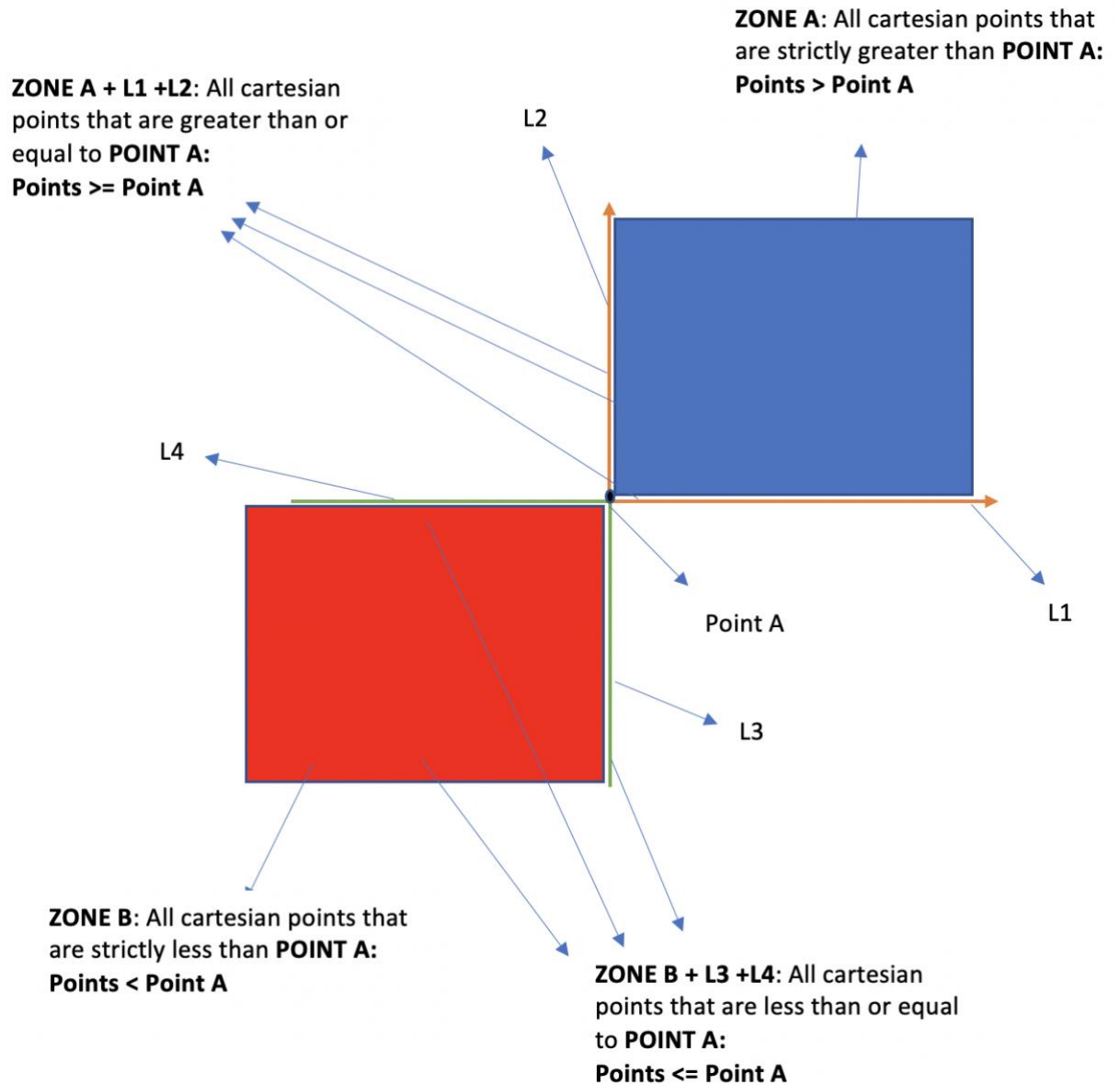


Figure 3-4: Cartesian diagram showing areas representing various comparison operators

Figure 3-4 shows the areas representing various comparison operators with respect to point A. Zone A (blue rectangle) is the zone with all the points strictly greater than point A. All the points

located in Zone A + the points located on line L1, and the points located on line L2 forms the set of points greater than or equal to point A. So, in summary we have:

$$(Points > Point A) = Zone A$$

$$(Points \geq Point A) = Zone A + L1 + L2$$

$$(Points = Point A) = Point A$$

Following the same logic, we have the following summary for less than operators:

$$(Points < Point A) = Zone B$$

$$(Points \leq Point A) = Zone B + L3 + L4$$


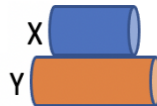


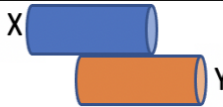


$$(Points = Point A) = Point A$$

3.4 Allen's Interval Relationships

Allen's interval relationships describe all possible positional relationships between two time periods along a common timeline[37]. These relationships were originally proposed by James F. Allen in 1983 [16] to represent temporal knowledge and temporal reasoning. Besides temporal databases, Allen's relations are also used in temporal pattern matching ([38], [39]). Queries based on Allen's relations will be used to evaluate our model. Table 3-1 shows a summary of Allen's relations. We are going to explain each relationship in the following sections.

For each Allen's interval relationship, we provide the visualizations on a spatial context, so that these queries could be designed accordingly. This is not a one-to-one mapping of conditions based on start and end times of intervals compared to the traditional model.

Table 3-1: Summary of Allen's relations

Relations	Inverse Relation	Visual Example
X BEFORE Y	Y AFTER X	
X DURING Y	Y CONTAINS X	
X EQUALS Y	Y EQUALS X	
X MEETS Y	Y MET BY X	
X OVERLAPS Y	Y OVERLAPPED BY X	
X STARTS Y	Y STARTED BY X	
X FINISHES Y	Y FINISHED BY X	

3.4.1 Before/After Relationship

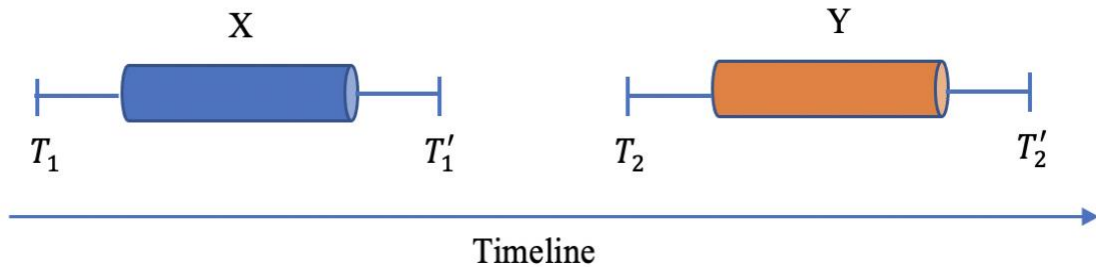


Figure 3-5: Diagram showing Allen's before/after relationship

Figure 3-5 represents Allen's *before/after* relationship. This shows the relationship between two events (X, $[T_1, T_1']$) and (Y, $[T_2, T_2']$) where X happened *before* Y meaning that $T_1 < T_1' < T_2 < T_2'$. The cartesian diagram in Figure 3-6 will help us demonstrate how we can use our model to process such temporal queries. Zone B in the diagram represents the location of all the events that happened before the interval Y that is all the X events, and the inverse is shown by Zone A. To retrieve such data with our model, we will use the reference point B for before query and point A for after query. The area covered by zone B represents all the points that are less than reference point B (T_2, T_2). So, the cypher query used to retrieve all events X that happened before Y is provided in Table 3-2.

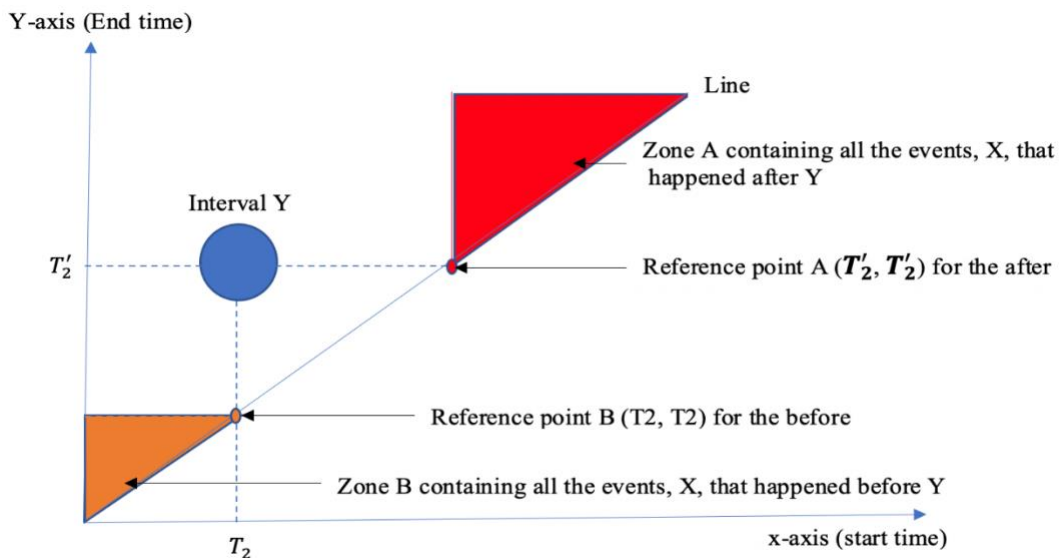


Figure 3-6: Cartesian representation of before/after relationship

We should note that using comparison operators ‘>’ and ‘<’ on a cartesian point is enough to satisfy these queries. In other words, we do not need to compare with each axis.

Table 3-2: Summary of our model query compared to that of the traditional mode for before/after relation

Temporal relation	Our model query	Traditional model query
Before	Match (<i>n</i> : event) Where <i>n.interval</i> < <i>point</i> (<i>{x:T₂, y:T₂}</i>) return <i>n</i>	Match (<i>n</i> : event) Where <i>n.endTime</i> < <i>T₂</i> return <i>n</i>
After	Match (<i>n</i> : event) Where <i>n.interval</i> > <i>point</i> (<i>{x:T'₂, y:T'₂}</i>) return <i>n</i>	Match (<i>n</i> : event) Where <i>n.startTime</i> > <i>T'₂</i> Return <i>n</i>

3.4.2 During Relationship

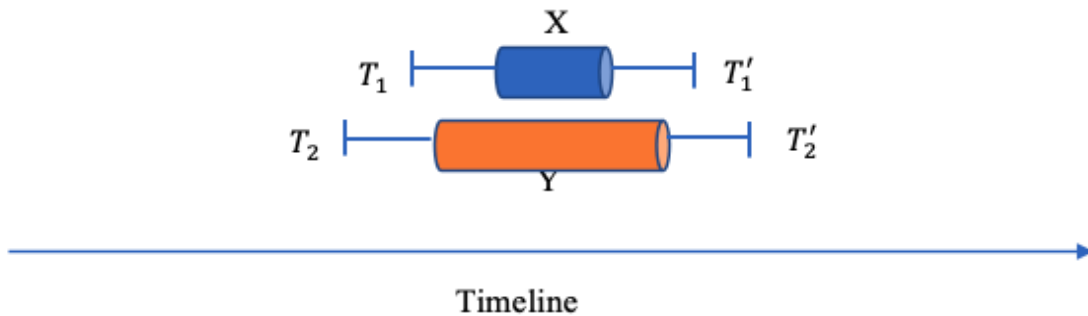


Figure 3-7: Diagram showing Allen's during relationship

Figure 3-7 represents Allen's during relationship. This shows the relationship between two events ($X, [T_1, T_1']$) and ($Y, [T_2, T_2']$) where X happened during Y meaning that: $T_2 < T_1 < T_1' < T_2'$. Figure 3-8 will help us demonstrate how we can use our model to process such temporal queries. From Figure 3-8, all the events that happened during interval Y are in Zone A. To retrieve this data, we will use the reference points A and B. The points in Zone A are all the points that are

greater than reference point B but less than reference point A. So, the general query to get all the events X that happened during interval Y is shown in Table 3-3.

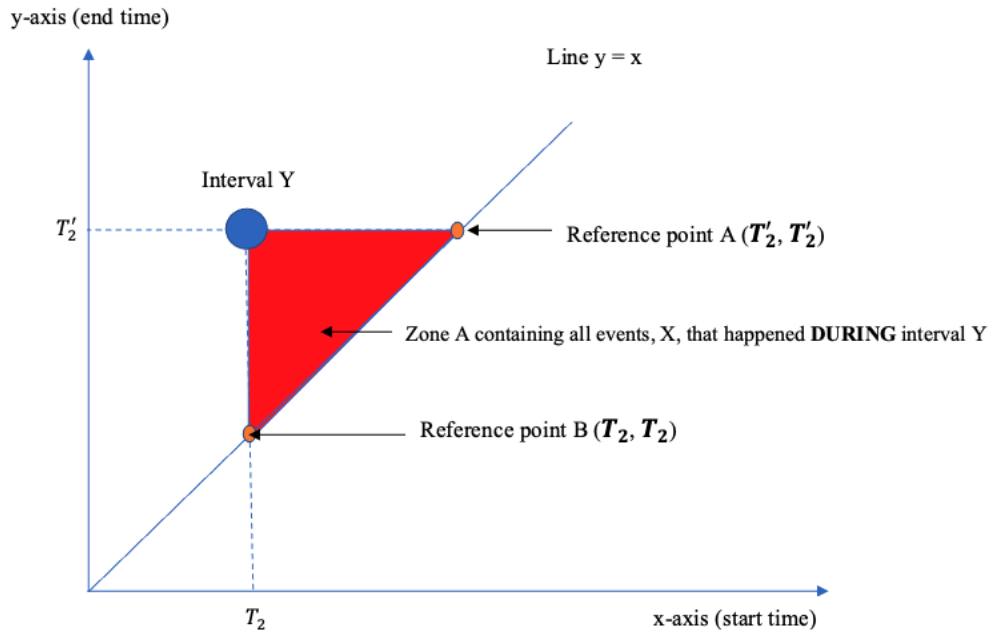


Figure 3-8: Cartesian representation of during relationship

Table 3-3: Summary of our model query compared to that of the traditional model for during relation

Temporal relation	Our model query	Traditional model query
<i>During</i>	<p><i>Match (n: event)</i></p> <p><i>Where n.Interval ></i> <i>point ({x:T₂, y:T₂'})</i> <i>AND</i> <i>n.Interval <</i> <i>point ({x:T₂' , y:T₂'})</i> <i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where n.startTime ></i> T₂ <i>AND</i> <i>n.endTime <</i> T₂' <i>return n</i></p>

3.4.3 Meets / Met By Relationship

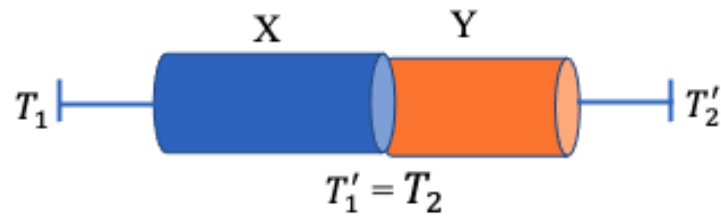


Figure 3-9: Diagram showing Allen's meets/met by relationship

Allen's *meets/met by* relationship is a relationship where the end time of one event equals the start time of another event. In Figure 3-9, the end time of event X equals the start time of event Y so, event X meets event Y, or Y is met by X. This implies that $T_1 < T'_1 = T_2 < T'_2$. Figure 3-10 shows how these events lie on the cartesian system and how they can be accessed. All the points located on L_1 are the intervals that meet Y. The points on the line L_2 represent all the events that are met by Y. Table 3-12 shows our model query compared to the traditional model query.

