On efficient temporal subgraph query processing

On Efficient Temporal Subgraph Query Processing

by

Kaijie Zhu

On Efficient Temporal Subgraph Query Processing by Kaijie Zhu

A catalogue record is available from the Eindhoven University of Technology Library

ISBN:978-90-386-5368-6

SIKS Dissertation Series No. 2021-24
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.
Keywords: Temporal graph, network generation, modeling, concurrent set size, temporal join, temporal clique, checkpoints, query processing, database system
Printed by: Ipskamp Printing

On Efficient Temporal Subgraph Query Processing

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 9 November 2021 om 11:00 uur

door

Kaijie Zhu

geboren te Zhuzhou, China

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	Prof.dr. Johan J. Lukkien
1^e Promotor:	Prof.dr. George H.L. Fletcher
Copromotor:	Dr. Nikolay Yakovets
Leden:	Prof.dr. Boudewijn van Dongen
	Prof.dr. James Cheng (Chinese University Hong Kong)
	Prof.dr. Toon Calders (Antwerp University)
	Prof.dr. Hamamache Kheddouci (Lyon 1 University)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

To my wife, parents, supervisors, friends, colleagues, and all the people I sincerely concern

Table of Contents

Summary

Graph-structured data (i.e., networks) is becoming increasingly ubiquitous across various application domains. Dominant examples include social networks, transportation networks, communication networks, citation networks, knowledge graphs, chemical databases, and biological networks. Time, as a data attribute, commonly occurs in many contemporary graph datasets. For example, in a social network, interactions between participants only hold at particular times. In transportation networks, each vehicle trip is associated with a start time and an end time. A common way to extract valuable information from graph data is through execution of subgraph(-matching) queries. This makes the investigation of efficient processing methods for subgraph queries to be of great interest to data scientists. Prior studies, however, have primarily focused on optimization of queries which are constrained *only* by topology such as edge labels, value predicates, and join predicates. In recent years, several works have also focused on queries constrained by *both* topology and time. Query processing approaches proposed in these studies, however, are mostly straightforward extensions of the approaches used to study non-temporal queries and primarily focus on leveraging selectivity of topological predicates in a query. Since they neglect to fully consider temporal factors, these studies fail to capture the significant impact of temporal predicates on query processing and generally prove to be inefficient.

In this thesis, we study efficient processing of temporal subgraph queries, i.e., queries constrained by both topology and time. Motivated by the need to fully understand this problem, we start by developing approaches that aim to accurately model the underlying temporal and topological characteristics of real-world networks. With better understanding of the behavior of real-world networks, we are able to capture the most important factors which affect temporal query selectivity. With this, we develop a family of algorithms that focus on efficient processing of temporal query predicates. These algorithms form the core of our novel processing approaches for general temporal subgraph queries. Based on careful theoretical analysis and experimental evaluation, we show that our proposed methods outperform the state of the art by a wide margin. Summarizing, the main contributions in this thesis are as follows:

- **Modeling of temporal networks.** The aim is to provide the insights into real-world networks as a foundation for further investigation in query processing. We study the characteristics of real-world networks and propose a network modeling approach. Our proposed model can cap-

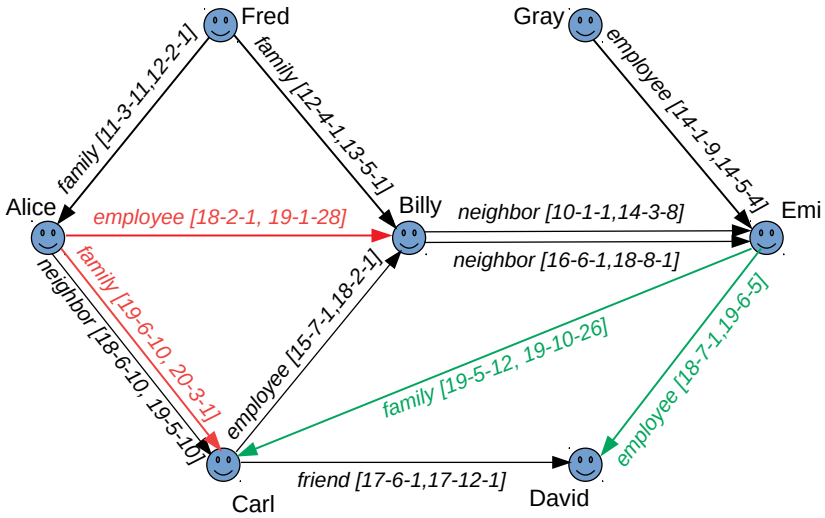
ture important network characteristics that prior studies did not consider. Moreover, our model can be used as a benchmark to generate realistic temporal networks in various applications. Our theoretical analysis and experimental evaluation demonstrate that our approach results in a controllable benchmark that can efficiently simulate real networks.

- **Processing of temporal predicates.** Temporal-predicate processing in our investigated query can be realized as a problem of temporal k-clique enumeration, i.e., to find all k-sized subsets of edges jointly overlapping in time. To the best of our knowledge, this general problem has never been studied or identified before. We first propose an improved framework for temporal-predicate processing. With our framework, the state of the art in an interval join problem (i.e., finding all the pairs of overlapping tuples from two relations) can be easily adjusted to our problem and the resulting methods are much less complex than straightforward processing. Next, based on a careful analysis of these adjusted approaches, we propose a novel method to overcome the shortcomings of previous approaches and improve processing efficiency, especially in very large datasets. Then, we develop checkpoint mechanisms to further speed up query processing in our proposed approach. Specifically, we discuss four checkpointing strategies and highlight their benefits. Experimental evaluation demonstrates that our new approaches can provide efficient temporal-predicate processing, which lays a solid foundation for efficient temporal subgraph query processing.
- **Processing of temporal subgraph queries.** Since existing studies primarily focus on leveraging the selectivity of network topology, we determine that it is important to focus our investigation on the relatively neglected impact of the selectivity of time predicates in a query. We first propose a processing approach which follows "time *then* topology" pipeline and focuses on leveraging the temporal selectivity. Then, based on a careful analysis of the proposed approach, we propose a second processing approach which follows "time *and* topology" pipeline and focuses on leveraging the selectivity of both time and topology. Experimental evaluation demonstrates that our proposed methods outperform current methods by a wide margin at a small additional storage cost. In this way, we succeed in solving the efficient processing problem of temporal subgraph queries.

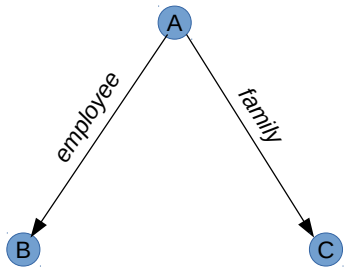
1 Introduction

Graphs are widely used to represent complex real-world systems consisting of multiple relationships (i.e., edges) among entities (i.e., vertices). Nowadays, numerous real-world systems are time-related. That is, the relationships among entities can evolve over time. To represent such evolvement, a prevalent solution is to associate the relationships with intervals recording their period of validity. To be more specific, Figure 1.1(a) presents a graph-structured social dataset that is time-related. The vertices represent residents in a community while the edges represent the relationships among residents. Each edge is associated with a time interval representing its period of validity. For example, an "employee" relationship from Alice to Billy would end on January 28th, 2019, when the company accepted Alice's resignation. Later, a new "family" relationship from Alice to Carl would appear on June 10th, 2019, when she married Carl. It is important to note that multiple edges are allowed to exist between the same pair of vertices. For example, two "neighbour" relationships can be found from Billy to Emi, representing that Billy used to be Emi's neighbor twice. This type of representation is widely known as a *temporal graph* and arises in various contemporary applications beyond the social domain. Some illustrative examples are shown as follows.

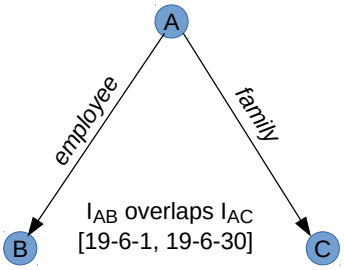
- *Transportation*: Consider the road traffic in New York City. A temporal network can be constructed where vertices represent road interactions and edges represent the flow of vehicles in road segments. Each edge is labeled with a status of 'fluid' or 'congested' and carries with it a time interval representing the period of the status.
- *Networking*: Consider Internet traffic. A temporal network can be constructed where vertices represent IPs and edges represent the connections among them. Each connection is labeled with a protocol type (e.g., TCP, UDP) and carries a time interval representing the period of the connection.



(a) a graph-structured social dataset



(b) non-temporal 2-star query Q_n



(c) temporal 2-star query Q_t

Figure 1.1: Examples of (a) temporal graph, (b) non-temporal subgraph query, and (c) temporal subgraph query. Note that the red and green-colored subgraphs are both matches of (b). However, only the green-colored subgraph is a match of (c).

- *Collaboration*: Consider cases of scientific collaboration over the past few decades. A temporal network can be constructed where vertices represent the authors and edges represent the collaborations among them. Each edge carries an interval representing the period of collaboration.

Database systems have been widely used for decades to store and manipulate real-world data. Query, where the users' desire for data manipulation is formalized in query language, is one of the most important interfaces provided by database systems. In recent years, the functionality of database systems has

been developed in two important directions: First, following SQL:2011 [1] standard, database systems have started to provide support for temporal data. Specifically, records in a database system are allowed to be associated with intervals to represent their valid periods. Various operations such as temporal join and temporal aggregation have been studied to support the query for temporal data. Second, advances in the data model have stimulated the development of graph databases (e.g., Neo4j [2]), in which graph-structured data can be stored and manipulated. All these developments provided the possibility and support for the storage and manipulation of temporal graphs.

With the recognized graph representation and support from database systems, graph analysis has attracted great interest as a key to the understanding of real-world systems. Subgraph query processing has emerged as an important graph analysis operation for capturing the often hidden underlying structures in real-world graphs. However, prior studies have primarily focused on queries constrained by only topological predicates (i.e., *non-temporal subgraph queries*). Such queries search for non-temporal matches of specified subgraphs, but fail to provide pathways for the structural exploration of real-world systems since they do not incorporate temporal information. Though several studies have been conducted on processing queries constrained by both topological and temporal predicates (i.e., the *temporal subgraph query*), they primarily follow the processing pipeline of non-temporal subgraph queries without investigating the behavioral characteristics of queries in a temporal context. As a result, methods proposed in these studies can be inefficient in many scenarios.

Summarizing, though temporal subgraph query processing is a field of great interest, it still remains to be investigated in depth. Motivated by the great need, we will start a comprehensive investigation of temporal subgraph query processing in this thesis.

1.1 For expressiveness: temporal subgraph query

Data scientists have demonstrated an increasing interest in using subgraph query processing to discover and reveal interesting topologies in ever increasing amounts of real-world data. Current works have primarily focused on non-temporal subgraph queries, which aim to retrieve matches for topology structures of interest (e.g., star, chain, triangle, clique) from graphs. Considering a query Q_n "finding individuals A, B, C from social dataset such that A works for B and A is a family member of C", the queried topology structure in

this query can be materialized as a 2-star subgraph as shown in Figure 1.1(b). Specifically, A is the center of the star and is attached with two out-going edges to B and C, denoted as (A, B) and (A, C). (A, B) is "employee"-labeled to represent a "work for" relationship while (A, C) is "family"-labeled to represent a "family of" relationship. We call the queried topology structure the *topological predicate* of the query. As a result of query processing, two triples (Alice, Billy, Carl) and (Emi, David, Carl) would be identified as the complete matches of Q_n in original graph. In the first match, A, B, and C in the query are respectively mapped to Alice, Billy, and Carl. While in the second match, A, B, and C are respectively mapped to Emi, David, and Carl. Note that the complete matches can be realized as obtained view of original graph by filtering irrelevant topology.

A primary demerit of the non-temporal query is its poor expressiveness. Specifically, the query does not specify the time-related constraints, which results in matches which do not make (logical) sense since they are not filtered according to time attributes. For example, in the first match, the "family" relationship between Alice and Carl was built up long after the end of "employee" relationship. In the second match, however, the "family" and "employee" relationships coexist in a period from May 12th to June 5th in year 2019. The recognition of this highlights the great need for constructing more expressive queries where filtering would be conducted in both topology and temporal aspects. As a result, temporal subgraph queries are proposed to retrieve time-respecting patterns of interest from temporal graphs. Generally, they can be viewed as the temporal extension on corresponding non-temporal subgraph queries. Specifically, revisiting an example graph, constructed query, and our intent, we desire to find the historical patterns which are guaranteed to be valid at a specific time in a specified period. For this aim, we could construct the following temporal subgraph query Q_t by extending Q_n to produce matches: "finding individuals A, B, C such that, *at some moment in June*, A works for B and A is a family member of C", which is materialized in Figure 1.1(c). Compared to Q_n in Figure 1.1(b), I_{AB} and I_{AC} are used, for ease of expression, to represent the associated time intervals of (A, B) and (A, C). The "overlaps" is used to constrain that the interaction of I_{AB} and I_{AC} should be non-empty. Obviously, the query result would only include (Emi, David, Carl) since Emi is both an employee of David and a family member of Carl during the period from June 1st to 5th in year 2019. The key elements of this query include : (1) the topological structure of interest (i.e., the 2-star pattern) and (2) a time window (i.e., the whole July), in which all edges of the chain pattern jointly overlap in time (i.e., the temporal structure of interest). We call the temporal

structure the *temporal predicate* of the query to distinguish it from the topological predicate.

In general, a temporal subgraph query looks for all embeddings of a topological structure in a temporal graph occurring in a given time window such that all edges of the embedding form a "temporal clique". Here "temporal clique" emphasizes that the edges are *tightly interconnected in time*, in addition to satisfying the topological pattern of interest. This is in contrast to traditional "cliques"' in which vertices are *tightly interconnected in topology*. This basic problem arises in a wide range of applications beyond the social domain.

- In the transportation network, for traffic planning, engineers are interested in finding all traffic jams involving 4 roads that occurred on 14 April 2011 between 5 pm and 7 pm, i.e., during rush hour. In a traffic jam, road flows should all jointly overlap in time, indicating the congestions occur at a time point.
- For malicious network attack detection on the Internet, find all Denial-of-Service attack occurring last night between 11 pm and 3 am, where attackers, bot machines, and victims were connected at the same point in time.
- For a deeper understanding of scientific collaborations in a bibliographic database, find all triangles in which 3 people collaborated with each other at the same time, at some point in time in the 1990s.

It is important to note that in all of these applications, it is not sufficient to obtain pattern matches that overlap with the query window but do not necessarily jointly occur at a given point in the window, i.e., do not form a temporal clique. The joint overlap in time is crucial for obtaining correct query results (e.g., a traffic jam does not happen if congestion on the roads of the chain occurs on different days in April 2011 for different edges of the chain).

It is also important to note that being able to specify a time window (instead of just a time point or a small fixed window size) for the search is fundamental to the analyses in each of these applications. Indeed, the query time window captures the period of user interest. Depending on the nature of the application, it can range from seconds to decades. Furthermore, while it is possible to convert the search for matches in a time window into a set of queries, one query for each timestamp in the query window, independently solving each of these queries leads to highly inefficient query evaluation. Indeed, such an approach can create a tremendous amount of redundant work at each time point, which could be shared and reused across the time points in the window.

1.2 For efficient processing: query selectivity

Big data in real-world applications motivates the need for efficient query processing. In this thesis, we focus on efficient query processing via smart leverage of *query selectivity*. The notion "selectivity" refers to the frequency of certain embeddings in an original graph. That is, we say an embedding has high selectivity (or is selective) when its existence is not frequent in the graph. In query processing, smart leverage of query selectivity can help to produce fewer partial intermediate results and prevent the cost of manipulating irrelevant entities. In this way, query processing efficiency can be improved. To be more specific, a general query processing method based on query selectivity in current database systems can be described as follows: First, the query is decoupled into a series of selective sub-queries. Then, the sub-queries are evaluated and concatenated in order by their selectivity until complete matches are obtained. For non-temporal subgraph query, the decoupling is carried out based on the selectivity of topology (e.g., edge labels, value predicates, and join predicate). Continuing our query example Q_n in Figure 1.1(b) and for efficient processing, a 2-way join operation on the source vertices in edge candidates of (A,B) and (A,C) will be first executed because the query vertex A, which is constrained by both "employee" and "family"-labeled edges, is obviously more selective than B and C in the original graph. Specifically, Alice and Billy will be returned as bindings of A. Then, the bindings of B and C can be easily determined through an extensive search from A. In this way, the complete matches of query Q_n can be produced, which is expected to be more efficient than a naive extensive search starting from either B or C.

For temporal subgraph queries, the selectivity of topology can be realized as the selectivity of topological predicates. Continuing our temporal subgraph query example Q_t, the straightforward processing is to extend the above processing of Q_n. That is, for each match of Q_n, we check if edges in the match jointly overlap in time. However, we should note that this processing might not be sufficiently efficient since the involved temporal predicates (e.g., the joint overlapping among edge intervals) can be more selective than the topology. Specifically, a better way to process this is to first enumerate all 2-sized temporal cliques composed of "employee" and "family"-labeled edges in the query window. Then, we check if the source vertices of edges in each temporal clique can be joined. If so, the temporal clique forms a match of Q_t. Since the set of desired temporal cliques in the original graph includes only {(Emi, David), (Emi, Carl)}, this processing is expected to be more efficient than the straightforward processing.

To summarize, the selectivity of a temporal subgraph query can be divided into *topological selectivity* and *temporal selectivity* based on its predicates. Both types of selectivity can impact query processing efficiency. In this thesis, we would consider the strategies for leveraging both topological and temporal selectivities to achieve more efficient temporal subgraph query processing.

1.3 Research questions

Motivated by practical interest, in this thesis, we aim at the efficient temporal subgraph query processing by investigating the full leverage of selectivity in temporal subgraph queries. However, this is not easy work since prior research [3] has defined the NP-hardness of non-temporal subgraph query processing. Futhermore, the involvement of temporal predicates would significantly increase the complexity of query processing. Thus, our methodology of investigation is to start from solving basic relevant questions and progressively approach our final aim. Figure 1.2 presents the schematic diagram of our specific questions that we would like to answer in this thesis.

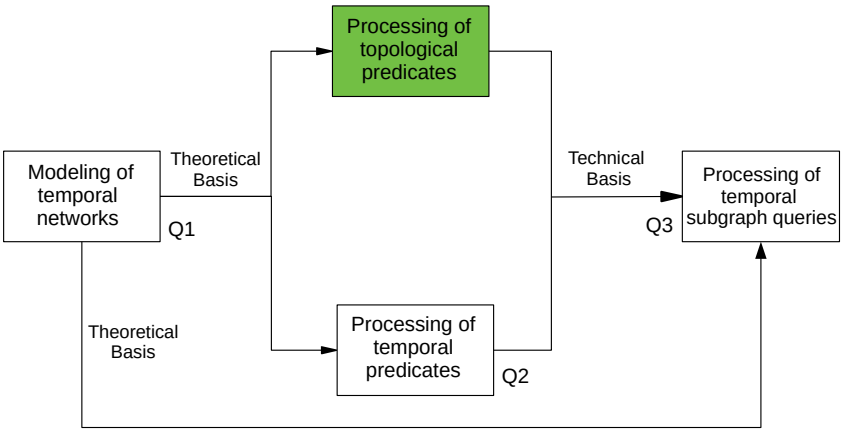


Figure 1.2: Schematic diagram of our research questions. The green-colored block demonstrates that the question has been primarily investigated in the state of the art. Note that we would not enter a question until all its preceding questions are investigated.

Modeling of temporal networks. Our first specific question is the modeling

of temporal networks. Answering this question helps to lay the theoretical basis for investigating the remained questions. That is, realistic and comprehensive model for temporal networks can guide us to capture the structures and better understand temporal networks in the real world. Moreover, such a model might provide us with the inspirations of leveraging selectivity for efficient query processing. However, prior works for network modeling mostly focused on the most fundamental characteristics. Therefore, we have a strong motivation to develop a modeling method for temporal networks that can capture more complex characteristics. This work can help to understand both networks and query processing in temporal contexts. We formalize our first research as follows.

Q1: How to develop a consistent model of temporal networks in the real world that can provide us with a clearer understanding of their structure? What network characteristics to capture in order to develop such a realistic model?

In Chapter 3, we propose a novel method for temporal network modeling based on our empirical findings for temporal networks. The straightforward aim is to capture the concurrent set size (CSS), a characteristic that is important for understanding temporal networks and query processing. Meanwhile, our proposed model can also capture other popular characteristics (e.g., degree, inter-event time, duration) under the constraint of the CSS. Theoretical analysis and empirical experiments demonstrate the effectiveness of our novel model.

Processing of temporal predicates. As we have discussed, two key elements in temporal subgraph query are the topological and temporal predicates. Thus, investigating the processing of these predicates helps to lay the technical basis for temporal subgraph query processing. Since prior works have primarily focused on topological-predicate processing, our second specific question is the efficient temporal-predicate processing. Temporal-predicate processing in our investigated queries can be realized as a problem of *temporal-k clique enumeration*. That is, we would like to enumerate all k-sized temporal cliques (for short, the temporal k-cliques) in which edges jointly overlap at a time point. Note that k is the parameter used to specify the number of edges in the queried pattern. However, the general temporal k-clique enumeration problem has not been identified and studied before. Prior works [4, 5] primarily studied the interval join problem, which aims to find all pairs of overlapping records from two relations and can be realized as a special case of our investigated problem with k=2. Therefore, we have a strong motivation to investigate the

temporal k-clique enumeration. To this end, we formalize our second specific question as follows.

Q2: How to enumerate temporal k-cliques efficiently in order to derive a method for temporal-predicate processing?

In Chapter 4, we propose a family of algorithms for efficient temporal k-clique enumeration. We then optimize the algorithms with checkpoints to overcome efficiency bottlenecks. Experimental evaluation demonstrates that our proposed algorithms can provide more efficient temporal k-clique enumeration.

Processing of temporal subgraph queries. The developed processing approaches of predicates enable us to investigate our third specific question, i.e., the processing of general temporal subgraph queries, which is also our ultimate question. Existing studies of temporal subgraph query processing are limited and primarily focus on the leverage of topological selectivity. Specifically, a traditional solution is to process queries using an existing pipeline in graph database systems, where the associated intervals are treated as edge properties. However, this solution can be inefficient since it follows the "topology then time" pipeline which ignores the selectivity of temporal predicates. We consider the temporal predicates as an important factor which can have a significant impact on query processing costs. Yet, there has been relatively little work on leveraging temporal selectivity in temporal subgraph query processing. Therefore, our final aim is to provide efficient processing for temporal subgraph queries which can fully leverage the query selectivity. We formalize our ultimate question as follows.

(Ultimate question) Q3: How to process temporal subgraph queries efficiently with full leverage of their selectivity?

In Chapter 5, we propose two methods for temporal subgraph query processing. Experimental evaluation demonstrates that our proposed methods can outperform current methods by a wide margin.

By answering Q1 to Q3, we succeed in efficient processing of temporal subgraph queries. In brief, our research procedure can be described as follows: First, we start by proposing a realistic model for temporal networks. Such a model can capture the structures and guide our understanding of real-world networks. Then, with our prior knowledge of temporal networks, we propose our methods for temporal k-clique enumeration. These methods are further

used to provide efficient processing of temporal predicates in subgraph queries. Finally, for temporal subgraph query processing, we analyze the demerits of the current methods and propose novel methods for efficient processing.

1.4 Contributions

In this thesis, our main contributions can be summarized as follows:

(1) For temporal network modeling,

- We study the characteristics of real-world networks. Particularly, we focus on a concurrent set size (CSS), an important characteristic of temporal networks but hardly ever considered in the prior works. We present our empirical findings for the CSS and discuss how it is related to the query processing in temporal contexts.
- Based on above discussion, we propose a novel *competition-driven* model (CDM) as a framework to generate networks constrained by CSS. We present a theoretical analysis of our CDM to demonstrate how it affects several important characteristics in generated networks. Further, we carry out an in-depth experimental study and our results demonstrate that CDM can simulate the real-world networks effectively and the generation process in CDM is scalable.

(2) For temporal-predicate processing,

- We propose a framework for temporal k-clique enumeration, which can be used to adjust existing sweep-based interval join algorithms to our problem. Compared to the most straightforward and naive solutions, the proposed adjusted algorithms have much lower complexity in temporal k-clique enumeration. Then, we carry out a careful analysis of the weaknesses in these algorithms and propose a novel method, the start time index (STI) algorithm, for more efficient k-clique enumeration.
- We develop checkpoint mechanisms to further improve query processing in STI. We discuss four checkpointing strategies and highlight their benefits. In addition to STI, these strategies are of independent interest and could also be applied in combination with other adjusted algorithms.

We carry out an in-depth experimental study and results demonstrate the significant improvements in scalability and performance introduced by our new methods.

(3) For temporal subgraph query processing,

- We first propose a method based on STI named TIME for processing queries with general patterns, which focuses on leveraging temporal selectivity. The rationale is to extend temporal k-clique enumeration with breath-first-based subgraph query processing. We further discuss several strategies for improving the efficiency of TIME.
- We further propose a novel method named leapfrog TSRJOIN, which leverages both topological and temporal selectivities for more efficient query processing. The rationale of this method is to inject the processing of temporal predicates into leapfrog triejoin, a worst-case optimal (WCO) join algorithm which has presented its excellent performance in solving various conjunctive queries in the state of the art. We further develop several mechanisms to optimize TSRJoin's processing efficiency.

We present the results of an in-depth experimental study which demonstrates significant improvement in performance introduced by our new methods.

1.5 Thesis overview and organization

Through a series of theoretical and empirical studies, this PhD thesis makes substantial contributions to the state of the art research in temporal subgraph query processing. The studies that comprise different chapters of this thesis have appeared in peer-reviewed conferences and journals. In order to make the construction of the dissertation more coherent, we make every effort to ensure that each chapter is consistent in definitions, notations, and so forth. To be more specific, the thesis is organized as follows.

Chapter 2 We present the common notations for both the data model and query model used in this thesis and conduct a comprehensive review of the existing research related to this thesis including network modeling, resources, and subgraph query processing.

Chapter 3 We investigate the problem of temporal network modeling. We first present our findings for real networks from an investigation of the existing literature and empirical observation. Then, based on our findings, we propose a novel method for modeling and generating temporal networks. Our theoretical analysis and experimental evaluation demonstrate that our proposed method results in a controllable benchmark, which can be used efficiently to simulate

and generate various networks. This chapter is an extension of our previously published peer-reviewed paper:

- **Kaijie Zhu**, George Fletcher, and Nikolay Yakovets. Competition-driven modeling of temporal networks. EPJ Data Science, 2021, 10(1): 1-24.

Chapter 4 We investigate the problem of temporal-predicate processing, i.e., temporal k-clique enumeration. We analyze the complexity of a straightforward solution and propose a processing framework with lower complexity. Based on our framework, we first propose three methods (i.e., CE-EBI, CE-gFS, and CE-bgFS) which are much less complex than the straightforward solution. Next, based on a careful analysis of the adjusted algorithms, we propose a novel STI algorithm to provide more efficient processing. Then, we develop four checkpoint mechanisms to further improve the processing in our proposed methods. We discuss four checkpointing strategies and highlight their benefits. Our experimental evaluation demonstrates that our proposed methods significantly improve processing scalability and performance. This chapter is an extension of our previously published peer-reviewed paper:

- **Kaijie Zhu**, George Fletcher, Nikolay Yakovets, Odysseas Papapetrou, and Yuqing Wu. Scalable temporal clique enumeration. In Proceedings of the 16th International Symposium on Spatial and Temporal Databases, 2019: 120-129.

Chapter 5 We investigate the problem of temporal subgraph query processing. We note that the state of the art in subgraph query processing primarily focused on leveraging the selectivity of topological predicates in queries. Thus, we first propose a method (i.e., TIME) which focuses on leveraging temporal selectivity. Then, based on a careful analysis of TIME, we propose a novel method (i.e., leapfrog TSRJOIN) which focuses on leveraging both temporal and topological selectivities. Our experimental evaluation demonstrates that our proposed method can provide much more efficient query processing than the state of the art. This chapter is an extension of our previously published peer-reviewed paper:

- **Kaijie Zhu**, George Fletcher, and Nikolay Yakovets. Leveraging temporal and topological selectivities in temporal-clique subgraph query processing. In Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2021: 672-683.

Chapter 6 We conclude the thesis with a discussion of future work based on our proposed temporal network modeling and query processing approaches. We believe that the methods in this thesis and their described results are of value to the research community as a basis for understanding the merits of the approaches and for further research on temporal graph analysis.

2 Background

2.1 Common notations

In this section, we provide a set of common notations used throughout this thesis. Note that in several chapters, some additional symbols will be used locally. An overview of notations across the thesis is presented in Table 2.1. In the following, we present the details for these notations.

2.1.1 Data model

Compared to traditional graph-structured data which considers only the interactions, temporal graphs associate each interaction with a *time window* to represent the evolution of data structures over time. Formal definitions are presented as follows:

Definition 2.1.1 (Time window) *A* time window *is an ordered pair of non-negative integers* $[i, j]$ *such that* $i \leq j$*. We refer to* i *and* j *as* timestamps*. We say time window* $[i, j]$ contains *time window* $[k, l]$ *if* $k \geq i$ *and* $l \leq j$*, which we denote by* $[k, l] \sqsubseteq [i, j]$*. We say* $[i, j]$ *and* $[k, l]$ overlap *if* $i \leq l$ *and* $k \leq j$*, i.e., there is a time window* w *contained in both* $[i, j]$ *and* $[k, l]$*. The* length *of window* $w = [i, j]$ *is the value* $|w| = j - i$*.*

Moreover, given two time windows $[i, j]$ and $[k, l]$, we define the operation $[i, j] \cap [k, l]$ as follows:

- If (1) both windows are non-empty; and, (2) $i \leq l$ and $k \leq j$ (i.e., $[i, j]$ and $[k, l]$ *overlap*), then $[i, j] \cap [k, l] = [\max\{i, k\}, \min\{j, l\}]$, i.e., the maximum (w.r.t. $\sqsubseteq$) time window contained in both time windows.
- Otherwise, $[i, j] \cap [k, l] = \emptyset$.

Symbol	Notation
Notations for Graph	
G	The original temporal graph, i.e., $G = (V, E, \eta, \lambda, \tau)$
R	The non-graph-structured temporal relation, i.e., $R = (E, \tau)$
V	The relation of vertices
E	The relation of edges
L	The set of labels
T	The set of timestamps
$\|V\|$	The number of vertices in V
$\|E\|$	The number of edges in E
$\|L\|$	The number of labels in L
$\hat{T}$	The maximal timestamp in T
$\eta(e)$	The endpoint pair of edge $e \in E$, i.e., $\eta(e) = (u, v)$
$\lambda(e)$	The label of edge $e \in E$, i.e., $\lambda(e) = l$
$\tau(e)$	The associated time window of edge $e \in E$, i.e., $\tau(e) = [t_s, t_e]$
$source(e)$	The source of edge e, i.e., $source(e) = u$
$destination(e)$	The destination of edge e, i.e., $destination(e) = v$
$starttime(e)$	The start time of edge e, i.e., $starttime(e) = t_s$
$endtime(e)$	The end time of edge e, i.e., $endtime(e) = t_e$
R_E	The edge stream representation of G
$G(t)$	The snapshot of graph G at time t
$E(t)$	The set of edges active at time t in E
Notations for Query	
q	The original temporal subgraph query, i.e., $(e_1, \ldots, e_k) \leftarrow l_1(u_1, v_1), \ldots, l_k(u_k, v_k), [q_s, q_e]$
$v_1^q \ldots v_{2k}^q$	The query vertices in q, i.e., $u_1, v_1 \ldots u_k, v_k$
$e_1^q \ldots e_k^q$	The query edges in q, i.e., $l_1(u_1, v_1), \ldots, l_n(u_k, v_k)$
$[q_s, q_e]$	The query time window of q
ϵ	A complete match of q, i.e., $(e_1, \ldots, e_k, [\epsilon_s, \epsilon_e])$
$[\epsilon_s, \epsilon_e])$	The life-span of ϵ, i.e., $\tau(e_1) \cap \cdots \cap \tau(e_k)$
Temporal clique	$S = (R, \tau)$ is a temporal clique if there exists a time window t such that $t \sqsubseteq \tau(r)$ for $\forall r \in R$
Partial match	$(e'_1, \ldots e'_m, [\epsilon'_s, \epsilon'_e])$ is a partial match of q if: (1) For each $i \in [1, m]$ there exists $j \in [1, k]$ such that $e'_i \sim e_j^q$ (2) $[\epsilon'_s, \epsilon'_e] = \tau(e'_1) \cap \cdots \cap \tau(e'_m)$ and $[\epsilon'_s, \epsilon'_e] \cap [q_s, q_e] \neq \emptyset$
Edge match	An edge e is an edge match of e_i^q if $\lambda(e) = e_i^q$, i.e., $e \sim e_i^q$
Clique match	A temporal clique in $[q_s,q_e]$ of edges $\{e_1 \ldots e_k\}$ is a clique match of q if $e_i \sim e_i^q$ for $\forall i \in [1, k]$

Table 2.1: Common notations and their symbols across the thesis

Definition 2.1.2 (Temporal graph) *Let L be a set of labels and T be a set of timestamps. A* temporal graph *is a structure $G = (V, E, \eta, \lambda, \tau)$, where: V and E are respectively relations of vertices and edges; $\eta : E \to V \times V$ is a function assigning to each edge an ordered pair of vertices, denoted $\eta(e) = (u, v)$, where $e \in E$ and $u, v \in V$; $\lambda : E \to L$ is a function associating each edge with a label, denoted $\lambda(e) = l$, where $e \in E$ and $l \in L$; $\tau : E \to T \times T$ is a function assigning to each edge a time window, denoted $\tau(e) = [t_s, t_e]$ where $e \in E$, $t_s, t_e \in T$, and $t_s \leq t_e$.*

For convenience, we call l, u, v, t_s, t_e respectively the *label*, *source*, *destination*, *start time*, and *end time* of e. We overload the $source()$, $destination()$, $starttime()$, $endtime()$ functions, allowing them to take an edge e as input and return its u, v, t_s, t_e respectively [1]. We say an edge e is active at a certain timestamp t if $t \in [starttime(e), endtime(e)]$. We use $|V|, |E|, |L|$ to represent the number of vertices, edges, and labels in G.

If one concerns only the graph topology, the notation can be simplified to a *non-temporal graph* denoted $G^s = (V, E, \eta, \lambda)$, where function τ to specify the temporal structure is omitted. This is also the primarily focused model for graph analysis in the state of the art. Similarly, if one concerns only the temporal aspect, the notation can be simplified to a non-graph-structured temporal relation where each element is only associated with a time window. We formalize the notation as follows, which is closely related to Chapter 4 in this thesis:

Definition 2.1.3 (Temporal relation) *A temporal relation is a structure $R = (E, \tau)$, where $\tau : E \to T \times T$ associates each element in E with a time window $[t_s, t_e]$.*

Compared to Definition 2.1.2, V, η, λ are omitted in the notation of temporal relation as R is non-graph-structured. For convenience, we say each element $r \in E$ is an element in temporal relation R, denoted $r \in R$. The size of temporal relation R is $|E|$, denoted $|R| = |E|$.

Since property graph is widely supported in current database systems, given a temporal graph G, the most general implementation in existing works is the *temporal property graph* which stores the start and end time as the edge properties. Besides, there are still other representation alternatives in current works. Different representations reflect researchers' various interests in

[1] For label l, there is already $\lambda(e) = l$.

graphs. Here we present the outline of two most popular alternatives. The first alternative is the *edge stream*, which is defined as follows.

Definition 2.1.4 (Edge stream) *Given a temporal graph G, its edge stream is a relation R_E where:*

- *For each edge $e \in E$, there is a 6-tuple (e, $source(e)$, $destination(e)$, $\lambda(e)$, $starttime(e)$, $endtime(e)$) $\in R_E$,*
- *For each tuple (e, u, v, l, t_s, t_e) $\in R_E$, there is $e \in E$ such that $\eta(e) = (u, v)$, $\lambda(e) = l$, $\tau(e) = [t_s, t_e]$.*

Edge stream is a practical and easy implementation for computation purposes on a graph (e.g., computation of graph statistics). However, such representation fails in expressing the topology structure of graphs, which can be a crucial component in many scenarios of graph analysis (e.g., subgraph query processing). Therefore, the second alternative primarily focuses on the instant topology, which is named *snapshot sequence* and defined as follows.

Definition 2.1.5 (Snapshot sequence) *Given a temporal graph G, its snapshot sequence is a series of non-temporal graphs $G(1), \ldots G(\hat{T})$ where $\hat{T}$ is the maximal timestamp in T. Each $G(t) = (V, E(t), \eta, \lambda)$ such that $t \in T$ is a non-temporal graph named snapshot where $E(t)$ is the set of edges active at time t.*

A snapshot sequence can clearly reflect the instant status of an original graph for each $t \in T$. However, such implementation fails to capture the accurate dynamic of G (e.g., start and end time of edges) and can be inefficient in storage when G is large. For this reason, in this thesis, we consider the temporal property graph as the general implementation of graphs for its expression power in both topological and temporal structures.

2.1.2 Query model

We start by presenting the definition of *temporal clique*, an important underlying structure constrained by the temporal predicates in temporal subgraph query.

Definition 2.1.6 (Temporal clique) *Given a temporal relation $G = (E, \tau)$, if there exists a time window t such that for every element $e \in G$ it is the case that $t \sqsubseteq \tau(e)$, then we say the elements in G forms a* temporal $|E|$-clique.

Moreover, given a time window w, if $t \sqsubseteq w$, we say the elements in G forms a temporal $|E|$-clique in w.

Here "temporal clique" emphasizes that the edges are *tightly interconnected in time*, in addition to satisfying the topological pattern of interest. This is in contrast to traditional "cliques"' in which vertices are *tightly interconnected in topology*.

Next, we present the definition of temporal subgraph query.

Definition 2.1.7 (Temporal subgraph query) *A temporal subgraph query is a pattern q of the form*

$$(e_1, \ldots, e_k) \quad \leftarrow \quad l_1(u_1, v_1), \ldots, l_k(u_k, v_k), [q_s, q_e]$$

where $e_1, \ldots, e_k, u_1, v_1, \ldots, u_k, v_k$ are variables (possibly with repetition); $l_1, \ldots, l_k \in L$; and, $q_s, q_e \in T$ where $q_s \leq q_e$. Given a temporal graph $G = (V, E, \eta, \lambda, \tau)$, the evaluation of q on G is the set of all matches $\epsilon = (e_1, \ldots, e_k, [\epsilon_s, \epsilon_e])$ such that:

1. *$e_1, ..., e_k \in E$ and $\epsilon_s, \epsilon_e \in T$;*
2. *there exists a function $f : \{u_1, v_1, ..., u_k, v_k\} \to V$ such that $f(u_i) = source(e_i)$, $f(v_i) = destination(e_i)$, and $\lambda(e_i) = l_i$, for $\forall i \in [1, n]$; and,*
3. *$[\epsilon_s, \epsilon_e] = \tau(e_1) \cap \cdots \cap \tau(e_k)$ and it holds that $[\epsilon_s, \epsilon_e] \cap [q_s, q_e] \neq \emptyset$.*

For convenience, we call $l_i(u_i, v_i)$ the ith query edge of q, denoted e_i^q. We call u_i, v_i the *query vertices* of q, denoted v_{2i-1}^q, v_{2i}^q. We call each match $\epsilon : (e_1, \ldots, e_k, [\epsilon_s, \epsilon_e])$ a *complete match* of q. We call $source(e_i)$ and $destination(e_i)$ the *vertex bindings* of v_{2i-1}^q and v_{2i}^q, resp. We call $[\epsilon_s, \epsilon_e]$ the lifespan of ϵ. We call the set of all complete matches the *complete result* of q. We call constraint (2) the *topological predicate* of q since this ensures the topological structure in a match. Similarly, we call constraint (3) the *temporal predicate* of q since it ensures the temporal overlapping behavior in a match.

Note that our defined query aim to find matches for query edges instead of vertices, since multiple edges (e.g., associated with different time intervals) can exist between the same pair of vertices. Two matches ϵ_1 and ϵ_2 are viewed as distinct matches if they differ on their bindings of at least one query edge. Our presented query example Q_n, which aims to find matches for query vertices, can be realized as a reduction of our definition. In Chapter 5, we would present more concrete examples of our defined temporal subgraph queries.

The processing pipeline for query is generally not atomic, where numerous intermediate tuples can be produced before they become complete matches. We present the following definitions for the convenience of investigating the intermediate status.