Table 3-4: Summary of our model query compared to that of the traditional model for meets/met by relationship

Temporal relation	Our model query	Traditional model query
<i>Meets</i>	<p><i>Match (n: event)</i></p> <p><i>Where n.Interval > =</i></p> <p style="padding-left: 40px;"><i>point ({x: 0, y: T₂})</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.Interval < =</i></p> <p style="padding-left: 40px;"><i>point ({x: T₂, y: T₂})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p style="padding-left: 40px;"><i>n.endTime = dateTime(T₂)</i></p> <p><i>return n</i></p>
<i>Met By</i>	<p><i>Match (n: event)</i></p> <p><i>Where n.Interval > =</i></p> <p style="padding-left: 40px;"><i>point ({x: T'₂, y: T'₂})</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.Interval < =</i></p> <p style="padding-left: 40px;"><i>point ({x: T'₂, y: Y_{max}})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p style="padding-left: 40px;"><i>n.startTime = dateTime(T'₂)</i></p> <p><i>return n</i></p>

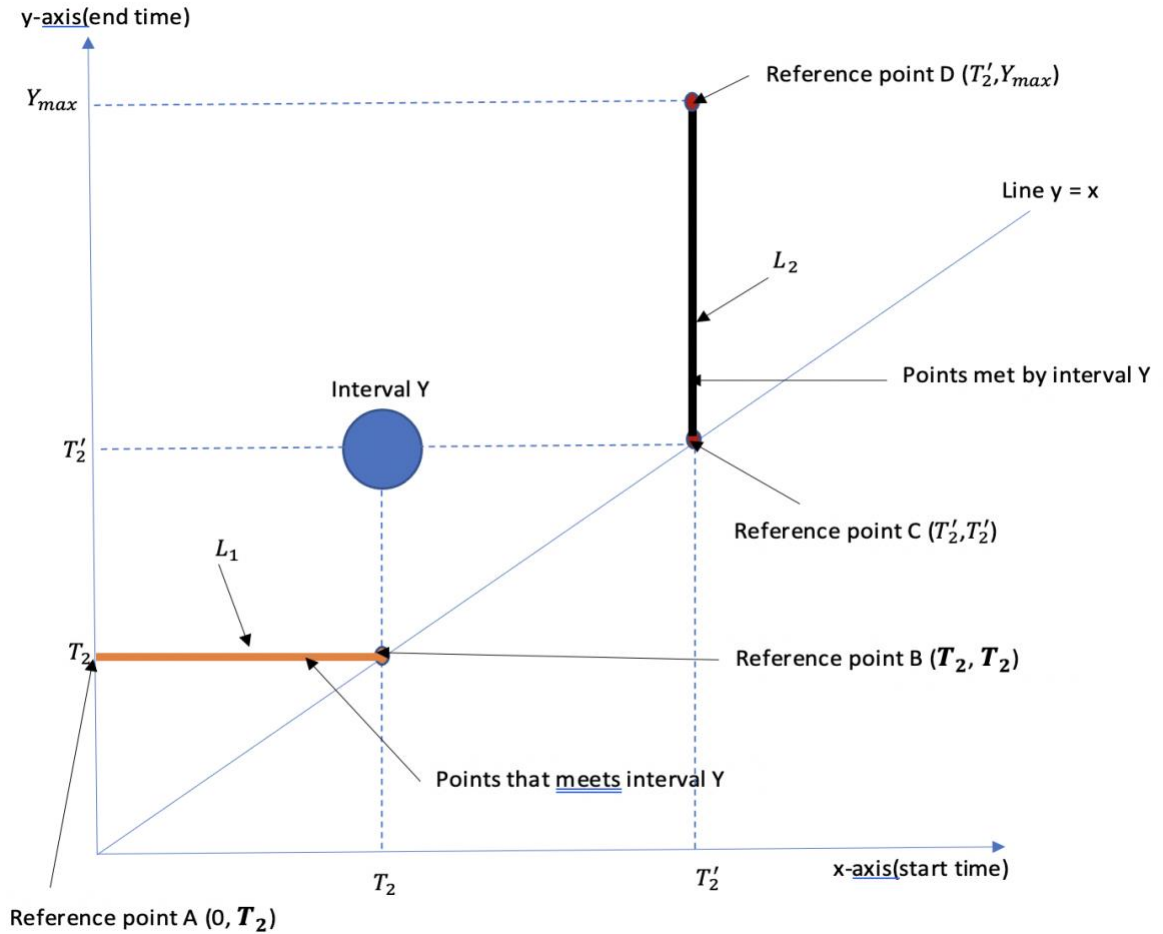


Figure 3-10: Cartesian representation of meets/met by relationship

3.4.4 Overlaps/Overlapped By Relationship

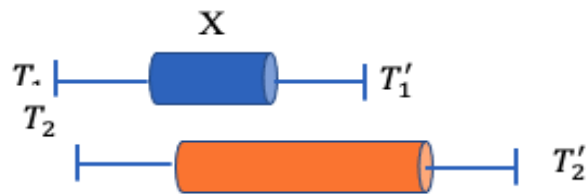


Figure 3-11: Diagram showing Allen's overlaps / overlapped by Relationship

Allen's *overlaps* relationship portrays a relationship where a portion of an event happens during the same time as another event. In Figure 3-11, the end time of event X is greater than the start time of event Y but less than the end time of Y, meaning that event X *overlaps* event Y. This

implies that $T_1 < T_2 < T'_1 < T'_2$. Overlapped By is the inverse of this relationship. We demonstrate how such events can be accessed with our model with the help of the cartesian plan in Figure 3-12. All the points located in zone A are the points that overlap with interval Y, and the ones located in Zone B are the ones that are overlapped by interval Y. Table 3-5 shows how we can get these points using our proposed model.

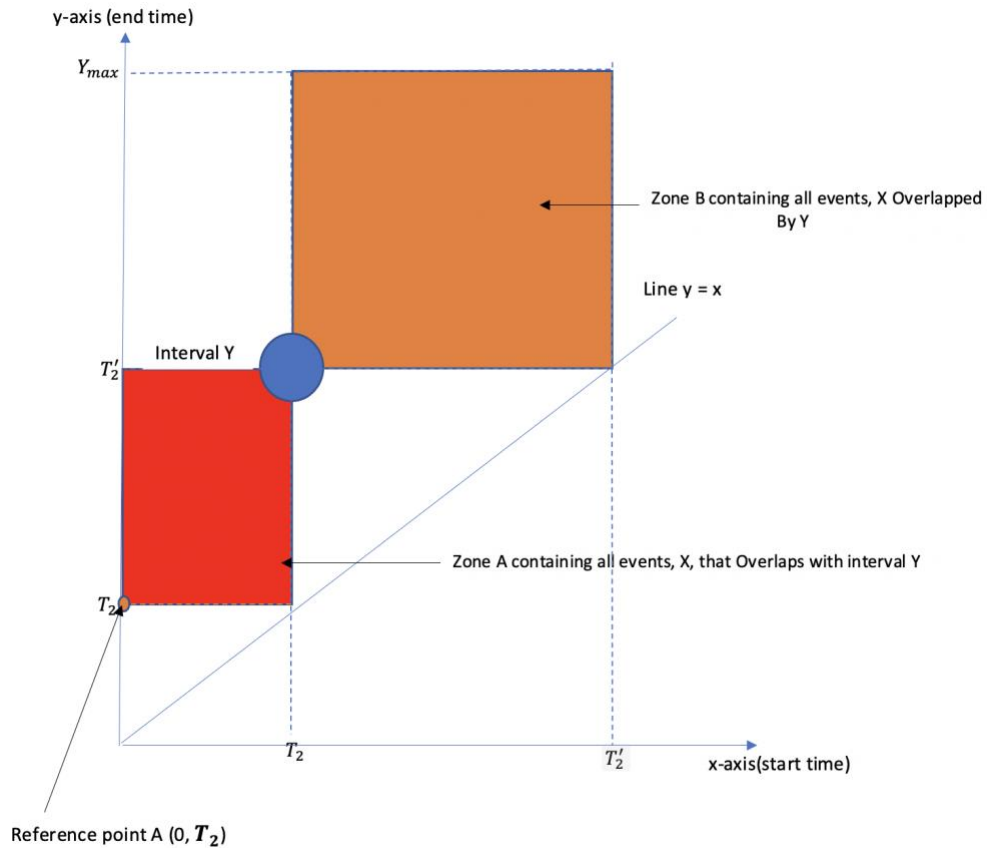


Figure 3-12: Cartesian representation of overlaps/overlapped by relationship

Table 3-5: Summary of our model query compared to that of the traditional model for overlaps/overlapped by relationship

Temporal relation	Our model query	Traditional model query
<i>Overlaps</i>	<p><i>Match (n: event)</i></p> <p><i>Where n.Interval ></i></p> <p style="padding-left: 40px;"><i>point ({x:0, y:T₂})</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.Interval <</i></p> <p style="padding-left: 40px;"><i>point ({x:T₂, y:T'₂'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where n.startTime <</i></p> <p style="padding-left: 40px;"><i>dateTime(T₂)</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.endTime ></i></p> <p style="padding-left: 40px;"><i>dateTime (T₂)</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.endTime <</i></p> <p style="padding-left: 40px;"><i>dateTime (T'₂)</i></p> <p><i>return n</i></p>
<i>Overlapped By</i>	<p><i>Match (n: event)</i></p> <p><i>Where n.Interval ></i></p> <p style="padding-left: 40px;"><i>point ({x:T₂, y:T'₂'})</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.Interval.x <</i></p> <p style="padding-left: 40px;"><i>point ({x:T'₂, y: Y_{max}'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where n.startTime ></i></p> <p style="padding-left: 40px;"><i>dateTime(T₂)</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.startTime <</i></p> <p style="padding-left: 40px;"><i>dateTime (T'₂)</i></p> <p style="padding-left: 40px;"><i>AND</i></p> <p style="padding-left: 40px;"><i>n.endTime ></i></p> <p style="padding-left: 40px;"><i>dateTime (T'₂)</i></p> <p><i>return n</i></p>

3.4.5 Starts/Started By

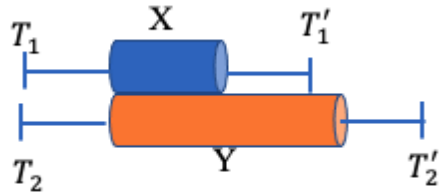


Figure 3-13: Diagram showing Allen's start/started by relationship

In Allen's *starts* relationship, the two events have the same start time, but one ends before the other. In Figure 3-13, the start time of event X equals the start time of event Y, but it ends before event Y ($T_1 = T_2 < T'_1 < T'_2$) implying that event X *starts* event Y. Started by is the inverse of this relationship. Figure 3-14 shows how such events could be accessed. The queries are summarized in Table 3-6.

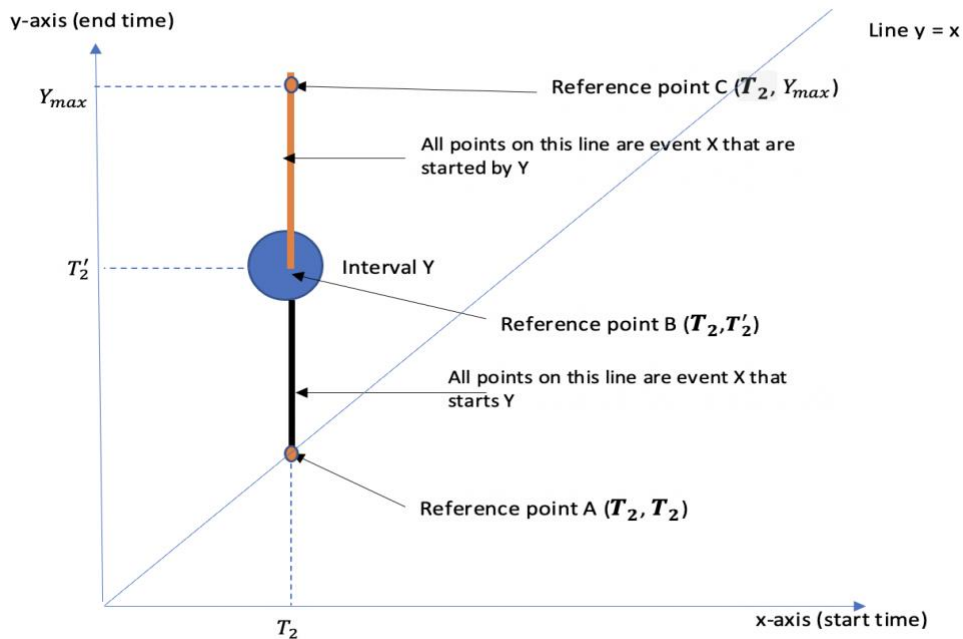


Figure 3-14: Cartesian representation of starts/start by relationship

Table 3-6: Summary of our model query compared to that of the traditional model for starts/started by relationship

Temporal relation	Our model query	Traditional model query
<i>Starts</i>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>Where n.Interval > =</i> <i>point ({x: T₂, y: T₂'})</i> <i>AND</i> <i>n.Interval < =</i> <i>point ({x: T₂, y: T₂'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where n.startTime =</i> <i>dateTime(T₂)</i> <i>AND</i> <i>n.endTime <</i> <i>dateTime (T₂')</i></p> <p><i>return n</i></p>
<i>Started By</i>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>Where n.Interval > =</i> <i>point ({x: T₂, y: T₂'})</i> <i>AND</i> <i>n.Interval < =</i> <i>point ({x: T₂, y: Y_{max}'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where n.startTime =</i> <i>dateTime(T₂)</i> <i>AND</i> <i>n.endTime ></i> <i>dateTime (T₂')</i></p> <p><i>return n</i></p>

3.4.6 Finishes/Finished By Relationship

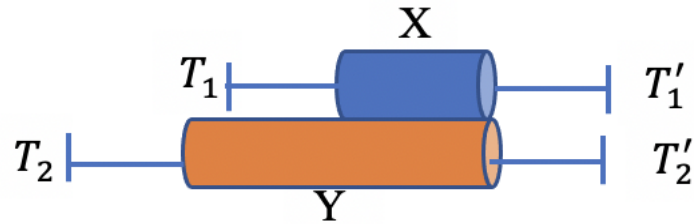


Figure 3-15: Diagram showing Allen's finishes/finished by relationship

This relationship is like the mirror of start/started by relationship. The two events have the same end time, but one starts before the other. In Figure 3-15, the start time of event X is greater than the start time of event Y and both have the same end time ($T_2 < T_1 < T'_1 = T'_2$), implying that event X *finishes* event Y. The inverse to this relation is Finished by. Figure 3-16 shows how such events could be accessed. The queries are summarized in Table 3-7.

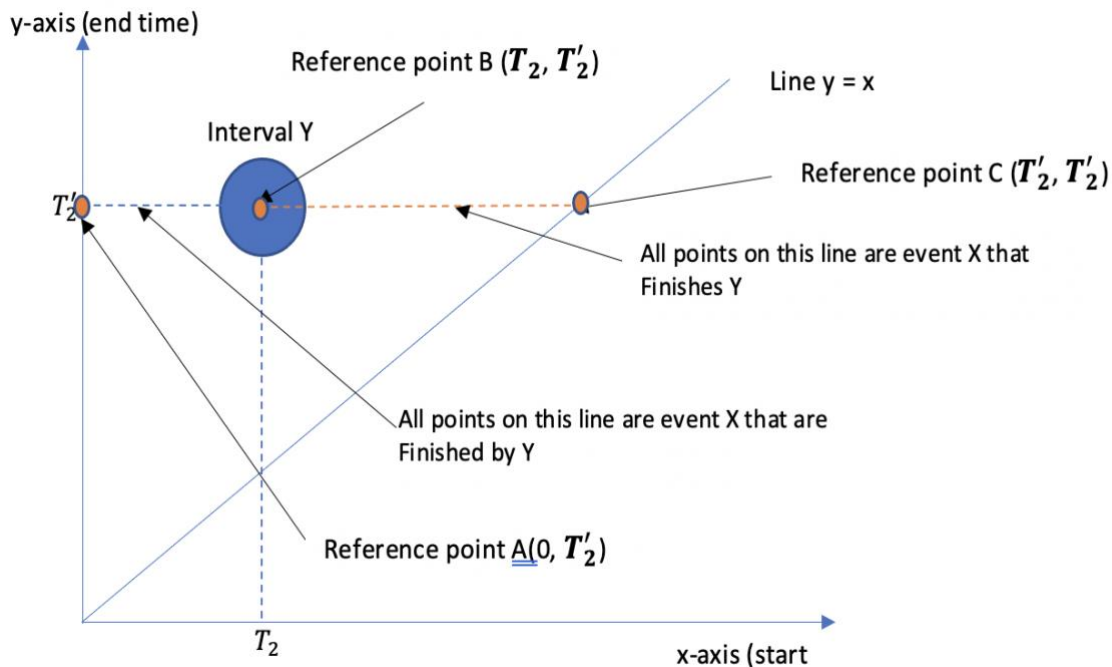


Figure 3-16: Cartesian representation of finishes / finished by relationship

Table 3-7: Summary of our model query compared to that of the traditional model for finishes/finished by relationship

Temporal relation	Our model query	Traditional model query
<i>Finishes</i>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>Where n.Interval > =</i> <i>point ({x:T₂, y: T'₂'})</i></p> <p><i>AND</i></p> <p><i>n.Interval < =</i> <i>point ({x: T'₂, y: T'₂'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>n.endTime =</i> <i>dateTime(T'₂)</i></p> <p><i>AND</i></p> <p><i>n.startTime ></i> <i>dateTime (T₂)</i></p> <p><i>return n</i></p>
<i>Finished By</i>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>Where n.Interval > =</i> <i>point ({x:0, y: T'₂'})</i></p> <p><i>AND</i></p> <p><i>n.Interval < =</i> <i>point ({x:T₂, y: T'₂'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>n.endTime =</i> <i>dateTime(T'₂)</i></p> <p><i>AND n.startTime <</i> <i>dateTime (T₂)</i></p> <p><i>return n</i></p>

3.4.7 Equals Relationship

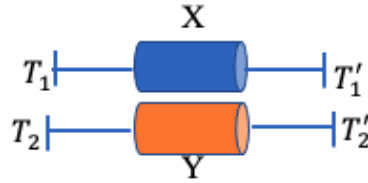


Figure 3-17: Diagram showing Allen's Equals relationship

In this form of relationship, the two events have the same time interval as seen in Figure 3-17: meaning $T_1 = T_2 < T_1' = T_2'$. This relationship does not have any inverse. This relationship is straightforward so, a graph is not needed to show how the queries from Table 3-8 are obtained.

Table 3-8: Summary of our model query compared to that of the traditional model for equals relationship

Temporal relation	Our model query	Traditional model query
<i>Equals</i>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>n.Interval =</i> <i>point({x: T₂, y: T'₂'})</i></p> <p><i>return n</i></p>	<p><i>Match (n: event)</i></p> <p><i>Where</i></p> <p><i>n.endTime =</i> <i>dateTime(T'₂)</i></p> <p><i>AND</i></p> <p><i>n.startTime = dateTime (T₂)</i></p> <p><i>return n</i></p>

3.5 Other Queries

In this section, we explain 4 more types of queries other than Allen's intervals: i) previous/next, ii) conditional before/after, iii) conditional time instance, and iv) conditional temporal graph traversing. Previous/next querying provides the immediate previous/next event. Conditional before/after just returns a subset of the results after filtering based on a condition. Time instance querying focuses on returning the time of event based on a condition. Conditional graph traversing query traverses the graph and returns the results that satisfy a condition. We provide 4 examples here.

3.5.1 Previous/Next

Previous/next query considers the immediate before/after query based on a condition. Handling this type of query first involves applying Allen's before/after relation, then organizing the data in descending order for previous or ascending order for next to get the top object from the list. Let us consider we want to get the previous event from today, with today interval given as T_2 - T'_2 . The query to get the previous event is shown in Table 3-9.

Table 3-9: Our model compared to traditional model for previous queries

Relationship	Our model query	Traditional model query
<i>Previous</i>	<i>Match (n:event)</i> <i>Where n.Interval <</i> <i>point({x: T_2, y: T'_2})</i> <i>AND (Conditions if any)</i> <i>return n</i> <i>ORDER BY</i> <i>n.Interval</i> <i>DESC</i> <i>LIMIT 1</i>	<i>Match(n:event)</i> <i>Where n.endTime <</i> <i>datetime(T_2)</i> <i>AND (Conditions if any)</i> <i>return n</i> <i>ORDER BY</i> <i>n.starttime</i> <i>DESC</i> <i>LIMIT 1</i>

3.5.2 Conditional Before/After

Conditional before/after just returns a subset of the results after filtering based on a condition. This is very similar to the previous/next except for the ordering and filtering the results to a single event.

Taking the temporal interval $T_2 - T'_2$ as reference, the temporal query is shown in Table 3-10.

Table 3-10: Our model compared to traditional model for conditional after query

Relationship	Our model query	Traditional model query
<i>Conditional After</i>	<i>Match (n:event)</i> <i>Where n.Interval ></i> <i>point({xT₂' : , y: T₂'})</i> <i>AND (Conditions)</i> <i>return n</i> <i>ORDER BY</i> <i>(ranking if available)</i>	<i>Match (n:event)</i> <i>Where n.starttime ></i> <i>datetime(' T₂')</i> <i>AND (Conditions)</i> <i>Return n</i> <i>ORDER BY</i> <i>(ranking if available)</i>

3.5.3 Conditional Time Instance

Time instance querying focuses on returning the time of an event based on a condition. This type of query could be based on any of Allen's intervals and will return where the time of the event in consideration will be accessed. The temporal query is provided in Table 3-11.

Table 3-11: Our model compared to traditional model for conditional time instance

Relationship	Our model query	Traditional model query
<i>Time Instance</i>	<i>Match (n:event)</i> <i>Where</i> <i>Temporal conditions</i> <i>AND</i> <i>Filtering Conditions</i> <i>return n.Interval.x</i> <i>Ranking (if any)</i>	<i>Match(n:event)</i> <i>Where</i> <i>Temporal conditions</i> <i>AND</i> <i>Filtering Conditions</i> <i>return n.startTime</i> <i>Ranking (if any)</i>

3.5.4 Conditional Temporal Graph Traversing

Conditional graph traversing query traverses the graph and returns the results that satisfy a condition. The general query structure for such queries is shown in figure Table 3-12.

Table 3-12: Our model compared to traditional model for conditional temporal graph traversing

Relationship	Our model query	Traditional model query
<i>Conditional temporal graph traversing</i>	<i>Match (n:event)</i> <i>Where</i> <i>Temporal condtions</i> <i>AND</i> <i>Filtering conditions</i> <i>return n</i> <i>Ranking (if any)</i>	<i>Match (n:event)</i> <i>Where</i> <i>Temporal condtions</i> <i>AND</i> <i>Filtering conditions</i> <i>return n</i> <i>Ranking (if any)</i>

3.6 Summary

This chapter showed a detailed overview of our model and how we can use the cartesian coordinate to design temporal queries. It also shows the query structure for other forms of queries. Not all forms of temporal interval organization are represented by Allen's intervals, but the same framework used for the Allen's relations can be used to come up with any other form of queries. The next chapter will show a performance comparison between the queries generated here and the queries generated for a traditional model.

Chapter 4 Experiments

Chapter 3 explained how we can use our model to build queries based on Allen's intervals and then compared those query structures to the query structure of a traditional temporal querying method. In this chapter, we will demonstrate these queries on actual examples and compare the performance between our model and the traditional model. In the traditional model, the event interval is stored as attributes composed of start time and end time for the event (Figure 4-1). To do this, we performed two sets of experiments. The first one will involve query examples and performance analysis based of Allen's intervals. The second experiment set will demonstrate performance related to temporal ordering.

The experiments and demonstrations were done using Neo4j. Neo4j comes with the point spatial geometry which has multiple characteristics [38] relevant to this research. Just like with any geometric point, Neo4j *spatial point* has up to three dimensions. We are going to focus only on two dimensions in this research (x, y). Neo4j *points* can be assigned to nodes and relationship as properties (Figure 4-2). With this feature, we can represent the temporal property of an event as a point. The points can be indexed using a spatial index. This will be useful for future research when speed optimization will be of the essence. All these features make it possible to represent temporal data using cartesian points through an existing database structure.

In the following section, we describe the dataset for our experiments. Then the results of experiments and performance results are provided. We additionally give examples of 4 types of queries other than Allen's intervals: i) previous/next, ii) conditional before/after, iii) conditional time instance, iv) conditional temporal graph traversing.

```
{
  "identity": 4450,
  "labels": [
    "TemporalGames"
  ],
  "properties": {
    "starttime": "2013-04-17T14:45:00Z",
    "endtime": "2013-04-17T16:15:00Z",
    "id": 135.0,
    "teamA": "west ham",
    "teamB": "manchester united"
  }
}
```

Figure 4-1: Sample neo4j node with traditional temporal interval storage start and end time represented as two distinct properties

```
{
  "identity": 107,
  "labels": [
    "CartesianGames"
  ],
  "properties": {
    "id": 135.0,
    "event_interval": point({srid:7203, x:20130417.1445,
    y:20130417.1615}),
    "teamA": "west ham",
    "teamB": "manchester united"
  }
}
```

Figure 4-2: Sample neo4j node with the point property representing the temporal interval

4.1 Dataset

The two sets of experiments will be performed on a soccer dataset of more than 4000 games (4343). These are the premiere league games of 21 English premiere league teams from 2011 to 2021. Each node represents a game with a start and end time, and each edge represents the timeline that links the game of each team. We have two representations of the node in the same graph database. The first representation is the one where the temporal interval is represented using the traditional model (Figure 4-3) and the second representation is with our model (Figure 4-4).

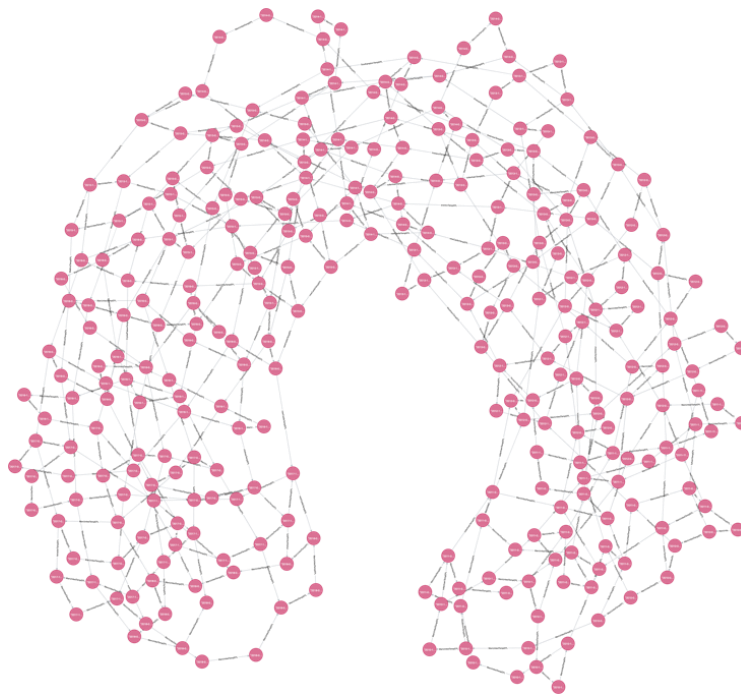


Figure 4-3: Soccer dataset with traditional temporal interval representation

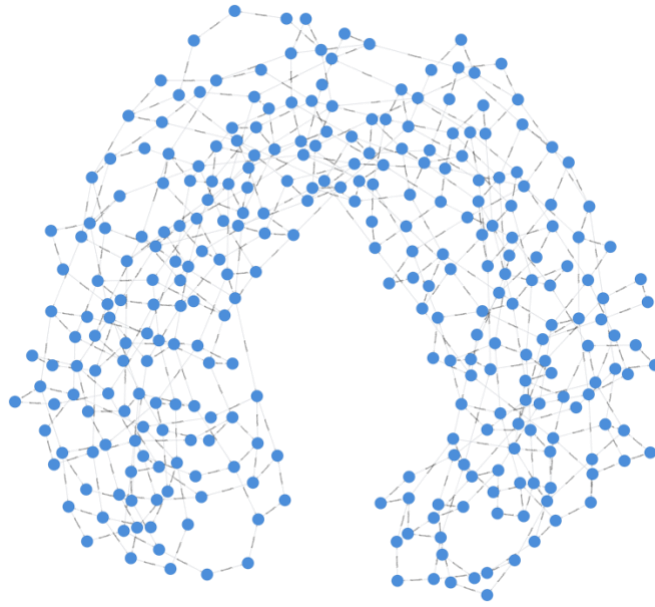
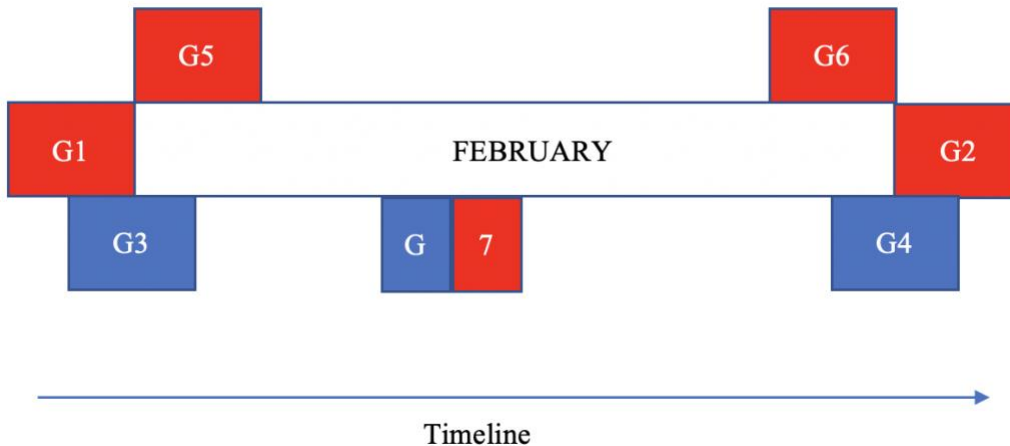


Figure 4-4: Soccer dataset with proposed model temporal interval representation

We have also developed a mini dataset for validating query results. ***The mini dataset consists of 7 carefully selected games. These games were selected in such a way that we will have a nonnull response for each of the queries generated based on Allen's intervals.*** Figure 4-5 shows how these games are aligned on the timeline. The games follow the timeline of two main teams: Manchester United and Chelsea. The edges between all Manchester United games are used to group Manchester United events together, and the edges between all Chelsea games are used to group Chelsea games together. Figure 4-6 shows the representation of these games in Neo4j graph database. We built 2 graph models for the mini dataset: one where the temporal interval is represented with our proposed model (Figure 4-6) and the other where the temporal interval is represented with the traditional interval representation (Figure 4-7).