Definition 2.1.8 (Edge match) *Given a temporal subgraph query q, an edge e is an edge match of query edge e_q^i if $\lambda(e_i) = l_i$, denoted $e_i \sim e_i^q$. Also, we call e an edge candidate of query q. Specifically, given a complete match $(e_1, \ldots, e_k, [\epsilon_s, \epsilon_e])$, there is $e_i \sim e_i^q$ for $\forall i \in [1, k]$.*

Definition 2.1.9 (Partial match) *Given a temporal subgraph query q, $(e'_1, \ldots e'_m, [\epsilon'_s, \epsilon'_e])$ is a partial match of q if:*

1. *$e'_1, ..., e'_m \in E$ and $m < k$;*
2. *For each $i \in [1, m]$ there exists $j \in [1, k]$ such that $e'_i \sim e_j^q$; and,*
3. *$[\epsilon'_s, \epsilon'_e] = \tau(e'_1) \cap \cdots \cap \tau(e'_m)$ and $[\epsilon'_s, \epsilon'_e] \cap [q_s, q_e] \neq \emptyset$.*

Definition 2.1.10 (Clique match) *Given a temporal subgraph query q, a temporal clique in $[q_s, q_e]$ composed of edges $\{e_1, \ldots e_k\}$ is a clique match of q if $e_i \sim e_i^q$ for $\forall i \in [1, k]$. Specifically, given a complete match $(e_1, \ldots, e_k, [\epsilon_s, \epsilon_e])$, $\{e_1, \ldots, e_k\}$ is a clique match of q.*

2.2 Network modeling

In the past few decades, numerous studies on temporal network modeling have been carried out to capture the theoretical grounding and characteristics of temporal networks in the real world. The most straightforward method [6, 7] is to model the generation of temporal networks by associating each edge in non-temporal graphs with timestamps. Currently, the most well-studied network model is the activity-driven network (ADN) model proposed by Perra et al. [8]. This model initializes each vertex v with a firing rate a_v drawn from a given probability distribution $F(x)$. At each timestamp t and with probability a_v, vertex v becomes active and generates m instant outgoing edges linked to the other vertices randomly. Several studies have been carried out to extend the model in both structural and temporal fields. For structural extension, prior studies concentrated on selecting the edge destination [9, 10, 11, 12]. Alessandretti et al. [9] extended each vertex with an *attractiveness* value representing its probability of being selected as the destination of edges. Other works extended the model with a reinforcement mechanism, which exhibits the preference of vertices to connect to previously contacted vertices [10, 11, 12].

Another collection of works concentrated on the incorporation of community structure [11, 13]. Laurent et al. [11] introduced focal closure and cyclic closure, which gives rise to the community structure in the network. Nadin et al. [13] initialized each vertex to a community. In each turn, a vertex could either connect other vertices within (or outside) the same community with probability μ (or 1 - μ). For temporal extension, Sunny et al. [14] introduced the duration for edges so that edges are lasting entities rather than instant ones. Besides ADN, there are also other categories of temporal network generation models. The Renewal process model extends the Gillespie algorithm [15] to model the network generation where each vertex is modeled as a Poisson process and the superposed vertices are regarded as the inter-event time distribution [16, 17]. Starnini et al. [18] and Zhang et al. [19] modeled the generation as a process involving agents performing a random walk in the unit square. Each agent interacts with its neighbors every time a random walk is performed. To deal with situations where information of entities is missing, Cho et al. [20] proposed a self-exciting process model where the event rate between each pair of entities is modeled as a Hawkes process.

2.3 Benchmark resources

We have considered diverse resources for temporal networks, which can be divided into two categories: *real-world datasets* and *synthetic generators*. These allow a comprehensive study on temporal networks and experimental evaluation of our proposed methods in this thesis. Here we present an overview of the network resources.

2.3.1 Real-world datasets

In the following, we present several popular repositories of temporal graph-structured datasets covering social, transportation, and networking domains. Corresponding temporal graphs can be constructed on the datasets at a small cost.

Social. *Stanford Network Analysis Project (SNAP)*[2] and *Koblenz Network Collection (KONECT)*[3] collect the networks of interaction in both the real-world (e.g., college message, email) and online communities (e.g., StackOverflow, Wikipedia, Google). For example, the largest network in SNAP records the

[2] `https://snap.stanford.edu/data/`

[3] `http://konect.cc/networks/`

interactions of comments, questions, and answers among users on StackOverflow, which has 2,601,977 vertices and 63,497,050 edges in total. Besides, *SociaPattern*[4] provides 14 temporal interaction networks in much smaller communities such as workplaces, schools, and conferences.

Transportation. Many real transportation datasets record the vehicle trips associated with their valid period, source, and target locations. Temporal networks can be built conveniently on these datasets, where vertices and edges respectively represent the locations and trips. Several instances are presented as follows. *NYC Taxi&Limousine Commission*[5] collected the historical vehicle trips (yellow taxi, green taxi, for-hire vehicles) in New York starting from 2009. *NYC citibike*[6], *Divvy Data*[7], *Metro bike*[8], collected the bike trips from different cities in the US. *US department of transports*[9] collected the national historical traffic statistics from various transportation domains including aviation, maritime highway, etc..

Networking. *Center for Applied Internet Data Analysis (CAIDA)*[10] collects various statistics that can be used for benchmarking. The most straightforward category is the snapshots of networking relationships. For example, relationships among different autonomous systems in hundreds of snapshots are provided in [21]. Another category is traffic statistics. With knowledge in network engineering, researchers can construct temporal networks of connections by extracting and aggregating the traffic packets in datasets. For example, a temporal network of anonymized passive traffic traces can be constructed on [22] by aggregating the packets into connections associated with periods.

2.3.2 Synthetic generator

Real-world temporal networks are not numerous for their difficulty to be collected and subject to confidentiality agreements. For many researchers, a more practical choice is to generate synthetic networks which can well reproduce the characteristics of real networks. The most straightforward method is to generate synthetic networks with the implementable works on temporal network

[4] http://www.sociopatterns.org/datasets/

[5] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[6] https://www.citibikenyc.com/system-data

[7] https://www.divvybikes.com/system-data

[8] https://bikeshare.metro.net/about/data/

[9] https://www.transtats.bts.gov/

[10] https://www.caida.org/data/

modeling [6, 7, 8, 9, 10, 11, 12, 11, 14, 13] as we summarized. A drawback of the implementable models is their focus on the simulation of most fundamental characteristics (e.g., degree distribution, burstiness phenomenon) in temporal networks. The state-of-the-art methods have considered the simulation of more complex structures (e.g., clustering, motifs) in temporal network modeling and generation. Gorke et al. [23] proposed a method for generating clustered temporal random networks, in form of snapshot sequences. The authors generate the first snapshot by Gilbert's non-temporal graph model and the rest from their prior snapshot by atomic updates based on pre-computed probability. Similarly, Leeuwen et al. [24] extended *gMark*, a powerful schema-driven benchmark for non-temporal networks, with their proposed algorithm to generate snapshot sequences which satisfy the monotonicity and are significantly more stable than those generated. Purohit et al. [25] proposed a method using the computed temporal motifs distribution to generate monotonic increasing networks in which the temporal evolution of the local structures is preserved. Zeno et al. [26] carried out empirical studies on temporal motifs in real-world graphs and proposed dynamic motif activity (DMA) model for sampling synthetic dynamic graphs with parameters learned from an observed network. Zhou et al. [27] presented a deep generative framework named *TagGen*. The framework started by sampling temporal random walks from real-world datasets and generating synthetic randoms walks with a family of defined local operations. Then, a discriminator is trained over the sampled random walks and used to determine the plausible synthetic random walks. Finally, the plausible walks are fed to an assembling module for network generation. The advantage of the framework is its independence from prior structural assumptions, so it can be used to generate networks without any prior knowledge.

2.4 Query processing

We divide the current research on query processing into the following three categories: (1) on topological predicates (i.e., the processing of non-temporal subgraph queries); (2) on temporal predicates (i.e., the enumeration of temporal cliques); and (3) on temporal subgraph query (i.e., the processing of queries involving both topological and temporal predicates).

2.4.1 Topological predicates

Topological predicates are generally processed by executing a guided search over a given graph. During the search, query vertices are bound to graph vertices to produce (partial) matches. Several different search strategies exist along with pruning strategies which aim to minimize the part of the graph explored during the search. Specifically, existing works can be divided into depth-first-based and breath-first-based.

In depth-first-search-based approaches, the matches are extended by matching query vertices to vertices in a graph, i.e., vertex-at-a-time. The first method of this category is Ullman's backtracking algorithm [28]. In the past few decades, numerous studies have been carried out to improve the efficiency of Ullman's backtracking algorithm by leveraging the query selectivity (e.g., optimizing the matching order [29, 30, 31, 32], pruning false-positive candidates [33, 34]). In the current database systems, a series of WCO-join algorithms (e.g., NPRR, Leapfrog Triejoin [35], Generic-join [36], Minesweeper [37]) have been proposed as the core of this category.

In breadth-first-search-based approaches, the matches are produced by processing a query graph edge-at-a-time. This category first decomposes a query into a set of basic query units [38, 39, 40, 41]. Then, each unit is processed to produce its partial matches. Finally, binary joins (BJ) are performed to concatenate all intermediate results. To sum up, this category is based on BJs which extend the partial match by matching query edges to corresponding edges in a graph.

2.4.2 Temporal predicates

Current research primarily focused on the interval join problem, which can be viewed as a special case to the related processing of temporal predicates. Specifically, given two temporal relation R_1 and R_2, the problem aims to enumerate all pairs of elements (r, s) such that $r \in R_1, s \in R_2$, and $\tau_1(r) \cap \tau_2(s) \neq \emptyset$. Research on interval join processing can be classified in index-based, partition-based, and plane-sweep methods. Index-based methods construct and maintain specialized data structures in order to speed up query processing. A bi-temporal index that could be used to compute interval joins on two temporal dimensions (i.e., both system and application time) is proposed in [42]. An algorithm based on a two-layer flat index (Overlap Interval Inverted, O2i) is presented in [43]. Indexed segment tree forest (ISTF), in which the temporal nesting relationships are represented by a binary tree and

joins are enumerated by searching related trees, is proposed in [44]. Partition-based methods cluster intervals into smaller buckets based on their similarity and join processing is done for certain pairs of buckets to reduce the unproductive evaluations. Dignos et al. [45] proposed a self-adjusting algorithm named OIP. The algorithm divides intervals into n equal-sized consecutive granules with a proposed method for a best n parameter, which could lead to a minimal compromise of query costs and unproductive join ratio when the timeline is divided into the same number of granules. Cafagna et al. [46] proposed DIP to divide temporal relation into partitions containing non-overlapped tuples, which also reduces the number of unproductive join operations in evaluation.

Currently, the best performing solutions for interval joins are based on plane-sweep methods [5]. Piatov et al. [4] proposed two memory plane sweep-based interval join algorithms EBI and LEBI based on endpoint index, which outperform OIP and prior plan-sweep methods. Bouros et al. [5] proposed two optimized algorithms based on forward scan named gFS and bgFS. In Chapter 4, we will present more details about both the state of the art.

2.4.3 Temporal subgraph query

Current research on temporal subgraph query processing is limited. Franzke et al. [47] proposed a method which creates an index to record the occurrences of basic motif structures (e.g., triangle) in a graph. In processing, the index is used to fast locate the candidates and reduce the search space. Moreover, several pruning rules are further used to refine the candidate sets. Semertzidis et al. [48] proposed an indexed method for obtaining the top-k durable matches. Though these methods can be used for our investigated problem, their considered graph implementation is the snapshot sequence so that they are not suitable for query processing in general scalable temporal property graph implementation. Xu et al. [49] proposed a method named TCGPM-E for temporal subgraph query processing. For each query, TCGPM-E first produces the non-temporal matches over the subgraphs centering at its selective edges. Then the algorithm filters the matches with pruning rules based on temporal predicates. A similar processing method can be found in modern database systems where query models over property graphs and hybrid planning engines [50, 51] are supported. Its common idea is to treat the temporal predicates of a query as general selection properties. In this way, physical plans, which are composed of join operators to process topological predicates and selection operators to filter the intermediates that do not satisfy temporal predicates, can be generated and used for temporal subgraph query processing. However, these methods can

be very inefficient since the selectivity of temporal predicates is not fully leveraged during query processing. To be more specific, the non-temporal matches can be extremely large while the temporal predicates are naively used as a filter to select valid matches among them.

Summarizing, currently there is no general and efficient processing method for temporal subgraph queries, which is also one of the motivations in this thesis.

3

Modeling of temporal networks

3.1 Motivation

With the aim of efficient temporal subgraph query processing, our first investigated question is how to model the graph-structured data (i.e., temporal networks) in the real world. The modeling problem is of various interest to researchers. First, an implementable temporal model can be used as the benchmark to generate numerous synthetic networks as a complement for the limitation in available real-world networks. Second, a consistent model for temporal networks can capture the essential graph structures and help researchers to understand real-world systems. Specifically, in this thesis, such a model can provide us with the guidance of leveraging selectivities in query processing. Thus, we have a strong motivation to develop a realistic model for temporal networks.

For network modeling, a key topic is to capture the characteristics of interest. In this chapter, we primarily focus on a characteristic named concurrent set size (CSS). Given a temporal network, *concurrent set (CS)* represents the collection of edges active at time $t \in T$. CSS distribution reflects the evolvement of edges' density over time. By summarizing the existing models and synthetic benchmarks of temporal networks in Chapter 2, we note that the distribution of *concurrent set size* (CSS) has never been considered in the state of the arts. We first formalize these notions as follows.

Definition 3.1.1 (Concurrent sets) *Given a temporal graph* $G = (V, E, \eta, \lambda, \tau)$ *and a timestamp* t*, we call the largest temporal clique* $S \subseteq E$ *in* $[t, t]$ *the Concurrent Set at* t*, denoted* $CS(t)$*. In other words,* $CS(t)$ *consists of all records that are active at timestamp* t*, i.e.,* $CS(t) = \{e \in E | t \in \tau(e)\}$ *The size of* $CS(t)$ *is denoted by* $C(t)$*.*

Definition 3.1.2 (CSS distribution) *Given a temporal graph G and its snapshot sequence $\{G(1) \dots G(\hat{T})\}$, a CSS distribution is represented as a function $C(t) = |E(t)|$ for $\forall t \in T$. That is, the CSS distribution indicates the number of active edges at each timestamp t.*

Note that these notions can be also applied to a non-graph-structured temporal relation since they are determined by only E and τ. Formally, given a temporal relation R, corresponding $CS(t)$ (or $C(t)$) represents all active elements (or the number of active elements) at time t in R. In Chapter 4, we would present more usage examples in non-graph-structured context.

We consider CSS as an important metric for temporal networks. First, many real-world networks are constrained by specific CSS distribution (i.e., the CSS-constrained networks). The distribution in these networks reflects the aggregation phenomenon of temporal events in temporal aspects. For example, in a transportation network, peaks in CSS distribution capture the traffic rush hours. This demonstrates that studying CSS distribution can lead to a better understanding of real-world networks and that modeling CSS can help to generate realistic synthetic temporal networks. Second, CSS is a factor which can impact graph analysis approaches. For example, the interval join algorithm EBI [4] can be impacted by CSS since it maintains real-time active records in memory during the whole procedure. Specifically, higher CSS value at timestamp t (i.e., $C(t)$) will generally lead to higher maintenance costs on real-time active records at time t in EBI. In Section 3.4.3, we would further discuss how CSS can be used to impact temporal subgraph query processing, i.e., our ultimate question (Q3) that is going to be investigated in this thesis.

3.2 Problem statement

Motivated by above discussion, in this chapter, we focus on the modeling problem of CSS-constrained networks. Our goal is to find a better way to model temporal networks which can capture the CSS distribution in real-world temporal networks. Formally, given a set of vertices V and the target CSS distribution $C(t)$, we aim to generate a temporal network G such that: given its snapshot sequence $\{G(1) \dots, G(\hat{T})\}$, there is $|E(t)| = C(t)$ for $\forall t \in T$.

In order to model and generate realistic synthetic networks, our model should also capture other important network metrics besides CSS distribution. Table 3.1 presents all our concerned metrics in modeling. In Chapter 3.4, we would present the notations for these listed metrics except $|E|$ and $C(t)$.

Concerned metrics	
$\|E\|$	Number of generated edges
$A(v)$	Relative degree
$\mathcal{I}$	Inter-event time distribution
$\mathcal{D}$	Duration Distribution
$C(t)$	CSS distribution

Table 3.1: Overview of the important metrics of the generated networks

Such a model would guide our understanding of temporal networks and upcoming studies on query processing. Besides, it can be used to generate realistic synthetic networks in various applications.

3.3 Contributions

In this chapter, our main contributions can be summarized as follows.

- We study the important characteristics of temporal networks in the real world. For our concerned fundamental characteristics, we summarize their related findings from the state of the art. For the characteristics which have never been investigated, we carry out empirical observation on real-world datasets and summarize our general findings.
- Based on the above study, we propose the competition-driven model (CDM) for modeling and generating the temporal networks constrained by CSS. This model can guide us to better capture and understand the characteristics of real-world datasets, with our aim of leveraging selectivities and providing efficient query processing.
- We carry out both theoretical analysis and experimental evaluation on CDM. The results demonstrate that CDM is controllable and can well-simulate the temporal networks in the real world.

3.4 Characteristics in real-world networks

A popular research method for network modeling can be described as follows. First, researchers observe real-world datasets empirically and capture network characteristics. Next, based on the empirical observation, researchers propose their modeling method to mimic their findings and analyze the model theoret-

Research	Description	Result
Activity rate [8]	Each individual is associated with an activity rate. Those with higher activity rate have a higher probability to be selected as sources.	Heterogeneities of out-going degree
Global popularity [9]	Each individual is associated with a value of attractiveness. Those with higher attractiveness have a higher probability to be selected as destinations.	Heterogeneities of in-going degree
Reinforcement mechanism [10, 11, 12]	Individuals tend to start more interactions towards existing strong ties and less new interaction for weak ties.	Emergence of strong and weak ties
Focal and cyclic closure [11]	Time-stamped interactions between individuals give rise to temporal motifs.	Emergence of triangles and clusters
Burstiness [52, 14]	There can be a long period of quiet time and nothing happens. Then, many events can suddenly happen in a short period.	Emergence of collective phenomena and heavy-tail phenomenan in IET and duration distribution

Table 3.2: Several recognized network characteristics in existing works.

ically. Finally, experiments are carried out to verify the validity of the model. Following the common method, in this section, we first discuss the recognized network characteristics in existing research, of which an overview is presented in Table 3.2. Then, we carry out our observation over several real-world networks, analyze the empirical findings for CSS, and discuss how CSS is related to our ultimate question (Q3). Our final proposed model in this chapter should be compatible with the recognized characteristics.

3.4.1 Characteristics of topological structure

The most fundamental behaviors of topological structure in systems are *individual activity* and *engaging preference*. In temporal networks, these two behaviors demonstrate the propensity of vertices to involve in interactions (i.e., to become the source or destination of edges respectively), which can be cap-

tured generally by the out-going and in-going degree of vertices.[1] Empirical observation demonstrates that they both perform variability in many real-world cases and generally result in the emergence of heterogeneities and hubs. Several works have been carried out to simulate the two behaviors. For individual activity, the activity-driven network (ADN) [8] framework captured the heterogeneities by initializing each vertex with an *activity* rate from an activity potential function, which represents the probability for each vertex to become active and generate out-going edges at each time-point. For engaging preference, things become more complex since this characteristic can be affected by various phenomena and mechanisms. Global popularity captured (and mimic) the heterogeneous opportunities of vertices to be selected as targets in interactions, i.e., a vertex with higher attractiveness has a higher probability to be selected as targets [9]. Reinforcement mechanisms captured the emergence of strong and weak ties in networks caused by memory, i.e., individuals tend to start more interactions towards existing strong ties and fewer new interactions for weak ties [10, 11, 12]. Focal and cyclic closure captured the emergence of clusters [11].

In this chapter, we primarily focus on the out-going degree. For the convenience of comparison and analysis, the degree of a vertex needs to be stable in its distribution across networks of different sizes. For this purpose, we define a *relative* degree of a vertex as follows.

Definition 3.4.1 (Relative degree) *Given a temporal graph* $G = (V, E, \eta, \lambda, \tau)$ *and* $\forall v \in V$*, we call* $A(v) = \frac{\delta(v)}{|E|}$ *the* relative *degree of* v*, where* $\delta(v)$ *is the number of edges outgoing from* v*. As defined,* $A(v)$ *denotes the proportion of edges starting from a given vertex* v*.*

Note that vertex degree also provides partial guidance of topological selectivity for query processing. Specifically, a vertex with lower degree demonstrates that joins on this vertex would be more selective. Such relevance is also a motivation for us to capture the relative degree in network modeling.

3.4.2 Characteristics of temporal structure

The most fundamental metric used to capture the timing behavior in systems is *inter-event time* (IET). Considering a temporal relation in which events are

[1] In temporal context, the notion of degree varies according to various application and researchers' interest. For example, it can be either the number of instantly attached edges in a time temporal, or the accumulating edges over time.

temporally sorted by start time in ascending order, IET is the period between any two consecutive events. From the individual level, IET captures the activity ratio of vertices over time (i.e., for each vertex, we construct a temporal relation consisting of its out-going edges). While from the system level, IET captures the intermittence pattern in the whole system (i.e., we construct a temporal relation consisting of all edges in a graph).

By assuming a constant activity rate in prior research, the timing of activities can be modeled with a Poisson process, in which IET follows an exponential distribution. However, recent empirical observations in many datasets have captured the *burstiness* [52] behavior for IET. That is, there can be a long period of quiet time and nothing happens. Then, many events can suddenly happen in a short period. Burstiness results in the emergence of collective phenomena and apparent *heavy-tail* phenomenon in IET distribution: first, most emerged IETs aggregate in the short collection. Second, compared to the exponential distribution, the observed decaying is much slower, which allows the emergence of extremely long IET. Specifically, it has been recognized that IETs in many real-world systems follow a power-law or power-law cut-off distribution. In recent years, numerous studies [53, 54, 55, 56] have been carried out to explain and understand the burstiness pattern via various models. These efforts make burstiness a well-documented phenomenon in temporal network modeling. Similar characteristics (i.e., the heavy-tail phenomenon) can also be found in network duration distribution [14], to which related works are much more limited than IET.

In this chapter, we focus on the distribution of IET and duration to capture the activity behavior of vertices and edges in the temporal aspect respectively. These notations are defined as follows.

Definition 3.4.2 (IET distribution) *The distribution captures the activity behavior of vertices. Given temporal graph* $G = (V, E, \eta, \lambda, \tau)$ *and* $\forall v \in V$*, we collect the distinct start times of edges outgoing from* v*, denoted* $\theta(v) = \{t_1^v, t_2^v, \ldots, t_\epsilon^v\}$ *where* $t_i^v \in [1, T]$ *and* $t_i^v < t_{i+1}^v$*. For* $i \in [1, \epsilon)$*, we call* $\tau_i(v) = t_{i+1}^v - t_i^v$ *an inter-event time (IET). We assume that* τ *follows a probability distribution* $\mathcal{I}(f, \underline{\tau}, \overline{\tau})$*, where* f *is a parameter distribution function,* $\underline{\tau}$ *and* $\overline{\tau}$ *are minimum and maximum IETs respectively.*

Definition 3.4.3 (Duration distribution) *The distribution captures the activity behavior of edges. Given temporal graph* $G = (V, E, \eta, \lambda, \tau)$ *and* $\forall e \in E$*, we call* $d(e) = endtime(e) - starttime(e) + 1$ *the* duration *of edge* e*. We assume that edge duration* d *follows a distribution* $\mathcal{D}(f, \underline{d}, \overline{d})$*, where* f *is a*

parameter distribution function, $\underline{d}$ and $\overline{d}$ are minimum and maximum edge durations, respectively.

Though IET and duration are recognized as important temporal metrics in networks, they fail to provide guidance of temporal selectivity for query processing. This motivates us to consider the following question: what network characteristics should we capture for temporal selectivity? As a result, we turn to CSS and its distribution, which is defined at the very beginning of this chapter. In the following, we would discuss our findings about CSS and how it is related to temporal subgraph query processing.

3.4.3 Our observation for CSS

Here we carry out our observation and analysis on two real-world networks, **FHV** [57] and **Flight** [58]. **FHV** records the transport trips via for-hired vehicles in New York City, where the time unit is a minute. **Flight** records the trips via airline in the whole US, where the time unit is an hour. Figure 3.1 presents their short-termed (i.e., in a day) and long-termed (i.e., in a week) CSS distribution. Our general findings can be summarized as follows: First, we find that short-termed CSS in different systems is heterogeneous. Specifically, for **FHV**, the shorted-termed CSS distribution can be approximated via a Poisson distribution in the morning and then a normal distribution for the rest of the day. The apparent existence of two peaks (i.e., one in the morning and the other in the afternoon) corresponds to the rush hours in different periods of a day. For **Flight**, however, the short-termed CSS can be modeled via normal distribution. That is, short-termed CSS begins to increase before dawn and reaches its peak in the morning. Then it keeps stable for hours until it begins to decline in the evening. Second, in long-termed CSS, we note an apparent phenomenon of periodicity. Specifically, for most days in **FHV**, though the intensity of fluctuation varies, two CSS peaks can always be found. In **Flight**, the periodicity is more regular and stable.

Besides the general findings above, we further recognize that CSS distributions provide guidance of temporal selectivity in networks for query processing. Specifically, the valleys in the curves demonstrate that the number of overlapping edges are much lower at these timestamps. Considering a temporal subgraph query in which the query window focuses on some of these timestamps, the processing is expected to be more efficient if the temporal selectivity can be leveraged in a smart way. Thus, we recognize the CSS distribution as an important factor that is related to our investigation in this thesis.

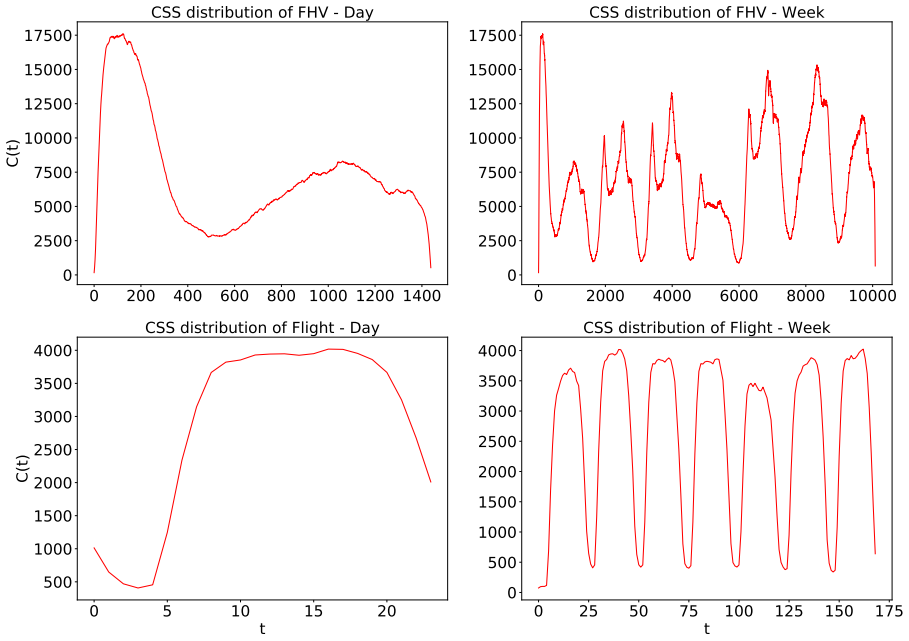


Figure 3.1: CSS distribution of several real networks. Left and right column plots respectively present the daily and weekly CSS distribution.

Unfortunately, to the best of our knowledge, currently there is not benchmark which takes CSS distribution into consideration for network modeling and generation. In the following, we propose the competition-driven model (CDM) to generate the CSS-constrained networks accurately and efficiently.

3.5 Proposed method: competition-driven model

Table 3.3 presents the input parameters used in our model. In CDM, each vertex is associated with a power value $\Pi(v)$ and next active time $nat(v)$. The former determines v's strength in edge generation, while the latter determines its next time to be active. That is, a vertex with higher $\Pi(v)$ has a higher chance to become the source of newly generated edges at time $nat(v)$. Values for $\Pi(v)$ are drawn from a parameter probability distribution f (e.g., the power value distribution) and values for $nat(v)$ are drawn from the IET distribution $\mathcal{I}$.

Based on the above model, the procedure of network generation is shown

Input parameters	
V	Set of vertices
f	Power value distribution
$\mathcal{I}$	Inter-event time distribution
$\mathcal{D}$	Duration distribution
$C(t)$	CSS Distribution

Table 3.3: Overview of the parameters used in the proposed model

in Algorithm 1. The generated graph is outputted in the form of edge stream R_E. That is, each generated edge is represented by a formalized 6-tuple as shown in Definition 2.1.4. For convenience, we use $tuple(e)$ to represent the corresponding tuple of e in R_E. Using generated R_E, general graph implementation (i.e., temporal property graph) and other alternatives (e.g., snapshots sequence) can be easily constructed. The basic idea of network generation can be described as follows: in the whole procedure, we maintain a dedicated *active-list* structure [4] named $Active$ to store the tuples of active edges in real time. Then for each valid time $t \in T$, the size of $Active$ (denoted $|Active|$) is adjusted according to the CSS distribution $C(t)$ at a certain timestamp t. Note that the notation $Active$ would also be used in Chapter 4 and 5 while their implementation can vary for the ease of maintenance in different applications. Here, we sort the stored tuples in $Active$ by their end time in ascending order and define the following two basic operations for maintenance.

- $insTuple(Active, tuple(e))$: insert $tuple(e)$ into $Active$; Return 1 for success and 0 for failure.
- $delTuple(Active, t)$: delete all tuples $tuple(e)$ s.t. $t > e.t_e$ from $Active$; Return the set of deleted edge tuples.

The complexity of $insTuple$ and $delTuple$ is logarithmic in $|Active|$ because the tuples in $Active$ are well-sorted. With the structure, the specific operations to be carried out at any given time t could be determined: when $C(t)$ is smaller than $|Active|$, some of the existing tuples should be forcibly deactivated and removed from $Active$ in order to satisfy $|Active| = C(t)$ constraint. We define one additional operation for $Active$ in order to deal with this situation:

- $PruneTuple(Active, t, m)$: select m tuples, set their end time to t, and delete them from $Active$; Return the collection of deleted tuples.

The procedure of $PruneTuple$ in our work is shown in Algorithm 2. Here,

we apply the end-time-first pruning strategy to prune $Active$. That is, we select the top-m tuples with minimal end time from $Active$, reduce their end time to t, and delete them from $Active$. Various pruning strategies can be used in $PruneActive$. We choose end-time-first-pruning for the following reasons. First, this strategy provides the best efficiency because tuples in $Active$ are sorted by their end time. Second, end-time-first-pruning also helps to preserve the duration distribution in the generated network, which is a desirable network characteristic.

Additionally, a total of $n = C(t) - |Active|$ edges should be generated and inserted into $Active$. The algorithm first collects the set of vertices $\Gamma(t) = \{p_1^t \dots, p_m^t\}$ with $nat(v) \leq t$. We call these vertices in the collection *participants* at current time t [2]. Continuously, the algorithm constructs a probability distribution $\mathcal{S}_t(p)$ by normalizing $\Pi(p_i^t)$ for each $i \in [1, m]$. We call $\mathcal{S}_t(p)$ the competition distribution and it reveals the probability for each participant to "win" in each turn of the coming competition at time t. With the constructed $\mathcal{S}_t(p)$, the algorithm carries out a n-turn competition to generate new edges. In each turn, a participant $p \in \Gamma(t)$ is first selected as the source of link according to $\mathcal{S}_t(p)$. Next, a duration d is generated from duration distribution $\mathcal{D}$, and another vertex v is selected uniformly from the remaining vertices as the destination. In this way, a new edge tuple $(id, p, v, \cdot, t, t + d - 1)$ is created and inserted into $Active$. And if it is the first time for p to win in this turn, the algorithm updates $nat(p)$ to $t + \tau$, where τ is drawn from $\mathcal{I}$ to determine its next time to be active. Similar turns are repeated until n turns have been carried out, which means n new edges have all been created in this competition. Note that if p does not win any turns in the competition, $nat(p)$ is not updated and p would be continuously considered as a participant in the competition at the next timestamp. This way, p's IET is prolonged until it can win at least one turn in a competition.

Though the in-going degree is not considered in Algorithm 1 for simplicity, existing approaches for selecting destinations can be integrated into CDM at small costs. For example, to integrate global popularity, each vertex can be associated with a value of attractiveness from a parameter probability distribution. Then in the generation procedure, destinations can be determined via turns of the same competitions as the determination of sources.

The iterative competitions are repeatedly carried out until $C(t)$ is com-

[2] If there is no $v \in V$ s.t. $nat(v) \leq t$, we collect the set of vertices u such that $nat(u) - t \leq \omega \cdot (t - nat'(u))$ and set each $nat(u)$ to t, where threshold $\omega \in (0, 1.0]$ and $nat'(u)$ is the last active time of u

Algorithm 1: The network generation using CDM

Input: Set of vertices V , power value distribution f, IET distribution $\mathcal{I}$, duration distribution $\mathcal{D}$, CSS distribution $C(t)$

Output: Edge stream R_E

Initialize $\Pi(v)$ and $nat(v)$ for each $v \in V$ by using f and $\mathcal{I}$
$t \leftarrow 1$
$t_{max} \leftarrow \max_{C(t_i) \neq \emptyset} t_i$
$id \leftarrow 0$
while $t \leq t_{max}$ **do**
    $D \leftarrow delTuple(Active, t)$
    $R_E \leftarrow R_E \cup D$
    $n \leftarrow C(t) - |Active|$
    if $n < 0$ **then**
        $D \leftarrow PruneTuple(Active, t-1, -n)$
        $R_E \leftarrow R_E \cup D$
    else
        Collect the participants set $\Gamma(t) = \{p_1^t \ldots, p_m^t\}$
        while $i \in [1, m]$ **do**
            $\mathcal{S}_t(p_i^t) \leftarrow \Pi(p_i^t) / \sum_{j=1}^{m} \Pi(p_j^t)$
        while $n > 0$ **do**
            Draw a participant p from $\mathcal{S}_t$ as source.
            if it is the first time for p to be drawn in this turn **then**
                Draw an IET τ from $\mathcal{I}$
                $nat(p) \leftarrow t + \tau$
            Draw a duration d from $\mathcal{D}$.
            Draw a destination v from $V - \{p\}$
            $insTuple(Active, (e, p, v, \cdot, t, t + d - 1))$
            $n \leftarrow n - 1$
    $t \leftarrow t + 1$
$R_E \leftarrow R_E \cup Active$
return E

pletely traversed in time. The complexity of the generation algorithm is $O(\hat{T} \cdot |V| + |E| \cdot \log |E|)$. Note that $|E|$ is determined by R_E instead of an input parameter in the CDM. That is, its exact value can only be known when the network is completely generated. Similarly, for each vertex $v \in V$,

Algorithm 2: PruneTuple

Input: active list $Active$, timestamp t, number of pruned tuple n
Output: the set of pruned tuple D

$D \leftarrow \emptyset$
while $n > 0$ **do**
  $r \leftarrow$ the first tuple in $Active$
  $D \leftarrow D \cup \{r\}$
  $Active \leftarrow Active - \{r\}$
  $n \leftarrow n - 1$
return D

relative degree $A(v)$ is also known after the generation since they depend on $|E|$. In the following section, we present the theoretical analysis of how values for $|E|$ and $A(v)$ in the produced networks are influenced by the generation algorithm.

3.6 Theoretical analysis

Two natural questions about the CDM are: (1) As the number of edges $|E|$ is not an input parameter to the algorithm, what is the expected cardinality for the generated network? (2) Similarly, what would the relative degree $A(v)$ be like? Answers to these questions respectively help to evaluate the necessary storage cost for generation and investigate the structural characteristics of the generated network. In this section, we provide an analysis to answer these two questions. For ease of analysis, we make the assumption that the activity behavior of both $v \in V$ and $e \in E$ follow the Poisson process and $\mathcal{I}, \mathcal{D}$ follow exponential distributions with λ_1, λ_2 parameters, respectively. Besides, we assume each participant in a competition can win at least one turn so that their IETs are not prolonged and follow $\mathcal{I}$ strictly.

3.6.1 Cardinality

For $t \in T$, let $O(t)$ demonstrate the number of edges that should be generated at timestamp t. The equation to describe the relation between network

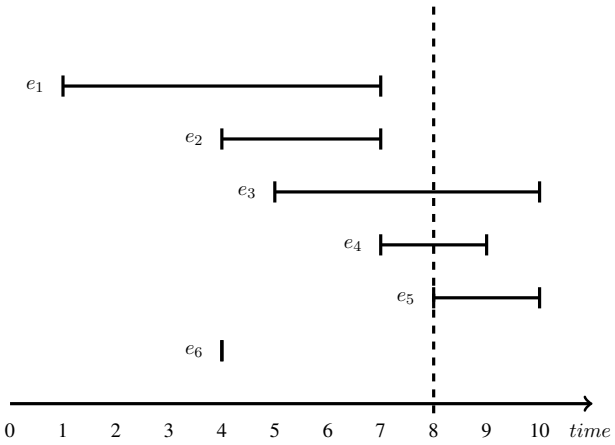


Figure 3.2: Example of an edge generation by the CDM. Dashed line corresponds to the $C(8) = 3$.

cardinality $|E|$ and $O(t)$ could be written down as follows:

$$|E| = \sum_{t=1}^{\hat{T}} O(t) \tag{3.1}$$

Let $R(t)$ demonstrate the number of remaining edges at time t after $delTuple$ is invoked. The equation to describe $O(t)$ is as follows:

$$O(t) = \begin{cases} C(t) - R(t) & C(t) > R(t) \\ 0 & C(t) \leq R(t) \end{cases} \tag{3.2}$$

That is, given timestamp t, $O(t)$ merely contributes to the cardinality when $C(t) > R(t)$. For example, Figure 3.2 presents a collection of edges generated using the CDM and $C(t) = \{1 : 1, \ldots, 3 : 1, \ldots, 5 : 3, 6 : 3, 7 : 4, 8 : 3\}$. Each edge is represented by its interval. Consider the edge generation at $t = 7$ is completed and we are going to generate the collection of edges at $t = 8$. $Active$ at $t = 7$ contains the edges e_1, e_2, e_3, e_4. Then $delTuple$ deletes e_1, e_2 from $Active$ since they both end at $t = 8$. In this way, only two edges e_3, e_4 survive in $Active$ after the edge deletion, hence $R(8) = 2$. Since $C(8) = 3$ and Equation 3.2 gives $O(8) = C(8) - R(8) = 1$, this means that a single edge needs to be generated at $t = 8$.

As cardinality analysis is generally used for network storage and construction time estimation, here we use the worst-case method to estimate the output of Equation 3.1. In this worst case, we assume that $R(t)$ is always smaller than $C(t)$. Then, the worst-case equation for the maximum number of generated edges $|E|_m$ is described as follows:

$$|E|_m = \sum_{t=1}^{T} (C(t) - R(t)) \tag{3.3}$$

From Algorithm 1, we know that $R(t)$ consists of the set of edges active at both $t-1$ and t. Then, $R(t)$ can be computed as follows:

$$R(t) = C(t-1) \cdot (1 - P_d(t)), \tag{3.4}$$

where $P_d(t)$ [3] represents the probability for each $e \in E(t-1)$ to end at time t. In our example, we can estimate that $P_d(8)$ is approximately 0.5 since there is $C(7) = 4$ and $R(8) = 2$. Here, we use the knowledge of the stochastic process to make further deduction on $P_d(t)$. Considering a probability event ϵ, Poisson process uses the following equation to express and compute the probability that ϵ *happens* k *times in duration* $[t, t+\tau]$:

$$P[N(t+\tau) - N(t) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \tag{3.5}$$

Note that $P_d(t)$ can be also described as the probability that *an edge active at* $t-1$ *is going to end at time* t. Based on our assumed Poisson process for e and exponential distribution for $\mathcal{D}$, $P_d(t)$ can be transformed into the following:

$$P_d(t) = P[N(t) - N(t-1) = 1] = \lambda_2 e^{-\lambda_2} \tag{3.6}$$

By substituting Equation 3.6, 3.4 into 3.3, we could obtain the following equation of describe the expected cardinality for synthetic network.

$$|E|_m = C(\hat{T}) + \sum_{t=1}^{\hat{T}-1} C(t) \cdot \lambda_2 e^{-\lambda_2} \tag{3.7}$$

With this equation, the complexity of CDM becomes more intuitive. Also, maximal memory cost in network generation can be evaluated.

[3] Since edges share the same $\mathcal{D}$ in the CDM, the ending probability is the same for $e \in E$.

3.6.2 Relative degree

Next, we give the derivation of the relative degree $A(v)$. The equation to describe $A(v)$ is as follows:

$$A(v) = \frac{\sum_{t=1}^{\hat{T}} o(v,t)}{|E|} \tag{3.8}$$

where $o(v,t)$ is the number of generated outgoing edges starting from v at time t. The value of $o(v,t)$ relies on whether v is active at t. Based on our assumption and letting $P_a(t)$ [4] be the probability for $v \in V$ to be active at t, the equation is as follows:

$$P_a(t) = P[N(t) - N(t-1) = 1] = \lambda_1 e^{-\lambda_1} \tag{3.9}$$

In this way, the equation to describe $o(v,t)$ is as follows:

$$o(v,t) = \begin{cases} 0 & \text{with probability } p = 1 - \lambda_1 e^{-\lambda_1} \\ \mathcal{S}_t(v) \cdot O(t) & \text{with probability } p = \lambda_1 e^{-\lambda_1} \end{cases} \tag{3.10}$$

According to Algorithm 1, $\mathcal{S}_t(v)$ could be computed as follows:

$$\mathcal{S}_t(v) = (\Pi(v) / \sum_{i=1}^{|\Gamma(t)|} \Pi(p_i^t)) \tag{3.11}$$

By substituting Equation 3.11 into 3.10, we could obtain:

$$o(v,t) = \begin{cases} 0 & \text{with } p = 1 - \lambda_1 e^{-\lambda_1} \\ \frac{\Pi(v) \cdot O(t)}{\sum_{i=1}^{|\Gamma(t)|} \Pi(p_i^t)} & \text{with } p = \lambda_1 e^{-\lambda_1} \end{cases} \tag{3.12}$$

The combination of Equations 3.8 and 3.12 reveals that in order to analyze $A(v)$, we only need to concentrate on the $o(v,t)$ in which v is active at timestamp t. We use $B(v) = \{b(v,1), \ldots, b(v,j), \ldots\}$ to demonstrate the collection of v's active timestamps $b(v,j)$ represent the jth active time of v. Equation 3.8 could be simplified into following format:

$$A(v) = \frac{\sum_{j=1}^{|B(v)|} o(v, b(v,j))}{|E|} \tag{3.13}$$

[4] Since vertices share the same $\mathcal{I}$ in the CDM, the active probability is the same for all $v \in V$.