Table 4-1: Mini dataset for temporal queries validation

Game label	Team A	Team B	Start Date	End Date
Game1	Man United	Liverpool	2021/01/31 10:30pm	2021/02/01 12:00 am
Game2	Man United	Arsenal	2021/02/28 11:59pm	2021/03/01 1:29am
Game3	Chelsea	Man City	2021/01/31 11:30pm	2021/02/01 1:00am
Game4	Chelsea	Arsenal	2021/02/28 11:30pm	2021/03/01 1:00am
Game5	Man United	Tottenham	2021/02/01 12:00 am	2021/02/01 1:30 am
Game6	Man United	Man City	2021/02/28 10:30 pm	2021/02/28 11:59 pm
Game7	Man United	Chelsea	2021/02/10 10:00 am	2021/02/10 11:30 am

**Figure 4-5: Mini dataset represented on a timeline. Red are Manchester's games**

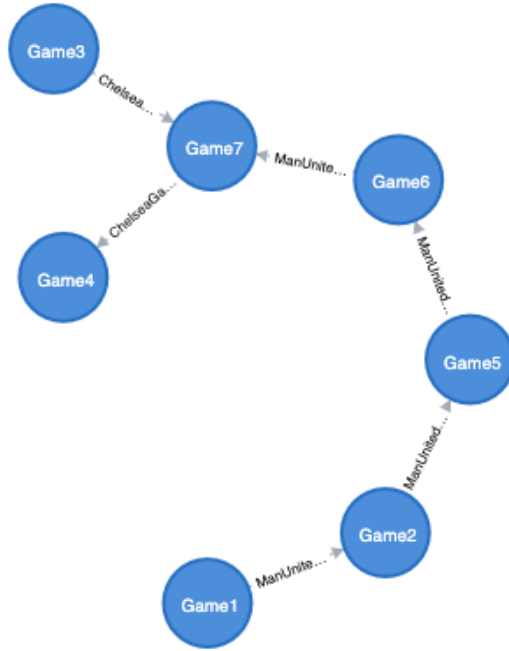


Figure 4-6: Graph representation of mini dataset in our model

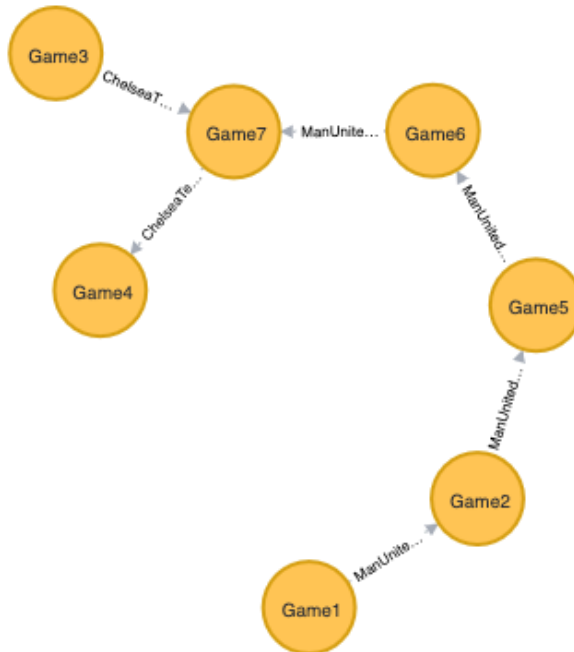


Figure 4-7: Graph representation of mini dataset in traditional model

4.2 Performance Comparison

In Chapter 3, we have developed a set of queries based on Allen's intervals. We will apply those queries using our model and the traditional model. Both models use a graph structure to store the dataset. The queries based on Allen's intervals include before/after relationships, during, meets/met by, overlaps/overlapped by, starts/started by, finishes/finished by, and equals relationships. We briefly provide performance analysis of these queries as follows. For each relationship, we provide a query example to show how the query can be structured and then provide the performance analysis.

For the experiments, we are going to use the queries developed in Chapter 3 then use the mini dataset shown in Table 4-1 for validation. We are then going to use these queries on the larger dataset for performance analysis between the traditional model and our model. For the performance analysis, we run each query 1000 times in series of hundreds and record the average availability time (time taken for the records to become available) and consumed time (time taken for records to be consumed by the server). The total query time is the sum of these two times, and it represents the query performance.

In our temporal database, events correspond to intervals. Events need to be compared with another interval for temporal queries based on Allen's intervals. Rather than using an existing event from the database, we explicitly defined intervals in our queries. The interval could be a day, a week, or a month, etc. If events need to be compared with another event as an interval, the queries will be executed based on its start and end time. Hence, the query structure is still the same. Some of the queries are based on a maximum y value. For this experiment, we will select our maximum y value as December 31st of the year 2100. So, in our model, this will be represented as $Y_{max} = 21001201$.

4.2.1 Before/After Relationship

Query example:

*What are all the games that were played **before (after) February 15th, 2021**?*

Query description:

This question contains two queries that relate to the before/after relationship proposed by Allen. The interval Y is represented by the 15th of February with $T_2 = 2021-15-01T00:00$ and $T'_2 = 2021-02-15T23:59$. All the events that happened before this interval represent the X from Figure 3-5, so the query wants us to get all the X that happened before Y and the inverse (after).

Getting all the games that happened before the 15th of February 2021 from the mini dataset should return Game 1, Game 3, Game 5, and Game 7. Table 4-2 provides the queries for both models using the Cypher query language. Figure 4-8 shows the query results for both models.

Table 4-2: Before queries using cypher query based on our model and the traditional model

Our model	Traditional model
<i>Match (n:CartesianGames)</i>	<i>Match (n:TemporalAttributeGames)</i>
<i>Where n.Event_Interval</i>	<i>Where n.end_Time</i>
<i>< point ({x:20210215, y:20210215})</i>	<i>< datetime('2021-02-15T0:0')</i>
<i>Return n</i>	<i>Return n</i>

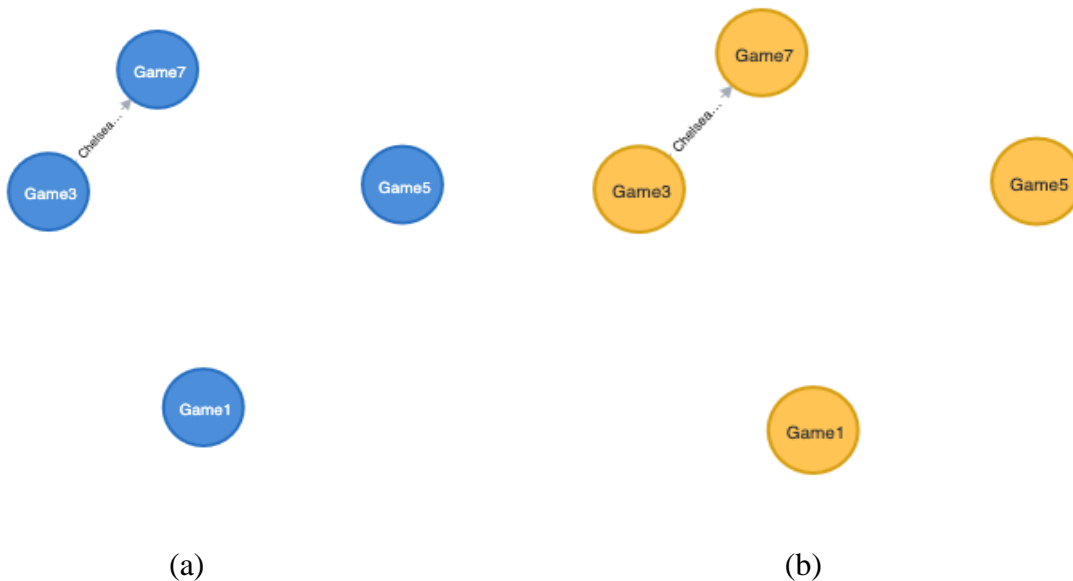
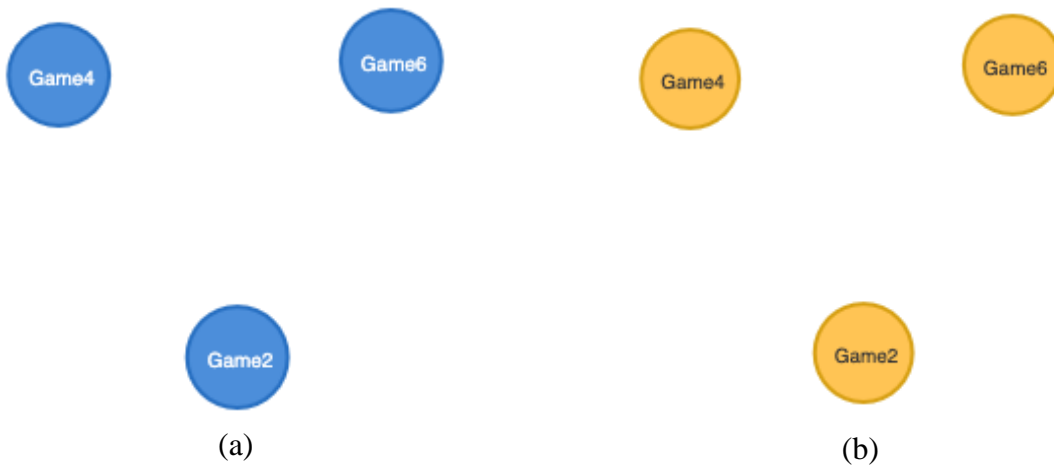


Figure 4-8: Results of before query (a) our model and (b) the traditional model

Getting all the games that happened after the 15th of February 2021 from the mini dataset should return Game 2, Game 4, and Game 6 (Figure 4-9).

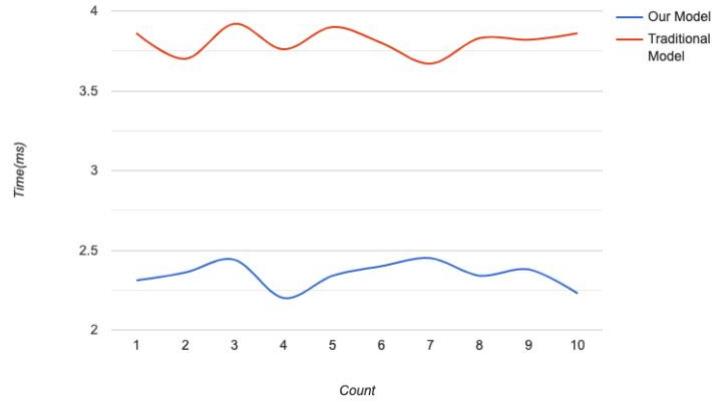
Table 4-3: After queries using Cypher query based on our model and the traditional mode

Our model	Traditional model
<i>Match (n: CartesianGames)</i> <i>Where n. Event_Interval > point</i> <i>({x:20210215.2359, y:20210215.2359})</i> <i>Return n</i>	<i>Match (n: TemporalAttributeGames)</i> <i>Where n.start_Time ></i> <i>datetime('2021-02-15T23:59')</i> <i>Return n</i>

**Figure 4-9: Results of after query (a) our model and (b) the traditional model**

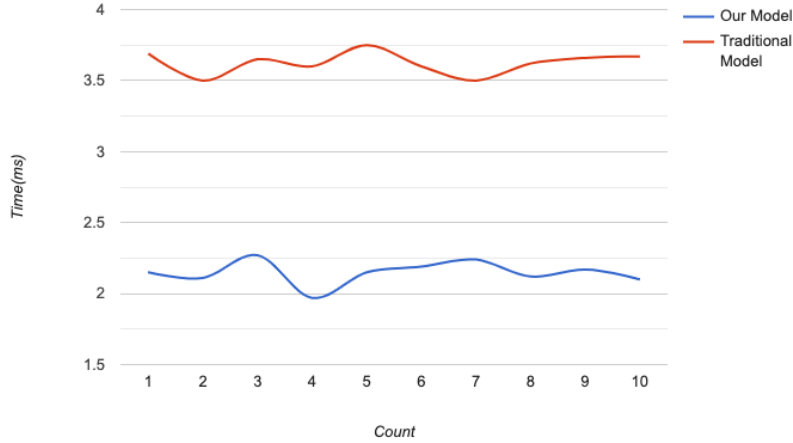
After validating the queries on the small dataset, we run the queries on the large dataset for performance analysis. Figure 4-10(a) shows the total query time which is the summation of available time and consumed time. In this graph, we see our model has a total query time of about 2.4ms and the traditional model has a total time of about 3.75ms so a difference of about 1.5ms between the two. Figure 4-10(b) and (c) show the consumed time and available time respectively between the two models. We can see that the availability time for both models are about the same. The difference between the 2 models originates from the consumed time.

Our model total time vs traditional model total time



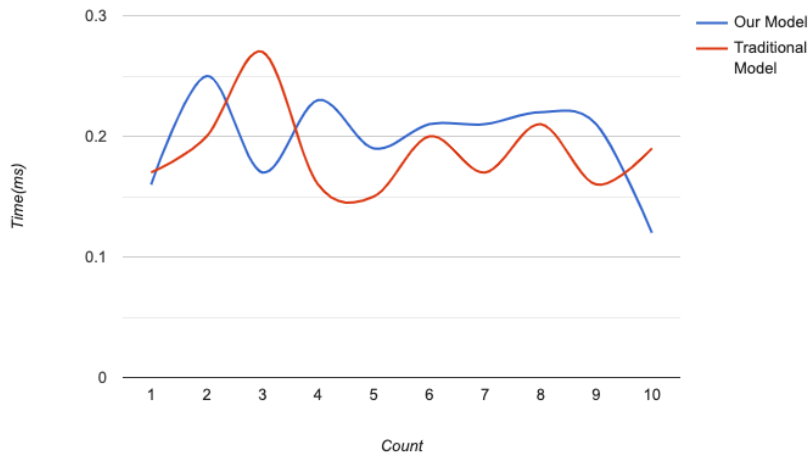
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-10: Our model (blue) vs traditional model query time comparison (red) for before relationship of Allen's interval (a) total time, (b) consumed time, and (c) available time

4.2.2 During Relationship

Query Example: *What are all the games that were played **during** the month of February 2021?*

Query Description:

This question is related to the during relationship proposed by Allen. The interval Y is represented by the whole month of February with $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-28T23:59$. All the events that happened during this interval represent the X from Figure 3-8. From our mini dataset, this query should return Game 5, Game 6, and Game 7. The queries are provided in Table 4-4.

Table 4-4: During queries using cypher query based on our model and the traditional model

Our model	Traditional model
<pre>match (n: CartesianGames) Where n.Event_Interval >= point ({x:20210201, y:20210201}) AND n.Event_Interval <= point ({x:20210228.2359, y:20210228.2359}) return n</pre>	<pre>Match (n: TemporalAttributeGames) Where n.start Time >= datetime('2021-02-01T0:0') AND n.end_Time <= datetime('2021-02-28T23:59') return n</pre>

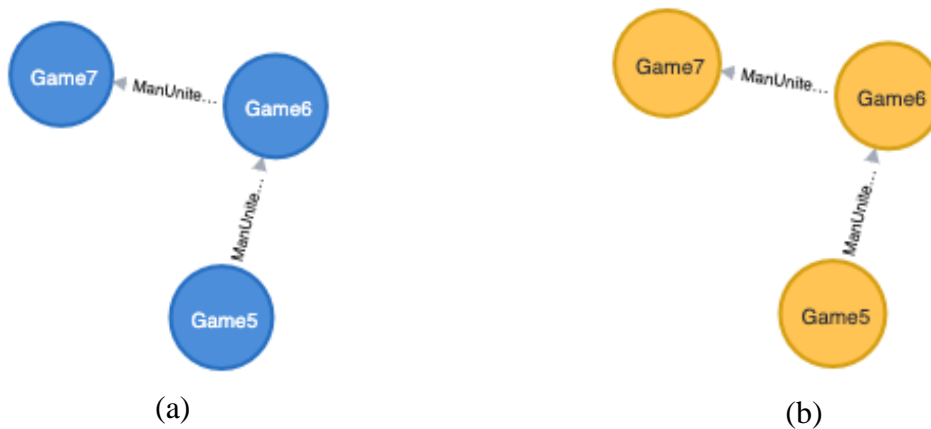
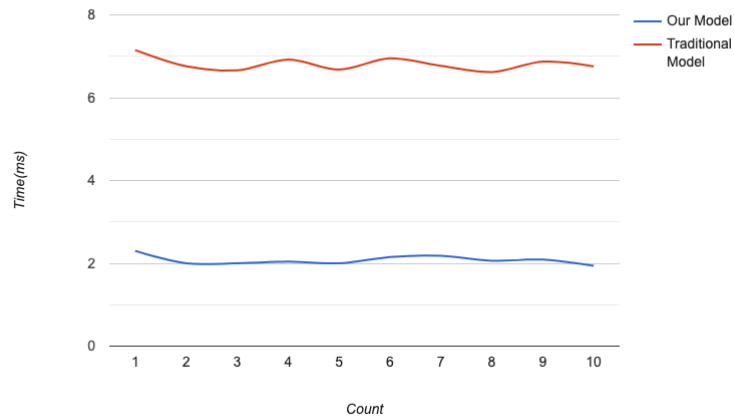


Figure 4-11: Results of during query (a) our model and (b) the traditional model

The query results for during relationship is shown in Figure 4-11.

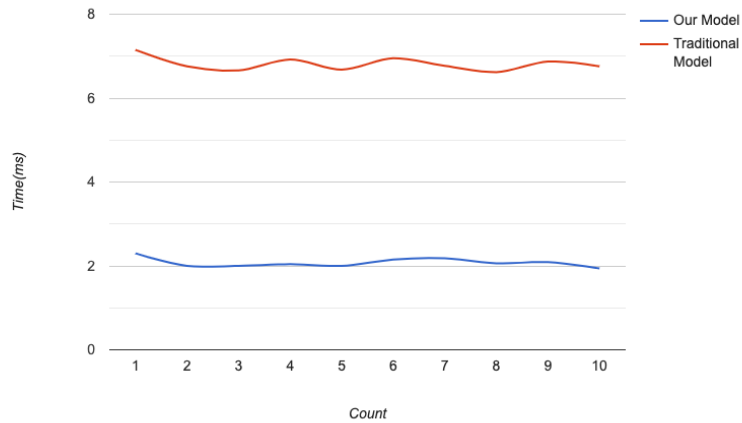
Just like with the *before* relationship, our model also performed better in this situation. Total query of our model time for this relationship is approximately 2ms while the traditional model's time is close to 7ms yielding a difference of about 5ms. Just like the before relationship, the difference between the two models mostly comes under the consumed time (Figure 4-12 (b)). The available time (Figure 4-12 (c)) for both models are about the same.

Our model total time vs traditional model total time



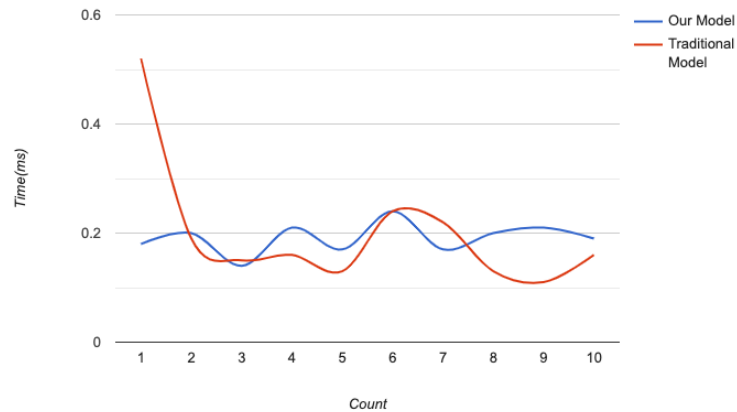
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-12: Our model(blue) vs traditional model query time comparison(red) for during relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time

4.2.3 Meets / Met By Relationship

Query example: *Find all the games that meet with the month of February.*

Query description:

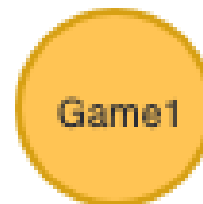
From the dataset shown in **Figure 4-5** the interval Y is represented as the month of February with $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-28T23:59$. The game that meets interval Y is G1, and the game met by interval Y is G2. The queries are provided in Tables 4-5 and 4-6. The query results for meets and met by are shown in **Figure 4-13** and **Figure 4-14** respectively.

Table 4-5: Meets queries using cypher query based on our model and the traditional model

Our model	Traditional model
<p><i>Match (n: CartesianGames)</i></p> <p><i>Where n. Event_Interval >=</i></p> <p><i>Point({x:0, y:20210201})</i></p> <p><i>AND</i></p> <p><i>n. Event_Interval <=</i></p> <p><i>Point({x: 20210201, y:20210201})</i></p> <p><i>Return n</i></p>	<p><i>Match (n: TemporalAttributeGames)</i></p> <p><i>Where</i></p> <p><i>n.end_Time = datetime('2021-02-01T0:0')</i></p> <p><i>Return n</i></p>



(a)



(b)

Figure 4-13: Results of meets query (a) our model and (b) the traditional model

Table 4-6: Met by queries using cypher query based on our model and the traditional model

Our model	Traditional model
<pre> Match (n:CartesianGames) Where n.Event_Interval >= point ({x: 20210228.2359, y: 20210228.2359}) AND n.Event_Interval <= point ({x: 20210228.2359, y: 21001201}) Return n </pre>	<pre> Match (n:TemporalAttributeGames) Where n.start_Time = datetime('2021-02-28T23:59') Return n </pre>



(a)

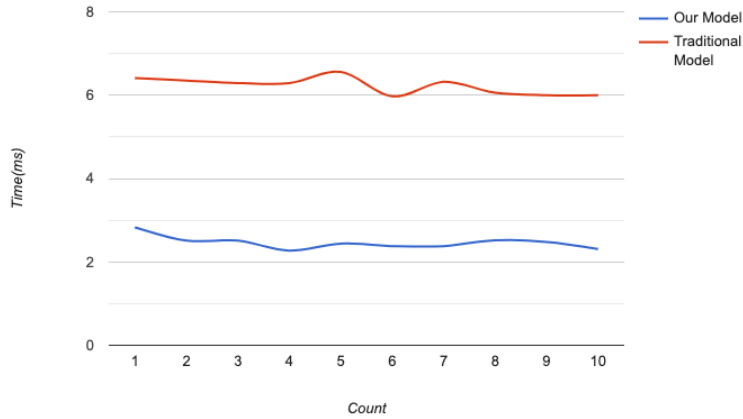


(b)

Figure 4-14: Results of met by query (a) our model and (b) the traditional model

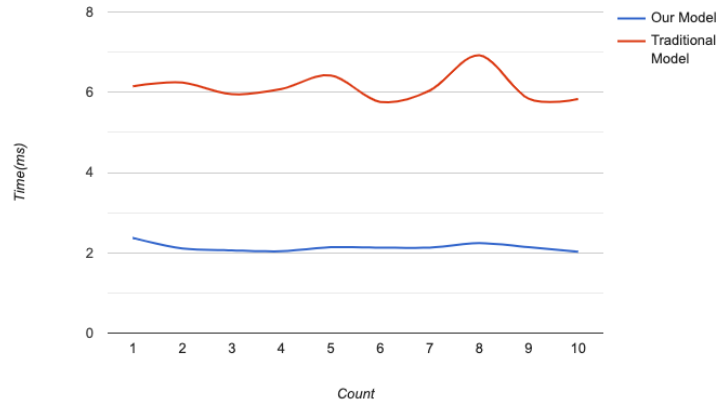
Our model performs better in this relationship too with an average time of 2.5ms compared to 6ms from Figure 4-15 (a) and just like with the other relationships, the difference in time is mostly contributed by the consumed time (Figure 4-15 (b)).

Our model total time vs traditional model total time



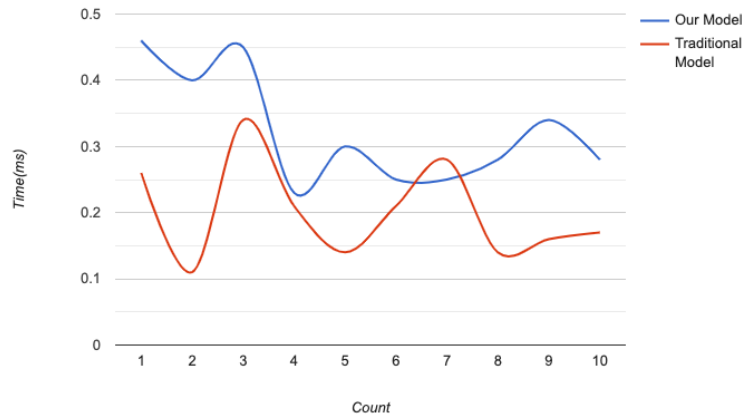
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-15: Our model(blue) vs traditional model query time comparison(red) for met by relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time

4.2.4 Overlaps/Overlapped By Relationship

Query example: *What are the games that overlaps or is overlapped by February 2021?*

Query description:

From the dataset shown in Figure 4-5 with interval Y represented as the month of February $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-28T23:59$. We have game 3 that overlaps Y and game 4 that is overlapped by Y. The queries are provided in Tables 4-7 and 4-8. The query results for overlaps and overlapped by are shown in Figure 4-16 and Figure 4-17 respectively.

Table 4-7: Overlaps queries using Cypher query based on our model and the traditional model

Our model	Traditional model
<pre> Match (n:CartesianGames) Where n.Event_Interval > point ({x:0, y:20210201.0000}) AND n.Event_Interval <point ({x:20210201.0000 , y:20210228.2359}) return n </pre>	<pre> Match (n:TemporalAttributeGames) Where n.start_Time < dateTime('2021-02-01T00:00') AND n.end_Time > dateTime ('2021-02-01T00:00') AND n.end_Time < dateTime ('2021-02-28T23:59') Return n </pre>



(a)



(b)

Figure 4-16: Results of overlaps query (a) our model and (b) the traditional model

Table 4-8: Overlapped by queries using cypher query based on our model and the traditional model

Our model	Traditional model
<p><i>Match (n: CartesianGames)</i></p> <p><i>Where n.Event_Interval ></i></p> <p><i>Point ({x:20210201.0000 , y:20210228.2359})</i></p> <p><i>AND</i></p> <p><i>n.Event_Interval <</i></p> <p><i>point ({x: 20210228.2359, y: 21001201})</i></p> <p><i>return n</i></p>	<p><i>Match (n:TemporalAttributeGames)</i></p> <p><i>Where n.start_Time ></i></p> <p><i>dateTime('2021-02-01T00:00')</i></p> <p><i>AND n.start_Time <</i></p> <p><i>dateTime ('2021-02-28T23:59')</i></p> <p><i>AND n.end_Time ></i></p> <p><i>dateTime ('2021-02-28T23:59')</i></p> <p><i>Return n</i></p>



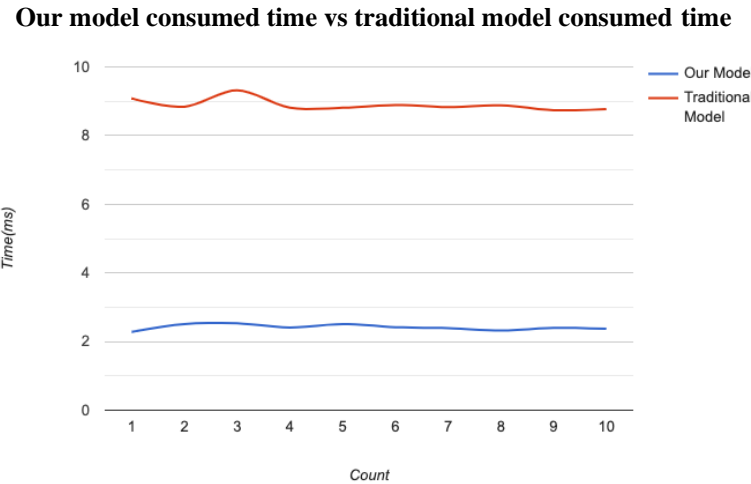
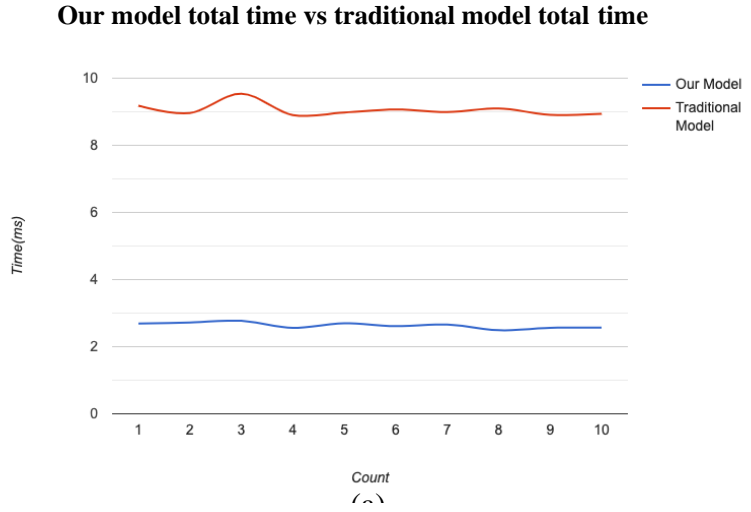
(a)



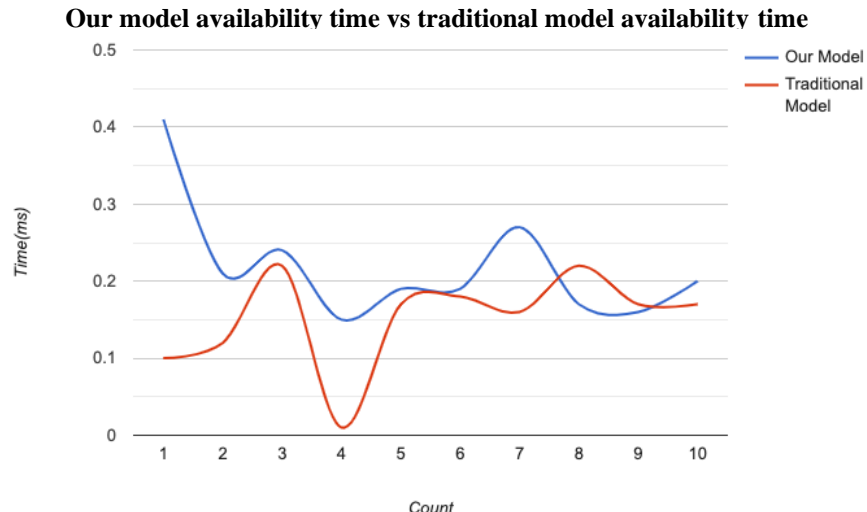
(b)

Figure 4-17: Results of overlapped by query (a) our model and (b) the traditional model

The performance comparison between the two models shows an average query time of about 3ms for our model and about 9ms from the traditional model giving a difference of about 6ms (Figure 4-18).