Aligning Equations 3.13 with 3.12, we could obtain the following equation which illustrates the factors impacting $A(v)$:

$$A(v) = \frac{1}{|E|} \cdot \sum_{j=1}^{|B(v)|} \frac{\Pi(v) \cdot O(b(v,j))}{\sum_{i=1}^{|\Gamma(b(v,j))|} \Pi(p_i^{b(v,j)})} \tag{3.14}$$

Equation 3.14 reveals that the relative degree of vertex v is influenced by following factors:

- $|B(v)|$, the number of timestamps when v is active (i.e., the number of competitions v participated). The larger $|B(v)|$ provides more opportunities for v to earn outgoing edges.
- $|\Gamma(t)|$, the number of participants in competition at time t. The larger $|\Gamma(t)|$ tends to weaken $\mathcal{S}_t(v)$, which in turn leads to less outgoing edges from v.
- $\Pi(v)$, the power value of v. The larger $\Pi(v)$ tends to enhance $\mathcal{S}_t(v)$, which in turn leads to more outgoing edges from v.
- $O(t)$, the number of generated edges at time t. The larger $O(t)$ leads to more outgoing edges from v when $\mathcal{S}_t(v)$ is fixed.

In order to mine more underlying factors on $A(v)$, we introduce the mean-field method to simplify the variables in the model and regard the inferred result as the benchmark. Let $\overline{A(v)}$ be the mean relative degree of vertex v. The equation to describe the mean field is as follows:

$$\overline{A(v)} = \frac{1}{|E|} \cdot \sum_{j=1}^{\overline{B}} \frac{\Pi(v) \cdot \overline{O}}{\sum_{i=1}^{\overline{\Gamma}} \Pi(p_i^{b(v,j)})} = \frac{\overline{O}}{|E|} \cdot \sum_{j=1}^{\overline{B}} \frac{\Pi(v)}{\sum_{i=1}^{\overline{\Gamma}} \Pi(p_i^{b(v,j)})} \tag{3.15}$$

where $\overline{B}$ is the mean number of competitions v participated. $\overline{\Gamma}$ is the mean number of participants at time t. $\overline{O}$ is the mean number of edges that should be generated at time t. Corresponding equations to describe these mean-field parameters are as follows:

$$\overline{B} = E[B(v)] = \lambda_1 e^{-\lambda_1} \hat{T} \tag{3.16}$$

$$\overline{\Gamma} = E[|\Gamma(t)|] = \lambda_1 e^{-\lambda_1} |V| \tag{3.17}$$

$$\overline{O} = |E|/\hat{T} \tag{3.18}$$

By substituting equation 3.16, 3.17, 3.18 into 3.15. The mean degree equation could be transformed as follows:

$$\overline{A(v)} = \frac{1}{\hat{T}} \cdot \sum_{j=1}^{\lambda_1 e^{-\lambda_1 \cdot \hat{T}}} \frac{\Pi(v)}{\sum_{i=1}^{\lambda_1 e^{-\lambda_1 \cdot |V|}} \Pi(p_i^{b(v,j)})} \tag{3.19}$$

Equation 3.19 reveals the two characteristics of the relative degree in our model: first, as the number of vertices $|V|$ increases, the relative degree of each vertex will drop because $|V|$ determines the sum of cumulative adding in denominator. Second, given the number of vertices $|V|$, more involved participants make the distribution of $A(v)$ much closer to $\Pi(v)$ as it makes $\sum_{i=1}^{|\Gamma(t)|} \Pi(p_i^{b(v,j)})$ closer to 1. That is, relative degree $A(v)$ is exactly reflected by $\Pi(v)$ in the most ideal situation. The larger vertices set size makes $A(v)$ closer to $\Pi(v)$.

3.7 Experiments

In this section, we present our experimental investigation for the CDM. We aim to answer the following questions. First, we would like to know if CDM could simulate real networks with both structural and temporal characteristics preserved. Second, we investigate to what extent various graph configuration parameters influence the synthetic networks generated by the CDM.

3.7.1 Setup

Environment. Our experiments were carried out on a server with 192GB RAM and 2 Intel(R) Xeon(R) CPU X5670 with 6 cores at 2.93GHz running a Linux operating system. We implemented the in-memory versions of the CDM in C++.

Datasets. We consider four real networks in the transportation domain: **Yellow** [57], **FHV** [57], **HVFHV** [57], and **Flight** [58]. **Yellow** records the trips on the yellow taxi in New York City on 2 January 2018 and each trip is labeled with an interval to represent its duration. **FHV** records the trips on for hired vehicles in New York City on 1 January 2018. **HVFHV** records the trips on high-volume for hired vehicles on 1 June 2019. **Flight** records the trips on airlines in the US in January 2019. An overview of statistics of the four networks is given in Table 3.4 and their CSS distributions are shown in Figure 3.3.

Name	$\|V\|$	$\|E\|$	$\hat{T}$
Yellow	253	236,522	1,440 minutes
FHV	261	585,691	1,440 minutes
HVFHV	260	823,629	1,440 minutes
Flight	335	566,942	744 hours

Table 3.4: Overview of the real-world networks used in the experiments.

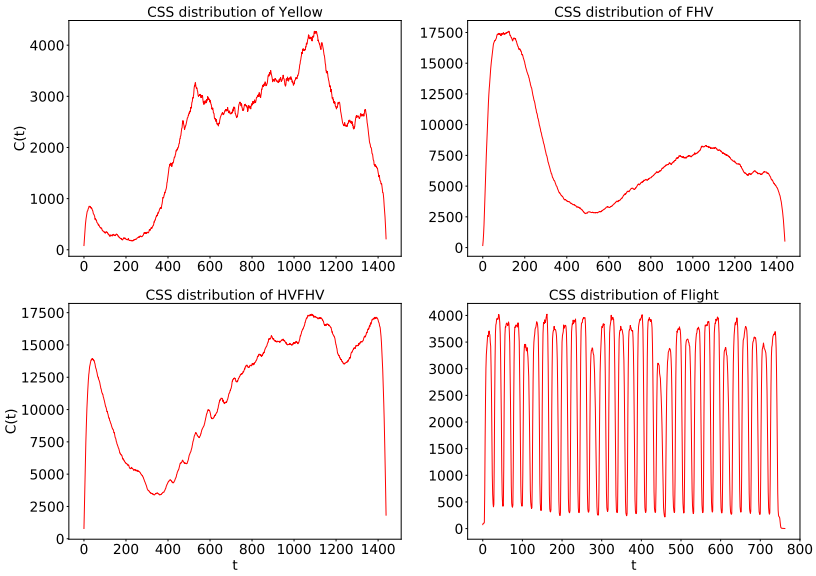


Figure 3.3: CSS distribution of the real-world networks used in the experiments.

Experiments. We run two categories of experiments to investigate the performance of CDM. The first category of experiments deals with the *quality* of network simulation by the CDM. We investigate real networks' relative degree, IET, duration, and CSS distribution to obtain a graph configuration to be used in network generation. Specifically, we obtain graph configurations in two ways. The first method is called the *frequency* configuration, in which statistical estimations of real measures are directly used as input schema. The second method is called the *fitted* configuration, in which for IET and duration distributions are used directly as estimated from real networks and we use the fitted result for $\mathcal{I}$ and $\mathcal{D}$ distributions. We use the power-law cut-off model $y = \beta \cdot \tau^{\alpha} \cdot e^{-\frac{\tau}{\tau_c}} + h$. The values used in frequency and fitted configuration for each network are shown in Figure 3.4. The parameter β is the CSS coefficient

which represents the times that basic CSS value is enlarged.

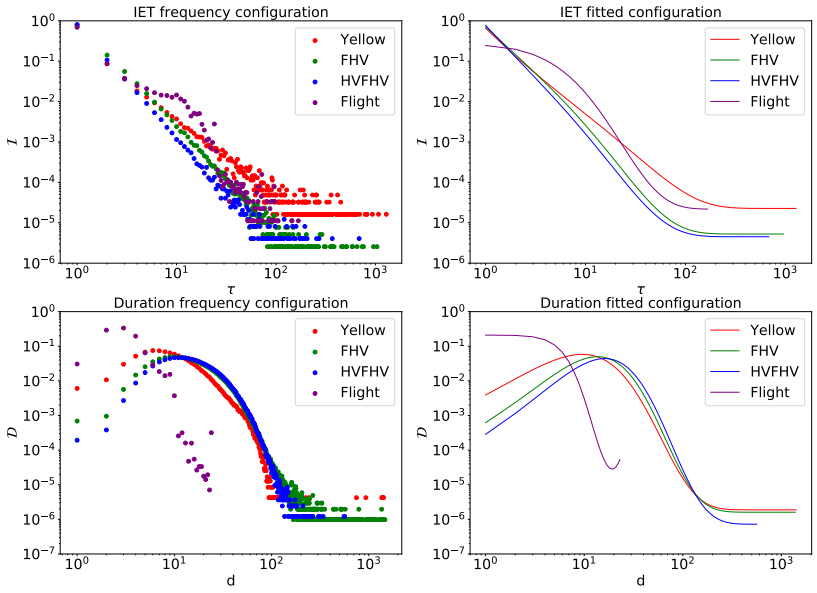


Figure 3.4: IET and duration values used in frequency and fitted configuration.

The second category of experiments investigates the *scalability* of the CDM. We use instances with two types of CSS: (1) the linear $C(t) \sim t$ and (2) the Gaussian $C(t) \sim N(702, 180.0^2)$, to investigate the CDM performance in both monotonous increasing and non-monotonous CSS respectively. The former case aims to investigate the CDM performance in monotonic increasing CSS and the later case aims to investigate the CDM performance in non-monotonic CSS. The default setup for the remainder of the configuration is shown in Table 3.5. These configuration parameters are either popularly used in existing benchmark for network modeling and generation [59] or supposed to impact the underlying structures in networks. To investigate such impact in various networks generated by using CDM, we vary the configuration parameters as follows. (1) We set $|V|$ in $[500, 750, 1000, 1250, 1500]$ to investigate the vertex cardinality impact in CDM. By setting various $|V|$, we obtain the networks involving either more or less entities. (2) We set CSS coefficient β in $[1, 5, 50, 500, 5000] \times 10^7$ to investigate the CSS coefficient impact. By setting various β, we obtain the networks with either higher or lower CSS value at each timestamp. (3) We set $|E|$ in $[20, 40, 60, 80, 100]$ million to investigate the edges cardinality impact[5]. By setting various $|E|$, we generate either small

[5]Every time $C(t)$ is consumed, we return to $C(0)$ and continue the generation iteration,

or large networks. (4) We set $\overline{\tau}$ in $[1, 10, 100, 1000, 10000]$ to investigate the IET impact. By setting various $\overline{\tau}$, the intensity of IET heavy tail (i.e., the existence of long IETs) in generated graph can be controlled. (5) We set $\overline{d}$ in $[1, 10, 100, 1000, 10000]$ to investigate the duration impact. By setting various $\overline{d}$, the existence of lasting edges can be controlled.

We use six measures to evaluate the result: the distribution of (1) network generation time, (2) relative degree, (3) closeness, (4) IET, (5) duration, and (6) stability [24]. Given a vertex $v \in V$, the *closeness* is a measure of how close v is to any other vertices in the network. The measure is computed as the inverse of the average distance from v to any other vertices in the network, which is shown as follows:

$$closeness(v) = \frac{|V| - 1}{\sum_{u \in V - \{v\}} dist(v, u)} \tag{3.20}$$

where $dist(v, u)$ represents the minimal distance (i.e., number of hops) from vertex v to u. The *stability* is a summary of v's evolving degree structure in time, which is measured based on the notion of *degree rank*. Given the set of snapshots $\{G(1) \dots G(\hat{T})\}$, the *degree rank* of v in snapshot $G(t)$ is computed as follows.

$$rank(t, v) = \frac{\delta_t(v)}{\max_{u \in V} \delta_t(u)} \tag{3.21}$$

where $\delta_t(v)$ is the number of out-going edges from v in $G(t)$. With these notions, given the set of snapshots $\{G(1) \dots G(\hat{T})\}$ the *stability* of v is computed as follows.

$$stability(v, \{G(1) \dots G(\hat{T})\}) = 1 - 2\sigma_v \tag{3.22}$$

where σ_v is the standard deviation of v's degree rank over all snapshots. As a trade-off between measuring accuracy and efficiency, for each network in this experiment, we compute the *stability* based on 1000 uniformly selected snapshots.

until the desired cardinality is reached.

Schema	Value	Description
$\|V\|$	500	Number of vertices
$\|E\|$	200,000,000	Number of edges
f	$\sim x^{-1.5}$	Power value distribution
$\mathcal{I}$	$\sim \tau^{-1.5}, \underline{\tau} = 1, \overline{\tau} = 1000$	IET distribution
$\mathcal{D}$	$\sim d^{-1.5}, \underline{d} = 1, \overline{d} = 1000$	Duration distribution
β	50×10^7	Coefficient in $C(t)$

Table 3.5: Default configuration for the scalability experiments.

3.7.2 Results and Analysis

Quality of network simulation. Table 3.6 reports the generation time for the two types of simulations (frequency and fitted configurations) for different networks. We note that even the largest network with around 800K edges could be constructed in less than 15 seconds. This indicates that CDM is highly efficient in network generation. Figures 3.5, 3.6, 3.7, 3.8, 3.9 report the measures in the real, frequency-simulation, and fitting-simulation networks. We note that in each subplot, the trend of different curve is similar to each other and the differences are minimal. This indicates that CDM could simulate real networks well.

Name	$\|E\|$	construction cost (ms)
FHV-frequency	563,243	12176.3
FHV-fitting	533,572	12574.7
Yellow-frequency	237,275	7859.7
Yellow-fitting	199,298	8770.9
HVFHV-frequency	826,523	14980.0
HVFHV-fitting	755,707	14714.8
Flight-frequency	569,483	10717.4
Flight-fitting	583,964	11266.1

Table 3.6: Generation time of simulation result

Scalability of network generation. Next, we investigate the performance of CDM in different categories of networks. Table 3.7 reports the construction time of various networks in CDM and following results could be drawn from it. First, by varying β, the construction time in the monotonic increases steadily.

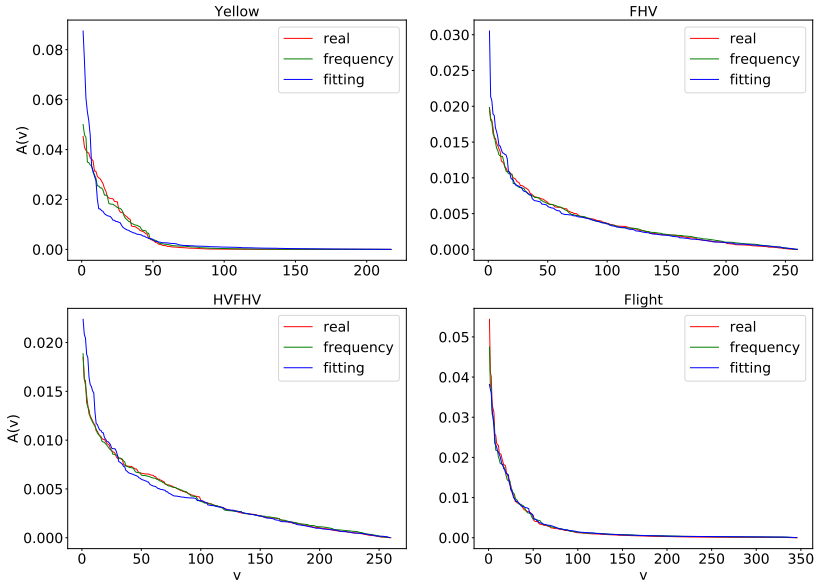


Figure 3.5: Relative degree of simulation result.

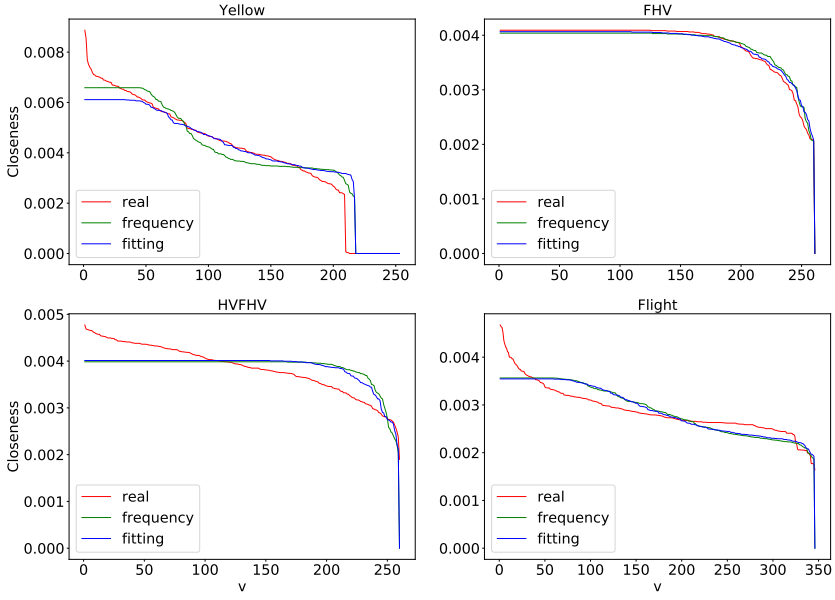


Figure 3.6: Closeness distribution of simulation result.

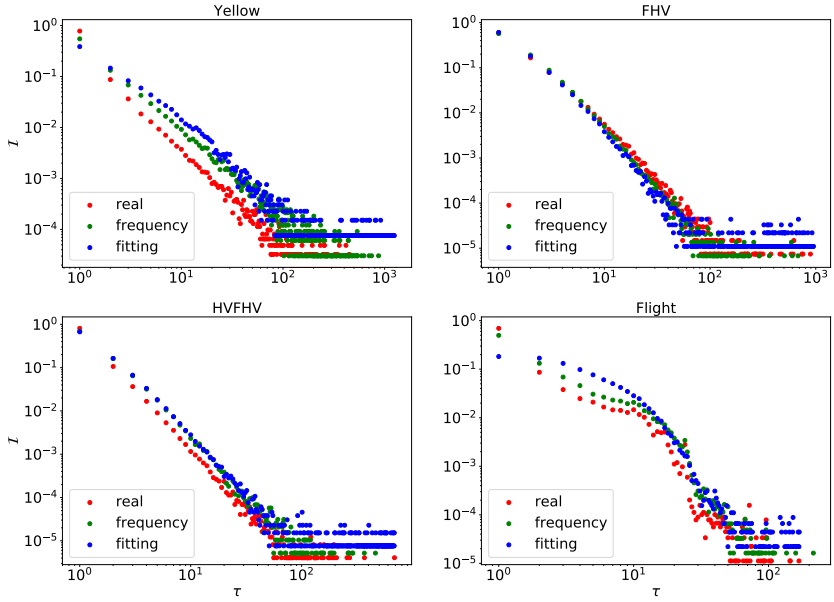


Figure 3.7: IET distribution of simulation result.

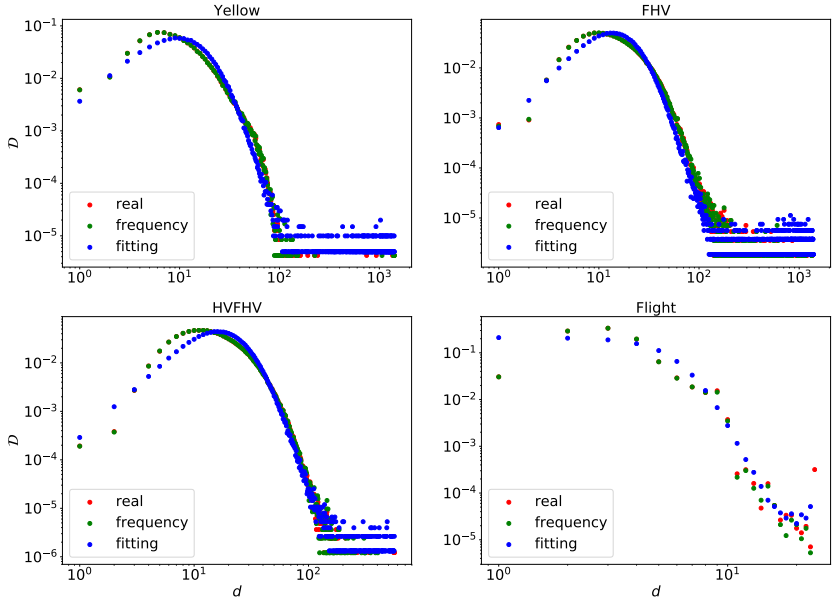


Figure 3.8: Duration distribution of simulation result.

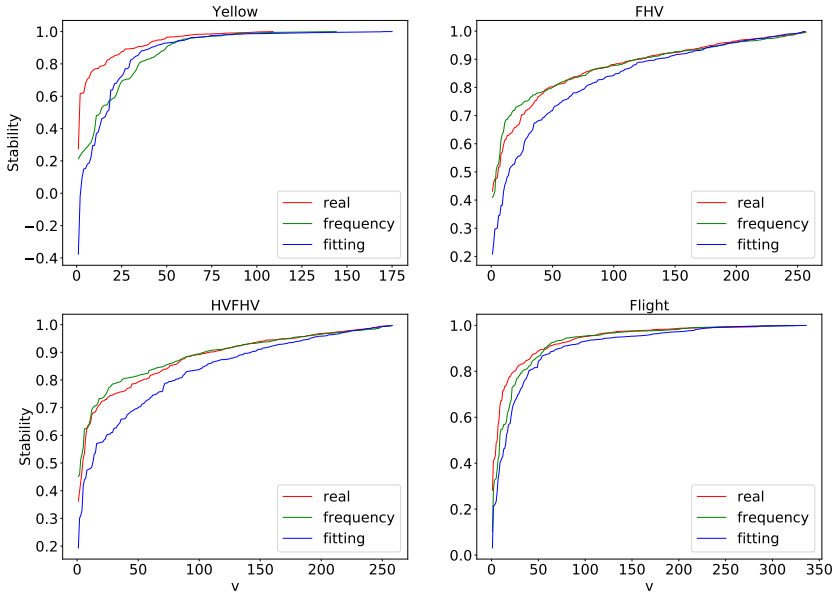


Figure 3.9: Stability distribution of simulation result.

This is expected because higher β leads to larger $|Active|$ in each competition so that the insert of a newly generated edge becomes more costly. In the non-monotonic, however, the construction time in the non-monotonic sharply decreases at the very beginning and then increases steadily. This is because the low β in the non-monotonic significantly increases the times of invoking $PruneTuple$, which is unproductive to the generation of new edges. Second, by varying $|V|$, the construction time keeps increasing. This is expected because higher $|V|$ generally leads to more participants in a competition which further makes each turn more costly. Third, by varying the desired $|E|$, the construction time steadily increases because more competitions are carried out to generate larger networks. Fourth, by varying $\overline{\tau}$, the construction time steadily decreases because long IET significantly reduces the number of participants in a competition. Finally, by varying $\overline{d}$, the construction time slightly increases because more lasting edges increases the maintenance cost of $Active$. Overall, the statistics demonstrate that CDM could generate both small and large networks efficiently.

Next, we concentrate on the remaining structural and temporal measures. Figure 3.10 reports the degree distribution in various networks. Several observations can be made here. First, we note that higher $|V|$ pulls down the degree proportion of each vertex, which is expected in Equation 3.19. Second, the

$\beta \times 10^{-7}$	1	5	50	500	5000
t_m	865.5	868.7	889.2	892.7	908.6
t_n	2251.8	1010.6	833.6	870.7	921.2
$\|V\|$	500	750	1000	1250	1500
t_m	889.2	887.9	908.6	925.7	920.5
t_n	833.5	867.8	908.6	925.7	948.9
$\|E\|$	200M	400M	600M	800M	1000M
t_m	889.2	1735.4	2680.1	3573.1	4465.7
t_n	833.5	1671.0	2539.8	3390.2	4226.2
$\overline{\tau}$	1	10	100	1000	10000
t_m	942.2	914.0	878.1	889.2	834.5
t_n	967.0	921.2	873.6	833.5	804.0
$\overline{d}$	1	10	100	1000	10000
t_m	821.4	810.3	832.2	889.2	881.4
t_n	804.1	819.2	801.0	833.5	840.1

Table 3.7: Generation time in both monotonic (denoted t_m) and non-monotonic (denoted t_n) networks with respect to schemas (secs)

higher β lifts the front of the distribution curve while the tail still keeps stable. This is because a higher coefficient value provides more opportunities for higher-power vertices to obtain outgoing edges in each competition so that the high-power vertices could fully take their advantage in their involved competitions. Third, higher $\overline{\tau}$ pulls down the front and lifts the rest of the curve. This is because the appearing of longer inactive period allows lower-power vertices to participate in more competitions without competing with higher-power vertices. Finally, higher $\overline{d}$ pulls down the front part because lasting edges lead to a limited number of edges to be generated at each timestamp. This restricts the degree advantage of high-power vertices.

Figure 3.11 reports the IET distribution in various networks. The dashed line is there to illustrate the "ends" of the lines, they cannot be seen otherwise because of the significant overlap between the lines that correspond to different studied parameters. We start by drawing two general conclusions about the IET results. First, the configured $\mathcal{I}$ is well modeled in networks generated

by the CDM since we can observe the heavy-tails in power-law distribution. Second, the networks with a higher proportion of small IETs tend to have smaller maximal IET. For convenience, we call this the *IET aggregation nature* in the generated networks. Third, the maximal IET in a generated network can be larger than configured $\overline{\tau}$. This is because vertices with lower $\Pi(v)$ may never win in a competition so their IET would be continuously prolonged.

The rest of the results observed in Figure 3.11 include the following. First, IET aggregation of the non-monotonic networks tends to be weaker than the monotonic networks. This is because the generation of non-monotonic networks involves the invoking of $PruneTuple$, which introduces the period with no competitions and extends vertices' IETs. Second, higher β enhances the IET aggregation because this provides more opportunities for lower-power vertices to win in competitions. Third, higher $\overline{\tau}$ weakens IET aggregation as expected. Finally, higher $\overline{d}$ also weakens the IET aggregation because lasting edges reduce the opportunities for lower-power vertices to win in competitions.

Figure 3.12 reports the duration distribution in various networks. We first note that the configured $\mathcal{D}$ is also modeled well because of the observed heavy-tails. Second, higher $\overline{d}$ pulls down the durations' aggregation on small values for the similar reason as in IET. Besides, duration proportion in networks generated by the CDM tends to be much more stable since they are hardly impacted by other factors (in comparison to the IET).

Finally, we present the result and analysis of vertex stability in various networks. A general situation drawn from the resulted statistics is that low-power vertices are generally more stable than higher-power vertices. This is expected because the temporal degree of the low-power is generally small or even negligible comparing to the global maximal degree. To be more specific, considering a vertex $v \in V$, the global maximal temporal degree might probably vary in v's inactive period. Since small $\Pi(v)$ generally leads to small degree, lower-power vertices tend to be less sensitive to the variation of the global maximal degree. Oppositely, higher $\Pi(v)$ generally leads to in-negligible degree. This makes high-power vertices much more sensitive to the variation.

Figure 3.13 reports the vertex stability in various networks and several results could also be drawn from it: first, the higher $|V|$ lifts the stability curve in both monotonic and non-monotonic networks. It is because the higher $|V|$ leads to the lower degree distribution so that a batch of higher-power vertices become less sensitive and more stable. Second, the higher β lifts the stability curve in both categories because a higher CSS coefficient leads to the increase of the maximal and a higher proportion of stable vertices. Third, higher $\overline{\tau}$ pulls

down the curve in both categories because vertices' longer in-active period can intensify the variation of the maximal. Fourth, higher $\overline{d}$ pulls down the curve in both categories because lasting edges can increase vertices' temporal degree and intensify the variation of the maximal. Finally, we note that vertices in the non-monotonic are less stable than that in the monotonic when the rest of the configuration is the same. This demonstrates that $PruneActive$, which mainly considers the generation efficiency and duration distribution in this paper, weakens the stability of vertices in generated networks. So in the future, we would consider various methods used in $PruneTuple$ and investigate their influence on stability.

3.8 Chapter summary

Motivated by the desire to understand real-world networks, in this chapter, we investigate the modeling problem of temporal networks. We first present the important characteristics of real-world networks via both existing literature and empirical observation. Particularly, we focus on CSS, a characteristic which is related to our topic but hardly investigated in prior works. Then, we propose CDM for the modeling and generation of CSS-constrained networks. The theoretical analysis and experimental evaluation demonstrate that CDM results in a controllable benchmark which can simulate real networks well and generate synthetic networks efficiently.

By investigating temporal network modeling, we partially build up the theoretical basis for our remained research questions, which provides us a better understanding of real-world networks. In the rest of the thesis, we will focus on the query processing problem. Our enhanced understanding will provide us guidance of query selectivity leverage in order to reach better processing efficiency.

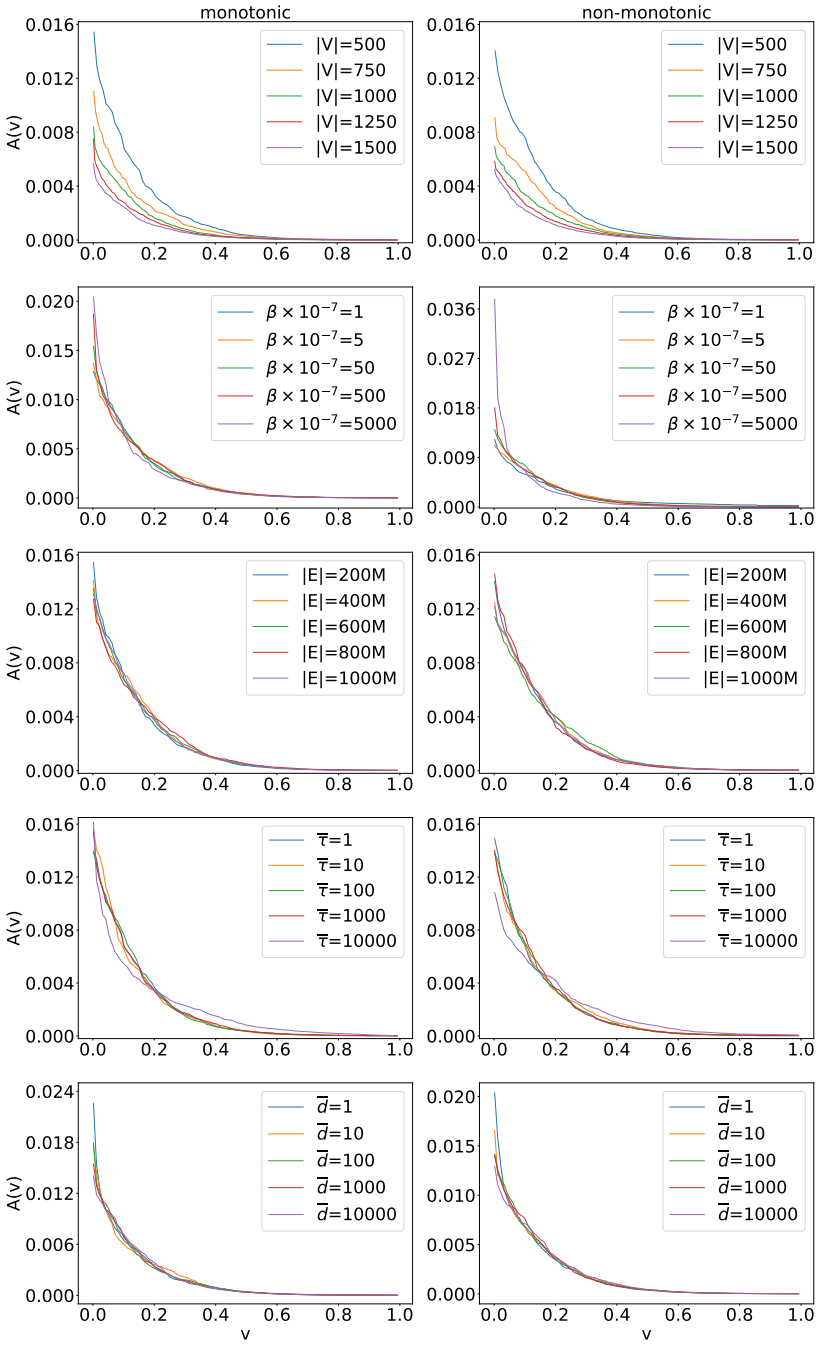


Figure 3.10: Relative degree in various networks.

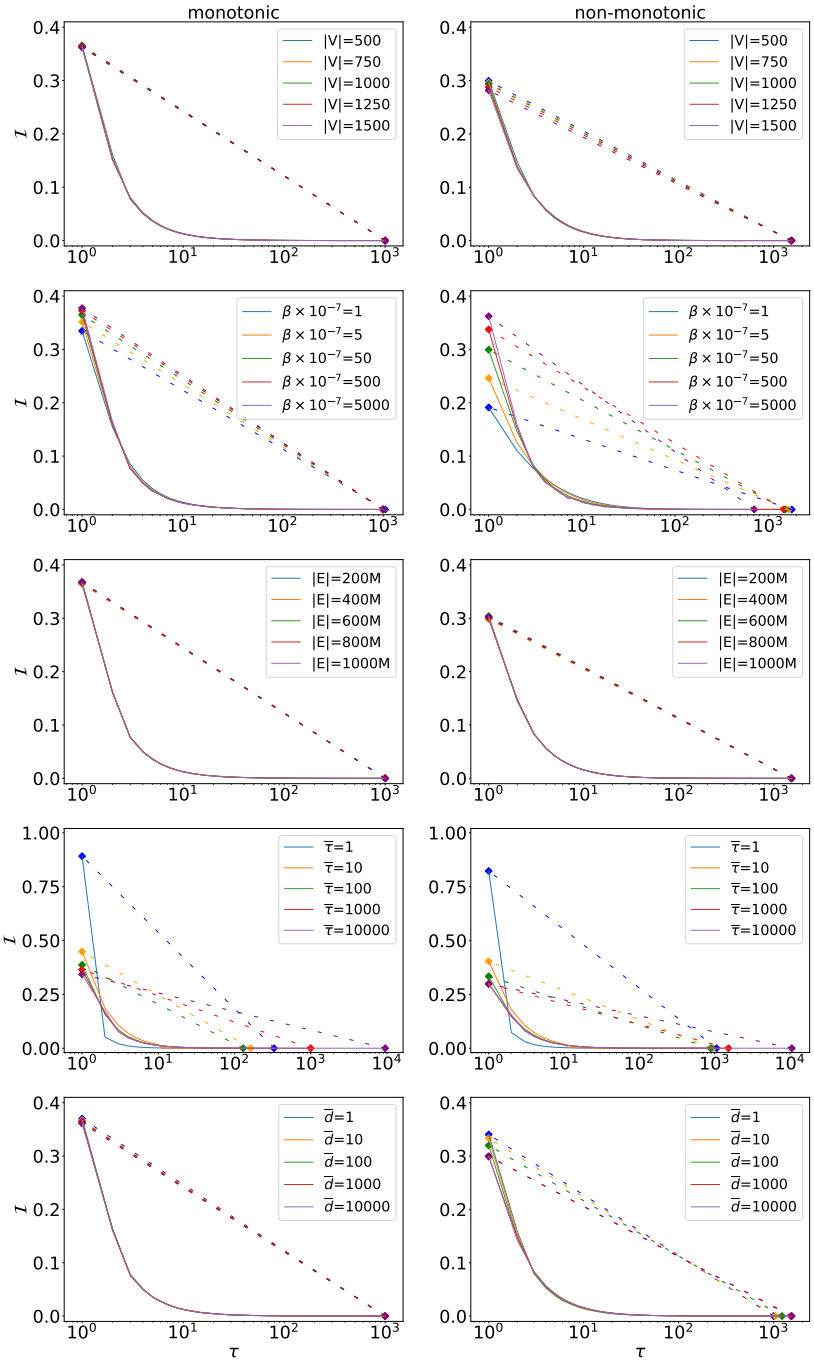


Figure 3.11: IET distribution in various networks.

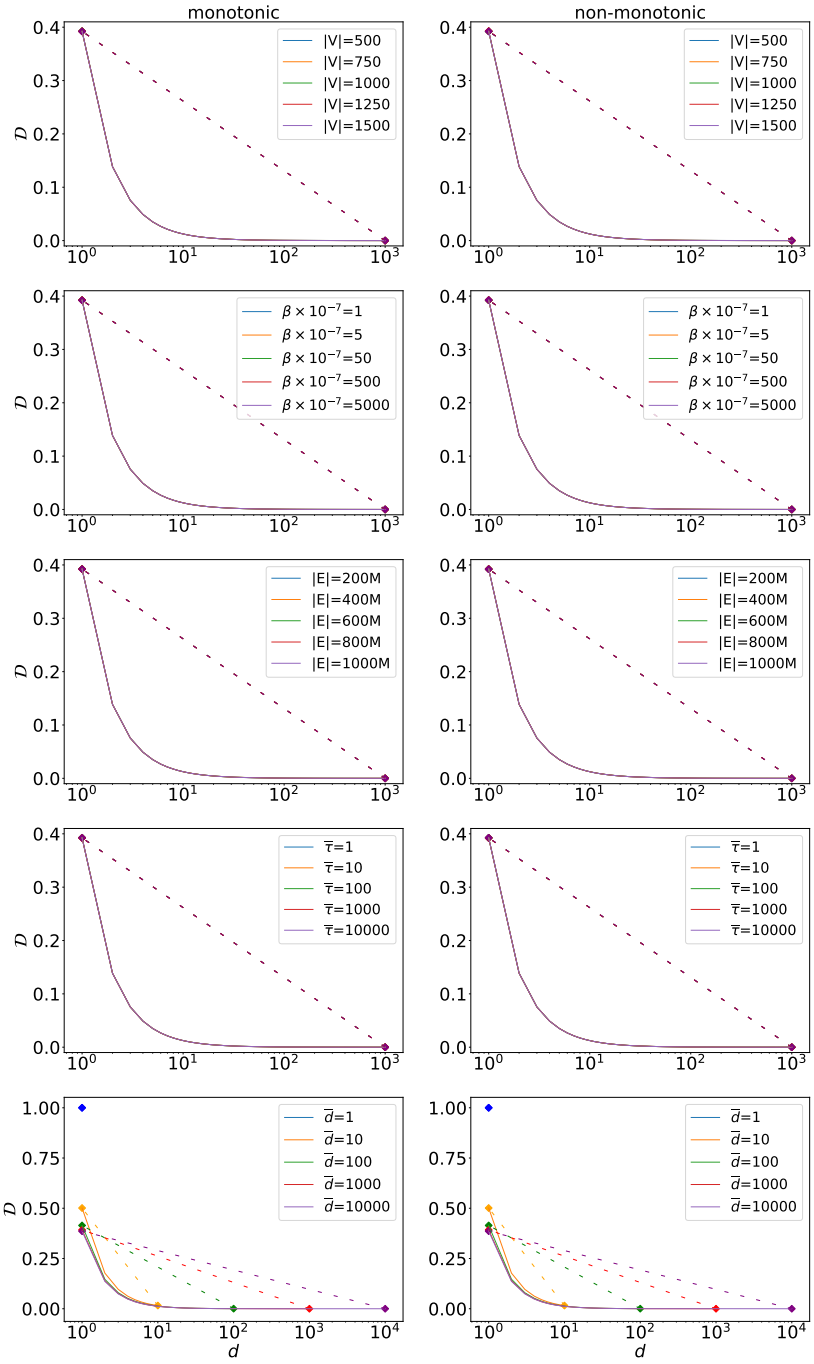


Figure 3.12: Duration distribution in various networks.