(b)



(c)

Figure 4-18: Our model(blue) vs traditional model query time comparison(red) for overlaps relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time

4.2.5 Starts/Started By

Query example: *Get all queries that starts the month of February*

Query description:

From the dataset shown in **Figure 4-5** with interval Y represented as the month of February $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-28T23:59$. We have game 5 that starts Y. When we execute both queries (Table 4-9), we get the results shown in **Figure 4-19**. The performance results (**Figure 4-20**) show overall gain of around 5.5ms for our model (2.5ms vs 7ms).

Table 4-9: Starts queries using cypher query based on our model and the traditional model

Our Model	Traditional Model
<pre> Match (n:CartesianGames) Where n.Event_Interval >= point({x: 20210201, y: 20210201}) AND n.Event_Interval <= point({x: 20210201, y: 20210228.2359 }) Return n </pre>	<pre> Match (n:TemporalAttributeGames) Where n.start_Time = dateTime('2021-02-01T0:0') AND n.end_Time < dateTime('2021-02-28T23:59') Return n </pre>



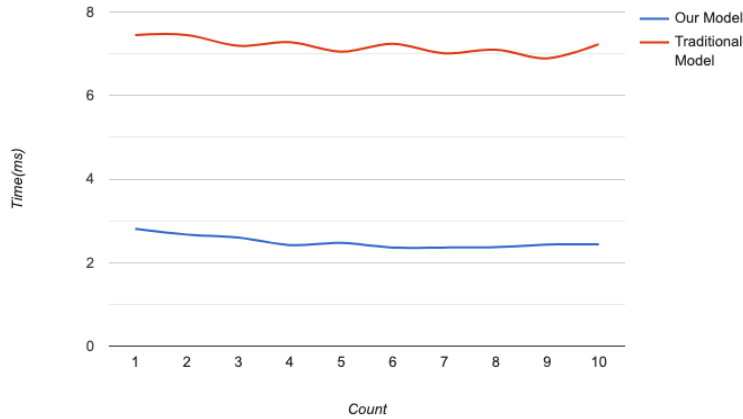
(a)



(b)

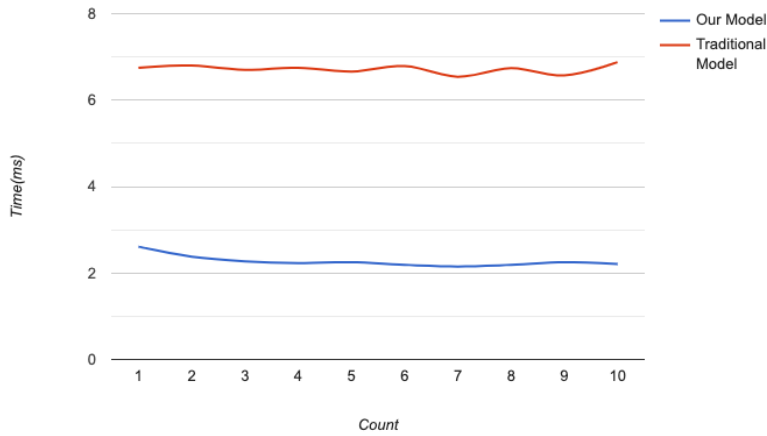
Figure 4-19: Results of starts query (a) our model and (b) the traditional model

Our model total time vs traditional model total time



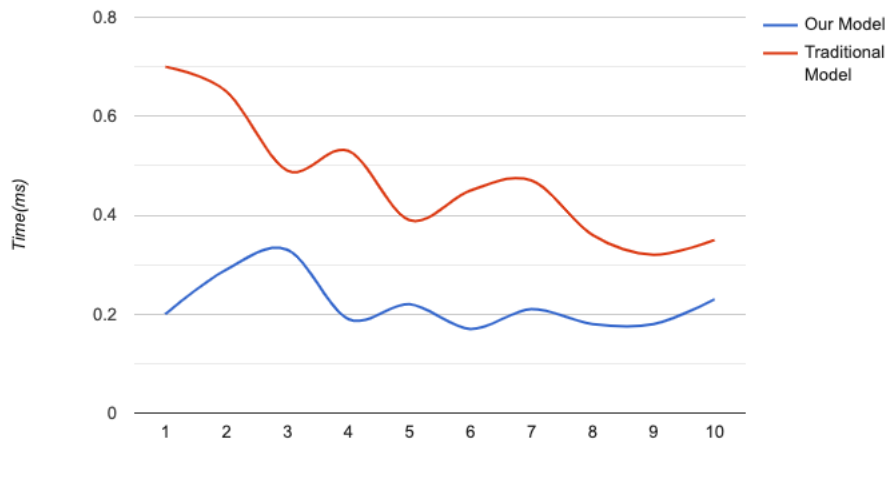
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-20: Our model(blue) vs traditional model query time comparison(red) for starts relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time

4.2.6 Finishes/Finished By

Query example: *What are the games that finishes the month of February?*

Query description:

From the dataset shown in Figure 4-5 with interval Y represented as the month of February $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-28T23:59$. We have game 6 that Finishes Y. When we execute both queries (Table 4-10), we get the results shown in Figure 4-21. The performance results (Figure 4-22) show overall gain of around 3.5ms for our model (3ms vs 6.5ms).

Table 4-10: Finishes queries using cypher query based on our model and the traditional model

Our model	Traditional model
<pre>Match (n:CartesianGames) Where n.Event_Interval >= Point({x: 20210201, y: 20210228.2359 }) AND n.Event_Interval <= Point({x:20210228.2359, y:20210228.2359 }) Return n</pre>	<pre>Match (n:TemporalAttributeGames) Where n.end_Time = dateTime('2021-02-28T23:59') AND n.start_Time > dateTime('2021-02-01T0:0') Return n</pre>



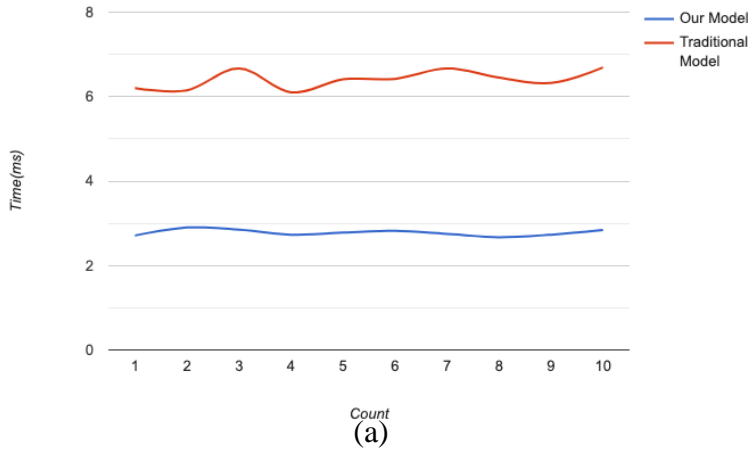
(a)



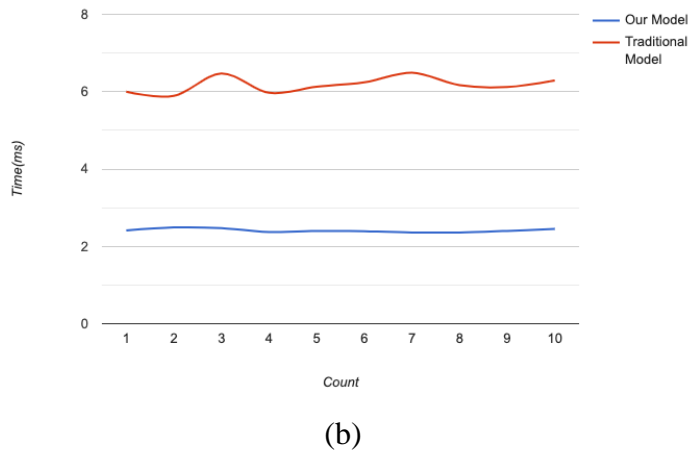
(b)

Figure 4-21: Results of finishes query (a) our model and (b) the traditional model

Our model total time vs traditional model total time



Our model consumed time vs traditional model consumed time



Our model availability time vs traditional model availability time

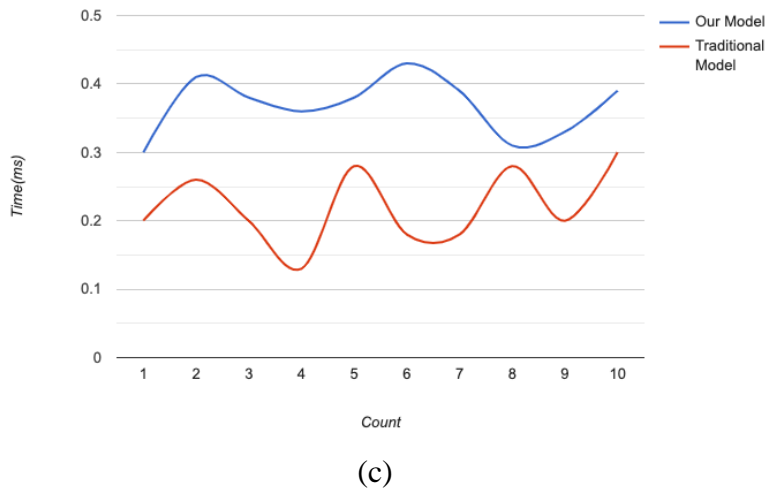


Figure 4-22: Our model(blue) vs traditional model query time comparison(red) for finishes relationship of Allen’s interval (a) total time, (b) consumed time, and (c) available time

4.2.7 Equals

Query example: *What are the games that **started** at midnight on **February 1st, 2021**, and **ended** at **1:30 am** of the same day?*

Query description:

This query uses the time interval of game 5, so using the equals interval from Allen's relation should return game 5. From the query, we have: $T_2 = 2021-02-01T00:00$ and $T'_2 = 2021-02-01T01:30$. The performance results summarized in Figure 4-24 show overall gain of around 5ms for our model (2ms vs 7ms). Table 4-11 shows the query used and Figure 4-23 shows the query results for both our model and the traditional model.

Table 4-11: Equal queries using cypher query based on our model and the traditional model

Our model	Traditional model
<pre>Match (n: CartesianGames) Where n.Event_Interval = Point ({x:20210201, y:20210201.0130}) return n</pre>	<pre>Match (n: TemporalAttributeGames) Where n.start_Time = dateTime('2021-02-01T0:0') AND n.end_Time = dateTime('2021-02-01T01:30') Return n</pre>



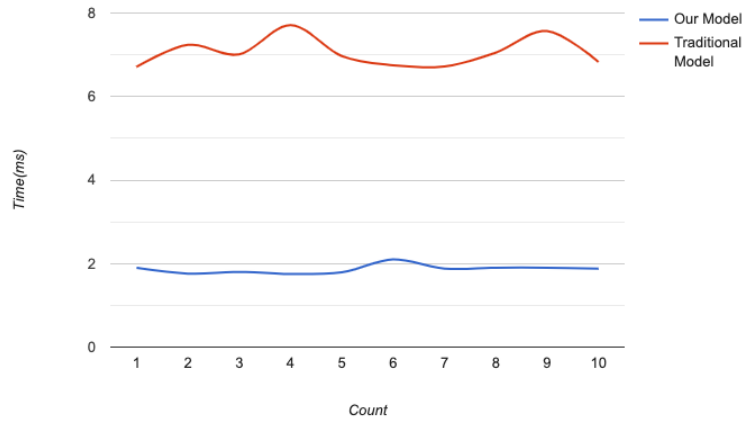
(a)



(b)

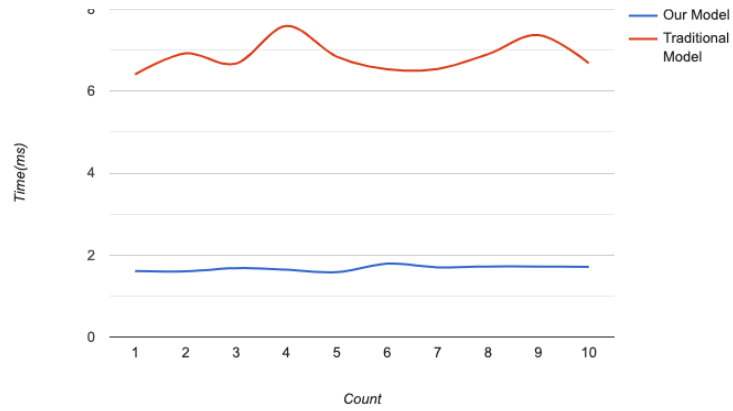
Figure 4-23: Results of equals query (a) our model and (b) the traditional model

Our model total time vs traditional model total time



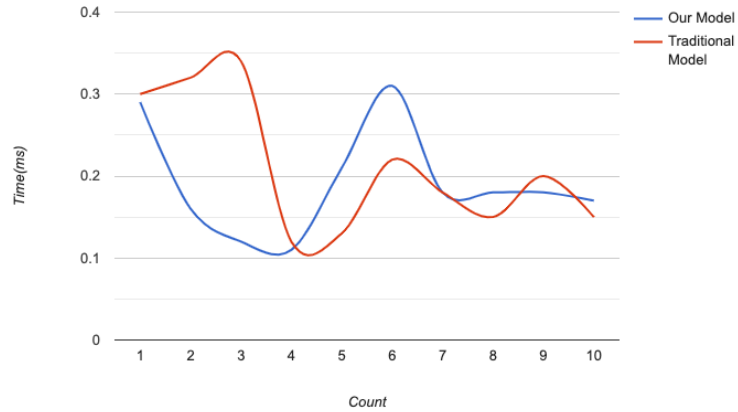
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-24: Our model(blue) vs traditional model query time comparison(red) for Equal relationship of Allen's interval

4.3 Other Query Examples

The second set of experiment involves examples for the four types of queries (previous/next, conditional before/after, conditional time instance, and conditional temporal graph traversing) introduced in Chapter 3. Just like with the Allen's intervals, the experiment will involve a performance comparison between our model and the traditional model.