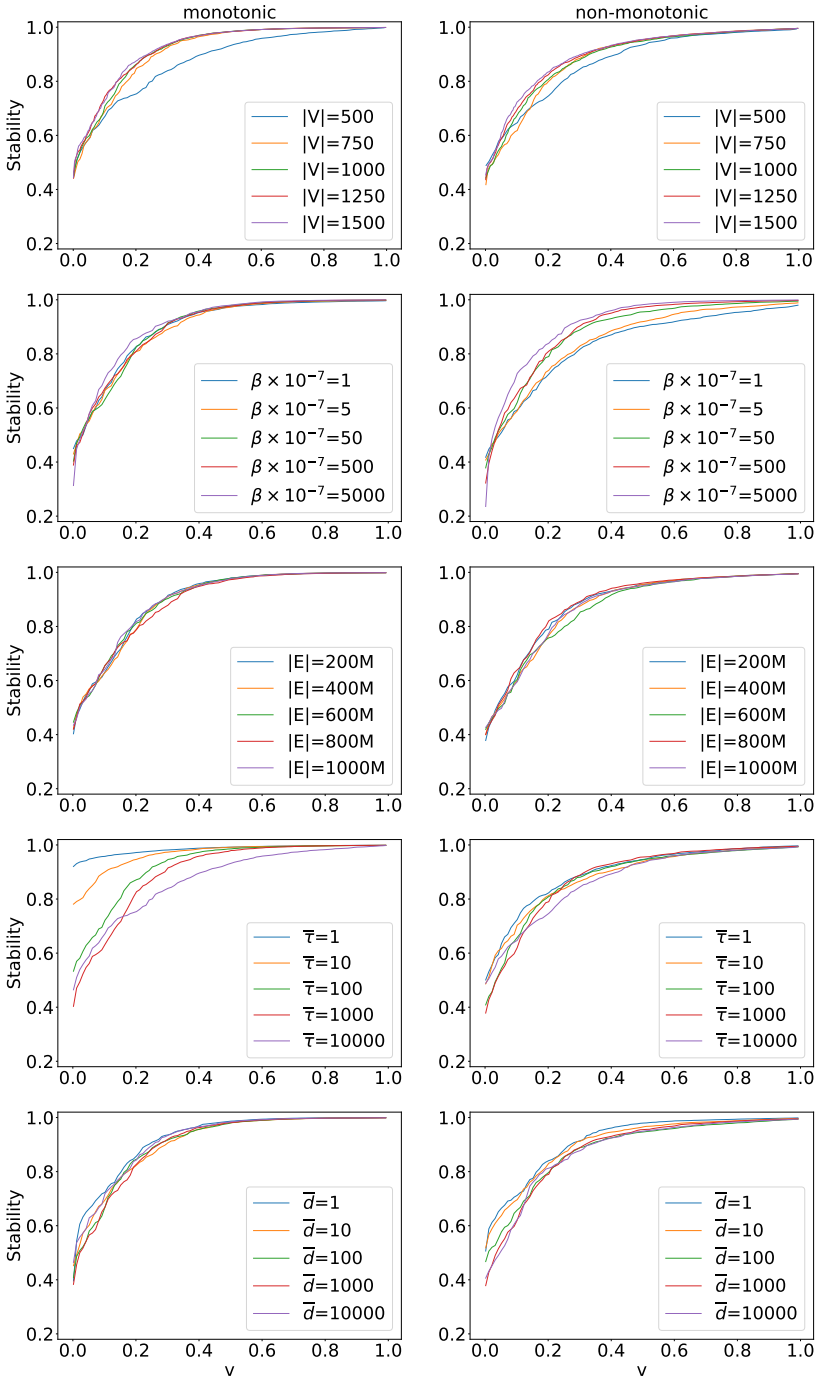


Figure 3.13: Vertex stability in various networks.

4

Processing of temporal predicates

4.1 Motivation

With a better understanding of temporal networks, we enter the processing problem of temporal predicates, i.e., temporal k-clique enumeration: Given a (1) temporal relation R (2) a query time window $[q_s, q_e]$; and (3) a positive integer k, enumerate all $S \subseteq R$ where S is a temporal k-clique in $[q_s, q_e]$. That is, objects in each S are all mutually overlapping at some time point in the query window. The investigated problem arises in a wide range of applications. Some illustrative examples follow.

- In case of disease eruption and its transmission in a community, find all groups of k people whose infectious periods all pairwise overlap in a given timeframe.
- For a deeper understanding of scientific collaborations in a bibliographic database, find all groups of k people who have tightly collaborated with each other at the same time, in a given time period.
- For calling a meeting, given the availability of one or more time slots per committee member, determine possible meeting schedules in a given time period, based on the need to reach a quorum of k available members.
- A typical lion pride consists of 8 to 9 adults whereas a large pride consists of 30 to 40 adults.[1] Most social animal groups likewise have well-defined bounds on membership size. In an ecological animal database, identify large lion prides visiting a particular location in a given time window (i.e., $k = 30$ adult lions which temporally co-occur at the location).
- For analytics on a temporal graph (i.e., a graph where each edge in the

[1] *https://cbs.umn.edu/research/labs/lionresearch/social-behavior*

graph has an associated time window), identify k-sized subgraphs which temporally co-occur in a given time window, where k is the target subgraph size. For example, towards targeted recommendations in a social network, identify small groups of people ($k < 5$) who are all mutually socially connected in a given time window.

Prior studies on the interval join problem [5, 60, 4] can be viewed as a special case in which clique size $k = 2$. To the best of our knowledge, the general case has never been identified and studied before. Moreover, the problem arises as a basic challenge in the context of spatial and uncertain data management [61]. Specifically, by investigating the problem, we would like to improve the computation costs in temporal-predicate processing so that we can better focus on the leverage of selectivities in temporal subgraph queries. According to our discussion in Chapter 3, temporal predicates can be selective in real-world networks, especially when the value of CSS is small. Thus, efficient temporal k-clique enumeration algorithms can be used to filter numerous non-overlapping subsets and reduce the cardinality of partial results produced in temporal subgraph query processing. In a word, we investigate temporal-predicate processing for its various interest and direct relevance to our topic.

4.2 Problem statement

Our studied temporal k-clique enumeration problem is formalized as follows:

Definition 4.2.1 (Temporal k-clique enumeration) *Given a temporal relation R, a positive integer k, and time window $[q_s, q_e]$. Enumerate all $S \subseteq R$ where S is a temporal k-clique in $[q_s, q_e]$.*

Example. Figure 4.1 presents our running example for temporal k-clique enumeration in this chapter. Consider the visualized temporal relation $R_{ex} = \{r_1 : [0,2], r_2 : [4,6], r_3 : [5,10], r_4 : [7,9], r_5 : [8,10], r_6 : [4,4]\}$, time window $[5,8]$, and $k = 3$. There is exaclty one temporal 3-clique in w, namely, $\{r_3, r_4, r_5\}$. If $k = 2$, the collection of temporal 2-cliques is $\{(r_2, r_3), (r_3, r_4), (r_3, r_5), (r_4, r_5)\}$.

An efficient processing method for temporal k-clique enumeration can provide enhanced leverage of temporal selectivity and partially lay the foundation for investigating temporal subgraph query processing.

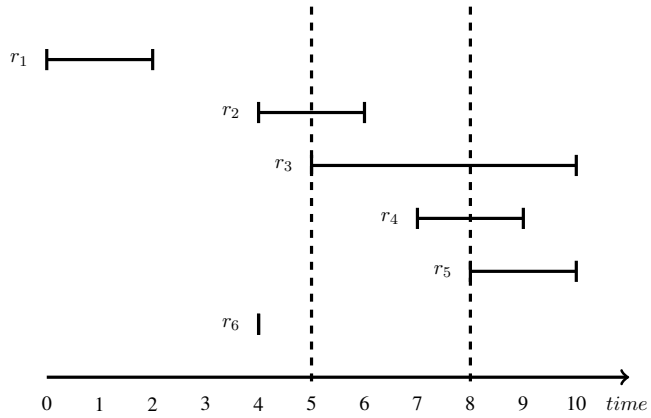


Figure 4.1: A running example of temporal k-clique enumeration. Temporal relation $R_{ex} = \{r_1 : [0, 2], r_2 : [4, 6], r_3 : [5, 10], r_4 : [7, 9], r_5 : [8, 10], r_6 : [4, 4]\}$ and query window $[5, 8]$.

4.3 Contributions

Our contributions in this chapter can be summarized as follows:

- We propose a linear-scan-based processing framework for temporal k-clique enumeration. The framework can be used to adjust existing sweep-based interval join algorithms to our investigated problem at a low cost. The adjusted algorithms have much lower complexity than the straightforward solution.
- Based on a careful analysis of the weakness of the adjusted algorithms, we propose a novel method, namely the Start Time Index (STI) algorithm, to provide more efficient processing for clique enumeration.
- We develop checkpoint mechanisms to further improve query processing in STI. We discuss four checkpointing strategies and highlight their benefits. In addition to STI, these strategies are of independent interest and could also be applied in combination with other state of the art methods.
- We present an in-depth experimental study of which the results demonstrate the significant improvements in scalability and performance introduced by our new methods.

4.4 Baseline: EBI and FS algorithms

As we have mentioned, the interval join processing can be regarded as a special case of our investigated problem with $k = 2$. Existing research on interval join processing can be classified into index-based [42, 43, 44], partition-based [45, 46], and plane-sweep methods [4, 5]. Currently, the best performing solutions for interval joins are based on plane-sweep methods [5]. Piatov et al. [4] proposed two memory plane sweep-based interval join algorithms EBI and LEBI based on endpoint index, which outperforms the prior plan-sweep methods. Bouros et al. [5] proposed two optimized algorithms based on forward scan named gFS and bgFS. Grouping and bucket indexing techniques are applied to reduce the cost caused by redundant comparison, which makes this algorithm competitive to Piatov's methods. We next discuss in detail these two general approaches, which are the state of the art methods.

Endpoint-based Index. EBI [4] is an internal-memory-based plane-sweep algorithm for processing an interval join between temporal relations R_1 and R_1. In EBI, each element $r \in R_1 \cup R_2$ with associated time window $[starttime(r), endtime(r)]$ is represented as a pair of endpoint events, where each event represented by tuple $(ti, ty, rid(r))$. Here, ti is the timestamp of an endpoint and is either $starttime(r)$ or $endtime(r)$, ty is the endpoint type and should be either $start$ or $stop$, and $rid(r)$ is the index of the record r. Given a pair of temporal relations R_1 and R_2, their endpoint indexes I_1 and I_2 are then constructed. The events are sorted by their ti in ascending order. As for join-processing, I_1 and I_2 are scanned concurrently from the beginning and each event is accessed forwardly. During the scan for each index, two *active-list* structures are maintained to store the concurrent set of relations in real time, denoted as A^1 and A^2. The active list is updated depending on the type of scanned endpoint. And for each scanned $start$ endpoint, EBI matches it with all the records in the opposite active list to produce joins.

Forward Scan Algorithm. Compared to EBI, forward scan (FS) [62] directly performs a linear scan on relations without using dedicated structures (i.e., active list). Relations R_1 and R_2 in forward scan algorithm are sorted by the start time of records. Two linear scans are carried out from the beginning of relation and stop each time at a new record. For each scanned record in a relation, FS matches it with all overlapping records in the opposite relation. In this way, all pairs of interval joins are produced. gFS and bgFS [5] are improved versions of FS. In gFS, similar consecutive intervals are grouped and matched with overlapping intervals in opposite relation instead of comparing

pairs of intervals one by one. In bgFS, the temporal domain is segmented into equal-sized dedicated buckets and intervals are put in corresponding buckets based on their start time. With the bucket index, the comparisons for join-matching need merely be made between interval groups in one relation and buckets in another. The two extensions reduce the cost of comparison and scanning in original FS.

4.5 Methodology

4.5.1 Framework on query processing

A straightforward method for answering temporal k-clique enumeration is to carry out $k-1$ instances of existing interval join over the temporal relation. The worst-case complexity of the straightforward idea is approximately $O(|R|^k)$, which is extremely large and obviously inefficient in practical application. Taking advantage of the fact that only self-joins are involved and temporal domain is linear, we aim to process the clique enumeration via a linear scan of temporal relation with much smaller complexity in theory. In this section, we propose a linear-scan-based framework for processing temporal k-clique enumeration. Using the framework, we can adjust interval join algorithms (i.e., EBI, gFS, and bgFS) to k-clique enumeration with much lower complexity than the straightforward method. Figure 4.2 presents our framework for processing temporal k-clique enumeration. The basic idea of answering temporal k-clique enumeration in this framework is to scan through the events in their temporal order overlapping the query window $[q_s, q_e]$, and generate k-cliques whenever a new record is encountered (at the timestamp of its start event). Hence, $[q_s, q_e]$ is the minimum window we have to scan through to generate all results.

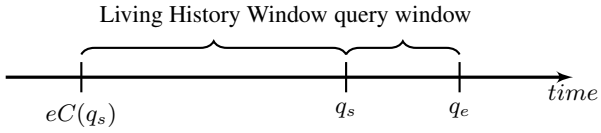


Figure 4.2: Our query processing framework via linear scan

In order to find all k-cliques within the query window, we are interested in first identifying the cliques at every timestamp where a start occurs. This timestamp corresponds to when a new clique may form within the query win-

dow. Based on this consideration, our framework takes the concurrent set CS as the core of our solution. To ensure scanning only $[q_s, q_e]$ and generating all results, we need to identify $CS(q_s)$ for every possible q_s in user queries. However, given that q_s can be any timestamp within the temporal domain of relation, it is impractical to store the concurrent set for every single timestamp. A simple remedy is to build the concurrent set on the fly at query processing time. We call the window that needs to be scanned to construct the concurrent set of a timestamp t the *Concurrent Set Construction Window (CSCW)* of t. Note that it is algorithm-dependent and the intuition is that we want it to be as small as possible.

A straightforward approach to construct $CS(q_s)$ is to start from the very beginning of temporal relation and scan through all the elements until one with a timestamp larger than q_s is met. This approach is obviously cumbersome since it introduces numerous unproductive scanning. Continuing our example and following the straightforward approach to construct $CS(5)$, the first scanned element r_1 would not be included in $CS(5)$ since it ends at $t = 2$, which is far before $t = 5$. The observation demonstrates the construction window in the straightforward processing is too large and could be reduced. For this aim, we consider an alternative approach as follows: Assuming no other information is available except the intervals associated with the elements in R, to construct the concurrent set of timestamp t, we need to go all the way back to the timestamp where the oldest record that is still active at t starts. We call the alternative approach the *Living History Window*, and its notion is defined as follows:

Living history window. Given a temporal relation R and a timestamp t, the *earliest concurrent* of t is the timestamp $eC(t)$ corresponding to the earliest start time of those records that are still active at t, i.e.,

$$eC(t) = \begin{cases} \text{undefined} & \text{if } CS(t) = \emptyset \\ \min_{r \in CS(t)} starttime(r) & \text{otherwise} \end{cases}$$

Note that $eC(t)$ only makes sense when $CS(t) \neq \emptyset$. We call the window $[eC(t), t]$ the *Living History Window (LHW)* of t, denoted $LHW(t)$; and the set of events in this window the Living History of t, $LH(t)$ = *events(*$LHW(t)$*)*

Example. Continuing our running example (Figure 4.1), two intervals occur at $t = 5$, i.e., $CS(5) = \{r_2, r_3\}$. The earliest record r_2 starts at $t = 4$, Hence, we have $eC(5) = 4$ and $LHW(5) = [4, 5]$.

Again, assuming no other information is available. The minimum set

of events we need to scan for the reconstruction of $CS(q_s)$ are the ones in $LHW(q_s)$. Hence, the ideal Concurrent Set Construction Window is the living history window of q_s. In this way, the scanning range in query processing is composed of two adjacent windows as shown in Figure 4.2: (1) the living history window and (2) the query window. The former is used to construct $CS(q_s)$ while the latter is used for clique enumeration. We could prove that no intervals that can contribute to the final join result would be missed in the method using $eC(q_e)$ compared to the basic method as follows

Theorem 4.5.1 *events($LHW(t)$)* covers all the records starting no later than and overlapping q_s.

Proof. Assuming a record r can contribute to the join result and is not included in *events($LHW(t)$)* , there must be (1) $starttime(r) < eC(q_s)$ and (2) $endtime(r) >= q_s$. Statement (1) and (2) illustrate that r is active at q_s. However, $eC(q_s)$ is defined as the start time of the earliest interval that is active at q_s, which contradicts the statement (1). So we know that no intervals associated with the final result are missed in the scan using $eC(q_s)$.

In the following sections, we will present a family of data structures and algorithms. The basic idea for the algorithms is adjusting the state of the art in interval join processing using our proposed framework to minimize the scan range.

4.5.2 Proposed method I: CE-EBI

In this section, we propose the adjusted EBI for temporal k-clique enumeration (CE-EBI, for short). The method consists of two main parts: (1) CE-EBI index, a B+-tree based data structure for temporally indexing the elements in temporal relation; and (2) CE-EBI algorithm, a processing pipeline which takes advantage of the CE-EBI index to answer the temporal k-clique enumeration problem efficiently.

Following the original EBI algorithm [4], we proceed to store and index temporal relation R in a CE-EBI index. The CE-EBI index is a B+-tree over R such that each tuple can be formalized as $(ti, ty, rid(r), eC(ti))$. Compared to the underlying index in [4], each tuple is extended with a new fourth position which records the earliest concurrent of ti for a fast location on the living history window of queries. We overload the $timestamp()$, $eventtype()$, and $eC()$ functions and allow them to take an index tuple as input and return the timestamp (first position), the event type (second position), and the earliest

concurrent of the timestamp (fourth position), respectively. The procedure to construct CE-EBI index can be described as follows:

- For each element $r \in R$, we construct two index tuples: $[starttime(r), rid(r), \mathsf{start}, -1]$ and $[endtime(r), rid(r), \mathsf{stop}, ()]$, representing the endpoints of r.
- We insert all these tuples into a B+tree, with the time element (i.e., first position of each entry) as primary search key and event type (i.e., the third position of each entry) as the secondary search key with the order defined as $\mathsf{start} < \mathsf{stop}$.
- As a final step of index construction, we scan the B+tree tuples from first to last, and update the fourth position of each tuple r of start type with $eC(timestamp(r))$, the starting time of the oldest element which is still active at the time-stamp of r.

We offer the following look-up methods for an CE-EBI index:

- $getEntry(t)$: given a time-stamp t, retrieve the first tuple r with the smallest time-stamp among all tuples that satisfy $t \leq timestamp(r)$.
- $getRecentStart(t)$: given a time-stamp t, retrieve the tuple with the largest time-stamp among all tuples that satisfy (1) r is of start type; and (2) $t \geq timestamp(r)$. If no such tuple satisfies both condition, then, return the first tuple of CE-EBI index.
- $startScan(r)$: start a linear scan of the index from tuple r and return a cursor sc for fetching each tuple.
- $nextEntry(sc)$: retrieve the next index entry of scan sc.
- $stopScan(sc)$: stop the scan sc in the CE-EBI index.

With CE-EBI index, the procedure of CE-EBI algorithm for temporal k-clique enumeration is presented in Algorithm 3. For each query q with a positive integer k and a window $[q_s, q_e]$, CE-EBI algorithm involves two B+tree look-ups and one linear scan of the tuples in CE-EBI index. The look-ups aim to identify the living history window for the following linear scan. The first CE-EBI index look-up uses q_s as the search key, invokes the $getRecentStart()$ method, and retrieves the tuple r for the most recent start event no later than q_s. The timestamp of r is marked as t_0 in Figure 4.3. The second CE-EBI index look-up uses $eC(t_0)$ as the search key and locates the left-most index entry with the timestamp. Figure 4.3 presents the determined scanning range of CE-EBI after the look-ups. Note that there may not be any event happening at q_s, the start point of the query window. Therefore, we do not have $eC(q_s)$ readily available and cannot identify the precise Living History Window of q_s. Instead, we identify the *Approximate Living History*

Window of q_s, $ALHW(q_s) = [eC(getRecentStart(q_s)), q_s]$, as described above. The scanning range of our CE-EBI is the concatenation of the two adjacent windows: the Approximate Living History Window of q_s and the query window.

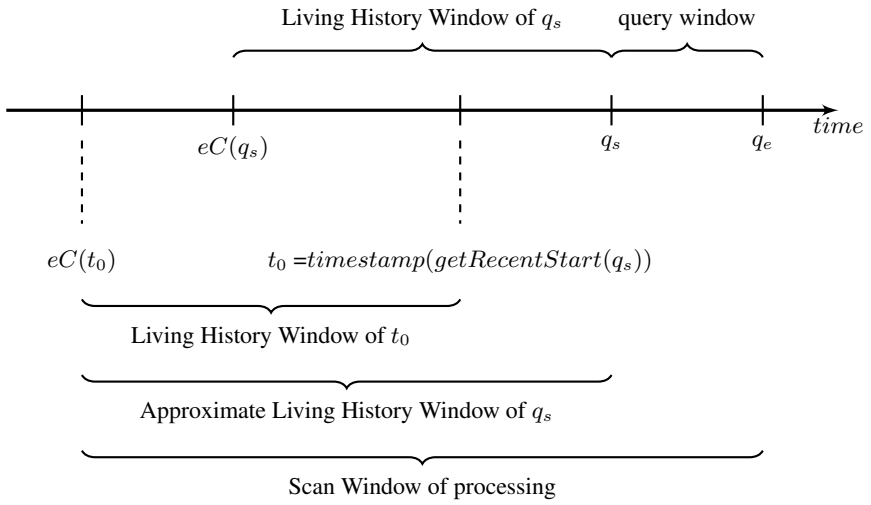


Figure 4.3: Our framework of temporal k-clique enumeration

During the linear scan, we maintain the active-list structure $Active$ to record currently active tuples. We define the following operations to manipulate the list:

- $insActive(Active, r)$: insert tuple r into active list $Active$.
- $delActive(Active, rid)$: delete tuple with tuple identifier rid from active list $Active$.
- $enumActive(Active, k)$: generate all subsets of size k over the elements of active list $Active$.
- $incEnumActive(Active, k, r)$: first invoke $insActive(Active, r)$ to insert a tuple r into active list $Active$, then generate all k-sized subsets over the elements in $Active$ which contains an occurrence of r.

CE-EBI algorithm takes actions whenever a tuple r is encountered. If r is a start tuple, algorithm inserts r into $Active$ or, if r is a stop tuple, the algorithm deletes r from $Active$. When the first tuple in a query window is encountered, all k-clique subsets in $Active$ are returned. From then on, every scanned tuple $curr$ would be matched with all $(k-1)$-cliques in $Active$ until either (1) the newly scanned $curr$ starts after the query window, or (2) the end of the relation

Algorithm 3: CE-EBI

Input: temporal relation R (with CE-EBI index I), query time window $[q_s, q_e]$, positive integer k

Output: outstream of k-size subset of tuples in R that forms k-clique in $[q_s, q_e]$

```
1  startTS ← eC(I.getRecentStart(q_s))
2  Active ← ∅, result ← ∅
3  curr ← I.getEntry(startTS)
4  inRange ← false
5  sc ← I.startScan(curr)
6  while curr ≠ NULL do
7      if timestamp(curr) < q_s then
8          if eventtype(curr) = start then
9              insActive(Active, curr)
10         else
11             delActive(Active, rid(curr))
12     else if timestamp(curr) ≤ q_e then
13         if inRange = false then
14             if eventtype(curr) = start then
15                 insActive(Active, curr)
16                 result ← result ∪ enumActive(Active, k)
17             else
18                 result ← result ∪ enumActive(Active, k)
19                 delActive(Active, rid(curr))
20             inRange ← true
21         else
22             if eventtype(curr) = start then
23                 result ←
                     result ∪ incEnumActive(Active, k,rid(curr)))
24             else
25                 delActive(Active, rid(curr))
26     else
27         break
28     curr ← getNext(sc)
29 if inRange = false then
30     result ← result ∪ enumActive(Active, k)
31 I.stopScan(sc)
```

is reached. These two situations show that all involved start-event tuples have been scanned and the algorithm should be halted.

Example. Consider the temporal relation R_{ex} and query window $q : [5, 8]$ in Figure 4.1. for 2-clique enumeration, we could obtain its CE-EBI index as follows:

$$I = \{(0, start, r_1, 0), (2, stop, r_1, 0), (4, start, r_2, 4), (4, start, r_6, 4), (4, stop, r_6, 4), (5, start, r_3, 4), (6, stop, r_2, 4), (7, start, r_4, 5), (8, start, r_5, 5), (9, stop, r_4, 5), (10, stop, r_3, 5), (10, stop, r_5, 5)\}$$

To enumerate all the 2-cliques in $q : [5, 8]$, CE-EBI algorithm firstly determines $[4, 5]$ to be the living history window by checking the $eC(5)$ from its index. So the algorithm starts the linear scan from $t = 4$ on the index. The procedure of the linear scan is summarized in Table 4.1, where for each scanned tuple, the status of $Active$ and conducted operations are listed. For example, the first encountered endpoint is $(4, start, r_2, 4)$. r_2 is added into $Active$ because it represents a start endpoint. And the status of $Active$ is changed to $\{r_2\}$ because of this operation.

Tuple	*Active*	*Operation*
(4,start,r_2,4)	$\{r_2\}$	insert r_2
(4,start,r_6,4)	$\{r_2, r_6\}$	insert r_6
(4,stop,r_6,4)	$\{r_2\}$	delete r_6
(5,start,r_3,4)	$\{r_2, r_3\}$	insert r_3 enumerate (r_2, r_3);
(6,stop,r_2,4)	$\{r_3\}$	delete r_2
(7,start,r_4,5)	$\{r_3, r_4\}$	insert r_4 enumerate (r_3, r_4)
(8,start,r_5,5)'	$\{r_3, r_4, r_5\}$	insert r_5 enumerate $(r_3, r_5), (r_4, r_5)$
R_{ex}.end()	$\{r_3, r_4, r_5\}$	End

Table 4.1: Example of CE-EBI processing

4.5.3 Proposed method II: CE-gFS and CE-bgFS

In this section, we present the adjusted FS algorithms for temporal k-clique enumeration, i.e., CE-gFS (i.e., the adjusted gFS) and CE-bgFS (i.e., the adjusted bgFS). We only discuss CE-gFS as an example and CE-bgFS can be obtained in the same way. The CE-gFS approach consists of two main parts: (1) *Start Time Index* (STI index), which is also a B+-tree based data structure

for temporally indexing the elements in temporal relation; and (2) CE-gFS algorithm, which follows the processing order of gFS to answer k-clique enumeration problem via STI. In STI index, only the start-event tuple in the form of [$starttime(r)$, $endtime(r)$, $rid(r)$, $eC(starttime(r))$] is used to represent each element in an STI index, instead of splitting a single interval into two event tuples. Given a temporal relation R, the procedure to construct a STI index can be described as follows:

- For each record $r \in R$, we construct an index tuple [$starttime(r)$, $endtime(r)$, $rid(r)$, -1], and insert all of these tuples into a B+tree, with the $starttime(r)$ (i.e., the first position of each entry) as primary search key.
- Similar to that in CE-EBI, as a final step of index construction, we scan the tuples in B+tree from first to last, and update the fourth position of each tuple with $eC(timestamp(starttime(r)))$.

STI index inherits all the look-ups from CE-EBI except $getRecentStart()$, which should be overloaded as follows.

- $getRecentStart(t)$: given a timestamp t, retrieve the first start-event tuple r with the largest timestamp among all index tuples that satisfy $t > starttime(r)$. Return the first start-event tuple of an STI if no such tuple exists.

The procedure of CE-gFS algorithm is presented in Algorithm 4. The scanning range can be located by the same look-ups as in CE-EBI. Following the original gFS [5] , a dedicated structure $Group$ is maintained to group "consecutive" tuples. In our CE-gFS, "consecutive" means that given the relation R and for any two time instances t_1 and t_2 with $t_1 < t_2$, all elements in R are considered "consecutive" if no interval ends in $[t_1, t_2]$. Then, all elements from R starting in $[t_1, t_2]$ can be grouped together to reduce comparisons. In other words, we can group all intervals which have succeeding endpoints of the same type. In $Group$, all elements are sorted by their end time in ascending order. We define the following operations to maintain $Group$.

- $insGroup(Group, r)$: insert the start-event tuple r into $Group$.
- $enumInternal(Group, k)$: generate all temporal k-clique subsets over the elements of $Group$;
- $enumExternal(Group, k, sc, qstop)$: generate all temporal k-clique subsets containing at least an element in $Group$, via initializing a forward scan starting from scanner sc. Note that $qstop$ is used as the boundary constraint for the forward scan

- $getMininalStop(Group)$: return the minimal end time among all elements in $Group$.

Each encountered tuple $curr$ in the living history window would only be added to $Group$ if its end time is larger than q_s. While for each $curr$ in the query window, the algorithm first checks if its start time is larger than the smallest end time in $Group$. If not, $curr$ is directly added to $Group$. Otherwise, the algorithm would first (1) enumerate all k-cliques in $Group$, (2) initialize a forward scan starting from scanner sc to generate all k-cliques by joining elements in $Group$ with tuples encountered in the forward scan, and (3) clears $Group$ after all these are done. Finally, the algorithms end when either (1) the newly scanned $curr$ starts after the query window, or (2) the end of the relation is reached, just like in CE-EBI.

Example. Continuing our temporal relation and 2-clique enumeration query, we could first construct following STI index for the temporal relation:

$$I = \{(0, 2, r_1, 0), (4, 4, r_6, 4), (4, 6, r_2, 4), (5, 10, r_3, 4), \\ (7, 9, r_4, 5), (8, 10, r_5, 5)\}.$$

CE-gFS algorithm firstly determines $[4, 5]$ to be the living history window by checking the $eC(5)$ from its index. Starting from $t = 4$ on the index, the procedure of running instance is summarized in Table 4.2. For example, when tuple $(7, 9, r_4, 5)$ is scanned, CE-gFS notes that r_4's start time 7 is larger than the minimal end time 6 (r_2's end time) in $Group$. Hence, CE-gFS first enumerates $\{r_2, r_3\}$ over $Group$ and then initializes a forward scan to join elements in $Group$ with encountered elements in the forward scan, during which $\{r_3, r_4\}, \{r_3, r_5\}$ would be generated.

4.5.4 Challenge

CE-EBI, CE-gFS, CE-bgFS can solve the temporal k-clique enumeration problem correctly as they derive from the state of the art in the interval join problem. Compared to the straightforward method, the adjusted algorithms are much less complex than the straightforward processing using $k - 1$ interval join instances since they follow a linear-scan-based framework. We taking CE-EBI as an example: For index-scanning, the entirety of CE-EBI index would be scanned in the worst case of processing, where the total cardinality of scanned tuples should be $2 \cdot |R|$ as CE-EBI index decouples each element in R into two

Algorithm 4: CE-gFS

Input: temporal relation R (with STI index I), query time window $[q_s, q_e]$, positive integer k

Output: outstream of k-size subset of tuples in R that forms k-clique in $[q_s, q_e]$

```
1  startTS ← eC(I.getRecentStart(q_s))
2  Group ← ∅,result ← ∅
3  curr ← I.getEntry(startTS)
4  sc ← I.startScan(curr)
5  while curr ≠ NULL do
6      if starttime(curr) < q_s then
7          if endtime(curr) >= q_s then
8              insGroup(Group,curr)
9      else if starttime(curr) ≤ q_e then
10         if getMininalStop(Group) < starttime(curr) then
11             result ← result ∪ enumerateInternal(Group, k)
12             result ←
                result ∪ enumerateExternal(Group, k, sc, q_e)
13             Group ← ∅
14         insGroup(Group,curr)
15     else
16         break;
17     curr ← getNext(sc)
18 result ← result ∪ enumerateInternal(Group, k)
19 I.stopScan(sc)
```

tuples. For maintenance, every encountered element would trigger an insertion or deletion to the $Active$ during the process of the scanning. The complexity of insertion and deletion is determined by the implementation of $Active$. Consider if $Active$ is a sorted list, the complexity of each insertion and deletion should be $O(\log |Active|)$. In this way, the worst-case complexity of CE-EBI is $O(|R| \cdot \log |R|)$[2] , which is much smaller than the complexity of straightforward processing (i.e., $O(|R|^k)$).

Note that we consider the index-scanning cost as the most important factor in processing efficiency. First, in practical applications, $|Active|$ is gener-

[2]In the worst case, the size of $Active$ should be the number of elements in relation R

Main scan	Forward Scan	Group	Operation
$(4, 4, r_6, 4)$	-	$\emptyset$	ignore r_6;
$(4, 6, r_2, 4)$	-	$\{r_2\}$	insert r_2;
$(5, 10, r_3, 4)$	-	$\{r_2, r_3\}$	insert r_3;
$(7, 9, r_4, 5)$	-	$\{r_2, r_3\}$	enumerate $\{r_2, r_3\}$; start a forward scan;
-	$(7, 9, r_4, 5)$	$\{r_2, r_3\}$	enumerate $\{r_3, r_4\}$;
-	$(8, 10, r_5, 5)$	$\{r_2, r_3\}$	enumerate $\{r_3, r_5\}$;
-	R_{ex}.end()	$\emptyset$	clear $Group$ exit forward scan
$(7, 9, r_4, 5)$	-	$\{r_4\}$	insert r_4;
$(8, 10, r_5, 5)$	-	$\{r_4, r_5\}$	insert r_5;
R_{ex}.end()	-	$\{r_4, r_5\}$	enumerate $\{r_4, r_5\}$; End

Table 4.2: Example of CE-gFS processing

ally much smaller than the cardinality of scanned tuples. Second, unlike the maintenance of $Active$ which always happens in memory, the index-scanning can happen either in memory (i.e., in-memory database) or hardware (in-disk database). The latter scenario can make a $nextEntry()$ operation on CE-EBI index much more expensive than insertion or deletion on $Active$.

Though the adjusted algorithms reduce the complexity of temporal k-clique enumeration, they still suffer from several inefficiencies in practical application. CE-EBI is inefficient in several aspects.

- For *index-scanning*, though the living history window is used, the scanning range is still much larger than the given query window. In the worst case, the scan could start from the leftmost event tuple, just like the time when the notation of living history window is not proposed. For example, if an element $r_7 : [0, 10]$ is added into R_{ex}, there will be $eC(t) = 0$ for $\forall t \in [0, 10]$. Such a scenario can happen in various applications because of the existence of extremely long duration as we discussed in Chapter 3. Besides, decoupling endpoints could lead the scanning cost to be much larger than the number of events that actually involve enumeration: (1) For elements starting and also stopping in the scanning range, the scanning cost doubles (e.g., r_2, r_6 in CE-EBI example). (2) For elements starting before scanning range but stopping in, their stop event tuples are still to be scanned though they do not involve the enumeration.
- For *active-list maintenance*, either an insertion or a deletion will be performed for each encountered tuple during the scan. Some of these per-

formed before q_s are irrelevant to the results (e.g. r_6). In the worst case, the number of operations performed on irrelevant tuples can be significantly larger than that of events which contribute to the final query results.

- Finally, for *index storage*, as the number of event tuples doubles the number of elements in relation, the storage cost increases significantly.

In a word, the essential reason for these additional costs is the decoupling of endpoints in CE-EBI. First, the decouple requires that an element should be represented via two indexed tuples. The cumbersome representation leads to additional costs in storage and scanning. Second, the decouple breaks the dependence between start and end time in elements. That is, CE-EBI algorithm cannot obtain the ending information of an active element until it encounters its stop event tuple. The lack of information finally leads to the unproductive cost of maintenance.

Next, we turn to the analysis of CE-gFS and CE-bgFS. Without decoupling endpoints in CE-gFS and CE-bgFS, the living history window scanning cost does not double. However, the worst case of the living history window still exists. A key inefficiency in the two algorithms is the production of duplicate index-scanning. That is, many elements tend to be scanned more than twice because of the inconsistent references in grouping and enumeration. In other words, the scanning range in grouping depends on the smallest end time of grouped elements while the scanning range in enumeration depends on the largest end time. For example, consider a relation $\{r_1 : [0, 2]$, $r_2 : [1, 10]$, $r_3 : [3, 12]$, $r_4 : [4, 6]$, $r_5 : [8, 10]$, $r_6 : [8, 10])\}$ and a query window $[0, 12]$. Note that R itself has been sorted by start time and earliest concurrent is not necessary since the query window covers all elements in R. A running instance of CE-gFS would first group $\{r_1, r_2\}$ and match them with $\{r_3, r_4, r_5, r_6\}$. Next, it would group $\{r_3, r_4\}$ and match them with $\{r_5, r_6\}$. And finally, $\{r_5, r_6\}$ would be grouped and matched with no elements. Note that even in such a small example, r_5 and r_6 are scanned three times. The essential reason for this phenomenon is that the existence of long intervals (e.g., r_2, r_3) significantly increases the necessary scanning range in each forward scan instance. Such impact can also be found in original gFS and bgFS for interval join processing easily, which makes gFS and bgFS lose their advantage in processing relations with many long intervals. Therefore, in FS family of algorithms, the existence of long intervals can potentially introduce high redundant scanning costs. Besides, the enumeration via forward scan can be costly when k is larger than 2.

4.5.5 Proposed method III: STI algorithm

Realizing the in-efficiency in the adjusted methods from analysis, we propose an optimized method based on STI index (namely, the STI algorithm), with the aim to provide more efficient temporal k-clique enumeration. The procedure of STI algorithm is presented in Algorithm 5. Our method uses STI index in CE-gFS and CE-bgFS as the underlying representation of data, but follows the processing order in CE-EBI. The aim of using STI index is to avoid inefficiencies in CE-EBI. Also, following CE-EBI processing order aims to avoid the duplicate scanning in CE-gFS and CE-bgFS algorithms. Just like in those adjusted algorithms, STI algorithm first determines the living history window via two look-ups. Then it carries out a linear scan to generate all k-cliques. During the linear scan, the in-memory active-list structure $Active$ is maintained to record the concurrent set in real time. The tuples in $Active$ are sorted by their end time in the ascending order. We define the following operations to maintain $Active$:

- $insActive(Active, r)$: insert the start-event tuple r into $Active$;
- $delActive(Active, t)$: delete all start-event tuples r s.t. $t > endtime(r)$ from $Active$;
- $enumActive(Active, k)$: generate all temporal k-clique subsets over the elements of $Active$;
- $incEnumActive(Active, r, k)$: first insert r into $Active$, then generate all temporal k-cliques over the elements in $Active$ which contains an occurrence of r.

The linear scan starts from the leftmost tuple in approximate living history window of q_s and scans forward. $Active$ is real-time maintained by inserting newly scanned start-event tuple $curr$ and deleting the expired tuples. When the first start-event tuple in a query window is scanned, all k-clique subsets in $Active$ are returned. From then on, every scanned start-event tuple $curr$ would be matched with all $(k-1)$-cliques in $Active$ until either (1) the newly scanned $curr$ starts after the query window, or (2) the end of the relation is reached. If there are no tuples in a query window, the algorithm directly enumerates all the k-clique subsets in $Active$ as the result. This way, all k-cliques in $[q_s, q_e]$ of relation R are enumerated.

Example. Continuing our temporal relation and 2-clique enumeration query, we could first construct the same STI index for the temporal relation as in CE-gFS example: then to enumerate the 2-cliques in $q = [5, 8]$, the algorithm firstly determines $[4, 5]$ to be the approximate living history window,

Algorithm 5: STI algorithm

Input: temporal relation R (with STI index I), query time window $[q_s, q_e]$, positive integer k

Output: outstream of temporal k-cliques in $[q_s, q_e]$ of R

```
1  startTS ← eC(starttime(I.getRecentStart(q_s)))
2  Active ← ∅,result ← ∅
3  curr ← I.getEntry(startTS)
4  inRange ← false
5  sc ← I.startScan(curr)
6  while curr ≠ NULL do
7      if starttime(curr) < q_s then
8          if endtime(curr) >= q_s then
9              insActive(Active, curr)
10     else if starttime(curr) ≤ q_e then
11         if inRange = false then
12             delActive(Active, q_s)
13             result ← result ∪ enumActive(Active, k)
14             inRange ← true
15         delActive(Active, starttime(curr))
16         result ← result ∪ incEnumActive(Active, curr, k)
17     else
18         break;
19     curr ← getNext(sc)
20 if inRange = false then
21     delActive(Active, q_s)
22     result ← result ∪ enumActive(Active, k)
23 I.stopScan(sc)
```

by retrieving $eC(5) = 4$ from $(5, 10, r_3, 4)$. Starting from the beginning of the approximate living history window, the processing procedure in shown in Table 4.3.

Maintenance. Maintenance of STI index under insertions and deletions of elements in R incurs the costs of B+tree maintenance and the cost of updating earliest concurrent values of affected tuples. When a new tuple r' is inserted, $eC(starttime(r'))$ could be obtained by using the information

$Tuple$	$Active$	$Operation$
$(4, 4, r_6, 4)$	$\emptyset$	ignore r_6
$(4, 6, r_2, 4)$	$\{r_2\}$	insert r_2
$(5, 10, r_3, 4)$	$\{r_2, r_3\}$	insert r_3; enumerate (r_2, r_3);
$(7, 9, r_4, 5)$	$\{r_3, r_4\}$	delete r_2; insert r_4; enumerate (r_3, r_4);
$(8, 10, r_5, 5)$	$\{r_3, r_4, r_5\}$	insert r_5; enumerate $(r_3, r_5), (r_4, r_5)$;
R_{ex}.end()	$\{r_3, r_4, r_5\}$	End

Table 4.3: Example of STI processing

provided by its adjacent tuples. In addition, we traverse each tuple r in $[starttime(r'), endtime(r')]$ and update $eC(starttime(r))$ to $starttime(r')$ if $eC(starttime(r)) > starttime(r')$. When an existing tuple r' is deleted, we first check if there exists a tuple r'' in STI such that $starttime(r'') = starttime(r')$ and $endtime(r'') \geq endtime(r')$. If so, no additional maintenance needs to be done since the deletion of r' will not incur any change on the earliest concurrent of tuples. Otherwise, we carry out a linear scan on tuples in $[starttime(r'), endtime(r')]$ with $Active$ maintained and update $eC(starttime(r))$ to $\min_{r \in Active} starttime(r)$ if $eC(starttime(r)) = starttime(r')$.