4.3.1 Previous/Next

Query example: *What is Manchester United's the last premiere league game?*

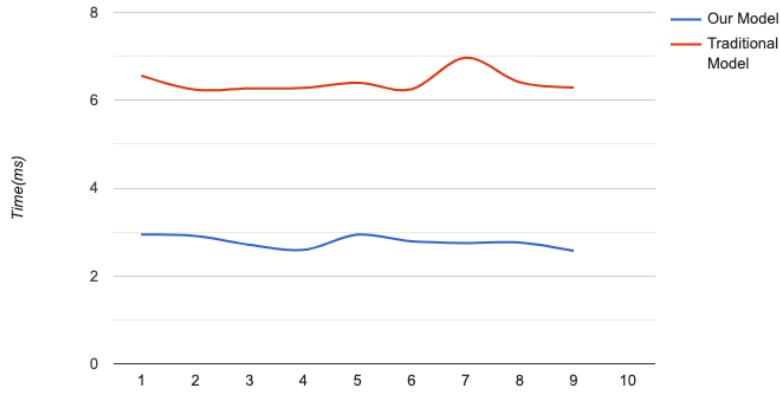
Query description:

This query refers to accessing the last game that happened from the date of today (the previous game). Let us assume that the date of today is March 15, 2013, which from our model is given as the interval **20130315 - 20130315.2359**. Applying the query structure proposed in Table 3-9 gives us the exact queries shown in Table 4-12.

Table 4-12: Our model compared to traditional model previous query example

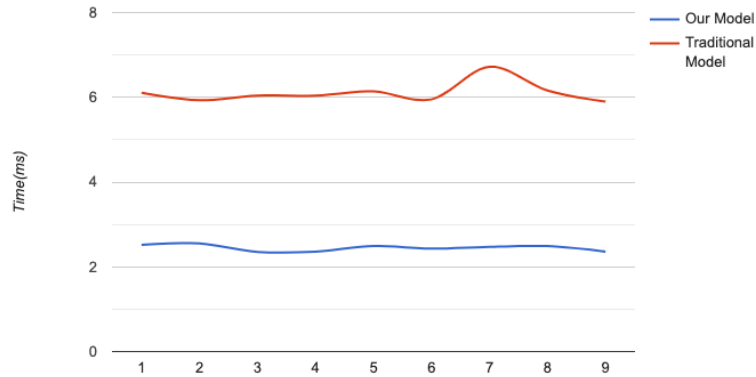
Our model	Traditional model
<i>Match (n:CartesianGames)</i> <i>Where n.event_Interval <</i> <i>point({x:20130315 , y:20130315})</i> <i>AND</i> <i>(n.teamA = 'manchester united'</i> <i>OR</i> <i>n.teamB = 'manchester united')</i> <i>return n</i> <i>ORDER BY</i> <i>n.event_Interval DESC</i> <i>LIMIT 1</i>	<i>Match(n:TemporalGames)</i> <i>Where n.endTime <</i> <i>datetime('2013-03-15T0:0')</i> <i>AND</i> <i>(n.teamA = 'manchester united'</i> <i>OR</i> <i>n.teamB = 'manchester united')</i> <i>Return n</i> <i>ORDER BY</i> <i>n.startime DESC</i> <i>LIMIT 1</i>

Our model total time vs traditional model total time



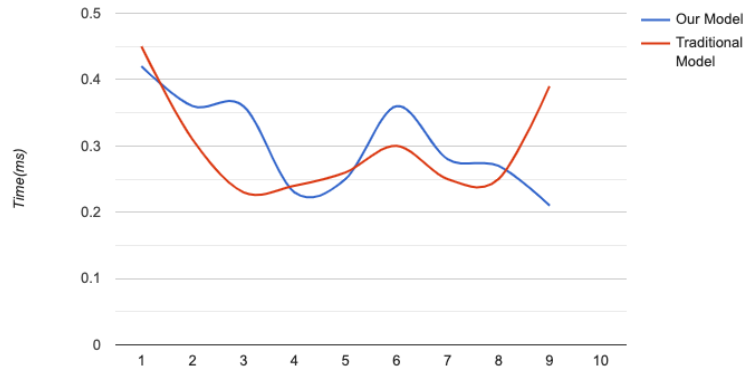
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-25: Our model(blue) vs traditional model query time comparison(red) for previous queries

The performance analysis shown in Figure 4-25 between our model and the traditional model shows a better performance from our model. The traditional model approximate query time is 6ms and our model approximate query time is 3, so a difference of about 3ms between the 2 models.

4.3.2 Conditional before/after

Query example: *What are all the **upcoming** Manchester United games to be played against Chelsea **ranked soonest to latest**?*

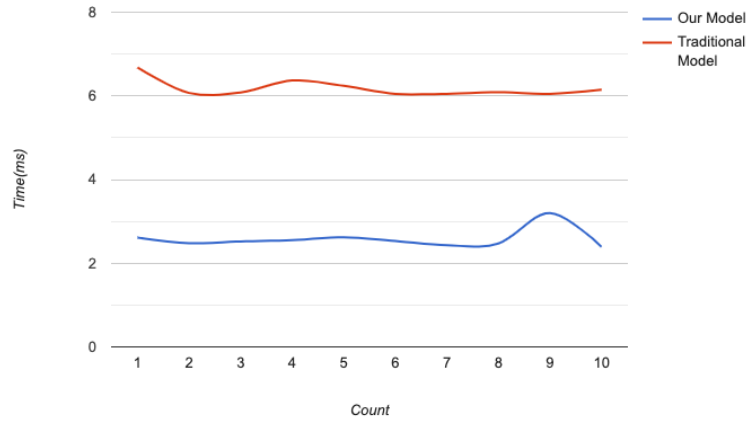
Query description:

This query falls under the conditional after category where the condition here is to select all Manchester United games against Chelsea. We also have some ranking involved here that organizes the data from soonest to latest. Let us assume that the date of today is March 15, 2013, which from our model is given as the interval **20130315 - 20130315.2359**. Based on Table 3-10, this query is given in Table 4-13.

Table 4-13: Our model compared to traditional model conditional after query example

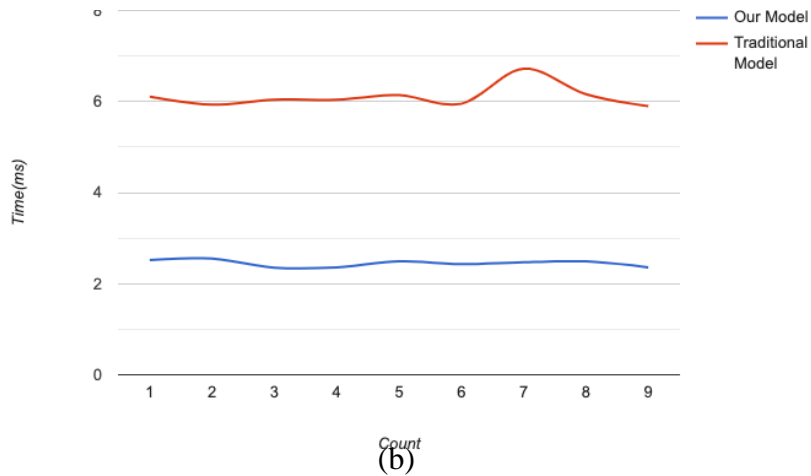
Our model	Traditional model
<pre> Match (n:CartesianGames) Where n.event_Interval > point({x: 20130315.2359 , y:20130315.2359}) AND (n.teamA = 'manchester united' OR n.teamA = 'chelsea') AND (n.teamB = 'manchester united' OR n.teamB = 'chelsea') return n ORDER BY n.event_Interval </pre>	<pre> Match (n:TemporalGames) Where n.starttime> datetime('2013-03-15T23:59') AND (n.teamA = 'manchester united' OR n.teamA = 'chelsea') AND (n.teamB = 'manchester united' OR n.teamB = 'chelsea') Return n ORDER BY n.starttime </pre>

Our model total time vs traditional model total time



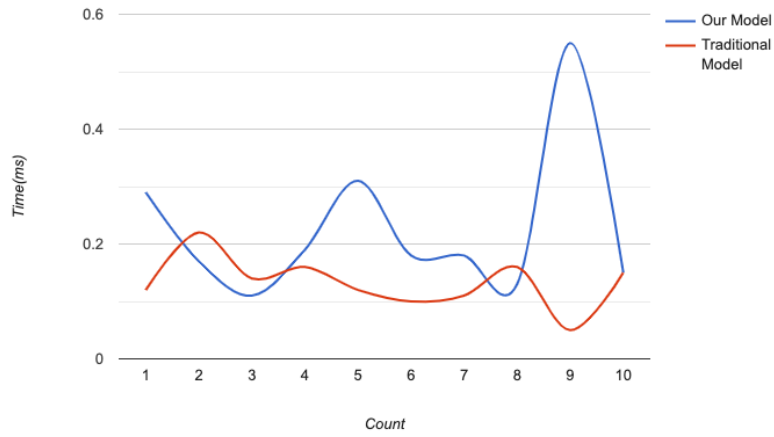
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-26: Our model(blue) vs traditional model query time comparison(red) for conditional after

The performance analysis shown in Figure 4-26 between our model and the traditional model shows a better performance for our model with a difference of about 3.5 ms between the 2 models (approximately 6 ms for the traditional model and 2.5ms for our model).

4.3.3 Conditional time instance

Query example: *When was the **last time** Tottenham played against Arsenal?*

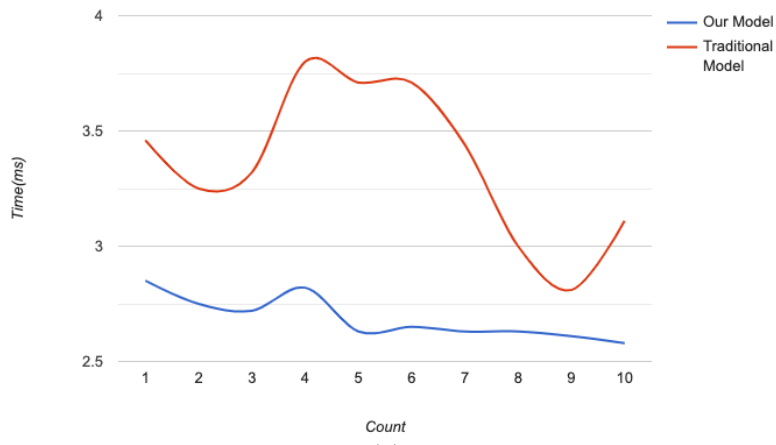
Query description:

This query is a time instance query because we need to access the time at which a specific event happened, and based on the general query structure provided in Table 3-11, the temporal condition is given as the previous relation and the filtering condition is given as games between Tottenham and Arsenal. To perform this query, we will assume the date of today is January 18, 2014. To get the last time Tottenham played against Arsenal, we need to get all the games between the 2 teams before January 18, sort it in temporal intervals and access the first element start date. The query is summarized in Table 4-14.

Table 4-14: Our model compared to traditional model conditional time instance query

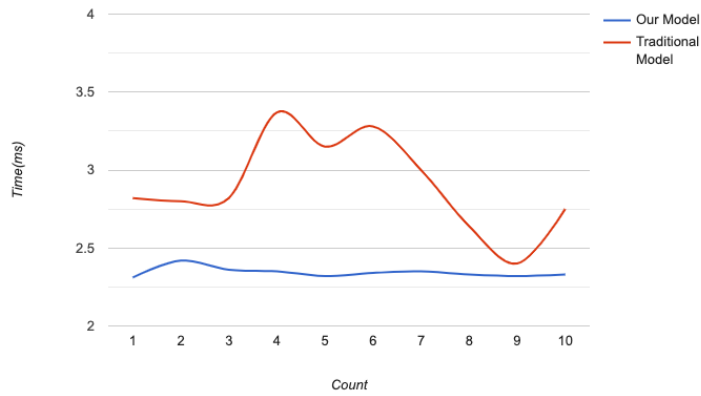
Our model	Traditional model
<p><i>Match (n:CartesianGames)</i></p> <p><i>Where n.event_Interval <</i> <i>point({x:20130315 , y:20130315})</i></p> <p><i>AND</i></p> <p><i>(n.teamA = 'tottenham'</i> <i>OR</i> <i>n.teamA = 'arsenal')</i></p> <p><i>AND</i></p> <p><i>(n.teamB = 'tottenham'</i> <i>OR</i> <i>n.teamB = 'arsenal')</i></p> <p><i>return n.event_Interval.x</i></p> <p><i>ORDER BY</i></p> <p><i>n.event_Interval DESC</i></p> <p><i>LIMIT 1</i></p>	<p><i>Match(n:TemporalGames)</i></p> <p><i>Where n.endTime <</i> <i>datetime('2013-03-15T0:0')</i></p> <p><i>AND</i></p> <p><i>(n.teamA = 'tottenham'</i> <i>OR</i> <i>n.teamA = 'arsenal')</i></p> <p><i>AND</i></p> <p><i>(n.teamB = 'tottenham'</i> <i>OR</i> <i>n.teamB = 'arsenal')</i></p> <p><i>return n.starttime</i></p> <p><i>ORDER BY</i></p> <p><i>n.starttime DESC</i></p> <p><i>LIMIT 1</i></p>

Our model total time vs traditional model total time



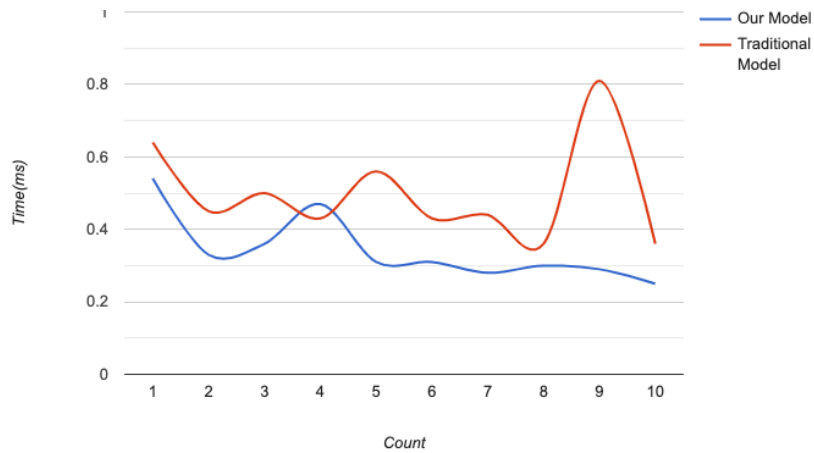
(a)

Our model consumed time vs traditional model consumed time



(b)

Our model availability time vs traditional model availability time



(c)

Figure 4-27: Our model(blue) vs traditional model query time comparison(red) for conditional time instance

The performance analysis shown in Figure 4-27 between our model and the traditional model shows a slightly better performance from our model on this type of query (about 1ms).