Complexity. The worst-case complexity of STI algorithm is $O(|R| \cdot \log R)$ as in CE-EBI and CE-bgFS. By comparasion, however, STI succeeds in better processing efficiency. Contrary to CE-gFS, duplicate scanning would not happen in STI approach since it follows CE-EBI processing order. Compared to CE-EBI with decoupling endpoints, STI requires less space for storing tuples and internal nodes in the B+Tree. The scanning cost in STI is improved: (1) For elements starting and also stopping in the scanning range, the scanning cost is halved because one element is represented by a single tuple rather than a pair; (2) For elements starting before the scanning range but not ending, no additional scanning cost is introduced because the start-event tuples are sorted by start time, which makes it impossible for such elements to appear in the scanning range. Additionally, operations on irrelevant tuples in active-list maintenance are completely avoided since their end time are compared with q_s. Finally, extra scanning costs introduced by the approximation of the living history window are minor compared to the scanning benefits gained from the STI, especially in datasets with extremely long intervals. However, we should

note that the case of the worst living history still exists. That is, the living history window could still be very large in its absolute temporal measurement.

Summarizing, STI addresses all the inefficiencies in CE-EBI and CE-gFS except the long living history window problem. In the rest of this section, we investigate the methods on how to solve this problem.

4.6 Optimization: checkpointing

We introduce Start Time Index with Checkpoints (STI-CP), which is a variant of STI enhanced with *checkpoints* (CPs), aiming to speed up the processing of long living history windows. A single checkpoint is a dedicated structure composed of a timestamp c and $CS(c)$, which represents the concurrent set at timestamp c.[3] Given a temporal relation R, necessary index structures in STI-CP include: (1) a STI index and (2) a set of checkpoints $C^P = \{c_1, c_2, \ldots, c_p\}$. Then, the linear scan in STI index could start from the timestamp of the latest checkpoint which is smaller than q_s rather than from the beginning of living history window. In the best case, $CS(q_s)$ could be directly obtained. The major difference between STI-CP and STI are:

- In STI-CP, an additional data structure that stores the concurrent set for each checkpoint in C^P is maintained. After the STI index is constructed, a checkpointing procedure proceeds to select some checkpoints in temporal domain and store them in dedicated structures for further use.
- In STI-CP, $LHW(t)$ starts from $max(eC(t), lateCP(t))$ where $lateCP(t) = \max_{c \in C^P \wedge c \leq t}(c)$. We call such timestamp a *history pointer* of time t, denoted $HP(t)$. That is, $LHW(t)$ in STI-CP is $[HP(t), t]$ rather than $[eC(t), t]$ in STI.

The STI-CP algorithm is shown in Algorithm 6. We provide the following functions are provided in STI-CP in addition to those in STI:

- $getHistoryPt(t)$: Given a timestamp t, returns the history pointer of t. It may be either $eC(t)$ or a timestamp of a checkpoint.
- $isCP(t)$: Given a timestamp t, returns true if $t \in C^P$ and false otherwise.
- $getConcurSet(t)$: Given a timestamp t, returns $CS(t)$ if $t \in C^P$ and $\emptyset$ otherwise.

Example. Continuing our running example, if a checkpoint c is set at $t = 5$,

[3] For brevity, c denotes a checkpoint at timestamp c

Algorithm 6: STI-CP temporal clique enumeraion

Input: temporal relation R (with STI-CP index I), query time window $[q_s, q_e]$, positive integer k

Output: outstream of temporal k-cliques in [q_s,q_e] of R

1 $startTS \leftarrow getHistoryPt(starttime(I.getRecentStart(q_s)))$

2 $Active \leftarrow I.getConcurSet(startTS)$

3 ... continue on with Algorithm 5 starting from line 3.

the scanning of living history window $[4, 5]$ becomes unnecessary, leading to a saving in scanning costs.

One might argue that checkpoints can introduce another scalability bottleneck. Consider a checkpoint c recording $CS(c)$ and a query window $[q_s, q_e]$. If only one element $r \in CS(c)$ overlaps the query window, the elements in $CS(c) - \{r\}$ are redundant and will introduce additional costs for the active-list maintenance in the living history window. This problem can be solved with smart implementation. Specifically, we implement the concurrent set of each checkpoint in the form of a relation in which elements are sorted by their end time, just like $Active$ in the STI algorithm. Every time a checkpoint c is obtained as the most recent checkpoint of q_s, STI-CP algorithm first locates the first element r with an end time larger than q_s and then takes its following elements (including r) as the initial $Active$. For each checkpoint $c \in C^P$, the additional cost introduced by this smart implementation is $O(\log C(c))$, where $C(c)$ represents the size of concurrent set at timestamp c (see Definition 3.1.1). In this way, elements which do not contribute to the query window can be easily filtered so that additional cost can be avoided.

Maintenance. Checkpoints could also be used to accelerate the maintenance of STI in Section 4.5.5. As for the maintenance of checkpoints themselves, an insertion of tuple r' would lead to inserting r' into $CS(c)$ for each $c \in [starttime(r'), endtime(r')]$. Similarly, a deletion of tuple r' would lead to removing it from the same group of checkpoints.

4.6.1 Problem statement

We note that the effectiveness of STI-CP depends on the selected CPs. Hence, in the following, we concentrate on the problem of checkpointing. We start by formalizing our problem as follows.

Definition 4.6.1 (Checkpointing problem) *Given a set Q of m queries in a workload, a set T of n checkpoint candidates, and storage budget B, a checkpointing problem $< Q, T, B >$ aims to obtain a solution $C^P = \{c_1, c_2, \ldots, c_p\}$ such that $\sum_{i=1}^{p} C(c_i) \leq B$, $p \leq n$, and $\forall c_i \in T$.*

That is, we select CPs under a limited storage budget B such that the total storage cost of CPs should not exceed B. For the best benefits from CPs, researchers might be interested in finding the optimal solution to our checkpointing problem. Intuitively, given a query workload Q and a CP solution C^P, the benefit from C^P is the scanning costs on indexed tuples saved by accessing CPs in C^P. However, one can easily show NP-completeness of the corresponding optimal checkpointing problem by producing a reduction to a 0-1 knapsack problem. In the rest of this section, we propose several heuristics for checkpointing, which could obtain effective solutions at small costs.

The most basic heuristic method is based on *random* checkpointing. This method is easy to implement, but it does not make good use of the budget as random placement of CPs might do little to improve query times. With the aim of obtaining more effective checkpointing strategies, we consider both data distribution and query workload. As a result, we propose four checkpointing strategies classified in two broad categories: *data-aware* and *workload-aware*.

Note that the initialization of the CP index consists of two phases: the CP selection and insertion phase. The complexity of the first phase depends on the checkpointing strategy while that of the second phase is strategy-independent.[4] So, for each strategy, we only analyze the complexity of its CP selection.

4.6.2 Data-aware strategies

Binary strategy. Our first checkpointing strategy is called *binary* strategy since it selects the CPs in a binary, breadth-first manner, until the storage budget is consumed. In every round, pairs of elements with the largest distance in which no CP exists is retrieved, and the middle point of the pairs in the duration is selected as the position to set a new CP. In this way, the CP distribution becomes even and pairs of CPs that are too close to each other (and, thus, potentially wasteful) could be avoided.

As Chapter 3 reveals that real-world data are not evenly distributed tem-

[4]Given N index tuples and p CPs, it takes $O(N \log N)$ time to scan the STI index while maintaining an active list and $O(p \log p)$ time to collect CSs and then insert them into the CP index bringing overall complexity to $O(N \log N + p \log p)$.

porally because of burstiness [52], we define two types of distances that could either be used in a binary strategy: (1) event and (2) temporal distance. Event distance reflects the distance measured in the number of events, while temporal distance reflects the distance measured in time. By considering temporal distance, we capture the burstiness of temporal data. To be more specific, Figure 4.4 and 4.5 presents the examples of event and temporal-binary checkpointing on R_{ex} respectively, in which we only consider the selection of the first three CPs.

The complexity of both binary methods is $O(p \log p)$, where p is the number of CPs. As the event and temporal distance are respectively tuple and timestamp-based, we define following mapping methods from timestamp to tuple:

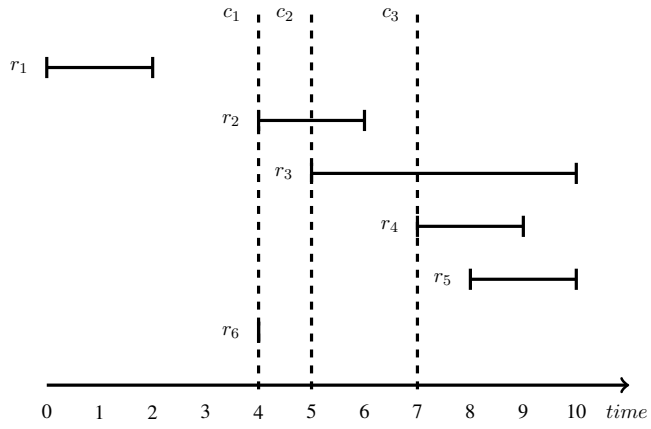


Figure 4.4: Example of event-binary checkpointing on R_{ex}, where dash line represents the selected CPs. For each CP c_i, its index i refers to its order to be selected in procedure.

- $first(t)$: given a timestamp t, returns the number of the first start-event tuple r s.t. $starttime(r) = t$. If there is no tuple starting at t, $getRecentStart(t)$ is returned.
- $last(t)$: given a timestamp t, returns the number of the last start-event tuple r s.t. $starttime(r) = t$. If there is no tuple starting at t, $getEntry(t)$ is returned.

Long-link-half strategy. Binary strategies do not consider the impact of long intervals in data. That is, extremely long intervals could influence a large number of tuples in index. Specifically, given a start-event tuple r and

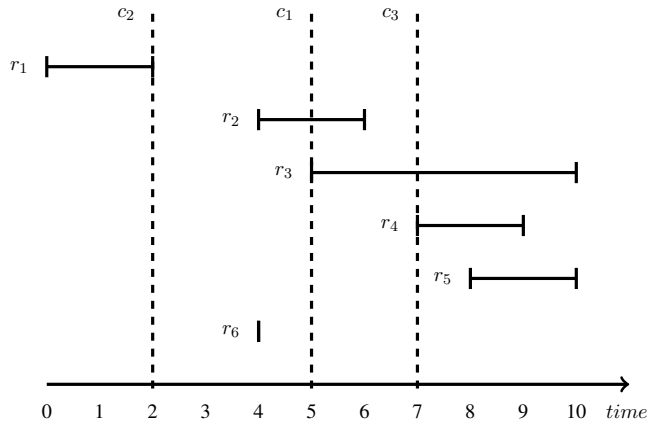


Figure 4.5: Example of temporal-binary checkpointing on R_{ex}, where dash line represents the selected CPs. For each CP c_i, its index i refers to its order to be selected in procedure.

a long interval r' such that $starttime(r) \in [starttime(r'), endtime(r')]$, $eC(starttime(r))$ should be at least as small as $starttime(r')$. Hence, to process such queries, algorithm needs to start the scan from $starttime(r')$ at least when no CP is present. Meanwhile, we should also note that queries are more likely to overlap long intervals when they are uniformly distributed.

Based on this, we propose the *long link half* strategy, which gives priority to long intervals to be assigned CPs in order to reduce their impact. The core of long link half strategy is a dedicated structure named *link map* constructed on *influential intervals* in STI. The aim of constructed link map is to provide guidance for later CP selection. We first present the notation of influential interval as follows.

Definition 4.6.2 (Influential Interval) *Given an STI index I and a start-event tuple r_0, if $\exists r \in I$ such that $eC(r) = starttime(r_0)$, we call $[starttime(r_0), endtime(r_0)]$ an influential interval in I.*

Influential intervals are the intervals which determine the earliest concurrent of tuples in STI index, which cover the whole domain of temporal relation. Considering queries in workload are uniformly distributed, longer influential intervals tend to have more opportunities to cover the uniformly distributed queries, which also tend to produce longer living history window. With the notation of influential intervals, our link map is constructed as follows.

- We collect all influential intervals via a linear scan on STI index, and sort them by $first(starttime(r_0))$ in ascending order.
- We refine the collection of influential intervals and build up the link map on them. Starting from the first interval, each interval is continuously combined with the following interval into a longer interval until they no longer significantly overlap (according to some threshold u). The newly generated interval would be continuously checked and determine if it could be combined with the next link. Such combination could help to avoid the potentially wasteful CPs in later selection phase.

In this way, we obtain the link map that could provide us an overview of the distribution of influential intervals in STI index. Given the link map, CPs can be selected under its guidance. In every round, the longest interval is taken out from the map, and the start time of its middle event is selected as a place to set CP. The CP segments the interval into two sub-intervals and this round would be carried recursively until the budget B is consumed. The selected CPs have a high tendency to segment long intervals, which turns out to weaken the long interval impact.

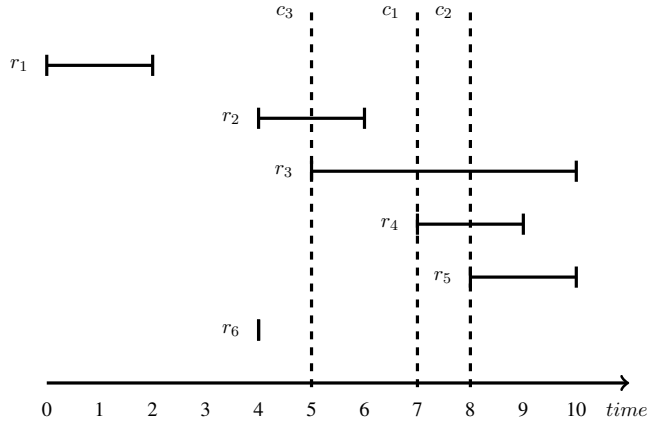


Figure 4.6: Example of long-link-half checkpointing on R_{ex} with threshold $u = 0$, where dash line represents the selected CPs. For each CP c_i, its index i refers to its order to be selected in procedure.

Example. Figure 4.6 presents our example for long-link-half checkpointing over R_{ex} with $u = 0$. The set of influential intervals in R_{ex} is $\{[0, 2], [4, 6], [5, 10]\}$, which would be directly used as the link-map since the threshold u for the determination of significant overlapping is set to 0. In the first round, the longest influential interval $[5, 10]$ is selected and a CP is set at

$t = 7$, which is exactly the start time of middle event r_4 and divides $[5, 10]$ into $[5, 7]$ and $[7, 10]$. Next in the second round, $[7, 10]$ is selected and a CP is set at $t = 8$, which divides $[7, 10]$ into $[7, 8]$ and $[8, 10]$. Then in the third round, $[4, 6]$ is selected and a CP is set at $t = 5$.

The complexity of link map construction is $O(|R|+\kappa \log \kappa)$, where κ is the number of link tuples in map. Considering influential intervals are collected through a single scan, we could move the collecting step into the initialization of the STI index. So the additional linear scan is unnecessary and the map construction complexity could be reduced to $O(\kappa \log \kappa)$. The complexity of selection is $O(p \log \kappa)$. Hence, the total complexity of the long link half strategy is $O((\kappa + p) \log \kappa)$.

4.6.3 Workload-aware strategies

The data-aware strategies do not assume any knowledge of the query workload. Without query workload, it may waste valuable storage budget to store CPs pointed in sections of data where no one queries, while the frequently queried data sections are not sufficiently checkpointed and the query performance is sub-optimal. In this section, we present our workload-aware strategies with prior workload information, besides the storage budget. Our goal is to choose a checkpoint set that maximizes performance gain for queries that fits this workload within the budget.

We model real-world query workloads as being composed of two parts: first, a small proportion of uniformly distributed queries, which represents the outlier behavior performed by some users. Second, a large proportion of queries distributed around several *hotspots* in the domain of a dataset. For example, considering a scheduling of the free time slots in classrooms in a university campus, most queries would aggregate in January and July since it is the time for final examinations. Using such queries, administration staff could find the pairs (or larger subsets) of classrooms available to simultaneously hold examinations. Considering a biological database recording retention period of zebras in Serengeti National Park in Africa, researchers might be interested in querying the pairs of zebras staying simultaneously in one place. Most of such queries would aggregate in the first half of each year as in the later half zebras would move to Masai Mara for abundant grass and water. This example also implies that burstiness patterns in real world could also be a factor in aggregation of queries.

Based on the prior knowledge, we propose a workload-aware strategy

named *query-set*, which consists of two phases: in the first phase, a batch of CPs is selected for clusters based on their importance. Secondly, if budget allows, another batch is selected for the uniformly distributed queries in order to improve the global efficiency. The basic idea for the first phase is to identify the hotspots in workload, clustering queries around each hotspot, and selecting the CPs for each cluster. Many existing works could be used to detect such clusters. Here, we choose the mean-ISI method [63] to obtain the aggregated pattern. Note that there are situations where some clusters could not receive a CP since budget B is limited, so it is necessary to determine which clusters should have a priority. For this reason, we introduce a metric named *cluster importance*, denoted as CI, to assist in making this decision.

Definition 4.6.3 (Cluster Importance) *Given a query cluster Cl and the minimal time window $[\overline{Cl}, \underline{Cl}]$ covering all q_s of queries in, its cluster importance CI could be calculated as follows:*

$$CI = |Cl| \cdot (LH(\overline{Cl}) + last(\underline{Cl}) - first(\overline{Cl} + 1))$$

where $|Cl|$ refers to the number of queries in Cl. We call the minimal time window $[\overline{Cl}, \underline{Cl}]$ the duration of Cl.

The idea of CI is to approximate the living history window scanning cost for the whole cluster. This estimation is efficient when Cl is large. We put all the query clusters in a list and sort them by their importance in descending order. Recursively, we select the cluster with the highest priority from the remaining clusters and set a checkpoint at $\overline{Cl}$, until the budget is consumed.

After the batch of initial CPs for each cluster are selected, duration of clusters would be sorted by the number of elements (i.e., last($\underline{Cl}$) - first($\overline{Cl}$ + 1)) inside in descending order. Ideas in event binary and long link half strategy are recursively applied to the re-sorted list until either (1) B is consumed or (2) the number of elements in every remaining duration is shorter than a pre-configured threshold x. That is, when (2) happens, it demonstrates most of queries in the cluster could benefit from CPs so we need to stop selecting CPs for clustered queries and turn to the next phase.

In the second phase, CPs are selected in the same way as in long link half strategy, aiming at the improvement on the processing of uniform queries and full use of the remaining budget.

Example. Figure 4.7 presents the query-set checkpointing over R_{ex} with query workload $Q = \{q_1 : [0, 1], q_2 : [6, 7], q_3 : [7, 8], q_4 : [8, 9]\}$. The

procedure starts by detecting the query cluster $Q_{2-4} : [6, 9]$, in which query q_2, q_3, q_4 are aggregated. Then, the first CP is set at $t = 6$, which is the start time of the query cluster. Following event-binary checkpointing in order to segment the cluster, the second CP is set at $t = 7$. Finally, following the long-link-half checkpointing, $[7, 10]$ (produced after the second CP is set) is selected and the third CP is set at $t = 8$.

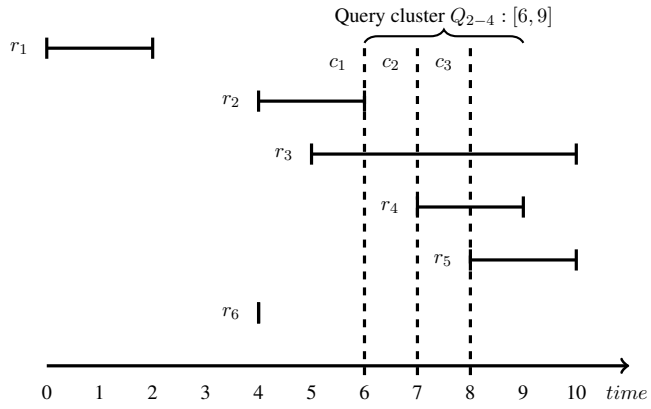


Figure 4.7: Example of query-set checkpointing on R_{ex} with query workload $Q = \{q_1 : [0, 1], q_2[6, 7], q_3 : [7, 8], q_4 : [8, 9]\}$, where dash line represents the selected CPs. For each CP c_i, its index i refers to its order to be selected in procedure.

The complexity of clustering is $O(m \log m)$, where m is the number of queries in workload. The further segmentation of cluster durations takes $O((m_c + p_0) \log m_c)$, where m_c is the number of clusters in workload and p_0 is the number of CPs selected in this step. The second phase has the same complexity as long link half strategy. So the total complexity of query-set strategy is $O(m \log m + (m_c + p_0) \log m_c + (p + \kappa) \log \kappa)$.

4.7 Experiments

In this section, we present our experimental investigation of STI and STI-CP approaches. We aim to answer the following questions. First, we would like to know if the STI approach can outperform the other methods on a number of diverse datasets and query workloads. Second, given a storage budget, we investigate to what extent could various checkpointing strategies improve the efficiency of the STI family.

4.7.1 Setup

Environment. Our experiments were carried out on a server with 192GB RAM and 2 Intel(R) Xeon(R) CPU X5670 with 6 cores at 2.93GHz running a Linux operating system. We implemented the in-memory versions of CE-EBI, CE-gFS, CE-bgFS, STI, and STI-CP approaches in C++. The length of buckets in CE-bgFS is set to 1000 units of time. We use an in-memory version of BerkleyDB B+tree in which we set the page size to 8KB and use a 12-byte search key to find 8-byte data values.

Query Generation. We consider two methods of query generation: (1) a uniform method, which aims to simulate the workload that has queries with their start times uniformly distributed in time, and (2) a clustering method, which aims to simulate the workload that has queries with their start times clustered in one or more *hotspots* in time. This query generation model requires three parameters to be specified: (1) the number of queries m in the workload, (2) the proportion of the query window size in relation to the entire time domain $\omega \in [0, 1]$, which determines the window size of generated queries[5], and (3) the clustering proportion $\xi \in [0, 1]$, which represents a proportion of clustered queries in a workload. Given ξ, we compute the number of queries that need to be clustered. Denoting the collection hotspots as $\{t_1, t_2, ...t_{m'}\}$ where t_i is the timestamp of the ith hotspot, we assume that each hotspot contains a predetermined number of queries which follow a normal distribution with $\mu = t_i$ and $\sigma = 100$.

The workload in our experiment is composed of above two categories of queries. For each workload, we generate 2000 queries in total which are then split into a training set and a test set, 1000 queries each. The training set is used for workload-aware STI-CPs to learn the clustering structures.

Types of experiments. We run two types of experiments: (1) we process queries with various query window sizes (which are determined by ω in [0,10^{-3},10^{-2},10^{-1},1,10,20]%), to investigate the performance of algorithms in dealing with both long and short queries; and (2) we experiment with datasets of different sizes. The largest dataset used in this experiment has 400 million elements[6] to investigate the scalability of algorithms with respect to the dataset size.

[5] We consider a special case when $\omega = 0$ to generate a workload of *instant* timestamp queries such that query's start time is the same as its end time.

[6] After cleaning, the size of original file for the 400-million dataset is more than 10GB

For each algorithm, we use three metrics to evaluate its efficiency: the average execution time[7] for each query (i.e., its processing cost), memory consumed by indexes (i.e., its storage cost), and index construction time (i.e, its preparation cost). For processing cost, we set the timeout to 10^5 seconds, after which we consider the instance to be not competitive.

For STI-CPs, we carry out two additional types of experiments to investigate the efficiency of various checkpointing strategies: (1) we set the budget B to [0.2, 0.4, 0.6, 0.8, 1]% of the dataset size to investigate the effect of budget size on different checkpointing strategies; (2) we set the clustering proportion ξ to [0.5, 0.8, 0.9, 0.95, 1] to investigate the effect of query hotspots on STI-CP evaluation.

Datasets. We consider four real-world datasets from telecommunication and transportation domains: **Yellow** [57], **CAIDA** [22], **FHV** [57], and **Bike** [64]. **Yellow** records the trips on the yellow taxi in New York City from 2010 to 2018 and each trip is labeled with an interval to represent its duration. **CAIDA** records the anonymized passive traffic traces from Center for Applied Internet Data Analysis (CAIDA) in 2018. Each session is labeled with an interval to represent its duration. **FHV** records the trips on free hired vehicles in New York City in the second half of 2017. **Bike** records the trips on citi-bikes in New York City in from 2013 to 2018.

4.7.2 Results and Analysis

Investigation for STI. We first investigate the effectiveness of our basic algorithm, the STI algorithm. Figure 4.8 reports the processing cost of CE-EBI, STI, CE-gFS, and CE-bgFS with respect to the size of the query window. We note that STI outperforms all of its competitors in the processing cost, especially when the size of query window increases to 0.1%, 1%, 10%, 20% of the time domain. Compared to CE-EBI, STI has lower processing cost in both living history and query window. Compared to CE-gFS and CE-bgFS, STI scales better with increasing size of the query window.

Next, we investigate the scalability of algorithms with respect to the size of the dataset. Figure 4.9 reports processing, storage and preparation costs for the algorithms with respect to the size of the dataset. The datasets for this experiment are obtained by selecting subsets of predetermined sizes from full datasets. We fix the size of the query window to 1% of the time domain

[7]The time cost of enumeration is not included.

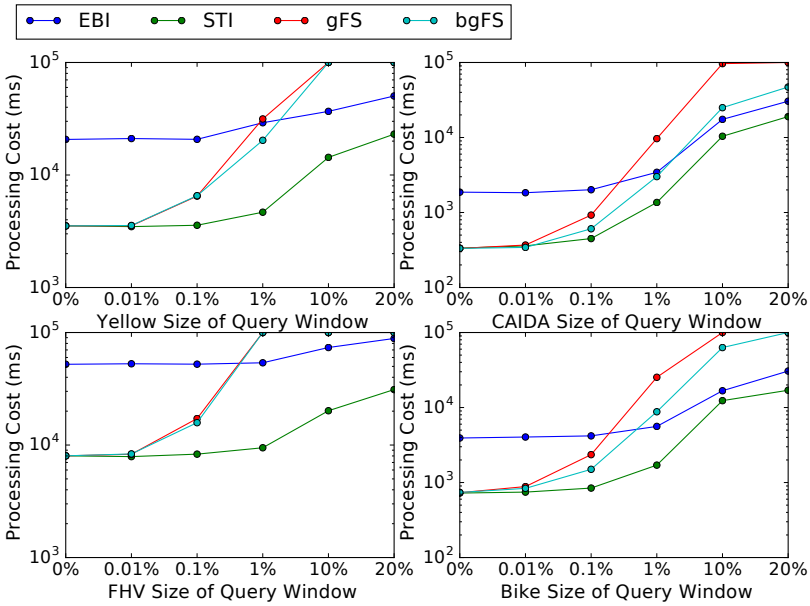


Figure 4.8: Performance for basic algorithms with respect to the query window size

and we set the clustering proportion to 0.9. The result demonstrates that STI scales better than its competitors with respect to the dataset size in all of the measured metrics. This result demonstrates that given the same budget (for index preparation and storage), STI will be significantly more effective than other methods in query processing in both small and large datasets.

Note that the processing cost of STI changes little as the size of query window increases. This clearly demonstrates that the processing cost within the query window is not the efficiency bottleneck for STI approach. In other words, the scanning cost in the living history window takes up the most time in STI's processing. In the rest of this section, we would present how various checkpointing strategies could reduce this cost and further improve the effectiveness of the STI approach.

Investigation for STI-CP Figure 4.10 reports the processing cost of several checkpointing strategies (STI-CPs) with respect to the size of the query window. We set the budget parameter B to 0.6% and clustering proportion to 0.9. We note that the query-set strategy outperforms all other strategies in processing short queries and its advantage diminishes as the size of the query window increases. This is expected because checkpointing strategies aim at reducing

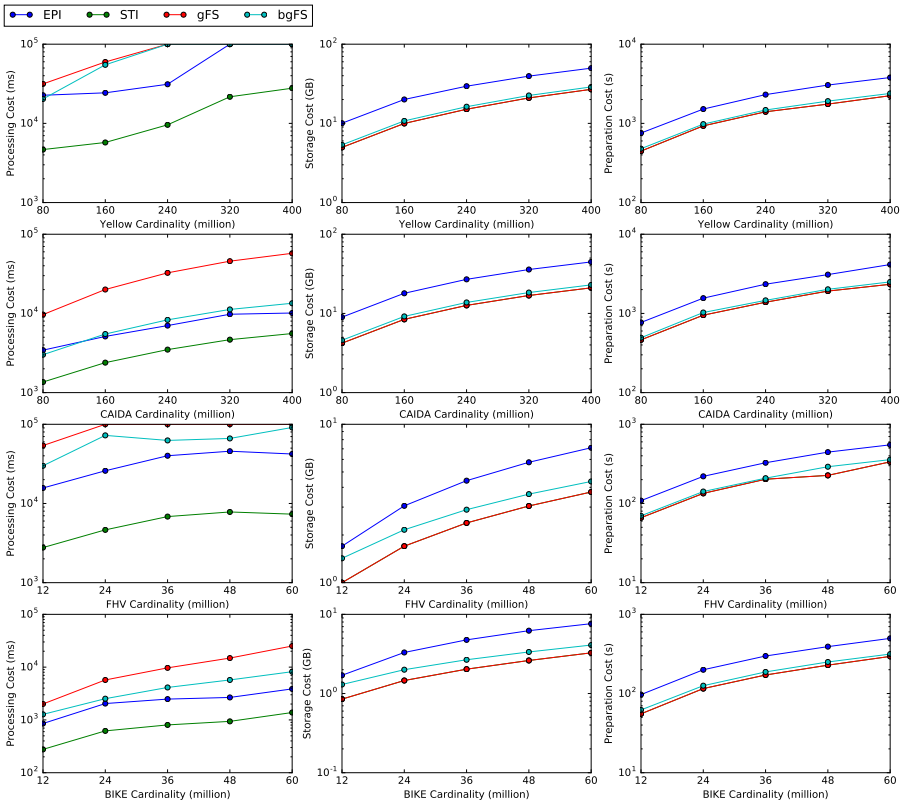


Figure 4.9: Performance of basic algorithms with respect to the dataset size.

the computation cost within the living history window.

In following experiments, we investigate the effectiveness of STI-CPs with respect to the size of the dataset, given checkpoint budget, and the distribution of queries in the workload. We use *instant* queries in these experiments since this isolates the effect of the size of the living history window on checkpointing strategies.

Figure 4.11 reports variation in query processing cost for STI-CPs with respect to the size of the dataset. We note that the three data-aware strategies perform close to each other but all outperform the random strategy. The most important result is that for all STI-CPs, the query-set strategy outperforms the data-aware strategies in both small and large datasets.

We also record the time consumption on checkpoint-selecting of various STI-CPs and the result demonstrates that the query-set strategy needs more

time to select the checkpoints. However, this cost is in the magnitude of milliseconds, which is a very small part of the total preparation cost of the STI index are could be negligible. So they are not reported in figures.

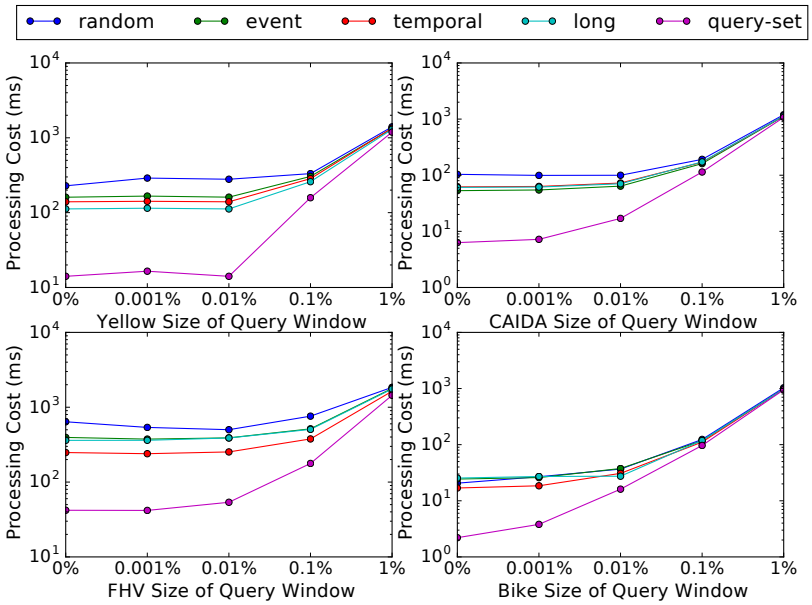


Figure 4.10: Performance of STI-CPs with respect to the size of the query window.

Figure 4.12 reports the performance of STI-CPs with respect to the checkpoint budget B. In most situations, the query-set strategy outperforms other strategies in processing time. The only exception is in FHV with $B = 0.2\%$ where it performs similar to the others. This is expected as average CSS of checkpoints in FHV is much larger than the other datasets, so query-set cannot set enough checkpoints for all clusters when the budget is low. As the budget increases, the advantage of the query-set strategy becomes apparent.

Figure 4.13 reports the performance of STI-CPs with respect to the distribution of queries in a workload. Observe that the query-set strategy performs best when the clustering ratio ξ is in $[0.8, 0.95]$. However, the advantage of the query-set strategy declines when $\xi = 0.5$ and $\xi = 1$. Query-set checkpointing strategy does not perform as well when $\xi = 0.5$ due to lack of clustered queries, hence limited query-aware optimization is possible. In other words, the checkpoints for clusters do not have much influence on overall processing cost. Query-set performance when $\xi = 1$ can be explained by the cluster-detecting method we use. By analyzing the details in checkpoint selection

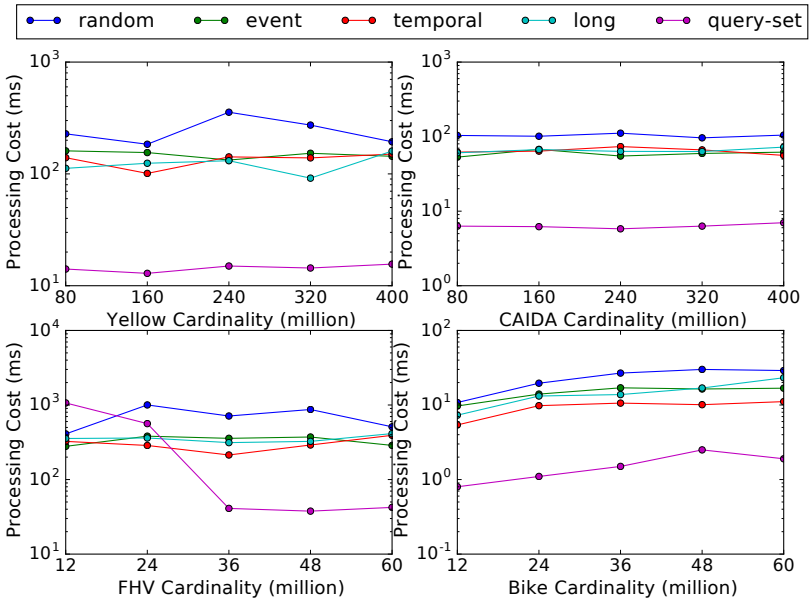


Figure 4.11: Performance of STI-CPs with respect to the dataset size.

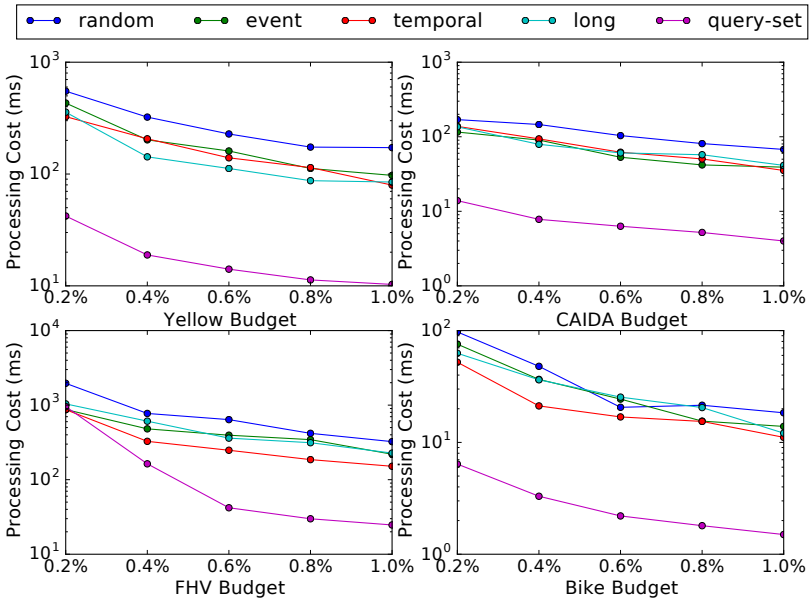


Figure 4.12: Performance for STI-CPs with respect to the budget

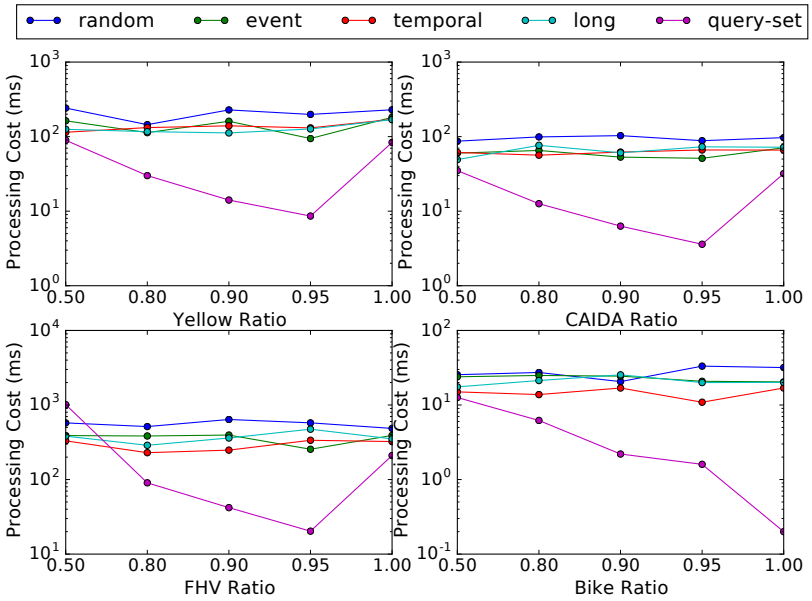


Figure 4.13: Performance for STI-CPs in varying query ratio.

procedure, we find that mean-ISI method cannot properly identify the duration of each cluster when there are no uniform queries in workload. That is, the identified duration does not completely cover all queries in a cluster yet it could identify the correct number of clusters. This is due to the threshold which is used for clustering being smaller than the possible maximum inter-time between two consecutive q_s in the same cluster, when queries are completely clustered. However, when some uniform queries are introduced (i.e., $\xi < 1.0$), the calculated threshold could be lifted so it can not properly cover cluster duration and filter uniform queries.

There are indeed algorithms that outperform the mean-ISI in workload with no uniform queries. For example, cluster duration identified by the histogram-based method proposed in [65] could properly cover all queries in cluster when $\xi = 1.0$. However, this method does not perform well when uniform queries involve as it usually fails to intercept short inter-time between consecutive queries. Besides, the result clearly demonstrates that the mean-value could achieve notable efficiency at low complexity cost. From this experiment, we could conclude that the effectiveness of workload-aware STI-CPs is influenced by: (1) distribution of queries and (2) cluster-detecting method used. More advanced method to identify the query cluster would certainly

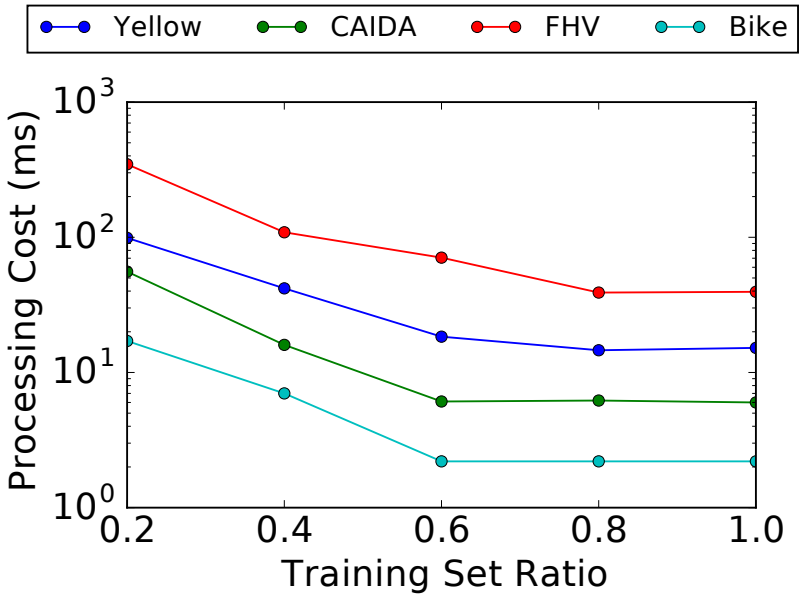


Figure 4.14: Performance for query-set STI-CP with respect to the size of the training set

improve its effectiveness.

To further understand the effect of the chosen clustering method, we carry out an additional experiment to test the performance of the query-set strategy with respect to the size of training set, reported in Figure 4.14. Comparing to the processing cost of other strategies (when $x = 0\%$ in Figure 4.10), we observe that the query-set one begins to outperform the others when training set increases to 0.4 of the test set, and its advantage becomes more stable and apparent when the ratio increases to 0.6-1.0. This is expected because test set used in experiments are small (1000 queries in each) so the smaller training set is not enough for the query-set strategy to learn the clustering structures especially when training set ratio is low. In other words, if we increase the size of processing set, even the training set at the size of 0.2 of processing set could present clear clustering structure information for the query-set strategy.

4.8 Chapter summary

Motivated by its practical significance in various applications and fundamental status in temporal subgraph query processing, in this chapter, we investigate the temporal k-clique enumeration problem, which can be regarded as the general case for interval join processing concerned in the state of the art. We start by proposing a processing framework for the investigated problem based on a linear scan. Next, we present how we could adjust existing interval join algorithms to our investigated problem via our proposed framework. The adjusted algorithms have much lower complexity than the straightforward solution to our problem. Then, we propose STI and STI-CP approaches for more efficient processing. STI is designed to overcome the efficiency bottlenecks in the adjusted algorithms and STI-CP uses checkpoints to further improve processing efficiency. Finally, our experimental results demonstrate that STI outperforms current state-of-the-art methods and all proposed checkpointing strategies outperform the random checkpoint selection method by a wide margin. In the rest of this thesis, we will investigate the temporal subgraph matching problem based on our proposed algorithms in this chapter.

5

Processing of temporal subgraph query

5.1 Motivation

With our developed methods of temporal-predicate processing, we enter the processing problem of temporal subgraph queries: given (1) a temporal graph G where each edge has an associated temporal window; (2) a subgraph query pattern q; and (3) a query time window, find all matches of q in G where the match life-span (i.e., the time interval on which all of the matched edges overlap) is non-empty and overlaps the query time window. Specifically, Figures 5.1 and 5.2 respectively present our examples of graphs G_1, G_2 and queries q_1, q_2, q_3 in this chapter. The subgraph query pattern in q_1, q_2, q_3 are respectively 3-star, 4-chain, 4-circle. And the query time window in q_1, q_2, q_3 are all $[10, 20]$.

As we mentioned, studies of subgraph query processing have primarily focused on the leverage of topological selectivity [66]. However, temporal selectivity in real-world networks can have a significant impact on query processing costs, yet there has been relatively little work on leveraging temporal selectivity in temporal subgraph query processing. Specifically, a straightforward processing approach of temporal subgraph queries is to use the existing pipeline in database systems (i.e., parser, optimizer, and operators), where associated time windows are stored as edge properties in temporal property graph implementation. To be more specific, matches are produced by a generated plan consisting of the following operators.

- ACCESS, which is used to get initial matches of query edges from underlying structures (e.g., edge list).
- JOIN, which is used to concatenate intermediate tuples containing the same vertices to solve topological predicates. Specifically, it can be either binary (e.g., nested-loop, sort-merge, hash) or WCO join operator.

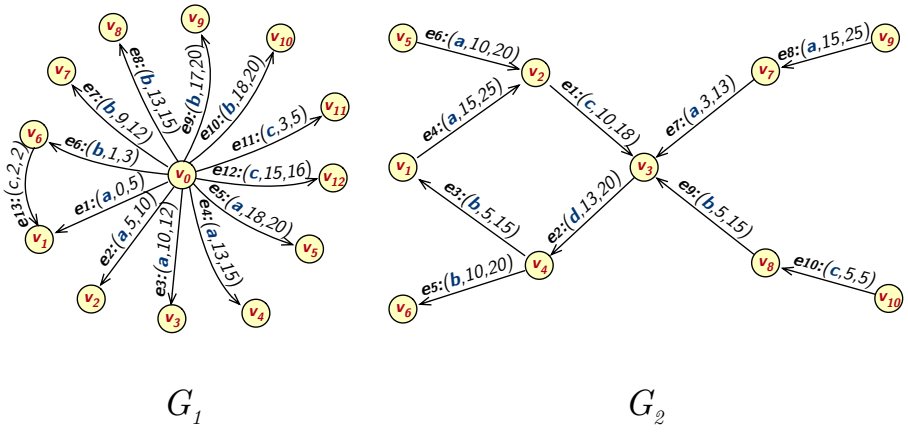


Figure 5.1: Example of temporal graphs in this chapter.

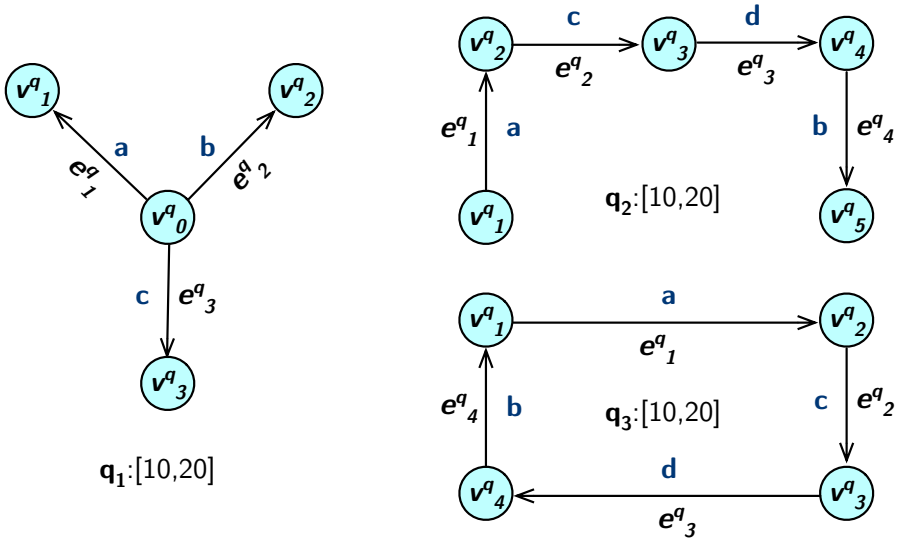


Figure 5.2: Three temporal-clique subgraph queries.

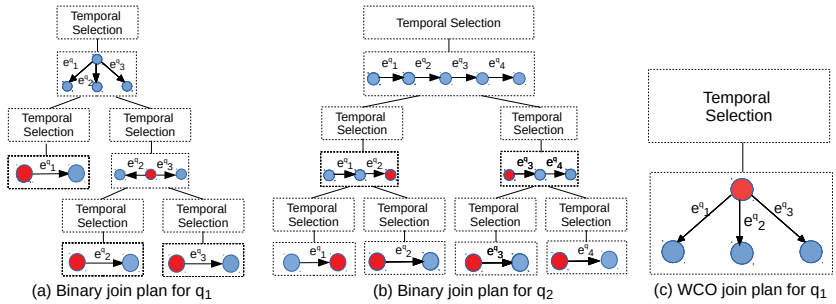


Figure 5.3: Examples of P^{T} processing pipelines, including (a) binary join processing for q_1; (b) binary join processing for q_2; and (c) WCO join processing for q_1; Red vertices highlight the topological joins.

- SELECTION, which is used to filter the intermediate tuples that do not satisfy the temporal predicates (e.g., the overlapping of time windows) and part of topological predicates (e.g., closure in cycle pattern).

To be more specific, Figure 5.3 presents three examples of query processing using the straightforward approach: (a) binary join processing for q_1 over G_1; (b) binary join processing for q_2 over G_2; (c) WCO join processing for q_1 over G_1. Temporal selections follow each topological join to filter the tuples which do not satisfy the temporal predicates. We call this class of methods "to**p**ology then **t**ime" (P^{T}) since the temporal predicates are processed after topological predicates. Such pipelines can be inefficient since the temporal selectivity is not fully leveraged.

5.2 Problem statement

We study the problem of temporal subgraph query processing. Given a temporal graph $G = (V, E, \eta, \lambda, \tau)$ (see Definition 2.1.2) and a temporal subgraph query $q : (e_1, \ldots, e_k) \leftarrow l_1(u_1, v_1), \ldots, l_n(u_k, v_k), [q_s, q_e]$ (see Definition 2.1.7), we aim to find the set of all complete matches of q over G.

Example. Considering query q_1 in Figure 5.2, there are query vertices $v_0^q, \ldots, v_3^q$; query edges $e_1^q = a(v_0^q, v_1^q)$, $e_2^q = b(v_0^q, v_2^q)$, and $e_3^q = c(v_0^q, v_3^q)$; and, query window $[10, 20]$. q_1 aims to return all complete matches of 3-star pattern overlapping $[10, 20]$ in a given temporal graph. For example, given the temporal graphs in Figure 5.1, processing q_1 over G_1 returns (e_4,

e_8, e_{12},$[15, 15]$). Moreover, according to Definition 2.1.9, (e_4, [13,15]), (e_8, [13,15]), (e_{12}, [15,16]), (e_4,e_8, [13,15]), (e_4,e_{12}, [15,15]), and (e_8,e_{12}, [15,15]) are all partial matches of the q_1. According to Definition 2.1.8, $e_1 \dots e_5$, $e_6 \dots e_{10}$, e_{11}, e_{12} are edges matches of e_1^q, e_2^q, e_3^q. According to Definition 2.1.10, $\{e_4, e_8, e_{12}\}$ is a clique match of q.

5.3 Contributions

In this chapter, our contribution can be summarized as follows

- We propose a novel method named TIME which follows a "**t**ime then to**p**ology" (T^P) pipeline. Compared to straightforward P^T methods, TIME focuses more on the leverage of temporal selectivity and can be more efficient in many scenarios, especially when temporal predicates are much more selective than topological predicates.
- Base on a careful analysis of the demerits in TIME method, we propose a novel method, leapfrog TSRJOIN for efficient temporal subgraph query processing following $\mathsf{T\&P}$ pipeline. That is, the method processes both temporal and topological predicates at the same time to fully take advantage of their selectivities and reach the best efficiency.
- We present the results of an in-depth experimental study which demonstrates significant improvement in performance introduced by our new methods.

5.4 Methodology following T^P

5.4.1 Proposed method: TIME algorithm

The procedure of our TIME is shown in Algorithm 7. The cores of the algorithm include (1) a binary join plan $\mathcal{P}$ and (2) an STI index I. $\mathcal{P}$ is the optimal plan to process topological predicates, which can be generated at the beginning based on graph statistics by the existing optimizer in database systems. I is constructed on the relation of edges in the temporal graph, which aims to provide an efficient enumeration of clique matches. We define the following operations to process $\mathcal{P}$:

- $Process(R, \mathcal{P})$: generate all results by processing $\mathcal{P}$ over elements in relation R.

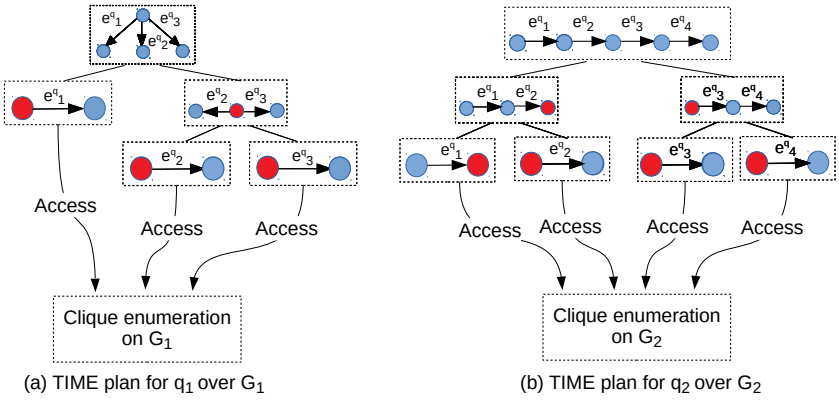


Figure 5.4: The examples of TIME processing example for (a) q_1 and (b) q_2. Red vertices highlight the topological joins.

- $Process(R, \mathcal{P}, r)$: generate all results by processing $\mathcal{P}$ over $R \cup \{r\}$ which contain at least an occurrence of r.

The basic idea of TIME can be summarized as follows: we leverage the framework STI (or STI-CP) algorithm in Chapter 4 for a fast enumeration of all clique matches. Then, we process $\mathcal{P}$ over the enumerated cliques to filter those which are inconsistent with the topological predicates and produce the complete matches. Specifically, TIME starts by determining the beginning of the living history window via STI look-ups, and initializing a linear scan starting from it, with the dedicated structure $Active$ maintained to record the clique matches of q in real-time. When the scanning cursor first reaches the query window, TIME first processes $\mathcal{P}$ over $Active$ to produce the set of complete matches which are formed before q_s. Then, for each scanned edge $curr$, if it is an edge candidate of q, $\mathcal{P}$ is processed over $Active \cup \{curr\}$ to produce all complete matches which contain at least an occurrence of $curr$. The algorithm's condition of terminating is the same as that in the STI algorithm. In this way, the complete result of a temporal subgraph query q can be produced.

Example. Figure 5.4 presents the example of TIME processing for (a) q_1 and (b) q_2. In TIME plans, STI algorithm is first invoked to solve the temporal predicates and produce the clique matches in a given query window. Then, a topological plan is used to produce matches based on the obtained temporal cliques.

Algorithm 7: Basic TIME algorithm

Input: Temporal subgraph query q, STI index I
Output: Complete matches $Result$

```
1  P ← generate the optimal binary join plan of l_1(u_1, v_1) ... l_k(u_k, v_k)
2  startTS ← eC(starttime(I.getRecentStart(q_s)))
3  Active ← ∅,result ← ∅
4  curr ← I.getEntry(startTS)
5  inRange ← false
6  sc ← I.startScan(curr)
7  while curr ≠ NULL do
8      if curr is an edge candidate of q then
9          if starttime(curr) < q_s then
10             if endtime(curr) >= q_s then
11                 insActive(Active, curr)
12         else if starttime(curr) ≤ q_e then
13             if inRange = false then
14                 delActive(Active, q_s)
15                 result ← result ∪ Process(Active, P)
16                 inRange ← true
17             delActive(Active, starttime(curr))
18             result ← result ∪ Process(Active, P, curr)
19             insActive(Active, curr)
20         else
21             break
22     curr ← getNext(sc)
23 if inRange = false then
24     delActive(Active, q_s)
25     result ← result ∪ Process(Active, P)
26 I.stopScan(sc)
```

The complexity of TIME is $O(|E| \cdot \log|E| \cdot \mathcal{F}(\mathcal{P}, Active))$, where $\mathcal{F}(\mathcal{P}, Active)$ is the average processing complexity of $\mathcal{P}$ over $Active$. Compared to the straightforward method following P^{T} pipeline, the TIME algorithm follows a "**t**ime then to**p**ology" (T^{P}) pipeline, which indexes the temporal characteristics of the graph and processes temporal predicates before topological predicates. Specifically, it starts by solving the temporal predicates

via generating all clique matches jointly overlapping in time, and then solve the topological predicates via processing a binary join plan over them. The T^{P} pipeline has advantages in the following aspects: first, as Chapter 3 has reported the existence of numerous CSS valleys and short edge durations in real-world networks, temporal predicates (i.e., joint overlapping) in query can be more selective than topological predicates. Hence, TIME tends to reduce the candidates which are going to be processed by binary join plan and produce smaller intermediate cardinality. Second, TIME decouples temporal-predicate processing from binary plans. That is, SELECTION operations used as temporal filters can be pruned so that the workload in the plan becomes much smaller. In this way, TIME is expected to be more efficient than the straightforward approach following P^{T} pipeline. However, there are several areas in which the TIME algorithm can be further improved, we summarize these opportunities as follows.

- The binary join plan $\mathcal{P}$ cannot always provide the best efficiency of topological-predicate processing since it is generated based on the global statistics of the graph. However, in Algorithm 7, $\mathcal{P}$ is processed over $Active$, in which practical statistics can vary as the linear scan goes on.
- In Algorithm 7, $\mathcal{P}$ is processed every time an edge candidate is encountered in the query window. Such incremental production of matches can introduce numerous computation costs. For example, if hash (or sort-merge) join is used in $\mathcal{P}$, every edge candidate will lead to the reconstruction of hash tables (or the re-sorting of relations) and cause the redundant cost.

In the following, we present additional optimization techniques to address these challenges.

5.4.2 Optimization

In this section, we propose an optimized version of the TIME algorithm, which aims to improve its efficiency in our summarized areas. Compared to basic TIME, the key extensions in optimized TIME include: (1) the division of active-list, and (2) the generation of binary join family.

Division of active list. The general idea of division is that, by aggregating edges which start at the same timestamp $t \in [q_s, q_e]$, we could reduce the computation cost in incremental production. We divide the active-list structure $Active$ into current-list (denoted $Current$) and delta-list (denoted $Delta$).

$Current$ and $Delta$ are both sorted in the same order as in $Active$. Given a timestamp t, $Current$ maintains the edges which start before t, while $Delta$ maintains the edges which newly start at t. Obviously, the union of $Current$ and $Delta$ is equivalent to $Active$ at time t. We make $Current$ and $Delta$ inherent all the operations in $Active$ for maintenance.

With the division, we could optimize the incremental production as follows: For each timestamp $t \in [q_s, q_e]$, we first collect all the encountered edges with $t_s = t$ and insert them into $Delta$. Then, we process $\mathcal{P}$ over $Current$ and $Delta$ with the guidance from a set of *delta queries* [67], denoted $\triangle q_1 \ldots \triangle q_k$. Delta queries are pre-constructed with the aim to guide ACCESS operations whether they should get initial edge matches from $Current$, $Delta$, or $Current \cup Delta$. For convenience, given a delta query $\triangle q$ that is used to guide ACCESS operators in a binary join plan $\mathcal{P}$, we say that $\mathcal{P}$ is processed with $\triangle q$. Let $Result^+(\mathcal{P}, t, i)$ be the matches obtained by processing $\mathcal{P}$ with $\triangle q_i$ at time t. Our constructed delta queries should meet the following two constraints for correctness and efficiency.

- $Result^+(\mathcal{P}, t) = Result^+(\mathcal{P}, t, 1) \cup \ldots \cup Result^+(\mathcal{P}, t, k)$
- $\forall i, j \in [1, k]$, $Result^+(\mathcal{P}, t, i) \cap Result^+(\mathcal{P}, t, j) = \emptyset$ if $i \neq j$.

The first constraint demonstrates the incremental matches at time t should be covered if we process $\mathcal{P}$ with $\triangle q_1 \ldots \triangle q_k$ one after another. The second constraint demonstrates that a match can only be produced once to avoid duplication. Such a set of delta queries can be constructed at a small cost. The procedure of constructing the set of delta queries is shown in Algorithm 8. For ease of expression, here we represent the delta queries in form of tuples. The notations C, D, $C \cup D$ denote that ACCESS operations should get initial edges matches from $Current$, $Delta$, and the union of $Current$ and $Delta$ respectively. To be more specific, in Line 1, the initial delta query $\triangle q \leftarrow \{C \cup D \ldots C \cup D, D\}$ demonstrates that for the $e_1^q \ldots e_{k-1}^q$, ACCESS should get their initial edge matches from the union of $Current$ and $Delta$. While for e_k^q, ACCESS should get its initial edge matches only from $Delta$.

Example. Continuing our 3-star query q_1, $GenerateDeltaQueries(3)$ can be invoked to construct the following set of delta queries.

$$\Delta q_1 : ((C \cup D), (C \cup D), D); \Delta q_2 : ((C \cup D), D, C); \Delta q_3 : (D, C, C)$$

In this way, $\mathcal{P}$ would be processed three times with $\Delta q_1, \Delta q_2, \Delta q_3$ respectively. In the first processing, ACCESS operations for e_1^q, e_2^q, e_3^q obtain their initial edge matches respectively from $Current \cup Delta$, $Current \cup Delta$,

Algorithm 8: GenerateDeltaQueries

Input: number of queried edges k
Output: delta query set DQ

```
1 △q ← {C ∪ D ... C ∪ D, D}
2 DQ ← {△q}
3 while k > 1 do
4 |   △q[k] ← C
5 |   △q[k − 1] ← D
6 |   k ← k − 1
7 |   DQ ← DQ ∪ {△q}
8 return DQ
```

and $Delta$. In the second processing, ACCESS operations obtain their initial edge matches respectively from $Current \cup Delta$, $Delta$, and $Current$. In the final processing, ACCESS operations obtain their initial edge matches respectively from $Delta$, $Current$, and $Current$.

After all the processing of all delta queries is done, we update $Current$ by inserting all edges from $Delta$ into $Current$ to prepare for the incremental production at time $t + 1$. In this way, the computation cost is expected to be reduced since the processing of $\mathcal{P}$ is carried out for each timestamp t instead of each encountered edge in the optimized production.

Binary join family. A binary join family is a set of binary join plans, each of which corresponds to an optimal (or approximate optimal) processing order in a certain scenario. With a binary join family and in each processing, the algorithm can use the local optimal plan according to the real-time statistics in $Active$ (or $Current$ and $Delta$) to reach better efficiency. In the most extreme case, the binary join family should cover all possible plans in space to deal with all possible real-time scenarios. However, such exhaustion of plans would introduce high additional costs. Hence, we implement the binary join family as a trade-off between query processing and plan construction: For each query edge e_i^q in q, we generate a binary join plan which considers e_i^q to be the most selective edge, denoted $\mathcal{P}(e_i^q)$. That is, our constructed binary join family can be denoted as $\{\mathcal{P}(e_1^q) \dots \mathcal{P}(e_k^q)\}$. In our optimized TIME, every time a query (or delta query) is going to be processed, we first determine the most selective query edge e^q based on the real-time statistics in $Active$ (or $Current$ and $Delta$) and then process $\mathcal{P}(e^q)$ to produce the matches. In

this way, the efficiency of processing topological predicates is expected to be improved since in most scenarios, we can choose a plan which is more efficient in real-time than the global optimal plan $\mathcal{P}$, to be processed.

Optimized TIME algorithm. The procedure of our optimized TIME algorithm is presented in Algorithm 9. We define the following method for the plan processing in our optimized TIME algorithm.

- $Process(Current, Delta, BJ, \Delta q)$: start by determining the local optimal $\mathcal{P}'$ from binary join family BJ according to delta query Δq and real-time statistics in $Current$, $Delta$. Then, generate matches by processing $\mathcal{P}'$ with Δq over $Current$, $Delta$.

Compared to Algorithm 7, our optimized TIME algorithm starts by generating the binary join family BJ and delta queries DQ. In the beginning of linear scan, each encountered edge candidate starting before q_s would be directly inserted into $Current$ if it overlaps the query window. When the scanner first enters the query window, optimized TIME chooses a local optimal plan from BJ and processes it with a default delta query $\{C \dots C\}$, which demonstrates that all ACCESS operations should get their initial edge matches from $Current$. Since then, all encountered edge candidates would be inserted into $Delta$ instead of $Current$. The following processing with delta queries is only carried out when the start time of an encountered edge is non-equal to the start time of last encountered edge, which demonstrates all edges with the same start time as the last encountered edge have been inserted into $Delta$. For each delta query $\Delta q \in DQ$, optimized TIME would select a local optimal plan $\mathcal{P}'$ for and then process $\mathcal{P}'$ with Δq. In this way, optimized TIME can produce all matches for query q more efficiently than the basic TIME.

5.4.3 Challenge

TIME algorithms are expected to be more efficient than the straightforward method following P^{T} in many scenarios, especially when the temporal predicates are more selective than the topological predicates. However, they could still be in-efficient as they scan all edges which are not candidates of q within the query window. Such unproductive costs can significantly increase in very large graphs. Besides, more maintenance costs will be introduced if CSS over the time domain is large. The essential reason for these inefficiencies is that TIME algorithms do not take full advantage of the topological selectivity. In some scenarios, topological predicates can be more selective than the temporal

Algorithm 9: Optimized TIME algorithm

Input: Temporal subgraph query q, STI index I
Output: Complete matches $Result$

```
1  Generate binary join family BJ : {P(e_1^q) ... P(e_k^q)}
2  DQ ← GenerateDeltaQueries(n)
3  startTS ← eC(starttime(I.getRecentStart(q_s)))
4  Current ← ∅, Delta ← ∅,result ← ∅
5  curr ← I.getEntry(startTS)
6  inRange ← false
7  lastStart ← −1, sc ← I.startScan(curr)
8  while curr ≠ NULL do
9      if curr is an edge candidate of q then
10         if starttime(curr) < q_s then
11             if endtime(curr) >= q_s then
12                 insActive(Current, curr)
13         else if starttime(curr) ≤ q_e then
14             if inRange = false then
15                 delActive(Current, q_s)
16                 result ←
                     result ∪ Process(Current, Delta, BJ, {C ... C})
17                 inRange ← true
18             else if starttime(curr) ≠ lastStart then
19                 delActive(Current, lastStart)
20                 for Δq ∈ DQ do
21                     result ←
                         result ∪ Process(Current, Delta, BJ, Δq)
22                 Current ← Current ∪ Delta
23             insActive(Delta, curr)
24         else
25             break
26     lastStart ← starttime(curr), curr ← getNext(sc)
27 if inRange = false then
28     delActive(Current, q_s)
29     result ←
         result ∪ Process(Current, Delta, BJ, {C ∪ D ... C ∪ D})
30 I.stopScan(sc)
```

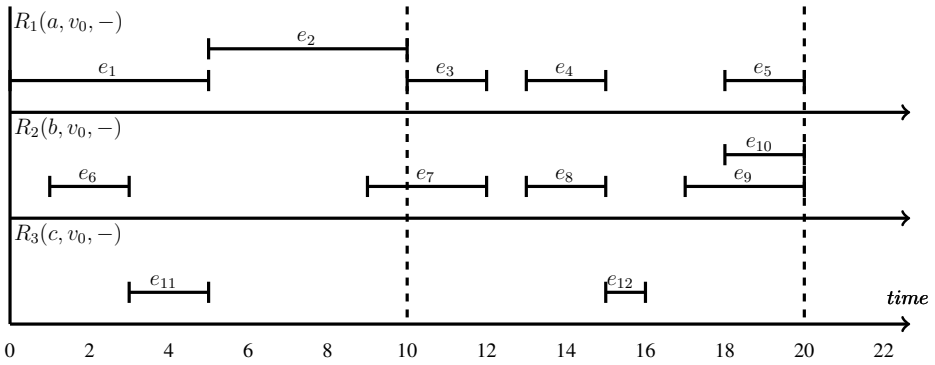


Figure 5.5: Collection of r-TSRs of query edges in q_1 under v_0. Dash lines represent the query window.

predicates (e.g., when some query edges are selective). In the following, we would investigate more efficient query processing methods which fully leverages both topological and temporal selectivities.

5.5 Methodology following T&P

5.5.1 Local notations

We start by presenting our local notations that are going to be used in the rest of the chapter as follows.

Definition 5.5.1 (Temporal selective relation (TSR)) *Given a temporal graph $G = (V, E, \eta, \lambda, \tau)$ and a label set L, a temporal selective relation R in G is a ternary relation $R(l, s, d)$, where*

1. *$l \in L$ is the label constraint,*
2. *s is the source constraint, which can be either a vertex $v \in V$ or $*$ (any vertex), and*
3. *d is the destination constraint, which can be either a vertex $v \in V$ or $*$.*

Let R represent a relation composed by edge $e \in E$ such that $\lambda(e) = l$ and $\eta(e) = (s, d)$. Specifically, $R(l, s, *)$ (or $R(l, *, d)$) denotes all of s's outgoing (or d's in-going) edges associated with label l.

Definition 5.5.2 (Relevant TSR (r-TSR)) *Suppose for a query edge $e^q = l(u^q, v^q)$ that v_1 and v_2 are respectively bindings of u^q and v^q. We say*

$R(l, v_1, v_2)$ is relevant to e^q under v_1, v_2, and $R(l, v_1, v_2)$ is the relevant TSR of e^q under v_1, v_2. If $v_1 = *$ (or $v_2 = *$), we call $R(l, *, v_2)$ (or $R(l, v_1, *)$) the r-TSR of e_q under v_2 (or v_1).

Definition 5.5.3 (Bound r-TSR) *Given a temporal subgraph query q and a binding v^b of query vertex v^q in a partial match of q, a r-TSR R is called a (v^q, v^b)-bound r-TSR in q if there exists a query edge e^q such that*

- *e^q is adjacent to v^q,*
- *e^q has not been matched yet, and*
- *R is relevant to e^q under v^b.*

Example. Considering graph G_1 in Figure 5.1, Figure 5.5 presents, for G_1, three TSRs $R_1(a, v_0, *) = \{e_1, \ldots, e_5\}$, $R_2(b, v_0, *) = \{e_6, \ldots, e_{10}\}$, and $R_3(c, v_0, *) = \{e_{11}, e_{12}\}$, which are respectively composed of v_0's outgoing edges associates with label a, b, and c in G_1. R_1, R_2, and R_3 are respectively r-TSRs of e_1^q, e_2^q, and e_3^q under v_0. If the edge matches of e_1^q, e_2^q, and e_3^q have not been determined, R_1, R_2, and R_3 are all (v_0^q, v_0)-bound r-TSRs in q_1.

5.5.2 Baseline: Leapfrog triejoin

We first describe the TRIEJOIN, which is the baseline used in our apporach. The TRIEJOIN is a WCO-join algorithm that is currently used in several state-of-the-art database systems (e.g., in LogicBlox [68], in AVANTGRAPH [69], and others). The basic idea of a TRIEJOIN is to iteratively extend the determined bindings for query vertices and filter the candidates by looking ahead similar to the depth-first search algorithm. We identify three key ingredients of a TRIEJOIN: (1) the *trie representation*, (2) the *binding production*, and (3) the *binding propagation*. The trie representation indexes the entities (e.g., labels, sources, and targets) of a graph in sorted order so that they can be used as a support for binding production. The binding production determines the vertex bindings in a sort-merge algorithm on a pre-constructed trie by using multi-way intersection and leapfrogging. The multi-way intersection technique joins multiple relations by a series of nested intersections, and leapfrogging technique skips over data that is guaranteed not to result in a binding. The binding propagation hands over the determined bindings to the parent operator of a TRIEJOIN so that they can be further extended in later processing.

We will now present the details of the binding production in TRIEJOIN since it is directly used in our proposed method. Considering that n sorted unary relations (e.g., each containing vertex IDs) are going to be processed,

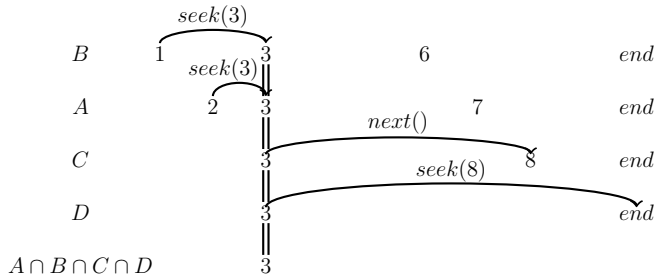


Figure 5.6: Example of binding production procedure for four relations ($n = 4$).

a method named *leapfrog-init()* is first invoked to initialize the relations. *leapfrog-init()* represents each relation by an iterator initially positioned at its first vertex, and then sorts the iterators by their positioned keys in ascending order. Following *leapfrog-init()*, the main workhorse *leapfrog-search()* is invoked to find the next binding in the intersection of the n relations. The basic idea of *leapfrog-search()* is that, in each turn, considering v_{max} is the current highest-value key among the n iterators, the method takes the iterator positioned the lowest-value key and *seeks* to v_{max} in the corresponding relation. If such key value does not exist, the iterator is positioned to the first key that is no smaller than v_{max} and updates the positioned key value as the new v_{max}. Otherwise, the algorithm returns v_{max} as a vertex binding. Subsequent bindings are obtained by invoking a method named *leapfrog-next()*. *leapfrog-next()* first positions current iterator at its next key and then invokes *leapfrog-search()* to find the next binding in the intersection. The procedure is repeated until the vertices in a relation are consumed. In this way, all bindings in the intersection of the k relations are produced.

The overall complexity of TRIEJOIN is $O(Q^* \log M)$, where Q^* is the upper bound of result size and M is the largest cardinality among the unary relations.

Example. Figure 5.6 presents a TRIEJOIN example of four relations $A = \{2, 3, 7\}, B = \{1, 3, 6\}, C = \{3, 8\}, D = \{3\}$. *leapfrog-init()* initially positions their iterators at $2, 1, 3, 3$ respectively and then sorts them by the positioned keys from the first to fourth row. Then *leapfrog-search()* is invoked and returns 3 as a binding. Later *leapfrog-next()* is invoked and no binding is produced until D's iterator comes to its end.

5.5.3 Proposed method: Leapfrog TSRJOIN

We propose an operator named Leapfrog TSRJoin (TSRJOIN, for short) designed for efficient processing of temporal subgraph queries. The basic idea of TSRJOIN is to extend TRIEJOIN into temporal aspects. For each temporal subgraph query, a physical plan composed of TSRJOINs can be constructed for efficient processing. To be more specific, Figure 5.7 presents the TSRJOIN processing plans for q_1, q_2, q_3. We choose TRIEJOIN as our baseline for its excellent performance in processing general conjunctive queries on graphs [70] which correspond to resolving the topological predicates in our investigated queries. Also, TRIEJOIN is easy to be implemented and understood. These merits of TRIEJOIN allow us to concentrate on a remaining challenge: how can we inject an efficient processing of temporal predicates? A straightforward solution is to insert selection operators after each TRIEJOIN as in P^{T}. An example of processing q_1 via the straightforward solution is presented in Figure 5.3(c). However, this solution suffers from *rigidity* in predicate ordering due to its fixed P^{T} order and vertex-at-a-time matching. In Section 5.6, we would further illustrate its inefficiency by detailed experiments. Our ultimate solution to the injection of temporal-predicate processing is to use our STI algorithm, the efficiency of which has been presented through our careful analysis and empirical experiment in Chapter 4.

Comparing to TRIEJOIN, our proposed TSRJOIN is composed of four key components: the *TSR representation*, *binding production*, *partial match production*, and *partial match propagation*. Our method starts from the TSR representation which indexes the TSRs as support for both binding and partial match production. Based on the represented TSRs, the binding production can produce a binding v^b of query vertex v^q as in TRIEJOIN. Using v^b, the (v^q, v^b)-bound r-TSRs $R_1 \dots R_n$ $(n \leq k)$ can be retrieved from the represented TSR. Then, partial match production determines the matches for subgraph composed of v^q's adjacent edges by processing a multi-way STI algorithm over $R_1 \dots R_n$. This, in fact, extends v_b to a series of partial matches, where the endpoints of edge matches are regarded as the bindings of corresponding vertices. Finally, partial match propagation hands over the partial matches to the parent operator so that match's lifespans and bindings can be later used in processing the remainder of the query. In the remainder of this section, we present the details of all the components in TSRJOIN.

TSR representation. Since the aim of TSR representation is to support other components, we should consider the following prerequisites. First, the ordering structure in a trie should be inherited to support the binding production.

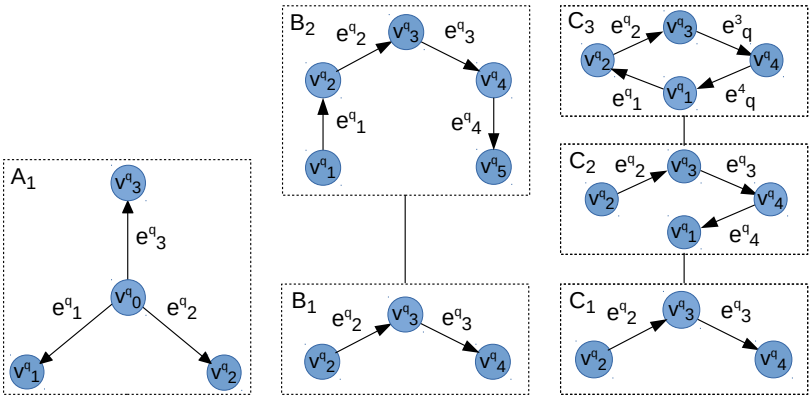


Figure 5.7: Examples of TSRJOIN processing for queries q_1, q_2, and q_3.

Second, the TSRs should also be temporally sorted to support the partial match production in which a multi-way STI algorithm is carried out.

Based on these prerequisites, we propose *temporal adjacency indexes* (TAIs) to represent the TSRs, which are the temporal extensions of a trie in TRIEJOIN and adjacency indexes in a database system. The proposed TAIs are composed of four distinct indexes named temporal LS, LD, LSD, LDS indexes. In each index, edges are categorized by the keys in the order as indicated in its name. The meaning of each capital character in the naming syntax can be summarized as follows: L for edge labels; S for sources; and, D for destinations.

The TAI is constructed as a trie in the order as prescribed by its name thus facilitating efficient topological binding production. Next, corresponding edge-values in a trie are sorted by their start time in ascending order, just like in STI index. As a result, the temporally sorted TSRs can be directly obtained from TAIs, which provides a support for partial match production. Table 5.1 presents the aim of each index.

Example. Figure 5.8 presents the LS and LD of G_1. We note the LS and LD-indexing structure (colored in yellow) in the two indexes have inherited ordering structure from tries, which can provide a support to binding production. Besides, we note that attached TSRs (colored in green) are sorted by start time so that they can be directly fed to multi-way STI algorithm used in latter

Name	Aim
LS	represent temporally sorted $R(l, s, *)$, with LS ordering for binding production
LD	represent temporally sorted $R(l, *, d)$, with LD ordering for binding production
LSD	represent temporally sorted $R(l, s, d)$, with LSD ordering for binding production
LDS	represent temporally sorted $R(l, s, d)$, with LDS ordering for binding production

Table 5.1: Aims of TAIs.

partial match production. More specifically, when necessary, start-time-sorted $R_1(a, v_0, *)$, $R_2(b, v_0, *)$, $R_3(c, v_0, *)$ can be directly obtained and processed with the STI, and also other plane-sweep algorithms.

The construction complexity of TAIs is $O(|E| \cdot (\log |L| + \log |V| + \log |E|))$. To save the cost, LDS can free its attached TSRs while keep the trie structure since $R(l, s, d)$ can still be obtained from the LSD alone.

Binding production. Consider a TSRJOIN operator in a physical plan. If the operator has no child operator (i.e., the bottom operator), binding production is carried out over the relevant trie structure in TAIs, just like in TRIEJOIN. Otherwise, a new trie of partial matches (namely, the PM trie) propagated from the child operator will be first constructed based on the value of vertex bindings that are going to be determined in this operator. To be more specific, if bindings of n different vertices are going to be determined in the operator, a PM trie can be constructed with $h \in [0, n]$, where h is the trie height and reflects the number of different vertices used for construction. A larger h value demonstrates that TSRJOIN relies more on the filtering power of topological selectivity. Then, binding production is carried out over the newly constructed trie and other relevant trie structures in TAIs.

Example. Considering that we are going to process q_1 over G_1 by TSRJOIN, binding production would be first carried out over following three unary relations $\{v_0\}, \{v_0\}, \{v_0, v_6\}$, in order to determine bindings of query vertex v_0^q. These relations are respectively composed of source vertices of a, b, and c-labeled edges in G_1, which can be directly obtained from LS index in Figure 5.8. As a result, v_0 would be produced as a binding of v_0^q

For bottom TSRJOINs, the complexity of binding production in TSR-

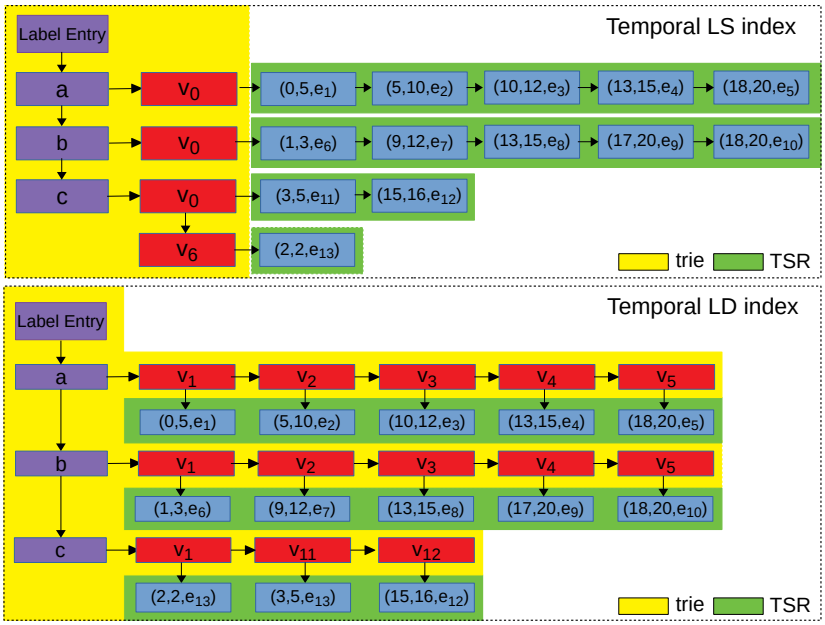


Figure 5.8: LS and LD structures of graph G_1. The yellow and green parts respectively refer to the ordering structure in a trie and TSRs

JOIN is $O(Q^*_{\mathsf{P}} \log M)$. $Q^*_{\mathsf{P}} \log M$ is the complexity of the original TRIEJOIN as presented in Section 5.5.2, where Q^*_{P} refers to the upper bound on the result size of the query with only topological constraints. For non-bottom TSRJOINs, the general complexity of binding production in TSRJOIN is $O(Q^*_{\mathsf{P}} \log M + P \cdot h \log M)$. $P \cdot h \log M$ is the complexity of PM trie construction, where P is the cardinality of the propagated partial matches. It is important to note that if $h = 0$, there will be no binding production, and propagated partial matches will be directly fed to partial match production. Thus, there is no need to construct the PM trie. Besides, it is also important to note that $h = n$ results in a full-sized PM trie to be constructed. A smaller h can be used to reduce the construction cost on PM trie but increase the cost in the other components since the filtering power of topological selectivity is not fully leveraged.