4.3.4 Conditional temporal graph traversing

Query example: *From earliest to latest, what Arsenal games were played **during** February 2017?*

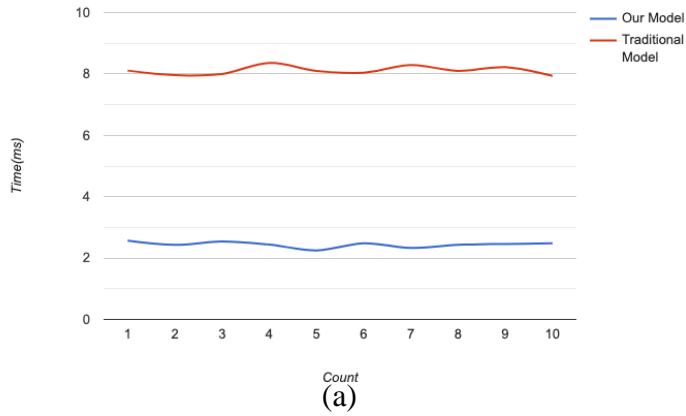
Query description:

This query is an example of the conditional temporal graph traversing query. The temporal condition is the Allen's during interval, and the filtering condition are Arsenal games. This query also involves some ranking. The start time for the interval is given as 20170201 and the end time is given as 20170228.2359. Based on this info and the query structure in Table 3-12, we have the query shown in Table 4-15.

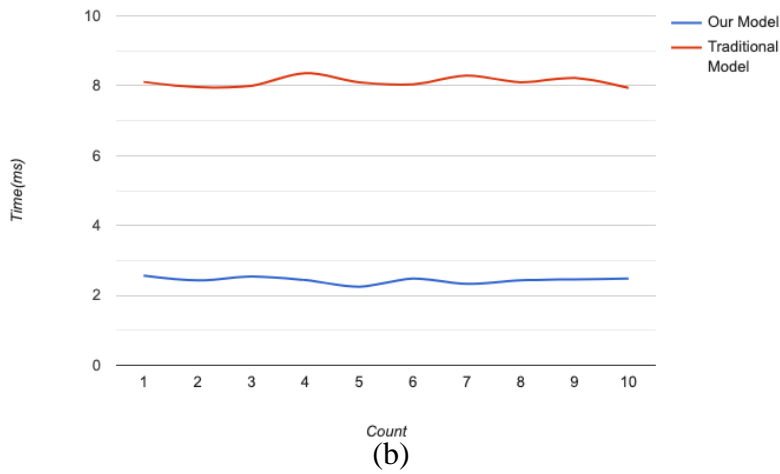
Table 4-15: Our model compared to traditional model conditional temporal graph traversing

Our model	Traditional model
<pre> Match (n:CartesianGames) Where n.event_Interval >= point ({x:20170201, y:20170201}) AND n.event_Interval <= point ({x:20170228.2359, y:20170228.2359}) AND (n.teamA = 'arsenal' OR n.teamB = 'arsenal') return n ORDER BY n.event_Interval </pre>	<pre> Match (n:TemporalGames) Where n.starttime >= datetime('2017-02-01T0:0') AND n.endTime <= datetime('2017-02-28T23:59') AND (n.teamA = 'arsenal' OR n.teamB = 'arsenal') return n ORDER BY n.starttime </pre>

Our model total time vs traditional model total time



Our model consumed time vs traditional model consumed time



Our model availability time vs traditional model availability time

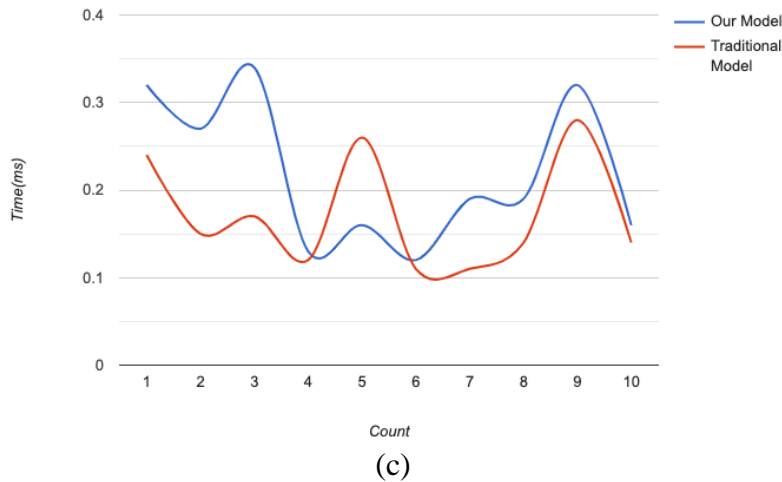


Figure 4-28: Our model(blue) vs traditional model query time comparison(red) for conditional temporal graph traversal

The performance analysis shown in Figure 4-28 between our model and the traditional model shows a better performance from our model with a difference of about 6 ms between the 2 models (approximately 8 ms from the traditional model and 2ms from our model).

4.4 Summary

The experiments compared the performance of two models. The total query time was given by the query *consumption time* and *availability time*. The experimental results show that total query time of our model was better than the traditional model query time with the time difference being mostly contributed by the consumption time. The difference between query execution times was as much as 6ms. If we look at the ratio of speedup, the speedup would be an average of 3.5 times.

Chapter 5 Conclusion and Future work

With the increasing complexity and availability of temporal data, it has become a necessity to find a way to appropriately store and retrieve this form of data. In this thesis, we proposed a cartesian point system where the x-axis represents the start time of the event and y-axis represents the end time of the event. To test our model, we used the spatial point system of an existing graph database (Neo4j) and used the cartesian coordinate system to show how queries can be built from our model. This work also contains a performance comparison between our model and a more traditional model where the start time and end time are represented as two separate attributes of the event based on Allen's interval relationships. The performance comparison shows that our model queries run faster than the traditional model (6ms difference and an average speed up of 3.5). Since we have not used spatial indexing at this stage, the performance difference most likely depends on the datatype being used. Further analysis on how spatial indexing could be beneficial for our model will be done. The major advantage of our method is that it can leverage an existing graph model without extending it.

5.1 Future work

The model we propose in this thesis can be extended as follows:

- 1) With the events being stored as a cartesian point, it is possible to have different events occupying the same positions in space (events with similar start and end time). To distinguish between such events, we could extend this work by adding a z index. The z index will increase the dimensionality of the temporal intervals to 3 and will be used to distinguish between events with start and end time being equal.
- 2) With the extensive amount of data being generated by computing systems, it is more likely to have data being stored in a distributed database system. Our proposed querying system can be extended for temporal queries on distributed graphs.

References

- [1] S. Agarwal and A. Sureka, “Applying Social Media Intelligence for Predicting and Identifying On-line Radicalization and Civil Unrest Oriented Threats,” *ArXiv151106858 Cs*, Nov. 2015, Accessed: Sep. 10, 2021. [Online]. Available: <http://arxiv.org/abs/1511.06858>
- [2] R. P. Khandpur, T. Ji, S. Jan, G. Wang, C.-T. Lu, and N. Ramakrishnan, “Crowdsourcing Cybersecurity: Cyber Attack Detection using Social Media,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, New York, NY, USA: Association for Computing Machinery, 2017, pp. 1049–1057. Accessed: Sep. 10, 2021. [Online]. Available: <https://doi.org/10.1145/3132847.3132866>
- [3] C. Dyreson *et al.*, “A consensus glossary of temporal database concepts,” *ACM SIGMOD Rec.*, vol. 23, no. 1, pp. 52–64, Mar. 1994, doi: 10.1145/181550.181560.
- [4] A. Debrouvier, M. Perazzo, E. Parodi, V. Soliani, and A. Vaisman, “A Model and Query Language for Temporal Graph Databases,” *VLDB J.*, vol. 30, Sep. 2021, doi: 10.1007/s00778-021-00675-4.
- [5] R. Snodgrass, “The temporal query language TQuel,” *ACM Trans. Database Syst.*, vol. 12, no. 2, pp. 247–298, Jun. 1987, doi: 10.1145/22952.22956.
- [6] C. S. Jensen, R. T. Snodgrass, and M. D. Soo, “The TSQL2 Data Model,” in *The TSQL2 Temporal Query Language*, R. T. Snodgrass, Ed. Boston, MA: Springer US, 1995, pp. 157–240. doi: 10.1007/978-1-4615-2289-8_10.
- [7] “SQL3 Support for Time.” <http://www2.cs.arizona.edu/~rts/sql3.html> (accessed Nov. 01, 2021).
- [8] J. Chomicki, D. Toman, and M. H. Böhlen, “Querying ATSQL databases with temporal logic,” *ACM Trans. Database Syst.*, vol. 26, no. 2, pp. 145–178, Jun. 2001, doi: 10.1145/383891.383892.
- [9] F. Rizzolo and A. A. Vaisman, “Temporal XML: modeling, indexing, and query processing,” *VLDB J.*, vol. 17, no. 5, pp. 1179–1212, Aug. 2008, doi: 10.1007/s00778-007-0058-x.
- [10] M. H. Böhlen, A. Dignös, J. Gamper, and C. S. Jensen, “Database Technology for Processing Temporal Data (Invited Paper),” p. 7 pages, 2018, doi: 10.4230/LIPICS.TIME.2018.2.
- [11] markingmyname, “Temporal Tables - SQL Server.” <https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables> (accessed Sep. 10, 2021).

- [12] C. Cattuto, M. Quaggiotto, A. Panisson, and A. Averbuch, “Time-varying Social Networks in a Graph Database: A Neo4j Use Case, in First International Workshop on Graph Data Management Experiences and Systems GRADES ’13 (ACM, New York,” USA, vol. 1, Jun. 2013.
- [13] A. Campos, J. Mozzino, and A. Vaisman, “Towards Temporal Graph Databases,” *ArXiv160408568 Cs*, May 2016, Accessed: Sep. 10, 2021. [Online]. Available: <http://arxiv.org/abs/1604.08568>
- [14] N. Prabhu and R. V. Babu, “Attribute-Graph: A Graph Based Approach to Image Ranking,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 1071–1079. doi: 10.1109/ICCV.2015.128.
- [15] J. Cheng, Q. Wang, Z. Tao, D. Xie, and Q. Gao, “Multi-View Attribute Graph Convolution Networks for Clustering,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Yokohama, Japan, Jul. 2020, pp. 2973–2979. doi: 10.24963/ijcai.2020/411.
- [16] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983, doi: 10.1145/182.358434.
- [17] Z. Liu, C. Wang, and Y. Chen, “Keyword Search on Temporal Graphs,” in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, Apr. 2018, pp. 1807–1808. doi: 10.1109/ICDE.2018.00261.
- [18] H. Memarzadeh, N. Ghadiri, and S. P. Zarmehr, “A Graph Database Approach for Temporal Modeling of Disease Progression,” in *2018 8th International Conference on Computer and Knowledge Engineering (ICCCKE)*, Oct. 2018, pp. 293–297. doi: 10.1109/ICCCKE.2018.8566311.
- [19] M. Yu, “A Graph-Based Spatiotemporal Data Framework for 4D Natural Phenomena Representation and Quantification—An Example of Dust Events,” *ISPRS Int. J. Geo-Inf.*, vol. 9, no. 2, Art. no. 2, Feb. 2020, doi: 10.3390/ijgi9020127.
- [20] L. Zheng, L. Zhou, X. Zhao, L. Liao, and W. Liu, “The Spatio-Temporal Data Modeling and Application Based on Graph Database,” in *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, Jul. 2017, pp. 741–746. doi: 10.1109/ICISCE.2017.159.

- [21] G. C. Durand, M. Pinnecke, D. Broneske, and G. Saake, “Backlogs and Interval Timestamps: Building Blocks for Supporting Temporal Queries in Graph Databases Work in progress paper.”
- [22] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, “Path problems in temporal graphs,” *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 721–732, May 2014, doi: 10.14778/2732939.2732945.
- [23] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, “Reachability and time-based path queries in temporal graphs,” in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, May 2016, pp. 145–156. doi: 10.1109/ICDE.2016.7498236.
- [24] H. Wu *et al.*, “Core decomposition in large temporal graphs,” in *2015 IEEE International Conference on Big Data (Big Data)*, Oct. 2015, pp. 649–658. doi: 10.1109/BigData.2015.7363809.
- [25] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, “Efficient Algorithms for Temporal Path Computation,” *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2927–2942, Nov. 2016, doi: 10.1109/TKDE.2016.2594065.
- [26] U. Khurana and A. Deshpande, “Efficient snapshot retrieval over historical graph data,” in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, Apr. 2013, pp. 997–1008. doi: 10.1109/ICDE.2013.6544892.
- [27] B. Salzberg and V. J. Tsotras, “Comparison of access methods for time-evolving data,” *ACM Comput. Surv.*, vol. 31, no. 2, pp. 158–221, Jun. 1999, doi: 10.1145/319806.319816.
- [28] G. Blankenagel and R. H. Güting, “External segment trees,” *Algorithmica*, vol. 12, no. 6, pp. 498–532, Dec. 1994, doi: 10.1007/BF01188717.
- [29] W. Huo and V. J. Tsotras, “Efficient temporal shortest path queries on evolving social graphs,” in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management - SSDBM '14*, Aalborg, Denmark, 2014, pp. 1–4. doi: 10.1145/2618243.2618282.
- [30] S. Huang, J. Cheng, and H. Wu, “Temporal Graph Traversals: Definitions, Algorithms, and Applications,” *ArXiv14011919 Cs*, Jan. 2014, Accessed: Nov. 10, 2021. [Online]. Available: <http://arxiv.org/abs/1401.1919>
- [31] J. Byun, S. Woo, and D. Kim, “ChronoGraph: Enabling Temporal Graph Traversals for Efficient Information Diffusion Analysis over Time,” *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 3, pp. 424–437, Mar. 2020, doi: 10.1109/TKDE.2019.2891565.

- [32] S. Mate *et al.*, *A Method for the Graphical Modeling of Relative Temporal Constraints (Preprint)*. 2019.
- [33] K. Wongsuphasawat, C. Plaisant, M. Taieb-Maimon, and B. Shneiderman, “Querying event sequences by exact match or similarity search: Design and empirical evaluation☆,” *Interact. Comput.*, vol. 24, no. 2, pp. 55–68, Mar. 2012, doi: 10.1016/j.intcom.2012.01.003.
- [34] “kim.pdf.” Accessed: Oct. 18, 2021. [Online]. Available: <http://www.cs.umd.edu/users/hjs//pubs/kim.pdf>
- [35] “15.2. Spatial Data Structures — CS3 Data Structures & Algorithms.” <https://opensa-server.cs.vt.edu/ODSA/Books/CS3/html/Spatial.html> (accessed Oct. 18, 2021).
- [36] X.-F. Jia, A. Trotman, and J. Holdsworth, “Fast Search Engine Vocabulary Lookup,” p. 8.
- [37] “Allen Relationship - an overview | ScienceDirect Topics.” <https://www.sciencedirect.com/topics/computer-science/allen-relationship> (accessed Sep. 16, 2021).
- [38] M. Körber, N. Glombiewski, and B. Seeger, “TPStream: Low-Latency Temporal Pattern Matching on Event Streams,” Aug. 2018.
- [39] S. Helmer and F. Persia, “ISEQL, an Interval-based Surveillance Event Query Language,” *Int. J. Multimed. Data Eng. Manag. IJMDEM*, vol. 7, no. 4, pp. 1–21, Oct. 2016, doi: 10.4018/IJMDEM.2016100101.
- [40] “Spatial values - Neo4j Cypher Manual,” *Neo4j Graph Database Platform*. <https://Neo4j.com/docs/cypher-manual/4.3/syntax/spatial/> (accessed Sep. 10, 2021).