Partial match production. As *leapfrog-search()* has been used as our first workhorse for binding production, here we define our second workhorse

named *leapfrog-temporaloverlap()* (denoted LFTO) for efficient partial match production. Every time a binding v^b of query vertex v^q is determined, LFTO is invoked to process the temporal predicates among (v^b, v^q)-bound r-TSRs, to find matches for v^q's adjacent edges, and to produce the collection of partial matches. Algorithm 10 presents the procedure of LFTO, which can be summarized as a multi-way STI algorithm over bound r-TSRs $R_1 \dots R_n$. Besides bound r-TSRs, a valid time window $[w_s, w_e]$ is used as the input parameter to filter the invalid edges. More specifically, if v^b is determined by the initial *leapfrog-search()*, the valid window should be exactly the query window $[q_s, q_e]$. If v^b is determined by a propagated partial match, the valid window should be its *lifespan*. Two groups of n-scanners are defined to support the plane-sweep on r-TSRs: the $Scan_{cur}$ and $Scan_{end}$. For each R_i, $Scan_{cur}[i]$ refers to the currently scanned edge in R_i. $Scan_{end}[i]$ refers to the ending of the edge-scanning in R_i. To start with, $Scan_{cur}$ is initialized at the first edge in TSRs (Line 2), which represent collection of starting points of the edge-scanners. Similarly, $Scan_{end}$ is initialized at the first edge which starts later than w_e in TSRs (Line 3), which represents the edge-scanning end point in each relation. A dedicated structure $Active$ is maintained to record the edges at current time that can be used to produce partial matches (Line 4). Note that here, edges in $Active$ are categorized by the r-TSR they belong to. Compared to the traditional $Active$ used in prior chapters, the categorzied $Active$ improves the efficiency of maintenance and enumeration. We use $Active[i]$ to represent R_i's currently active edges in $Active$ sorted by their end time in ascending order. We define following operations to maintain the categorized $Active$:

- $insActive(Active, e, i)$: insert the edge e into $Active[i]$;
- $delActive(Active, t)$: delete all edges e s.t. $t > \tau(e).t_e$ from $Active$;
- $enumActive(Active, e)$: enumerate all partial matches over the elements in $Active$ which contains exactly one occurrence of edge e.

In each iteration, the algorithm first obtains the scanner sc (Line 6), the positioned edge of which has the minimal start time, and reads the positioned edge (Line 7). If the edge overlaps the valid window (Line 8), the algorithm deletes the expired edges from $Active$ (Line 9), enumerates the matches in $Active$ containing e (Line 10), and inserts e into $Active$ (Line 11). Finally, the algorithm positions sc to its next edge (Line 12). If sc reaches its scanning end (Line 13), the algorithm closes sc (Line 14). The algorithm keeps iterating until all iterators are closed (Line 5). In this way, LFTO solves the temporal predicates and produces a collection of partial results extended from v^b.

Algorithm 10: Leapfrog temporal overlap

Input: bound r-TSRs $R_1, \ldots, R_n$, time window $[w_s, w_e]$
Output: partial match collection $Result$

for $i \in [1, n]$ **do**
  $Scan_{cur}[i] \leftarrow R_i.begin()$
  $Scan_{end}[i] \leftarrow R_i.upper(w_e)$
$Active \leftarrow \emptyset, Result \leftarrow \emptyset$
while $\exists j \in [1, n]$ s.t. $Scan_{cur}[j]$ is not closed **do**
  $sc \leftarrow Scan_{cur}[i]$ s.t. $\min_{i \in [1,n]} \tau(Scan_{cur}[i].e).t_s$
  $e \leftarrow sc.e$
  if $\tau(e) \cap [w_s, w_e] \neq \emptyset$ **then**
    $delActive(Active, \tau(e).t_s)$
    $Result \leftarrow Result \cup enumActive(Active, e)$
    $insActive(Active, e, i)$
  $sc.next()$
  if $Scan_{cur}[i] = Scan_{end}[i]$ **then**
    Close $Scan_{cur}[i]$ and $Scan_{end}[i]$
return $Result$

Example. Continuing our example in which v_0^q has been bound with v_0, R_1, R_2, R_3 shown in Figure 5.5 are first obtained as the (v_0^q, v_0)-bound r-TSRs. $Scan_{cur}[1, 2, 3]$ are initially set at e_1, e_6, e_{11} and $Scan_{end}[1, 2, 3]$ are set at the end of each relation. The processing procedure is shown in Table 5.2. In this way, the 3-star query q_1 is processed in a single TSRJOIN, as shown in Figure 5.7(a).

The complexity of Algorithm 10 is $O(n \cdot |R_{max}| \log |R_{max}|)$, where $|R_{max}|$ is the cardinality of the largest participating TSR.

Partial match propagation. Given a partial match produced by the LFTO algorithm, partial match propagation allows the match to be handed over to a parent operator so that it can be further extended to the remaining predicates in a query. Comparing to the binding propagation in TRIEJOIN which only hands over the bindings, partial match propagation hands over both bindings and a life-span of the partial result, which fully takes advantage of selectivities of both topological and temporal predicates.

We also propose a *planner* for processing queries with various patterns.

Table 5.2: Example of processing q_1 over G_1 using LFTO algorithm.

Edge	*Active*	*Enumerate*
e_1	$\emptyset$	$\emptyset$
e_6	$\emptyset$	$\emptyset$
e_{11}	$\emptyset$	$\emptyset$
e_2	[1]:$\{e_2\}$	$\emptyset$
e_7	[1]:$\{e_2\}$,[2]:$\{e_7\}$	$\emptyset$
e_3	[1]:$\{e_2, e_3\}$, [2] : $\{e_7\}$	$\emptyset$
e_4	[1]:$\{e_4\}$	$\emptyset$
e_8	[1]:$\{e_4\}$,[2]:$\{e_8\}$	$\emptyset$
e_{12}	[1]:$\{e_4\}$,[2]:$\{e_8\}$,[3]:$\{e_{12}\}$	$(e_4, e_8, e_{12}, [15, 15])$
e_9	[2]:$\{e_9\}$	$\emptyset$
e_{10}	[2]:$\{e_9, e_{10}\}$	$\emptyset$
e_5	[1]:$\{e_5\}$,[2]:$\{e_9, e_{10}\}$	$\emptyset$

The general idea of the planner is to first select the most selective query vertex and carry out iterative extension from it until all query edges and vertices are included. We use the following equation as the cost model to evaluate the selectivity of each query vertex:

$$cost(v^q) = \frac{\prod_{e^q \in N(v^q)} P_L(e^q)}{|N(v^q)| \cdot (1 + |N'(v^q) - N(v^q)|)}$$

where $N(v^q)$ is the collection of edges adjacent to v_q, $N'(v^q)$ is the collection of edges adjacent to edges in $N(v^q)$, and $P_L(e^q)$ is the proportion of edges in the graph that share the same label with e^q. The numerator of the cost model corresponds to the selectivity of the star centered at v_q with respect to labels. In the denominator, $|N(v^q)|$ corresponds to the number of query edges that are going to be processed in the first TSRJOIN. $1+|N'(v^q)-N(v^q)|$ corresponds to the potential for the v_q centered star to be further extended. Using this cost model, the v_q with smallest $cost(v^q)$ is selected and the processing order in the query is also determined. In this way, a plan composed of TSRJOIN operators for query processing can be generated.

Example. Consider we aim to process the 4-chain query q_2 and 4-circle query q_3 (shown in Figure 5.2) over G_2 (shown in Figure 5.1) with PM trie height $h = 0$. Figures 5.7(b) and 5.7(c) present the physical plans for processing. The plan for 4-chain query q_2, is composed of two TSRJOINS B_1 and B_2. B_1 produces the partial matches of the v_3^q-centered 2-star pattern (i.e. (e_2^q, e_3^q)) since v_3^q is the most selective query vertex according to our cost model. This determines edge matches for e_2^q, e_3^q and bindings for v_2^q, v_3^q, v_4^q. B_2 extends each partial match produced by B_1 with e_1^q, e_4^q so that the complete result of

q_2 can be processed. To be more specific, consider in B_1, *leapfrog-search()* has determined v_3 a binding of v_3^q. The following LFTO algorithm would process a 2-way interval join over $R(c, *, v_3)$, $R(d, v_3, *)$. From the interval join, (e_2^q,e_3^q) are matched with (e_1,e_2) and $(e_1, e_2, [13, 18])$ is produced as a partial match of q_2. The partial match in fact determines v_2,v_4 to be the bindings of v_2^q,v_4^q respectively. Based on the partial match, B_2 would process a 2-way interval join over $R(a, *, v_2)$, $R(b, v_4, *)$, matches (e_1^q, e_4^q) with (e_4, e_3), (e_4, e_5), (e_6, e_3), (e_6, e_5), and produce complete matches (e_4,e_1,e_2,e_3,[15,15]), (e_4,e_1,e_2,e_5,[15,18]), (e_6,e_1,e_2,e_3,[13,15]), (e_6,e_1,e_2,e_5,[13,18]). In this way, the complete result of q_2 over G_2 is produced.

Similarly, the plan for q_3 is composed of three TSRJOINs C_1, C_2, and C_3. C_1 produces the matches of the v_3^q-centered 2-star pattern as in q_2. Then C_2 extends each match propagated from C_1 with e_4^q to find edge matches for e_4^q and determine bindings for v_1^q. Finally, C_3 extends each match propagated from C_2 with e_1^q to produce complete matches for q_3. To be more specific, consider C_1 has produced the partial match $(e_1, e_2, [13, 18])$ as in q_2. C_2 extends the match with $R(b, v_4, \emptyset)$ and produces partial matches $(e_1, e_2, e_3, [13, 15])$,$(e_1, e_2, e_5, [13, 18])$. C_3 further extends the two partial matches with $R(a, v_6, v_2)$, $R(a, v_1, v_2)$ to close the circles. Finally, only $(e_4, e_1, e_2, e_3, [15, 15])$ is produced as a complete match of q_3. In this way, $(e_4, e_1, e_2, e_3, [15, 15])$ is finally produced as the complete result of q_3.

Challenges. The worst-case complexity of a TSRJOIN plan is $O(Q_{\mathsf{P}}^* \log M + Q_{\mathsf{P}}^* \cdot n_{max} \cdot |R_{max}| \log |R_{max}| + P \cdot n_{max} \log M)$, where $Q_{\mathsf{P}}^* \log M$ is the complexity of original TRIEJOIN ; $Q_{\mathsf{P}}^* \cdot n_{max} \cdot |R_{max}| \log |R_{max}|$ is the additional complexity introduced by the temporal overlap; and $P \cdot n_{max} \log M$ is additional cost of filtering based on the cardinality P of partial matches produced by TSRjoins. It is important to note that, in order to keep the upper bound parameter Q_{P}^*, PM tries in all non-bottom TSRJOIN should be full-sized.

However, according to our empirical observation, TSRJOINs with full-sized PM tries do not provide the best efficiency in most scenarios, though it fully leverages the topological selectivity. This is because that the efficiency of binding production in non-bottom TSRJOIN is in fact a trade-off between topology filtering overhead (i.e., cost on PM trie construction and leapfrogging interaction) and filtering power (i.e., the cardinality or proportion of filtered matches). Specifically, if additional filtering overhead exceeds the improvement via filtering power, TSRJOINs will become less efficient. Besides, in non-bottom TSRJOINs, temporal filtering power (i.e., partial match production) can be strong enough for filtering because of the numerous short intervals

in real world networks, as we discussed in Chapter 3. Specifically, this generally leads to the extremely short life-spans of produced partial matches, which can be fast filtered in partial match production if they are irrelevant. Summarizing, we argue that, from the level of practical application, the PM trie height h should be set with the aim to result in a trade-off between topology filtering overhead and power, in order to provide the best efficiency.

Compared to the existing methods following P^{T} and our prior TIME method following T^{P} pipelines, our TSRJOIN approach takes full advantage of both topological and temporal selectivities. Also, the logarithmic complexity of the TSR representation and partial match production guarantees that little additional cost is introduced in query processing. In this way, TSRJOIN is expected to be more efficient in temporal subgraph query processing than both P^{T} and T^{P} (i.e., TIME) approaches. However, there are several areas in which the efficiency of TSRJOIN can be further improved. We summarize these opportunities as follows:

- Many irrelevant edges can be scanned in partial match production. Continuing our running example in Table 5.2, e_1, e_6, e_{11}, e_2, e_7, e_3, e_9, e_{10}, e_5 are all irrelevant edges since only $(e_4, e_8, e_{12}, [15, 15])$ is produced as a match. This demonstrates that the edges can introduce significant scanning cost in processing selective queries.
- The enumeration of new partial matches can be costly since each scanned edge would lead to the invoking of $enumActive$, which normally traverses almost the whole $Active$.

In summary, while TSRJOIN addresses the inefficiencies in existing methods there are several possibilities for further acceleration of query processing. In the following, we present additional optimization techniques to address these challenges.

5.5.4 Optimization

Skipping irrelevant edges. We start by categorizing the irrelevant edges by the time they are scanned (i.e., start time). We call the irrelevant edges scanned before the first partial match is produced the *backward* edges. We call the irrelevant edges scanned after the last partial match is produced the *forward* edges. Continuing our running example, $\{e_1, e_6, e_{11}, e_2, e_7, e_3\}$ and $\{e_9, e_{10}, e_5\}$ are respectively the collection of backward and forward edges.

We start by presenting our methods of skipping backward edges. Remember that in Chapter 4, we first define the *earliest concurrent* to reduce the scan-

ning cost in temporal k-clique enumeration and store it as a property of each tuple. Here, we also consider to skip backward edges in TSRs by using their *earliest concurrent*. However, the "storing as property" can be in-efficient because of the heavy-tail phenomena of real world duration distribution, which has been discussed in Chapter 3. Specifically,the existence of extremely long intervals can make numerous temporally consecutive tuples share the same earliest concurrent. This generally leads to large storage redundancy since each edge is stored for at least three copies in TAIs. Hence, we propose *early coverage indexes* (ECIs) for more efficient earliest concurrent retrieving in TSRs. ECIs are composed of three indexes LS-EC, LD-EC, LSD-EC, which are respectively used to record the earliest concurrent distribution of TSRs in form of $R(l, s, *)$, $R(l, *, d)$, and $R(l, s, d)$. For each TSR, earliest concurrent distribution is represented by a series of early coverage tuples. Each early coverage tuple is formalized as $\theta : (c_s, c_e, ec)$, representing that the earliest concurrent of each time $t \in [c_s, c_e]$ is ec. For convenience of retrieving, tuples for each TSR are sorted by c_s in ascending order. We provide the following interface to perform a look-up in ECIs:

- $getCoverageTuple(R, t)$, given a timestamp t and a TSR R, retrieve the coverage tuple from corresponding ECI such that $t \in [c_s, c_e]$. Else, if no such tuple exists, return the first coverage tuple from corresponding ECI such that $c_s > t$. Otherwise, return $\emptyset$.

Example. Figure 5.9 presents the structure of LS-EC and LD-EC, where the early coverage tuples are categorized in the same way as in LS and LD. Using $getCoverageTuple(R(a, v_0, *), 1)$, tuple $(0, 5, 0)$ in LS-EC is returned. $(0, 5, 0)$ represents that in TSR $R(a, v_0, *)$, the earliest concurrent of time $t \in [0, 5]$ is 0. In this way, we obtain that $eC(1) = 0$ in $R(a, v_0, *)$. Similarly, using $getCoverageTuple(R(b, *, v_1), 1)$, $(1, 3, 1)$ in LD-EC is returned.

The construction complexity of ECIs are $O(|L| + |V|^2 + |E| + \Theta \cdot \log \Theta)$, where Θ is the number of coverage tuples. Comparing to "storing as property", ECIs store the earliest concurrent in form of early coverage tuples. This guarantees *distinct* earliest concurrent value is only stored once in a relation, which significantly improves the efficiency of storage.

Using ECIs, Algorithm 11 presents our final method to skip backward edges by computing the optimized starting points for $Scan_{cur}$s. The basic idea of the algorithm is to find the first collection of coverage tuples $\theta_1 \dots \theta_n$ for $R_1 \dots R_n$ such that the intersection of $[\theta_1.ec, \theta_1.c_e] \dots [\theta_n.ec, \theta_n.c_e]$ is not empty. Based on the notion of earliest concurrent, it can be deduced that each

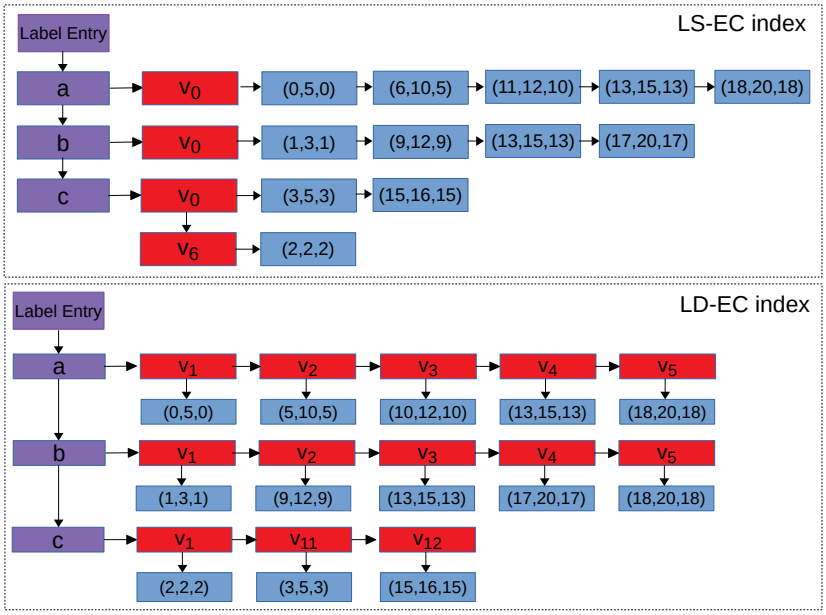


Figure 5.9: LS-EC and LD-EC structures of graph G_1.

coverage tuple θ_i reveals that the longest interval in R_i starting at $t = \theta_i$.ec is $[\theta_i.\text{ec}, \theta_i.c_e]$. In this way, the first non-empty $[\theta_1.ec, \theta_1.c_e] \cap \cdots \cap [\theta_n.ec, \theta_n.c_e]$ in fact demonstrates that the first match is going to be produced at the maximal time among $\theta_1.ec \ldots \theta_n.ec$. Also, edges in R_i with start time no smaller than $\theta_i.ec$ can be relevant to partial matches. In this way, $Scan_{cur}[i]$ can start from the $\theta_i.ec$, instead of very beginning of R_i.

Example. Continue our running example and use Algorithm 11 with $w_s = 10$. In the first round, the collection of early coverage tuples $(6, 10, 5), (9, 12, 9), (15, 16, 15)$ for R_1, R_2, R_3 will obtained. Since $[5, 10] \cap [9, 12] \cap [15, 16]$ is empty, the algorithm will start the second round with $t = 15$, i.e., the maximal ec among the tuples. Then, the collection of tuples $(13, 15, 13), (13, 15, 13), (15, 16, 15)$ for R_1, R_2, R_3 will be obtained. We note that $[13, 15] \cap [13, 15] \cap [15, 16]$ equals to $[15, 15]$, which implies that the first match is going to be produced at $t = 15$. Thus, $Scan_{cur}[1, 2, 3]$ can respectively start from e_4, e_8, e_{12} instead of e_1, e_6, e_{11}. In this way, the irrelevant edges $e_1, e_6, e_{11}, e_2, e_7, e_3$ can be skipped.

Algorithm 11: OptimizeStartPoint

Input: r-TSRs $R_1, \ldots, R_n$, start time of valid window w_s
Output: optimized starting point $t_1^s, \ldots, t_n^s$

```
1  t ← w_s
2  while true do
3      for i ∈ [1, n] do
4          θ_i ← getCoverageTuple(R_i, t_0)
5          if θ_i = ∅ then
6              return −1, ... − 1
7      if [θ_1.ec, θ_1.c_e] ∩ ... ∩ [θ_n.ec, θ_n.c_e] ≠ ∅ then
8          break
9      t ← max_{i∈[1,n]} θ_i.ec
10 return θ_1.ec, ..., θ_n.ec
```

Algorithm 12: delSkip

Input: active list $Active$, timestamp t, scanner list $Scan_{cur}$
Output: $false$ if subsequent edges-scanning are non-productive

```
1  for i ∈ [1, n] do
2      for e ∈ Active[i] do
3          if τ(e).t_e ≥ t then
4              break
5          Active[i] ← Active[i]/{e}
6      if Active[i] = ∅ and Scan_cur[i] is closed then
7          return false
8  return true
```

To skip the forward edges, we propose *delSkip* operation as a replacement of *delActive*. Besides removing expired edges from $Active$, *delSkip* can additionally identify and skip some forward edges. Algorithm 12 presents the procedure in *delSkip*. If the operation deletes all edges in an $Active[i]$ and finds $Scan_{cur}[i]$ is closed, it returns *false* indicating that the following scanned edges are all forward edges and should be skipped.

Example. Continuing our running example, when e_9 starting at $t = 17$ is scanned, *delSkip* is first invoked to remove the expired edges e_4, e_8, e_{12} from

$Active$. Since $Active[3]$ becomes empty after *delSkip* and R_3 has been closed when e_{12} is scanned, *delSkip* returns $false$ which indicates later edge-scans are irrelevant and should be stopped. In this way, the scanning on irrelevant edges e_5, e_{10} can be skipped.

Lazy enumeration. We introduce lazy enumeration to reduce the traversal cost on $Active$ in partial match production. Given current timestamp t and scanner $Scan_{cur}[i]$, the basic idea of lazy enumeration is not to carry out enumeration until all edges with $t_s = t$ in R_i have been scanned by $Scan_{cur}[i]$. A dedicated structure named *candidate list* (C) is maintained to record the edges starting at current time. For each edge e scanned by $Scan_{cur}[i]$, if $\tau(e).t_s = t$, algorithm adds e into C instead of traversing $Active$ to enumerate partial results containing e. Otherwise, if $\tau(e).t_s > t$ or sc is switched to another r-TSR, algorithm traverses $Active$, enumerates partial matches containing each element in C, and finally cleans all elements in C. We define the following operation to replace $enumActive$ and support lazy enumeration in LFTO.

- $enumLazy(Active, C)$: enumerate all partial matches over the elements in $Active$, which contains an occurrence of edge in C if $C \neq \emptyset$.

Considering m edges starting at time t, the complexity of enumeration at time t in Algorithm 10 is $O(m \cdot \prod_{j=1}^{i-1} |Active(j)| \cdot \prod_{j=i+1}^{n} |Active(j)|)$. Using the lazy enumeration, the complexity is reduced to $O(\prod_{j=1}^{i-1} |Active(j)| \cdot \prod_{j=i+1}^{n} |Active(j)|)$. In this way, the traversal cost on $Active$ is significantly reduced.

Optimized LFTO. Using our proposed optimization, Algorithm 13 presents the procedure of optimized LFTO method. Compared to the original LFTO in Algorithm 10, the optimized LFTO first invoke $OptimizeStartPoint$ (Algorithm 11) to find the starting points for each TSR (Line 1). If there are no valid starting points (Line 2), algorithm will be directly stopped since this demonstrates no valid match can be produced (Line 3). $enumLazy$ is only carried out when (1) either edges starting at a time point are all collected, or (2) current scanned TSR is switched (Line 16). Finally, algorithm will also be stopped if $delSkip$ (Algorithm 12) returns $false$ (Line 21). In this way, the scanning and enumeration costs in Algorithm 13 are significantly reduced according to our analysis in this section. In this way, the performance of the TSRJOIN is improved.

Example. Continuing our running example, the q_1 processing procedure in

Algorithm 13: Optimized Leapfrog temporal overlap

Input: bound r-TSRs $R_1, \ldots, R_n$, valid time window $[w_s, w_e]$
Output: Complete matches $Result$

$t_s[1, \ldots, n] \leftarrow OptimizeStartPoint(R_1 \ldots R_n, w_s)$
if $t_s[1] = -1$ **then**
  return $\emptyset$
for $i \in [1, n]$ **do**
  $Scan_{cur}[i] \leftarrow R_i.lower(t_s[i])$
  $Scan_{end}[i] \leftarrow R_i.edges.upper(w_e)$
$Active \leftarrow \emptyset, Result \leftarrow \emptyset, C \leftarrow \emptyset$
$inRange \leftarrow false, t' \leftarrow 0, i' \leftarrow -1$
while $\exists j \in [1, n]$ s.t. $Scan_{cur}[j]$ is not closed **do**
  $sc \leftarrow Scan_{cur}[i]$ s.t. $\min_{i \in [1,n]} \tau(Scan_{cur}[i].e).t_s$
  $e \leftarrow sc.e$
  if $e.t_s < w_s$ **then**
    if $e.t_e >= w_s$ **then**
      $insActive(Active, e, i)$
  else
    if $t' \neq e.t_e$ or $i' \neq i$ **then**
      if $inRange = false$ **then**
        $Result \leftarrow Result \cup enumLazy(Active, \emptyset)$
        $inRange \leftarrow true$
      else
        if $delSkip(Active, t', Scan_{cur}) = false$ **then**
          break
        $Result \leftarrow Result \cup enumLazy(Active, C)$
        $C \leftarrow \emptyset$
    $insActive(Active, e, i), C \leftarrow C \cup \{e\}$
  $t' \leftarrow e.t_s, i' \leftarrow i$
  $sc.next()$
  if $Scan_{cur}[i] = Scan_{end}[i]$ **then**
    Close $Scan_{cur}[i]$ and $Scan_{end}[i]$
return $Result$

Table 5.2 can be significantly optimized by using Algorithm 13, as shown in Table 5.3.

Edge	*Active*	*Enumerate*
e_4	[1]:{e_4}	$\emptyset$
e_8	[1]:{e_4},[2]:{e_8}	$\emptyset$
e_{12}	[1]:{e_4},[2]:{e_8},[3]:{e_{12}}	$(e_4, e_8, e_{12}, [15, 15])$
e_9	[2] : {e_9}	$\emptyset$

Table 5.3: Example of the optimized LFTO algorithm.

5.6 Experiments

In this section, we present an experimental study of TIME and TSRJOIN. From the investigation, we would like to understand the performance of TIME and TSRJOIN compared with the current methods in processing queries with various patterns, result size, window lengths, and network cardinality, over a number of diverse real-world networks.

5.6.1 Setup

Environment. Our experiments were carried out on a server with 192GB RAM and 2 Intel(R) Xeon(R) CPU X5670 with 6 cores at 2.93GHz running a Linux operating system. We implemented the in-memory versions of TIME and TSRJOIN in AVANTGRAPH.[1] In all experiments, we used vectorized execution and set the tuple output (i.e., the maximal number of tuples produced in each pull) of each operator to 1024.

Competitors. We use two standard state-of-the-art methods named HYBRID and BINARY following P^T as competitors to TIME and TSRJOIN. These two competitors are all implemented in AVANTGRAPH. In BINARY, a plan composed of binary joins and selection operators is used for each query's processing. In HYBRID, however, both binary and TRIEJOIN are used since *hybrid* plans are considered to be more efficient in many situations in non-temporal subgraph query processing [71]. Note that HYBRID is a straightforward extension of TRIEJOIN which allows to process temporal subgraph queries. In

[1] AVANTGRAPH is a new-generation graph processing engine developed in the Database Group at TU Eindhoven. For more information, please refer to `http://avantgraph.io`.

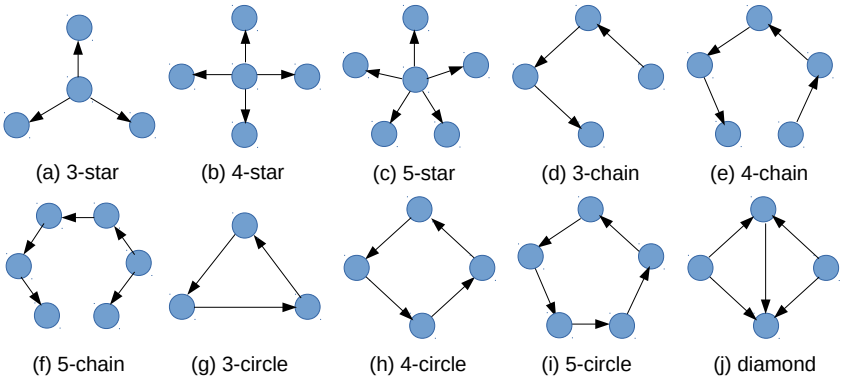


Figure 5.10: Subgraph patterns used in the experimental evaluation.

both BINARY and HYBRID, label adjacency indexes[2] are constructed to fully cover the access patterns during topological subgraph matching. All implementations are single-core.

Query generation. Our query generation model requires the following parameters to be specified: (1) the number of queries N in the workload, (2) the proportion of the size of the query window in relation to the entire time domain $\omega \in [0, 1]$, (3) the pattern type to be queried, (4) the edge label set L, and (5) the maximal result size M of each query (this is used to avoid extremely long running queries). Figure 5.10 presents the patterns used in our experimental evaluation. For each k-sized query, k distinct elements are uniformly drawn from L as the associated labels of query edges respectively. Then we process the query and add it to the workload if its result size is in $[1, M]$. For each workload, 100 queries are generated in our experiment. The M-parameterized uniform generating method can produce the workloads with appropriate selectivity in short time.

Types of experiments. We run four experiments to investigate the performance of the TIME and TSRJOIN: (1) We investigate the performance of algorithms with respect to query patterns shown in Figure 5.10. (2) We process the queries with maximum output sizes in [1K,10K,100K,1M,10M] to investigate the performance of algorithms with respect to query selectivity. (3) We process queries with varying query window ω in [10^{-2},10^{-1},1,10,20]% to investi-

[2]Label adjacency index is a B-tree structure where graph edges are sorted in LSD or LDS order to represent the non-temporal adjacency.

Name	$\|V\|$	$\|E\|$	$\hat{T}$
Yellow	261	20,000,000	89,741
Green	262	30,000,000	1,636,978
Bike	455	20,000,000	1,207,485
Divvy	1,097	20,000,000	3,245,197
Stack	2,465,111	20,000,000	961,820
CAIDA	31,379	714,016	121

Table 5.4: Overview of the real-world networks used in the experiments.

gate the scalability of algorithms in dealing with both long- and short-window queries. (4) We experiment with real-world networks of different sizes. The largest network used in our experiment has 100 million edges.

For each algorithm, we use the average execution time (i.e., its processing cost) and memory consumed by indexes (i.e., its storage cost) to evaluate its efficiency. Each workload is set to timeout in 10^5 seconds.

Datasets. Table 5.4 presents the overview of our six real-world temporal graphs from transportation, social, and networking domains. **Yellow** [57], **Green** [57], **Bike** [64], **Divvy** [72] record the trips from source to destination. **Stack** [73] records the interactions among users in StackOverflow. **CAIDA** [21] records the relationships among autonomous systems. Note that in original **Stack** each interaction is only associated with a time-instant representing its activity time. To make the interactions durable , we associate each interaction with an interval $[t, t + 1000]$ where t represents its activity time. The "seting fixed duration" method is commonly used in social networks.

5.6.2 Results

Figure 5.11 presents the processing cost of HYBRID, BINARY, TIME, and TSRJOIN with respect to queried patterns in various networks. We fix the length of query window to 10% of the time domain. The maximal result size of each query is set to $100K$ tuples. We note that the TSRJOIN outperforms all of its competitors in all situations since it fully leverages both topological and temporal selectivities. Also, we note that TSRJOIN becomes less efficient in **Bike** and **Divvy**, in which edge intervals are much smaller than those in **Yellow** and **Green**. This is because of existence of more shorter intervals which are more likely to become irrelevant compared to long intervals as we use

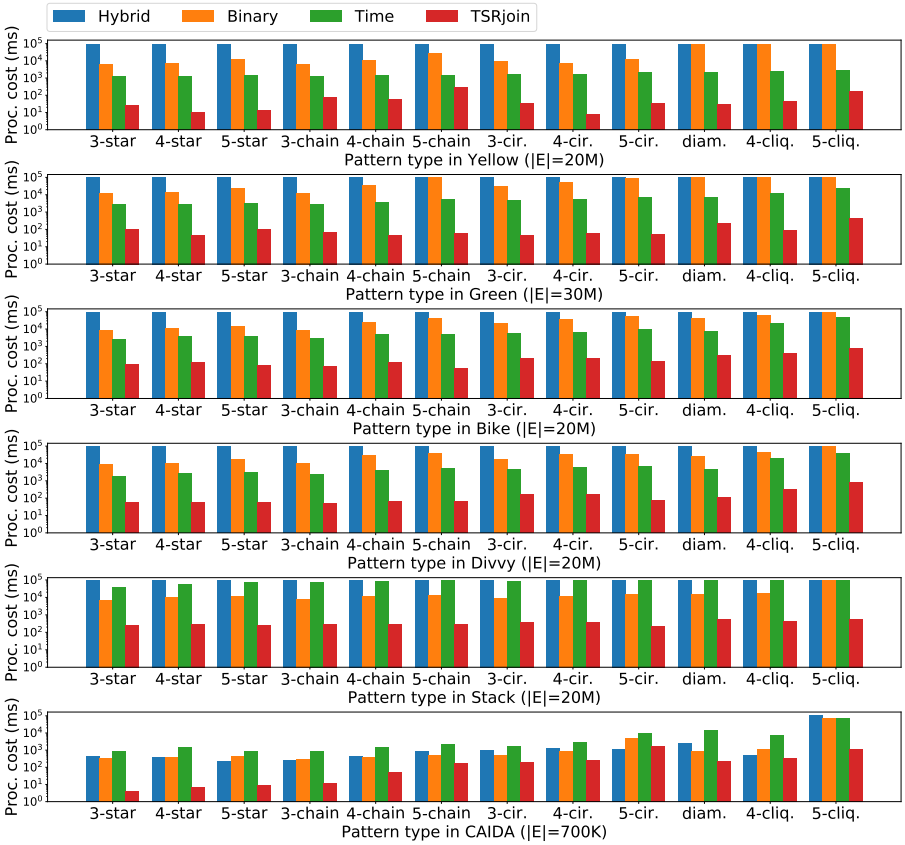


Figure 5.11: Performance of algorithms with respect to pattern types.

sweep-plane interval join algorithms. For TIME, we note that it outperforms two P^{T} methods in the four transportation networks, but loses the advantage in **Stack** and **CAIDA**. This is because the topological predicates are more selective than the temporal predicates in the two datasets. To be more specific, in **Stack**, the fixed duration at 1000 gives rise high CSS which increases the active-list maintenance cost in TIME. In **CAIDA**, the existence of extreme long interval which covers the whole domain increases the scanning cost in TIME. This demonstrates that TIME, which follows T^{P}, can only outperform the methods following P^{T} when temporal predicates are selective. Finally, we note that HYBRID performs the slowest processing in most cases, where all of its instances have timed out. This is expected because HYBRID allows for efficient processing of the topological part of the query by using TRIEJOIN, but effectively disallows the *injection* of temporal predicates into the TRIEJOIN.

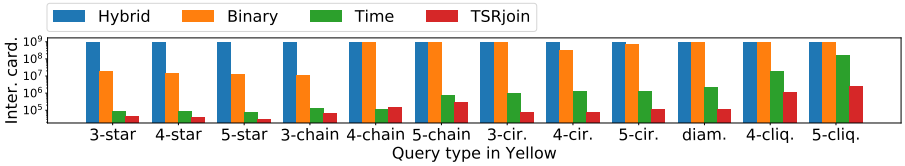


Figure 5.12: Intermediate cardinality of various subgraph patterns in **Yellow** dataset ($|E| = 20M$).

To further investigate the findings above, we carry out an additional experiment, in which the total intermediate result cardinality is used to evaluate each instance. A threshold of 10^9 tuples is set for produced intermediate cardinality in each instance, after which an instance would be forcefully stopped. To save space, Figure 5.12 presents the intermediate cardinality of the additional experiment in **Yellow** with query output size fixed to 1000, as an example. We first note that TSRJOIN produces the smallest intermediate cardinality so that it can outperform its competitors in most situations. We also note the intermediate cardinality produced in TIME vs. TSRJOIN while their difference in processing cost is more significant. This is due to the scanning of the irrelevant edges in STI-CP and the significant construction cost of the hash table especially in selective queries. Finally, we note that at most 2% of workload in HYBRID has completed which explains the inefficiency of the HYBRID approach in the previous experiment.

As TSRJOIN has performed its significant advantage over the other methods, we would primarily concentrate on its performance in the rest of experiments. Figure 5.13 presents the performance of methods with respect to query selectivity, where only results of 4-star, 4-chain, 4-circle in transportation networks are presented as examples. We note that the TSRJOIN outperforms the other methods in most situations. The only exception is the chain processing in **Yellow** network, in which our TSRJOIN becomes less efficient than TIME as selectivity decreases. This demonstrates that a TSRJOIN-only plan generated by our proposed planner is not suitable in processing non-selective chain queries. To illustrate this, consider a variation q' of our 4-chain query q_2 in which e_1^q, e_2^q, e_4^q are far more selective than e_3^q. Consider a TSRJOIN plan q' in which the bottom operator processes the 2-star sub-query centered at v_2^q for its selectivity. The intermediate cardinality produced by the next operator would be large because partial matches produced by the bottom operator could only be further extended with e_3^q which is regarded as non-selective. This example demonstrates that the inefficiency in chain processing is due to the low-arity of chain pattern, which leads to limited choices for TSRJOIN to bypass non-

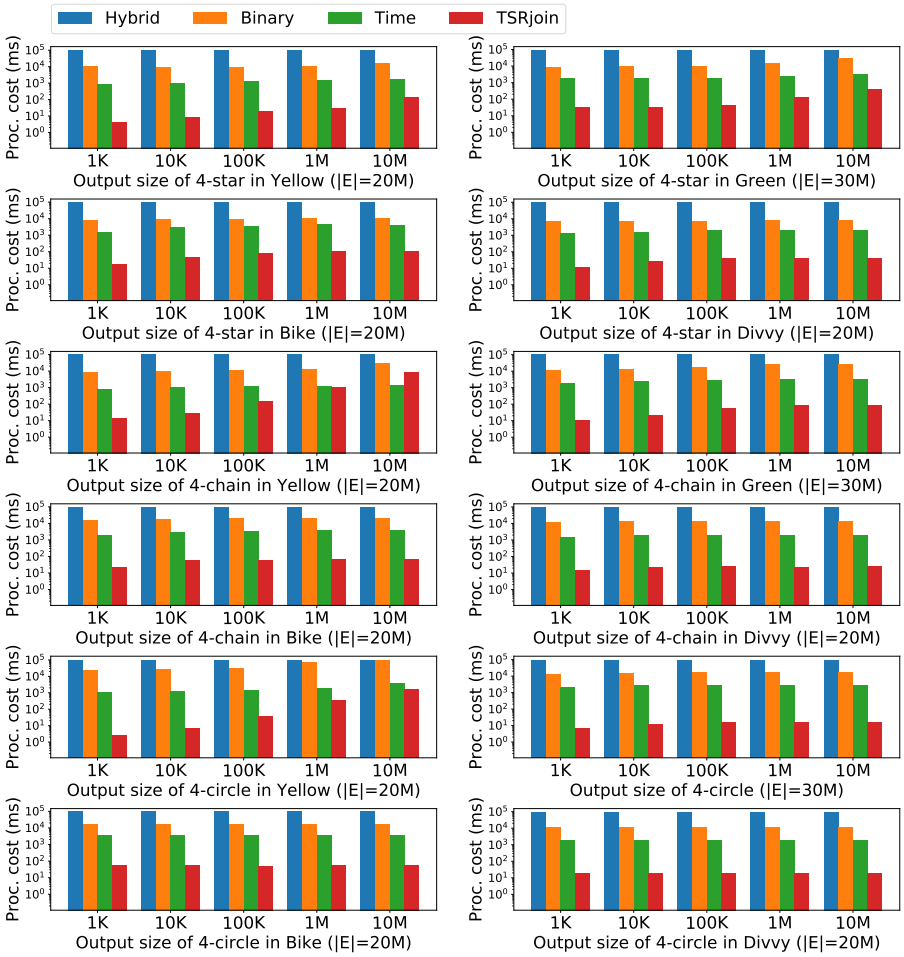


Figure 5.13: Performance of algorithms with respect to query output size.

selective edges. We defer the optimization of such queries to future work.

Figure 5.14 presents the performance of methods with respect to query length which shows that TSRJOIN scales better than its competitors. Similarly, Figure 5.15 presents the performance of methods with respect to network size. The networks in this experiment are obtained by selecting subsets of predetermined sizes from full networks. The result demonstrates that TSRJOIN scales better than its competitors with respect to network size.

Finally, Table 5.5 presents the index storage cost of each algorithm in various networks. We note that in most networks, TSRJOIN requires at most twice

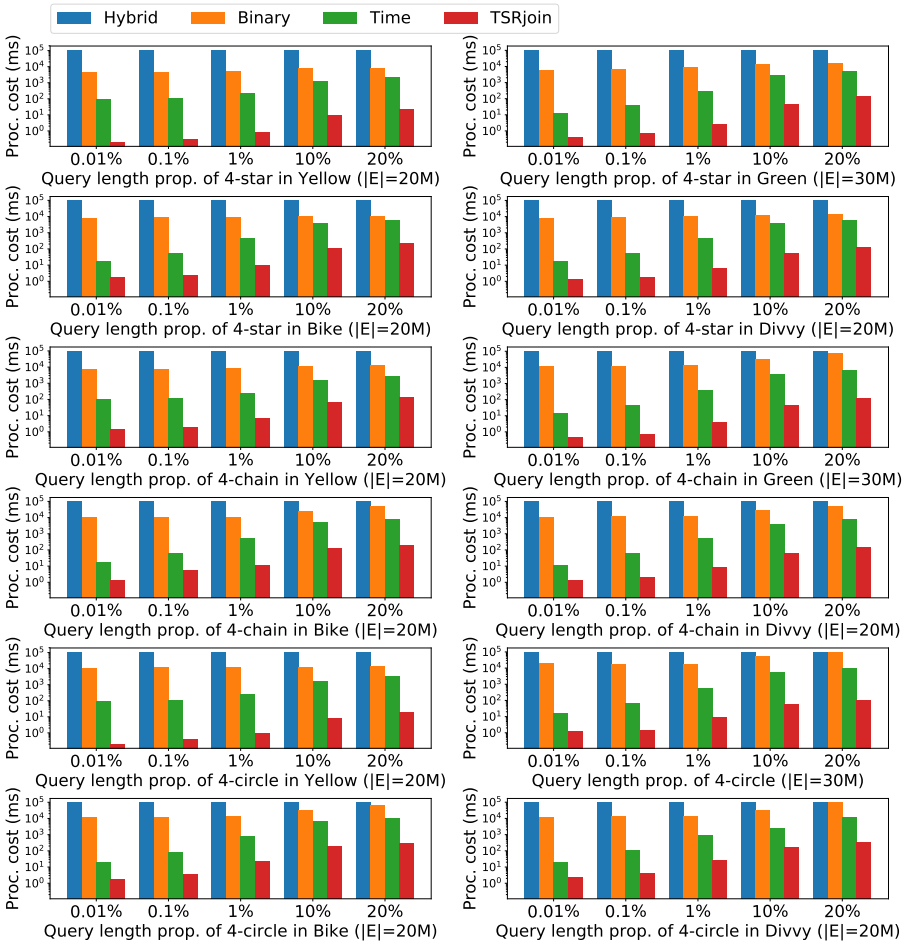


Figure 5.14: Performance of algorithms with respect to the query window.

as much space that its competitors in the worst case. And, in **Stack**, TSRJOIN requires higher space consumption. This is expected since TSRJOIN materializes additional structures to support efficient query processing. Comparing to BINARY and HYBRID in which label adjacency index is constructed, TSRJOIN additionally constructs LS and LD trie structure, a copy of edges, and ECIs for partial result production. Comparing to TIME in which STI-CP index is constructed, TSRJOIN stores additionally two copies of edges and the trie structures. Generally, the storage cost of the three copies is significantly compressed since the trie structure is used to index the edges sharing labels and endpoints. Moreover, ECIs significantly reduce the redundancy in the ear-

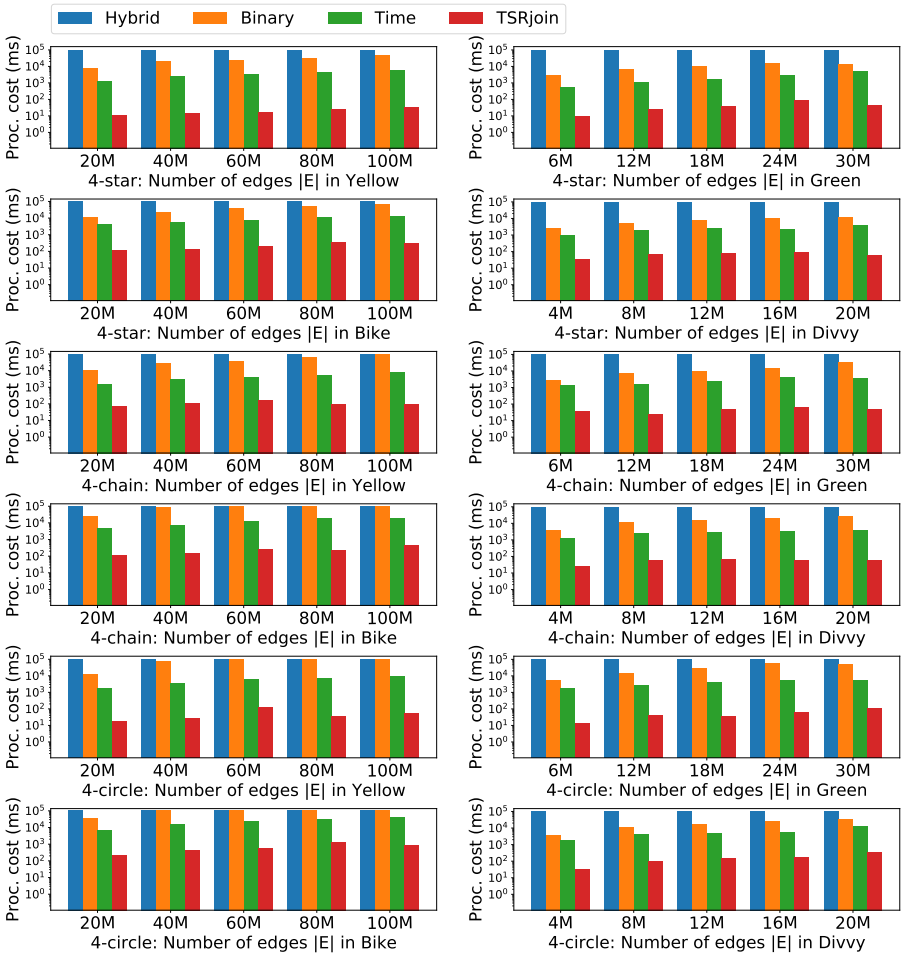


Figure 5.15: Performance of algorithms with respect to network size.

liest concurrent storage, which also improves the efficiency of index storage in TSRJOIN. However, when the number of vertices become much larger, more additional storage cost is introduced due to increase in the size of trie structures. We also present the pre-processing cost on index construction in Table 5.6, from which the similar conclusion can be drawn. Summarizing, we consider the time-space trade-off introduced by the TSRJOIN to be very reasonable as processing cost is decreased by several orders of magnitude for a small index construction and storage overhead.

Method	BINARY	HYBRID	TIME	TSRJOIN
Index	label adjacency	label adjacency	STI-CP	TAIs+ECIs
Yellow	4.5	4.5	5.0	7.4
Green	6.4	6.4	7.5	11.8
Bike	4.5	4.5	5.2	9.1
Divvy	4.5	4.5	4.8	9.0
Stack	5.9	5.9	4.5	17.4
CAIDA	0.2	0.2	0.2	0.4

Table 5.5: Storage cost of algorithms in various networks (GB).

Method	BINARY	HYBRID	TIME	TSRJOIN
Index	label adjacency	label adjacency	STI-CP	TAIs+ECIs
Yellow	60.3	61.2	87.7	118.9
Green	97.5	88.2	125.7	172.0
Bike	86.0	81.0	84.9	145.7
Divvy	86.3	77.2	79.7	133.4
Stack	66.6	67.8	270.0	196.8
CAIDA	1.2	1.2	7.0	4.1

Table 5.6: Pre-processing cost of algorithms in various networks (secs).

5.7 Chapter summary

Since prior works for temporal subgraph query processing primarily follow the P^{T} pipeline which ignores the leverage of temporal selectivity and can be inefficient in many scenarios, we propose two methods for efficient temporal subgraph query processing. We start by proposing TIME which primarily focuses on the leverage of temporal selectivity. Then we propose TSRJOIN which fully leverages both topological and temporal selectivities to provide better efficiency. Empirical experiments demonstrate that our proposed methods outperform current methods by a wide margin with a small additional storage cost. Thus, our TSRJOIN approach can provide efficient temporal subgraph query processing.

6 Conclusion

6.1 Research summary

Subgraph query processing arises in many application domains (e.g., social networks, life sciences, smart cities, telecommunications, and others), which guides a better understanding of graph-structured systems in the real world. However, existing research have primarily focused on non-temporal subgraph query processing, ignoring that real-world systems are generally dynamic and time-related. Though several investigations on temporal subgraph query processing have been carried out, the proposed methods primarily focused on the leverage of topological selectivity. Such methods can be inefficient in many scenarios, especially when temporal predicates are more selective. Thus, in this thesis, we present a comprehensive study on temporal subgraph query processing, intending to improve the efficiency in the state of the art.

Temporal network modeling. In Chapter 3, we outlined our findings for temporal networks and proposed the CDM method for their modeling and generation. CDM aims to generate the networks constrained by a specific CSS distribution, with other important characteristics (e.g., degree, IET, duration) captured meanwhile. To the best of our knowledge, this is the first method for modeling and generating CSS-constrained networks. Theoretical analysis and experimental evaluation demonstrated that CDM is a controllable benchmark that can simulate real network characteristics and efficiently generate various synthetic networks. Thus, CDM is effective and practical to model the temporal networks in the real world (Q1).

Temporal-predicate processing. In Chapter 4, we first proposed a general framework for the temporal k-clique enumeration problem. Based on the framework, the state of the art in interval join processing (i.e., EBI, gFS, bgFS)

can be adjusted to our investigated problem, which has much lower complexity than the straightforward solution. Then, based on a careful analysis of the adjusted approaches, we proposed the STI algorithm to provide more efficient query processing. STI combines the advantages of different adjusted algorithms and addresses almost all of their inefficiencies. We further proposed the STI-CP algorithm which aims to solve the remaining inefficiency in STI. STI-CP incorporates the checkpoints to reduce the scanning cost on living history in STI. We carried out an in-depth discussion on the checkpointing problem with a limited storage budget. Specifically, we discussed four checkpointing strategies and highlighted their benefits. Experimental evaluation demonstrated significant improvements in scalability and performance introduced by STI and STI-CP. Thus, our proposed methods can provide efficient temporal-predicate processing (Q2).

Temporal subgraph query processing. In Chapter 5, based on an analysis of the demerits in existing methods following the P^{T} pipeline, we first proposed our TIME method, which follows a T^{P} pipeline and focused on the leverage of temporal selectivity. Then, based on a careful analysis of TIME, we proposed TSRJOIN, which follows a $\mathsf{T\&P}$ pipeline and aims to improve the processing efficiency by leveraging both topological and temporal selectivity. Experimental evaluation demonstrated that our proposed methods, especially the TSRJOIN, outperform current methods by a wide margin with a small additional storage cost. Thus, our proposed methods can provide efficiently processing for temporal subgraph queries (Q3)

This thesis and its contributions are foundational in nature and lay the groundwork for a potentially rich set of future contributions. In the following, we will discuss some interesting topics for further investigation.

6.2 Future works

Based on the contributions of this thesis, we are particularly interested in the following directions:

6.2.1 Temporal network modeling

Theoretical investigation. In Chapter 3, we discussed the importance of CSS distribution and its impact on network generation. Currently, there are still many other important characteristics (e.g., temporal closeness and between-

ness [74], community structure [11]) for understanding temporal networks but are hardly investigated in the state of the art. In future work, we will investigate the modeling methods to capture these characteristics. The investigation is expected to help us better understand temporal networks.

Powerful generator. CDM intends to generate networks constrained by various CSS with several important characteristics (e.g., degree, IET, duration) captured. In future work, we will investigate more powerful generators for more various and complex networks. Specifically, we expect that our generator can generate (1) temporal networks containing various subgraph embeddings (e.g.,patterns listed in Figure 5.10); and (2) query workloads that are meaningful to generated networks. In this way, we will finally develop a more powerful benchmark which can generate realistic networks with more complex underlying structures and corresponding query workloads for evaluation.

6.2.2 Temporal-predicate processing

Predicate processing. In Chapter 4, the basic unit of our investigated temporal predicates is the interval "overlap" predicate, which is first introduced in Allen's Algebra [75]. Allen's Algebra is a defined set of binary relationships for reasoning about intervals and has been widely used in various applications. Besides Allen's Algebra, ISEQL [76] defines a set of parameterized binary relationships. However, the contemporary processing of these general predicates in database systems can be very inefficient. In future work, we will investigate the enumeration problem of subsets constrained by other temporal predicates in Allen's Algebra and ISEQL. Studying the processing of various temporal predicates will provide us a more general temporal-predicate processing that can be used for graph analysis in various applications.

Checkpointing strategies. We presented that checkpoints can be used to improve the processing efficiency in temporal k-clique enumeration. However, our investigated checkpointing strategies are all heuristic. Though the NP-completeness of optimal checkpointing problem can be theoretically proved, it is still interesting to investigate the optimal (or approximate optimal) checkpointing strategies to further improve enumeration efficiency. Besides, our best checkpointing strategy *query-set* assumes the aggregation structure in query workload. Such an assumption might be too strong in some scenarios. Our future work for checkpointing includes the following two sub-tasks: (1) We will investigate the problem of optimal checkpointing strategy to find a trade-off between checkpointing and processing. (2) We will investigate the assumption-

independent checkpointing strategy, which aims to improve the processing efficiency in more general cases.

6.2.3 Temporal subgraph query processing

Query optimizer. In Chapter 5, experimental evaluation demonstrated that our novel operator TSRJOIN outperforms existing methods but can still be less efficient in several scenarios. This is because our optimizer for TSRJOIN plan generation is too naive. We expect that an improved optimizer can realize the potential of TSRJOIN and generate a smarter plan for efficient processing. Our future work for optimizers includes the following two sub-tasks: (1) Given a temporal subgraph query, we will investigate the cost estimation problem. The estimation aims to evaluation query costs based on both topological and temporal selectivities. (2) With proposed cost estimation approaches, we will investigate the pipeline to generate a smarter plan (e.g., better joining order, incorporation of bushy structure) for more efficient query processing.

General processing. Compared to queries which might involve a more flexible definition of topology structure and various temporal predicates, our investigated query is a very specific case since it involves only the conjunctive format of topology definition and *overlap* predicates. Further investigation on other cases is of great interest to provide efficient processing for more general temporal subgraph queries. Our future work of general query processing includes the following two sub-tasks: (1) We will investigate the efficient processing of queries with various temporal predicates in Allen's Algebra and ISEQL involved. To start with, this can be carried out by extending TSRJOIN. Later, we also need to extend the optimizer with the cost estimation on general temporal predicates. (2) We will investigate the efficient processing of *regular path queries* in a temporal context, which provides a more flexible and navigational definition of topology structure. Specifically, we will first propose efficient traversal algorithms on temporal graphs as our query processing foundation. Then, we will focus on decomposing the path query to further leverage the query selectivity and improve the processing efficiency.

Bibliography

[1] *SQL:2011,* https://en.wikipedia.org/wiki/SQL:2011#Temporal_support (2011).

[2] *Neoj4,* https://en.wikipedia.org/wiki/Neo4j.

[3] R. Shamir and D. Tsur, *Faster subtree isomorphism,* Journal of Algorithms (1999).

[4] D. Piatov, S. Helmer, and A. Dignös, *An interval join optimized for modern hardware,* in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on* (IEEE, 2016) pp. 1098–1109.

[5] P. Bouros and N. Mamoulis, *A forward scan based plane sweep algorithm for parallel interval joins,* PVLDB **10**, 1346 (2017).

[6] P. Holme, *Epidemiologically optimal static networks from temporal network data,* PLoS Computational Biology **9**, e1003142 (2013), doi:10.1371/journal.pcbi.1003142.

[7] L. Speidel, R. Lambiotte, K. Aihara, and N. Masuda, *Steady state and mean recurrence time for random walks on stochastic temporal networks,* Physical Review E **91**, 012806 (2015), doi:10.1103/physreve.91.012806.

[8] N. Perra, B. Gonçalves, R. Pastor-Satorras, and A. Vespignani, *Activity driven modeling of time varying networks,* Scientific Reports **2**, 469 (2012), doi:10.1038/srep00469.

[9] L. Alessandretti, K. Sun, A. Baronchelli, and N. Perra, *Random walks on activity-driven networks with attractiveness,* Physical Review E **95**, 052318 (2017), doi:10.1103/physreve.95.052318.

[10] E. Ubaldi, A. Vezzani, M. Karsai, N. Perra, and R. Burioni, *Burstiness and tie activation strategies in time-varying social networks,* Scientific Reports **7**, 46225 (2017), doi:10.1038/srep46225.

[11] G. Laurent, J. Saramäki, and M. Karsai, *From calls to communities: a model for time-varying social networks,* The European Physical Journal B **88**, 301 (2015), doi:10.1140/epjb/e2015-60481-x.

[12] H. Kim, M. Ha, and H. Jeong, *Scaling properties in time-varying networks with memory,* The European Physical Journal B **88**, 315 (2015), doi:10.1140/epjb/e2015-60662-7.

[13] M. Nadini, K. Sun, E. Ubaldi, M. Starnini, A. Rizzo, and N. Perra, *Epidemic spreading in modular time-varying networks,* Scientific Reports **8**, 2352 (2018), doi:10.1038/s41598-018-20908-x.

[14] A. Sunny, B. Kotnis, and J. Kuri, *Dynamics of history-dependent epidemics in temporal networks,* Physical Review E **92**, 022811 (2015), doi:10.1103/physreve.92.022811.

[15] D. T. Gillespie, *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions,* Journal of Computational Physics **22**, 403 (1976), doi:10.1016/0021-9991(76)90041-3.

[16] M. Boguná, L. F. Lafuerza, R. Toral, and M. Á. Serrano, *Simulating non-markovian stochastic processes,* Physical Review E **90**, 042108 (2014), doi:10.1103/physreve.90.042108.

[17] N. Masuda and L. E. Rocha, *A gillespie algorithm for non-markovian stochastic processes,* SIAM Review **60**, 95 (2018), doi:10.1137/16m1055876.

[18] M. Starnini, A. Baronchelli, and R. Pastor-Satorras, *Modeling human dynamics of face-to-face interaction networks,* Physical Review Letters **110**, 168701 (2013), doi:10.1103/physrevlett.110.168701.

[19] Y.-Q. Zhang, X. Li, D. Liang, and J. Cui, *Characterizing bursts of aggregate pairs with individual poissonian activity and preferential mobility,* IEEE Communications Letters **19**, 1225 (2015), doi:10.1109/lcomm.2015.2437382.

[20] Y.-S. Cho, A. Galstyan, P. J. Brantingham, and G. Tita, *Latent self-exciting point process model for spatial-temporal networks,* Discrete and Continuous Dynamical Systems **19**, 1335 (2014), doi:10.3934/dcdsb.2014.19.1335.

[21] *The caida as relationships dataset, 2004-2007,* `https://www.caida.org/data/as-relationships/` ().

[22] *CAIDA UCSD anonymized internet traces dataset - [dates used],* `http://www.caida.org/data/passive/passive_dataset.xml` ().

[23] R. Görke, R. Kluge, A. Schumm, C. Staudt, and D. Wagner, *An efficient generator for clustered dynamic random networks,* in *Design and Analysis of Algorithms - First Mediterranean Conference on Algorithms, MedAlg 2012, Kibbutz Ein Gedi, Israel, December 3-5, 2012. Proceedings*, Lecture Notes in Computer Science, Vol. 7659, edited by G. Even and D. Rawitz (Springer, 2012) pp. 219–233.

[24] W. van Leeuwen, A. Bonifati, G. H. Fletcher, and N. Yakovets, *Stability notions in synthetic graph generation: a preliminary study.* in *Proceeding of the 20th International Conference on Extending Database Technology (EDBT), Venice, Italy* (2017).

[25] S. Purohit, L. B. Holder, and G. Chin, *Temporal graph generation based on a distribution of temporal motifs,* in *Proceedings of the 14th International Workshop on Mining and Learning with Graphs*, Vol. 7 (2018).

[26] G. Zeno, T. L. Fond, and J. Neville, *Dynamic network modeling from motif-activity,* in *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, edited by A. E. F. Seghrouchni, G. Sukthankar, T. Liu, and M. van Steen (ACM / IW3C2, 2020) pp. 390–397.

[27] D. Zhou, L. Zheng, J. Han, and J. He, *A data-driven graph generative model for temporal interaction networks,* in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, edited by R. Gupta, Y. Liu, J. Tang, and B. A. Prakash (ACM, 2020) pp. 401–411.

[28] J. R. Ullmann, *An algorithm for subgraph isomorphism,* J. ACM **23**, 31 (1976).

[29] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, *A (sub)graph isomorphism algorithm for matching large graphs,* IEEE Trans. Pattern Anal. Mach. Intell. **26**, 1367 (2004).

[30] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, *Taming verification hardness: an efficient algorithm for testing subgraph isomorphism,* Proc. VLDB Endow. **1**, 364 (2008).

[31] W. S. Han, J. Lee, and J. H. Lee, *Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases,* in *Acm Sigmod International Conference on Management of Data* (2013).

[32] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, *Efficient subgraph matching by postponing cartesian products,* , 1199 (2016).

[33] H. He and A. K. Singh, *Graphs-at-a-time: query language and access methods for graph databases,* in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, edited by J. T. Wang (ACM, 2008) pp. 405–418.

[34] P. Zhao and J. Han, *On graph query optimization in large networks,* Proc. VLDB Endow. **3**, 340 (2010).

[35] T. L. Veldhuizen, *Leapfrog triejoin: A simple, worst-case optimal join algorithm,* arXiv preprint arXiv:1210.0481 (2012).

[36] H. Q. Ngo, C. Ré, and A. Rudra, *Skew strikes back: New developments in the theory of join algorithms,* ACM SIGMOD Record **42**, 5 (2014).

[37] D. Nguyen, M. Aref, M. Bravenboer, G. Kollias, H. Q. Ngo, C. Ré, and A. Rudra, *Join processing for graph patterns: An old dog with new tricks,* in *Proceedings of the GRADES'15* (2015) pp. 1–8.

[38] T. Plantenga, *Inexact subgraph isomorphism in mapreduce,* Journal of Parallel and Distributed Computing **73**, 164 (2013).

[39] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, *Efficient subgraph matching on billion node graphs,* arXiv preprint arXiv:1205.6691 (2012).

[40] L. Lai, L. Qin, X. Lin, and L. Chang, *Scalable subgraph enumeration in mapreduce,* Proceedings of the VLDB Endowment **8**, 974 (2015).

[41] L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang, *Scalable distributed subgraph enumeration,* Proceedings of the VLDB Endowment **10**, 217 (2016).

[42] M. Kaufmann, P. M. Fischer, N. May, C. Ge, A. K. Goel, and D. Kossmann, *Bi-temporal timeline index: A data structure for processing queries on bi-temporal data,* in *IEEE International Conference on Data Engineering* (2015) pp. 471–482.

[43] J.-Z. Luo, S.-F. Shi, G. Yang, H.-Z. Wang, and J.-Z. Li, *O2ijoin: An efficient index-based algorithm for overlap interval join,* Journal of Computer Science and Technology **33**, 1023 (2018).

[44] B. Otlu and T. Can, *Joa: Joint overlap analysis of multiple genomic interval sets,* BMC bioinformatics **20**, 121 (2019).

[45] A. Dignös, M. H. Böhlen, and J. Gamper, *Overlap interval partition join,* in *SIGMOD* (ACM, 2014) pp. 1459–1470.

[46] F. Cafagna and M. H. Böhlen, *Disjoint interval partitioning,* The VLDB Journal **26**, 447 (2017).

[47] M. Franzke, T. Emrich, A. Züfle, and M. Renz, *Pattern search in temporal social networks,* in *Proceedings of the 21st International Conference on Extending Database Technology* (2018).

[48] K. Semertzidis and E. Pitoura, *Top-k durable graph pattern queries on temporal graphs,* IEEE Transactions on Knowledge and Data Engineering **31**, 181 (2018).

[49] Y. Xu, J. Huang, A. Liu, Z. Li, H. Yin, and L. Zhao, *Time-constrained graph pattern matching in a large temporal graph,* (2017).

[50] A. Mhedhbi and S. Salihoglu, *Optimizing subgraph queries by combining binary and worst-case optimal joins,* arXiv preprint arXiv:1903.02076 (2019).

[51] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré, *Emptyheaded: A relational engine for graph processing,* ACM Transactions on Database Systems (TODS) **42**, 1 (2017).

[52] M. Karsai, H. H. Jo, and K. Kaski, *Bursty human dynamics,* **10.1007/978-3-319-68540-3** (2018).

[53] A.-L. Barabási, *The origins of bursts and heavy tails in human dynamics,* Nature **435**, 207 (2005).

[54] Alexei, Vázquez, Joo, Gama, Oliveira, Zoltán, Dezs, Kwang-Il, Goh, and I. and, *Modeling bursts and heavy tails in human dynamics,* Physical Review E **73**, 36127 (2006).

[55] R. D. Malmgren, D. B. Stouffer, A. E. Motter, and L. A. N. Amaral, *A poissonian explanation for heavy-tails in e-mail communication,* CoRR **abs/0901.0585** (2009), arXiv:0901.0585 .

[56] Peng, Wang, Bing-Hong, Wang, Tao, Zhou, Xiao-Pu, and Han, *Modeling correlated human dynamics with temporal preference,* Physica, A. Statistical mechanics and its applications **398**, 145 (2014).

[57] *NYC TLC Trip Record Data,* `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page` (2019).

[58] *Airline On-Time Performance Data,* `https://www.transtats.bts.gov/` (2019), accessed 15 June 2020.

[59] G. Bagan, A. Bonifati, R. Ciucanu, G. H. Fletcher, A. Lemay, and N. Advokaat, *gmark: Schema-driven generation of graphs and queries,* IEEE Transactions on Knowledge and Data Engineering **29**, 856 (2016).

[60] J. Enderle, *Joining interval data in relational databases,* in *ACM SIGMOD International Conference on Management of Data, Paris, France, June* (2004) pp. 683–694.

[61] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia, *Efficient join processing over uncertain data,* in *International Conference on Information and Knowledge Management CIKM* (2006) pp. 738–747.

[62] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, *Efficient processing of spatial joins using R-trees*, Vol. 22 (ACM, 1993).

[63] L. Chen, Y. Deng, W. Luo, Z. Wang, and S. Zeng, *Detection of bursts in neuronal spike trains by the mean inter-spike interval method,* Progress in Natural Science **19**, 229 (2009).

[64] *NYC Citi Bike,* `https://www.citibikenyc.com/system-data` (2019).

[65] J. Cocatre-Zilgien and F. Delcomyn, *Identification of bursts in spike trains,* Journal of neuroscience methods **41**, 19 (1992).

[66] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets, *Querying Graphs* (Morgan & Claypool Publishers, 2018).

[67] C. Kankanamge, S. Sahu, A. Mhedbhi, J. Chen, and S. Salihoglu, *Graphflow: An active graph database,* in *Acm International Conference* (2017).

[68] *Logicblox,* `https://developer.logicblox.com/`.

[69] *Avantgraph,* `http://avantgraph.io/`.

[70] A. Hogan, C. Riveros, C. Rojas, and A. Soto, *A worst-case optimal join algorithm for sparql,* in *International Semantic Web Conference* (Springer, 2019) pp. 258–275.

[71] A. Mhedhbi and S. Salihoglu, *Optimizing subgraph queries by combining binary and worst-case optimal joins,* Proceedings of the VLDB Endowment **12**.

[72] *Divvy System Data,* `https://www.divvybikes.com/system-data` (2019).

[73] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection,* `http://snap.stanford.edu/data` (2014).

[74] H. Kim and R. Anderson, *Temporal node centrality in complex networks,* Physical Review E **85**, 026107 (2012).

[75] *Allen's interval algebra,* `https://en.wikipedia.org/wiki/Allen's_interval_algebra`.

[76] S. Helmer and F. Persia, *Iseql, an interval-based surveillance event query language,* International Journal of Multimedia Data Engineering and Management (IJMDEM) **7**, 1 (2016).

List of Figures

List of Tables

List of Acronyms

ADN	Activity-Driven Network
bgFS	Forward Scan with grouping and bucket indexing
CDM	Competition-Driven Model
CP	Check-Point
CS	Concurrent Set
CSCW	Concurrent Set Construction Window
CSS	Concurrent Set Size
EBI	Endpoint-Based Index
ECI	Earliest concurrent index
gFS	Forward Scan with grouping
IET	Inter-Event Time
LH	Living History
LHW	Living History Window
P^{T}	toPology then Time
STI	Start Time Index
STI-CP	Start Time Index with Check-Point
TAI	Temporal adjacency index
TSR	Temporal Selective Relation
r-TSR	Relevant Temporal Selective Relation
WCO	Worst-Case Optimal
T^{P}	Time then toPology
T&P	Time and toPology

Acknowledgments

I still remember it was Feb 23rd, 2018, when I started my Argonautica to Technische Universiteit Eindhoven (TU/e) in the Netherlands. Honestly, I felt a little bit sad when I was on the flight as it was my first travel to a new country which is far from my home. However, when the flight was going to land and when I could see the houses on the ground, I felt more and more excited since I knew I would start a new living and get the opportunities to experience numerous life stories which I would appreciate all my life. Eventually, the four-year story about the Golden Fleece is coming to its end. I appreciate the period of living and studying in TUe which broadens my horizon and makes me grow up. More importantly, I would like to express my gratitude to every one I met during the journey. It is you who contribute to the unique colorful painting of my 4 years' time.

I would like to thank my supervisors, prof. George Fletcher and dr. Nikolay Yakovets. Thank you for your guidance and support in the four years. I still remember the day when I first have my interview with George on Skype, from which I got the opportunity to live and study here. At the beginning of my PhD, I was really a freshman as I was concentrating on doing more engineering job than researches in my past years. However, George and Nikolay are always patient with my questions (even though some of them are naive) and generous to give their suggestions about my research. I remember that when I was writing my SSTD paper in 2019, Nikolay spent his weekend time in revising the paper and teaching me the techinques to improve my writing ability. When I felt frustrated about a paragraph of analysis in my ICDE paper, Nikolay spent his time discussing it with me for hours until we finally drew a plausible conclusion. Besides, they encourage me to think independently and concentrate on true value of research itself. I remember there was a time I was frustrated about publishing a paper in my first year, George told me that " What really matters is not the publication of paper but to complete some challenging tasks, since we want to change the world." This quote accompanies all my following time in TU/e and encourages me to investigate truly interesting work and improve myself.

I would like to thank the committee members for this PhD defense, for carefully reviewing this thesis and insightful feedbacks: prof.dr. Boudewijn

van Dongen from TU/e, prof.dr. James Cheng from Chinese University Hong Kong, prof.dr. Toon Calders from Antwerp University, prof.dr. Hamamache Kheddouci from Lyon 1 University. Also, I would like to thank Roger Olesen and Yulong Pei for their kind advice. Your insightful and detailed feedbacks helped me to make this dissertation more complete.

I would like to express my special thanks for Jianpeng Zhang and Odysseas Papapetrou for their precious help in my first year. I would like to thank Jianpeng as he took me to walk around, visit every office, and introduce me to every one, when I first arrived in the 7th floor in MetaForum. This helps me to acquaint with the living and studying environment. I would like to thank Odysseas for his straightforward and valuable suggestions on my research in the first year when I was doing a bad job. It is his opinions that make me begin to realize the fact that I was far from being a qualified researcher and try to to get rid of my demerits in order to make progress.

I would like to thank Simon van der Zon and dr. Wouter Duivesteijn for their organization of various entertaining social activities. Thanks Simon for your organizing the lake camping in Summer 2018. Thanks Wouter for organizing every lunch talk, Poker night, and movie night. I appreciate my experience of participating these activities in which I could relax myself , communicate with other people, and make new friends. Besides, I'm sorry for pouring beer and polluting your poker table from time to time!:)

I would like to thank all my friends including Guangming Li, Cong Liu, Yulong Pei, Xin Du, Xuming Meng, Shiwei Liu, Tianjin Huang, Yuhao Wang, and Zhaohuan Wang for the organization of various activities and their generous help towards me. I appreciate the moments when we organize the dumpling night, the roof BBQ, and movie nights. Specially, I would like to thank Yulong and Tianjin for helping me back home from central station on June 10th, 2019, the day when I hurt my feet in Cochem. Also, thanks Shiwei for taking care of my bike when I was stuck in China because of COVID-19.

I would like to thank Ricky Maulana Fajri, Pieter Gijsbers, and Anil Yaman for your warm welcome when I stepped into my first office. I appreciate the memories we were working together in the first year. I would also like to thank Yulong Pei, Xuming Meng, Wilco van Leeuwen for sharing the same office with me in the following three years, during which period the office is always full of various discussion and joy.

I would like to thank Jose, Ine and Riet for your kind services which make our working environment more convenient. Specially, thanks Jose for your concern and contact during my period stuck in China because of the COVID-

19. Because of this, I could feel the warmth of my European home, even though I was thousands of kilometers away from that.

I would like to thank other colleagues from database group including Daniele Bonetta, Hamid Shahrivari Joghan, Larissa Capobianco Shimomura, Daphne E. Miedema, Thomas Mulder, Stanley Clark, Bram van de Wall, and Akrati Saxena. I appreciate the moments when we were participating DB-DBDs, helping organizing SIGMOD, having online stand-ups, and my opportunity to witness the growing of our warm family in the four years, with every one's participation. I would also like to thank other colleagues from data mining group including prof. dr. Mykola Pechenizkiy, Decebal Mocanu, Sibylle Hess, Loek Tonnaer and Bilge Celik. It is really a pleasure for me to know all of you with whom I used to talk during lunch and coffee break.

I would like to thank prof. Paul De Bra as he sent my e-mail for research offer to George, when he was going to retired. Without your help, I could never obtain the opprtunites to experience all these stories. Honestly, I have always been desiring to show my gratitude every time I met you in academic conference, but all result in silence for various reasons. Anyway, I'm glad that I can show my gratitude here.

I would like to thank Roger Olesen for his concerning and guidance of my four-year life in Europe. I always enjoy our chat about culture, history, literature, and politics. Your patience and enlightening opinions always encourage me to open my mind and eyes to observe my life more carefully.

Last but not least, I would like to thank all my family members in China for their support in the four years. I would like to thank Di Chen, my girl friend, fiancee, and wife. Thank you for your durable awaiting and thank you for your accompany and encourage from cell phone when I was upset. I would like to thank my parents as they brought me up in my past years life and always encouraged me to face new challenges and pursuit my dreams. I would like to parents-in-law for their concern on me every time I departed from China to Netherlands. It is you, all my family members, who provide me resourceful power and energy to fight and pursuit.

Kaijie Zhu

Eindhoven, November, 2021

Curriculum Vitæ

Kaijie Zhu was born on 28-01-1991 in Zhuzhou, China. He received bachelor's degree from information engineering university in 2013. Then he received his master's degree in Compute Science and Engineering from National Digital Switching System Engineering & Technology Research Center (NDSC) in 2016. From 2018 he started a PhD project in the Department of Mathematics and Computer Science at Eindhoven University of Technology under the supervision of prof.dr. George Fletcher and dr. Nikolay Yakovets at Eindhoven, the Netherlands, of which the results are presented in this thesis.

Publications

Publications

5. **Kaijie Zhu**, George Fletcher, and Nikolay Yakovets. *Competition-driven modeling of temporal networks*. EPJ Data Science. 2021, 10(1): 30.

4. **Kaijie Zhu**, George Fletcher, Nikolay Yakovets, Odysseas Papapetrou, and Yuqing Wu. *Scalable temporal clique enumeration*. In proceedings of the 16th International Symposium on Spatial and Temporal Databases (SSTD). 2019: 120-129.

3. **Kaijie Zhu**, George Fletcher, and Nikolay Yakovets. *Leveraging temporal and topological selectivities in temporal-clique subgraph query processing.* In proceedings of the IEEE International Conference on Data Engineering (ICDE). 2021: 672-683.

2. Jianpeng Zhang, **Kaijie Zhu**, Yulong Pei, George Fletcher, Mykola Pechenizkiy. *Clustering Affiliation Inference from Graph Samples*. In proceedings of the 14th international workshop on mining and learning with graphs (MLG) at KDD. 2018.

1. Jianpeng Zhang, **Kaijie Zhu**, Yulong Pei, George Fletcher, and Mykola Pechenizkiy. Cluster-preserving Sampling from Fully-dynamic Streaming Graph. Information Sciences, 2019, 482: 279-300.

SIKS Dissertations

2011	01	Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models
	02	Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
	03	Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems
	04	Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference
	05	Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
	06	Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage
	07	Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction
	08	Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
	09	Tim de Jong (OU), Contextualised Mobile Media for Learning
	10	Bart Bogaert (UvT), Cloud Content Contention
	11	Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
	12	Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
	13	Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
	14	Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
	15	Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
	16	Maarten Schadd (UM), Selective Search in Games of Different Complexity
	17	Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
	18	Mark Ponsen (UM), Strategic Decision-Making in complex games
	19	Ellen Rusman (OU), The Mind's Eye on Personal Profiles

20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games

	40	Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
	41	Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
	42	Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
	43	Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
	44	Boris Reuderink (UT), Robust Brain-Computer Interfaces
	45	Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
	46	Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
	47	Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
	48	Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
	49	Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality
2012	01	Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
	02	Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
	03	Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
	04	Jurriaan Souer (UU), Development of Content Management System-based Web Applications
	05	Marijn Plomp (UU), Maturing Interorganisational Information Systems
	06	Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
	07	Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
	08	Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
	09	Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms

10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women

29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

	50	Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
	51	Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching
2013	01	Viorel Milea (EUR), News Analytics for Financial Decision Support
	02	Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
	03	Szymon Klarman (VU), Reasoning with Contexts in Description Logics
	04	Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
	05	Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
	06	Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
	07	Giel van Lankveld (UvT), Quantifying Individual Player Differences
	08	Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
	09	Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
	10	Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
	11	Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
	12	Marian Razavian (VU), Knowledge-driven Migration to Services
	13	Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
	14	Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
	15	Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
	16	Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
	17	Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid

18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art
39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games

	41	Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
	42	Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
	43	Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts
2014	01	Nicola Barile (UU), Studies in Learning Monotone Models from Data
	02	Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
	03	Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
	04	Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
	05	Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
	06	Damian Tamburri (VU), Supporting Networked Software Development
	07	Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
	08	Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
	09	Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
	10	Ivan Salvador Razo Zapata (VU), Service Value Networks
	11	Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
	12	Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
	13	Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
	14	Yangyang Shi (TUD), Language Models With Meta-information
	15	Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying

	38	Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
	39	Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
	40	Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
	41	Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
	42	Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
	43	Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
	44	Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
	45	Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
	46	Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
	47	Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval
2015	01	Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
	02	Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
	03	Twan van Laarhoven (RUN), Machine learning for network data
	04	Howard Spoelstra (OUN), Collaborations in Open Learning Environments
	05	Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
	06	Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
	07	Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
	08	Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
	09	Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems

10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning

	31	Yakup Koç (TUD), On the robustness of Power Grids
	32	Jerome Gard (UL), Corporate Venture Management in SMEs
	33	Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
	34	Victor de Graaf (UT), Gesocial Recommender Systems
	35	Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction
2016	01	Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
	02	Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
	03	Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
	04	Laurens Rietveld (VU), Publishing and Consuming Linked Data
	05	Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
	06	Michel Wilson (TUD), Robust scheduling in an uncertain environment
	07	Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
	08	Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
	09	Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
	10	George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
	11	Anne Schuth (UVA), Search Engines that Learn from Their Users
	12	Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
	13	Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
	14	Ravi Khadka (UU), Revisiting Legacy Software System Modernization
	15	Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
	16	Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward

17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies

	37	Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
	38	Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
	39	Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
	40	Christian Detweiler (TUD), Accounting for Values in Design
	41	Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
	42	Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
	43	Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
	44	Thibault Sellam (UVA), Automatic Assistants for Database Exploration
	45	Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
	46	Jorge Gallego Perez (UT), Robots to Make you Happy
	47	Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
	48	Tanja Buttler (TUD), Collecting Lessons Learned
	49	Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
	50	Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
2017	01	Jan-Jaap Oerlemans (UL), Investigating Cybercrime
	02	Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
	03	Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
	04	Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
	05	Mahdieh Shadi (UVA), Collaboration Behavior
	06	Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
	07	Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

	28	John Klein (VU), Architecture Practices for Complex Contexts
	29	Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
	30	Wilma Latuny (UvT), The Power of Facial Expressions
	31	Ben Ruijl (UL), Advances in computational methods for QFT calculations
	32	Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
	33	Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
	34	Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
	35	Martine de Vos (VU), Interpreting natural science spreadsheets
	36	Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
	37	Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
	38	Alex Kayal (TUD), Normative Social Applications
	39	Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
	40	Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
	41	Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
	42	Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
	43	Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
	44	Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
	45	Bas Testerink (UU), Decentralized Runtime Norm Enforcement
	46	Jan Schneider (OU), Sensor-based Learning Support
	47	Jie Yang (TUD), Crowd Knowledge Creation Acceleration
	48	Angel Suarez (OU), Collaborative inquiry-based learning
2018	01	Han van der Aa (VUA), Comparing and Aligning Process Representations

02 Felix Mannhardt (TUE), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots

	25	Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
	26	Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
	27	Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
	28	Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
	29	Yu Gu (UVT), Emotion Recognition from Mandarin Speech
	30	Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019	01	Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
	02	Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
	03	Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
	04	Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
	05	Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
	06	Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
	07	Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
	08	Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
	09	Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
	10	Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
	11	Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
	12	Jacqueline Heinerman (VU), Better Together
	13	Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
	14	Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses

15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming

	36	Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
	37	Jian Fang (TUD), Database Acceleration on FPGAs
	38	Akos Kadar (OUN), Learning visually grounded and multilingual representations
2020	01	Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
	02	Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
	03	Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
	04	Maarten van Gompel (RUN), Context as Linguistic Bridges
	05	Yulong Pei (TUE), On local and global structure mining
	06	Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
	07	Wim van der Vegt (OUN), Towards a software architecture for reusable game components
	08	Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
	09	Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
	10	Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
	11	Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
	12	Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
	13	Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
	14	Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
	15	Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
	16	Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
	17	Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences

	18	Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
	19	Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
	20	Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
	21	Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
	22	Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
	23	Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
	24	Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
	25	Xin Du (TUE), The Uncertainty in Exceptional Model Mining
	26	Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
	27	Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
	31	Gongjin Lan (VU), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models

03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
07 Armel Lefebvre (UU), Research data management for open science
08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Childrens Collaboration Through Play
10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), BIMODAL EMOTION RECOGNITION FROM AUDIO-VISUAL CUES
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems

21 Pedro Thiago Timb Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proena (LIACS), Robust rules for prediction and description
24 **Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing**