IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON CLOUD - FOG - EDGE COMPUTING IN CYBER-PHYSICAL-SOCIAL SYSTEMS (CPSS)

Received February 12, 2020, accepted February 26, 2020, date of publication April 2, 2020, date of current version April 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2985103

Cloud-Based K-Closest Pairs Discovery in Dynamic Cyber-Physical-Social Systems

JUNWEN LU[1], (Member, IEEE), GUANFENG LIU[2], (Member, IEEE), AND XIANMEI HUA[3]

[1]Engineering Research Center for Software Testing and Evaluation of Fujian Province, Xiamen University of Technology, Xiamen 361024, China
[2]Department of Computing, Macquarie University, Macquarie Park, NSW 2109, Australia
[3]Xiamen LiHan Information Technology Service Company, Ltd., Xiamen 361000, China

Corresponding author: Guanfeng Liu (guanfeng.liu@mq.edu.au)

ABSTRACT Given two object sets P and Q, a k-closest pairs (k-CP) query finds k closest object pairs from P×Q. This operation is common in many real-life applications such as GIS, data mining and recommender systems. However, the k-CP problem has not been well studied in Dynamic Cyber-Physical-Social Systems (D-CPSS), where temporal information and multiple attributes are associated with each edge. In D-CPSS, people would like to specify multiple constraints on these attributes within a time interval to illustrate their requirements. In this paper, we study the temporal multiple constraints k closest pairs (TMC-k-CP) in D-CPSS, which is NP-Complete. We propose a divide-and-conquer cloud-based algorithm (DC) to find TMC-k-CP efficiently and effectively. To the best of our knowledge, DC is the first algorithm supporting the TMC-k-CP query in D-CPSS. The experimental results on eight real D-CPSS datasets demonstrate that our algorithm outperforms the state-of-the-art methods in terms of both efficiency and effectiveness.

INDEX TERMS CPSS, cloud-based, divide and conquer, k-closest pairs.

I. INTRODUCTION

k-Closest Pairs Query (k-CPQ) has attracted the attention of billions of people and been widely used in many applications, such as urban planning [19], [29], resource management and recommender systems [2], [6], [26]–[28]. Traditionally, given two spatial object sets P and Q, a k-CPQ returns k closest object pairs from $P \times Q$ according to a certain similarity metric, such as the minimum distance between two points of interest on a road network. The following *Example 1* illustrates a typical application of k-CPQ.

Example 1: As shown in Fig.1, G_1 is a road network. Each element in the set $P = \{a, c, d, k\}$ shown as a blue vertex represents a tourist attraction, each element in the set $Q = \{i, j, h\}$ shown as a grey vertex represents a hotel, and white vertices represent other buildings. The weight on each edge represents the length of the corresponding road segment. Then, the 2-CPQ should return two vertex pairs (v_i, v_j), $v_i \in P, v_j \in Q$ with two minimum distances in all possible pairs. Therefore, the answers are *(c, j)* with distance *5* and *(d, h)* with distance *7*.

However, some graphs have attributes on the vertices and edges, such as the *Contextual Social Graph (CSG)* in D-CPSS [11]–[14], [24], in which social role information in a specific domain (e.g., an expert in the field of knowledge graph) is associated on a vertex, and each edge has social relationships (e.g., Lucy teaches Tom) and social trust information (e.g., Andy trust Bill in cooking). In a road network, each vertex has position information, and there may be many attributes associated with the edges, including travel time, travel cost and travel distance. In various applications of road networks, e.g., trip arrangement [1] and urban planning [17]–[19], [32], [36], [37], people are glad to consider plenty of contexts related to the road segment, which has a major impact on people's arrangement and decision making. For example, on a trip, a tour group wants to book

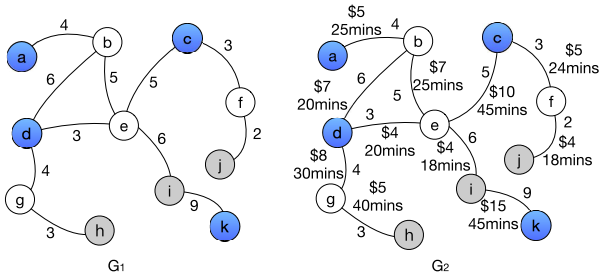
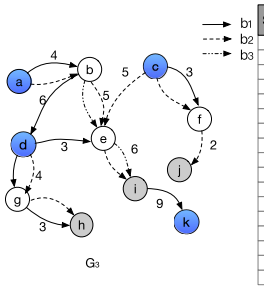


FIGURE 1. Road networks.

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaokang Wang.



Start	End	Departure Time	Duration (mins)	Cost ($)	Distance (km)	Time Point	Bus Route
a	b	8:00AM	25	5	4	t_1	b_1
a	b	9:00AM	20	6	4	t_2	b_2
b	d	8:30AM	20	7	6	t_3	b_1
b	e	8:30AM	25	7	5	t_4	b_2
b	e	8:45AM	30	8	5	t_5	b_3
c	e	8:45AM	45	10	5	t_6	b_2
c	f	9:00AM	24	5	3	t_7	b_2
c	f	9:30AM	35	8	3	t_8	b_1
f	j	9:45AM	18	4	2	t_9	b_2
d	e	9:00AM	20	4	3	t_{10}	b_1
d	g	9:00AM	30	8	4	t_{11}	b_1
d	g	9:20AM	25	6	4	t_{12}	b_2
g	h	9:40AM	40	5	3	t_{13}	b_1
g	h	10:00AM	20	10	3	t_{14}	b_2
e	i	9:10AM	18	4	6	t_{15}	b_2
e	i	9:30AM	25	5	6	t_{16}	b_3
i	k	10:20AM	45	15	9	t_{17}	b_1

FIGURE 2. Bus routes.

tickets for two attractions and order hotels for two nights in advance. The daily travel budget is *10* dollars, and the estimated travel time is *45* minutes. Then, this problem can boil down to finding two closest pairs under the constraints of time and money.

Example 2: As shown in Fig.1, G_2 is also a road network. The difference between G_1 and G_2 is that there are two other attributes on each road segment besides the length, namely, travel time and travel cost. Then the 2-CPQ returns two pairs with two minimum distances among all possible pairs which satisfy the two constraints — travel cost less than *$10* and travel time less than *45mins*. One is *(c, j)* with distance *5*, travel time *42mins*, travel cost *$9*, and the other one is *(d, i)* with distance *9*, travel time *38mins*, travel cost *$8*.

In addition to multiple attributes on edges, many real applications are modeled as D-CPSS, where a vertex is associated with another vertex at particular time instances [31]. For example, buses between stations depart according to the schedules [25], and connections between two people via telephone occur at certain moments [21]. Besides, the attributes associated with edges change over time. *Example 3* below discusses the scenario of a D-CPSS structure with multiple attributes on edges.

Example 3: Fig.2 is a temporal road network with two attributes on each edge. The associated table shows the departure time, the corresponding travel time, the cost of the ticket, and the distance between stations. We use *(u, v, t, r, c, d)* to denote a bus from u to v with departure time t, travel time r, and the cost is c, the distance between u and v is d. A tour group is interested in looking for two closest pairs from P to Q, which satisfy the below constraints: 1) the departure/arrival time is within *[8:30 AM, 10:30 AM]*; 2) the travel time needed does not exceed *45mins*; 3) the total budget is no more than *$10*. Based on the data shown in the table, we can get two result pairs.

- One pair is *(c, j)* with distance *5km*. The trip group members can take bus b_2 at site c at *9:00 AM*, and arrive at f at *9:24 AM*. Then take bus b_2 at *9:45 AM* at f, and reach j at *10:03 AM*. Thus the total travel time is *42mins* and the total cost is *$9*.
- The other pair is *(d, i)* with distance *9km*. The trip group members can take bus b_1 at site d at *9:00 AM*, and arrive at e at *9:20 AM*. Then take bus b_3 at *9:30 AM* at e, and reach i at *9:55 AM*. Thus the total travel time is *45mins* and the total cost is *$9*.

The example illustrates that *Temporal Multi-Constraint k-CPQ (TMC-k-CPQ)* is an important issue in D-CPSS, which incorporates temporal information into traditional static graphs, and considers multiple constraints on networks. When the number of the constraints is greater than one, TMC-k-CPQ becomes an NP-Complete problem because it subsumes the classical NP-Complete multi-constrained path selection problem [7], [9]. Thus, the challenge of our work is to propose techniques to effectively and efficiently support TMC-k-CPQ. Our contributions are summarized as follows.

- We propose a temporal objective function consisting of multiple constraints and value-changing attributes that indicates whether a temporal path is feasible to satisfy multiple constraints within a given time interval;
- We propose an approximation algorithm, called *Two-Pass*, which bidirectionally searches the temporal graph to find a temporal shortest path that satisfies the corresponding constraints.
- We propose a grid decomposition technique based on the divide-and-conquer method. It uses the k^{th} shortest distance in the currently obtained vertex pairs as the side length to construct a grid. The grid can filter the unpromising vertices, which speeds up the TMC-k-CPQ.
- The experiences conducted on 8 real graphs illustrate that our algorithm can save 56.63% execution time, and the average length of multi-constraints k closest pairs is 61.35% shorter compared with the results found by the state-of-art k closest pairs finding algorithm.

The rest of the paper is organized as follows. In Section II, we introduce the preliminaries and the definition of TMC-k-CPQ. The temporal objective function and the *Two-Pass* algorithm for Temporal Shortest Path with Multi-Constraints (TSP-MC) query are presented in Section III, followed by the grid decomposition technique based on divide-and-conquer for TMC-k-CPQ. Section IV reports the experimental results and Section V discusses the related works. Section VI concludes this paper.

II. PRELIMINARIES

A. MODELLING

1) D-CPSS

A *Dynamic Cyber-Physical-Social-System (D-CPSS)* can be modelled as a labelled directed temporal graph $G_A = (V, E, L, F)$, where

- V is a set of vertices, and each vertex v has spatial coordinate information;
- E is a set of edges, $e = (u, v, t, r)$, where $e \in E$ denotes an edge from vertex u to vertex v with departure time point t, running time r;
- L is a function defined on V. For each vertex v in V, $L(v)$ is a label for v. For example, the vertex label may represent a tourist attraction in road networks;

- $F = \{f^1, f^2, \ldots, f^j\}$ $(1 \leq j \leq W)$ is a function set defined on E. For each link (u, v, t, r) in E, $f^j(u, v, t, r)$ is an attribute for (u, v, t, r), like travel cost in a road network.

2) TEMPORAL PATH WITH MULTIPLE CONSTRAINTS

Given an ATG G_A, a source v_s and a destination v_d in G_A, W constraints on attributes as $\lambda_1, \lambda_2, \ldots, \lambda_W$, and a time interval $[t_\alpha, t_\beta]$, a feasible *temporal path with multiple constraints (TPMC)* in the given time interval is denoted as $p^M_{v_s,v_d}(t) = (e_1, e_2, \ldots, e_n)$, which is a time-ordered sequence of edges, and satisfies the given W constraints, where:

- $e_i = (v_i, v_{i+1}, t_i, r_i) \in E$ is a temporal edge with the departure time t_i, running time r_i for $1 \leq i \leq n$, and the departure time of v_i and arrival time of v_{i+1} are both in the time interval $[t_\alpha, t_\beta]$;
- $\Pi^j(p^M_{v_s,v_d}(t)) = \sum_{i=1}^{n} f^j(e_i)$, $\Pi^j(p^M_{v_1,v_{n+1}}(t)) \leq \lambda_j$;

where $\Pi^j(p^M_{v_s,v_d}(t))$ denotes the j^{th} aggregated attribute value of the path $p^M_{v_s,v_d}(t)$, $1 \leq j \leq W$.

Then, in a TPMC finding problem, multiple constraints on attributes can be given. For example, Total Travel Time *2* hours, Total Travel Cost *$500*, and Total Travel Distance *100km* for a temporal path finding in a travel route planning.

3) TEMPORAL MULTIPLE CONSTRAINT K CLOSEST PAIRS (TMC-k-CP)

Given an ATG G_A, W constraints on attributes (λ_1, λ_2, …, λ_W), point sets P and Q, and a time interval $[t_\alpha, t_\beta]$, then TMC-k-CPQ returns a set $R = \{\langle p, q\rangle\}$ of k temporal shortest paths with multiple constraints that satisfy the following conditions:

- $\langle p, q\rangle \in P \times Q$;
- $\forall \langle p, q\rangle$, $d(p^M_{p,q}) \leq d(p^M_{r,s})$, $\forall \langle r, s\rangle \in (P \times Q) \backslash R$, where $d(p)$ is the length of TPMC p;
- $|R| = k$;

B. TMC-k-CPQ PROBLEM

The main work of our study is to find the temporal multi-constraint k closest pairs, we define the TMC-k-CPQ problem in an ATG as follows.

Input: An ATG G_A, W constraints on attributes (λ_1, λ_2, …, λ_W), point sets P and Q, and a time interval $[t_\alpha, t_\beta]$.

Output: a set $R = \{\langle p, q\rangle\}$ of k pairs, where $\langle p, q\rangle \in P \times Q$, and $\langle p, q\rangle \in R$ have the k^{th} shortest distance, and the temporal path from p to q satisfies the W constraints within the time interval $[t_\alpha, t_\beta]$.

III. DIVIDE AND CONQUER CLOUD-BASED ALGORITHM

A. TEMPORAL OBJECTIVE FUNCTION

In this section, we propose a temporal objective function to investigate whether the aggregated attribute value of a temporal path satisfies the corresponding constraints. The temporal objective function of path $p^M_{v_s,v_d}(t)$ is defined as below Eq.(1):

$$\delta(p^M_{v_s,v_d}(t)) = max\{\frac{\Pi^1(p^M_{v_s,v_d}(t))}{\lambda_1}, \frac{\Pi^2(p^M_{v_s,v_d}(t))}{\lambda_2}, \ldots, \frac{\Pi^W(p^M_{v_s,v_d}(t))}{\lambda_W}\} \quad (1)$$

where $\Pi^w(p^M_{v_s,v_d}(t)) = \sum_{e(v_i,v_j) \in p^M_{v_s,v_d}(t)} f^w(e(v_i, v_j))$, $1 \leq w \leq W$, $f^1(e(v_i, v_j)), \ldots, f^W(e(v_i, v_j))$ are the values of the attributes on edge $e(v_i, v_j)$. If the temporal path is feasible, then $\delta(p^M_{v_s,v_d}(t)) \leq 1$, otherwise, $\delta(p^M_{v_s,v_d}(t)) > 1$.

B. TWO-PASS ALGORITHM FOR TSP-MC

In this section, we propose a *Two-Pass* algorithm that adopts two linear scans to conduct a bidirectional search of the D-CPSS from v_d to v_s to obtain the result of TSP-MC. In the algorithm, the first backward process calculates (1) the latest departure time (expressed as t^L) of all possible *backward temporal paths (BTP)* from v_i to v_d, expressed as $p^{Back}_{v_i,v_d}(t)$. The value is used to investigate whether the arrival time (expressed as t_a) of the next *forward temporal path (FTP)* from v_s to v_i (expressed as $p^{For}_{v_s,v_i}(t)$) is earlier than the latest departure time of the backward path from v_i to v_d. If $t_a \leq t^L$, it means there may be a temporal path from v_s to v_d. (2) The minimum temporal objective function value of the BTP from each vertex v_i to v_d at any time points (expressed as δ_{min}, and the corresponding departure time is t_{min}). It is used to investigate whether BTP can satisfy corresponding multiple constraints. In this process, the aggregate attribute value of BTP is stored in v_i and combined with the temporal path $p^{For}_{v_s,v_i}(t)$ to study the feasibility of the temporal combination path. If $p^{Back}_{v_s,v_d}(t) \leq 1$, it means that there is a feasible temporal path from v_s to v_d in the ATG. The next forward pass process from v_s to v_d is then performed to find the shortest temporal path that satisfies multiple constraints.

The Backward Pass: the above temporal objective function shows that the lower the δ value of the current search path, the greater the probability that the path is feasible. So the backward pass algorithm can be constructed based on the Dijkstra algorithm. And in a D-CPSS, if the BTP from v_i to v_d has a late departure time, then the FTP from v_s to v_i is more likely to form a path from v_s to v_d combined with it. Thus if there are two paths $p^{Back}_{v_i,v_d}(t_1)$ and $p^{Back}_{v_i,v_d}(t_2)$ from v_i to v_d with departure time t_1 and t_2 respectively, $\delta(p^{Back}_{v_i,v_d}(t_1)) \leq \delta(p^{Back}_{v_i,v_d}(t_2))$ and $t_1 \geq t_2$, then $p^{Back}_{v_i,v_d}(t_2)$ can be pruned without affecting the computation of δ, and we call path $p^{Back}_{v_i,v_d}(t_2)$ dominated path.

The steps of the backward pass procedure of TSP-MC are discussed as follows, and the corresponding pseudo-code is shown in Algorithm 1.

Step 1: Scan the edges in the ATG and check whether the incoming edge is within the given time interval $[t_\alpha, t_\beta]$. If so, go to *Step 2*, otherwise terminate the algorithm (*Lines 1-3, 18-21 in Algorithm 1*).

Algorithm 1 The Backward Pass

Input: G_A, v_s, v_d, W, $\lambda_1, \lambda_2, \ldots, \lambda_W$, $[t_\alpha, t_\beta]$.
Output: δ_{min}, t_{min}, t^L.

1 // edges in G_A are sorted in descending order of departure time. Elements in L_v are in the form of $(d_v, \delta(p^M_{v,v_d}), \Pi^1(p^M_{v,v_d}), \ldots, \Pi^W(p^M_{v,v_d}))$ where d_v, $\delta(p^M_{v,v_d})$, $\Pi^j(p^M_{v,v_d})$ is the departure time, the value of δ, the j^{th} aggregated attributed value of the temporal path from v to v_d;
2 **for** *each* $e = (v_i, v_{i+1}, t_i, r_i)$ **do**
3 **if** $t_\alpha \le t_i$ *and* $t_i + r_i \le t_\beta$ **then**
4 **if** $v_{i+1} = v_d$ **then**
5 update L_{v_d};
6 **end**
7 select the element in $L_{v_{i+1}}$ with $\delta(p^M_{v_{i+1},v_d})$ is minimum and $d_{v_{i+1}} \ge t_i + r_i$;
8 $\Pi^j(p^M_{v_i,v_d}) = \Pi^j(p^M_{v_{i+1},v_d}) + f^j(e)$, for all $1 \le j \le W$; $\delta(p^M_{v_i,v_d}) = max\{\frac{\Pi^j(p^M_{v_i,v_d})}{\lambda_j}\}$, $1 \le j \le W$;
9 $d_{v_i} = t_i$;
10 prune elements in L_v;
11 **if** $d_{v_i} > t^L(v_i)$ **then**
12 update $t^L(v_i)$;
13 **end**
14 **if** $\delta(p^M_{v_i,v_d}) < \delta_{min}$ **then**
15 update $\delta_{min}(v_i)$ and $t_{min}(v_i)$;
16 **end**
17 **end**
18 **else if** $t_i < t_\alpha$ **then**
19 break;
20 **end**
21 **end**
22 **return** δ_{min}, t_{min} and t^L;

Step 2: Update the departure time, δ value, and the aggregated attribute value of the temporal path from v_i to v_d by using the related information of the path from v_{i+1} to v_d. Then, prune the dominated paths (*Lines 4-10 in Algorithm 1*).

Step 3: Update the latest departure time, the minimum temporal objective function and the corresponding departure time of temporal path from v_i to v_d (*Lines 11-17 in Algorithm 1*).

Step 4: Return δ_{min}, t_{min} and t^L (*Lines 22 in Algorithm 1*).

The Forward Pass: the backward pass has calculated the latest departure time, and the minimum temporal objective function value and the corresponding departure time of all possible BTP from v_i to v_d. Then, the next forward pass is performed to find the TSP-MC. In the process, we prune the unpromising paths whose δ values are more than 1 with the help of the information calculated by backward pass and combine the current FTP with the BTP to find whether there exists a feasible temporal path from v_s to v_d. If the δ value of the combination path is greater than 1, then we can skip the current edge. And we select the shortest temporal path from v_s to v_d at last. Besides, if there are two paths $p^{For}_{v_s,v_i}(t_1)$ and $p^{For}_{v_s,v_i}(t_2)$ from v_s to v_i with arrival time a_1 and a_2 respectively, $\delta(p^{For}_{v_s,v_i}(t_1)) \le \delta(p^{For}_{v_s,v_i}(t_2))$ and $a_1 \le a_2$, then $p^{For}_{v_s,v_i}(t_2)$ can be pruned without affecting the calculation of TSP-MC, and we call path $p^{For}_{v_s,v_i}(t_2)$ dominated path.

Algorithm 2 The Forward Pass

Input: G_A, v_s, v_d, W, $\lambda_1, \lambda_2, \ldots, \lambda_W$, $[t_\alpha, t_\beta]$.
Output: len_{min}.

1 // edges in G_A are sorted in ascending order of departure time. Elements in F_v are in the form of $(a_v, \delta(p^M_{v_s,v}), \Pi^1(p^M_{v_s,v}), \ldots, \Pi^W(p^M_{v_s,v}), len(p^M_{v_s,v}))$ where a_v, $\delta(p^M_{v_s,v})$, $\Pi^j(p^M_{v_s,v})$, $len(p^M_{v_s,v})$ is the arrival time, the value of δ, the j^{th} aggregated attributed value and the length of the temporal path from v_s to v;
2 **for** *each* $e = (v_i, v_{i+1}, t_i, r_i)$ **do**
3 **if** $t_\alpha \le t_i$ *and* $t_i + r_i \le t_\beta$ **then**
4 **if** $\delta_{min}(p^M_{v_{i+1},v_d}) > 1$ **then**
5 continue;
6 **end**
7 **if** $t^L(p^M_{v_{i+1},v_d}) < t_i + r_i$ **then**
8 continue;
9 **end**
10 **if** $v_{i+1} = v_s$ **then**
11 update F_{v_s};
12 **end**
13 combine path $p^M_{v_s,v_i}$ with $p^M_{v_i,v_d}$ as path p^M_{com};
14 **if** $\delta(p^M_{com}) > 1$ **then**
15 continue;
16 **end**
17 select the element in F_{v_i} with $\delta(p^M_{v_s,v_i})$ is minimum and $a_{v_i} \le t_i$;
18 update $\Pi^j(p^M_{v_s,v_{i+1}})$, for all $1 \le j \le W$;
19 update $\delta(p^M_{v_s,v_{i+1}})$, and $a_{v_{i+1}}$, and $len(p_{v_s,v_{i+1}})$;
20 prune elements in $F_{v_{i+1}}$;
21 **end**
22 **else**
23 break;
24 **end**
25 **end**
26 **for** *each element in* F_{v_d} **do**
27 choose the shortest temporal path from v_s to v_d with len_{min};
28 **end**
29 **return** len_{min};

The steps of the forward pass of TSP-MC is shown as follows, and the pseudo-code is shown in Algorithm 2.

Step 1: Scan the edges in the ATG and check whether the incoming edge is within the given time interval $[t_\alpha, t_\beta]$. If so, go to *Step 2*, otherwise terminate the algorithm (*Lines 1-3, 22-25 in Algorithm 2*).

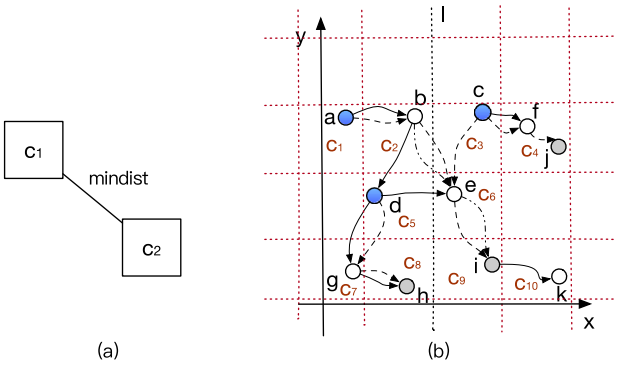


FIGURE 3. Grid.

Step 2: If the δ value of the BTP from v_{i+1} to v_d is greater than 1, or the arrival time of current FTP is later than the latest departure time of the BTP from v_{i+1} to v_d, skip the current edge (*Lines 4-9 in Algorithm 2*).

Step 3: Combine FTP and BTP, if the combination path satisfies multiple constraints, then update the minimum temporal objective function and the corresponding arrival time, the aggregated attribute value, and the length of temporal path from v_s to v_{i+1}. If the combination path cannot satisfy multiple constraints, skip the current edge. (*Lines 10-21 in Algorithm 2*).

Step 4: Find the shortest temporal path from v_s to v_d (*Lines 26-28 in Algorithm 2*).

Step 5: Return len_{min} (*Lines 29 in Algorithm 2*).

Time Complexity: In *Algorithm 1 (2)*, the initialization of variables takes $O(n)$ time, where n is the number of vertices. And the algorithm takes $O(m \log d_{max})$ time to process each edge, where searching and updating elements in priority queues takes $O(\log d_{max})$ time, and m is the number of temporal edges, and $d_{max} = max\{d_{in}, d_{out}\}$, where d_{in} (d_{out}) is the maximum in-degree (out-degree) of vertices in the ATG. Thus the time complexity of the forward pass (backward pass) procedure is $O(n + m \log d_{max})$. Then the time complexity of the *Two-Pass* algorithm is $O(n + m \log d_{max})$.

C. DC ALGORITHM

In this section, we introduce the divide-and-conquer cloud-based algorithm for TMC-k-CPQ. First of all, the set of vertices V is evenly divided by a vertical line l in the coordinate space according to the x value of each vertex, so that there are $n/2$ vertices on each side. Let P_1 (or P_2) be the set of vertices on the left (or right) side of l. Afterwards, we recursively find the temporal multiple constraints k closest pairs in P_1 and P_2, respectively. Finally, we need to merge the two halves to find the global temporal multiple constraints k closest pairs. The pairs (p_1, p_2) should satisfy $p_1 \in P_1$ and $p_2 \in P_2$, namely, p_1 and p_2 are from different sides. By the way, if a cell covers at least one point, it is called a non-empty cell. Let c_1 and c_2 be two non-empty cells, if their minimum distance is less than r, then c_1 is an r-neighbor of c_2, and vice versa. In Figure 3(a), the minimum distance between c_1 and c_2 is denoted as *mindist*. If *mindist* $\leq r$, then c_1 is an r-neighbor of c_2, and vice versa.

We use recursion to implement the divide-and-conquer cloud-based algorithm (see Algorithm 3). The core of the algorithm is the recursive algorithm (see Algorithm 4).

Algorithm 3 Answering TMC-k-CPQ

Input: G_A, W, $\lambda_1, \lambda_2, \ldots, \lambda_W$, P, Q, $[t_\alpha, t_\beta]$.
Output: O.
1 // the spatial information of vertex v in the coordinate is (x, y), then vertices are sorted in ascending order of x. A is an sorted array (e.g., $A[i].x \leq A[j].x$, for $i < j$) to store vertices. O is a priority queue, which stores the *TMC-k-CP* results;
2 $L = 0$;
3 $R = A.length$;
4 $O = Recursive(L, R)$;
5 **return** O;

Step 1: For all vertices in the space, call the recursive algorithm to calculate the k closest neighbor pairs (*Lines 1-4 in Algorithm 3*).

Step 2: If there is only one vertex left, return null. If there are two vertices left, one vertex in P and the other in Q, denoted as $u \in P$ and $v \in Q$, call *Algorithm 1* and *Algorithm 2* to calculate TSP-MC from u to v, then return the current result (*Lines 2-16 in Algorithm 4*).

Step 3: Construct a line perpendicular to the x-axis so that the number of target vertices on both sides of l is the same (see Figure 3(b), blue vertices and grey vertices are the target vertices, the number of target vertices is the same on both sides of the line l). Then recursively call *Algorithm 4* to find the TMC-k-CPQ result on the left/right of l, and update the TMC-k-CPQ result (*Lines 17-21 in Algorithm 4*).

Step 4: Build a grid in the data space, where each cell is an axis-parallel square with side length $r/\sqrt{2}$, r is the k^{th} shortest distance of the TMC-k-CPQ result, and l is a line in the grid (*Lines 22-25 in Algorithm 4*).

Step 5: For each cell c_1 on the left of l and each non-empty r-neighbor cell c_2 on the right of l, calculate the TSP-MC from p_1 to p_2 where $p_1 \in c_1, p_1 \in P, p_2 \in c_2, p_2 \in Q$ and the TSP-MC from p_2 to p_1 where $p_2 \in c_2, p_2 \in P, p_1 \in c_1, p_1 \in Q$ (*Lines 26-32 in Algorithm 4*).

Step 6: Return the TMC-k-CPQ result (*Lines 33 in Algorithm 4, Lines* 5 *in Algorithm 3*).

Time Complexity: The time complexity of *Algorithm 3* is $O(n^2 + nm \log n \log d_{max})$. Assuming that the total time complexity is $T(n)$, then in detail, (1) the time complexity of *line 2-16* in *Algorithm 4* is $O(n+m \log d_{max})$ due to the call of *Two-Pass Algorithm*; (2) the data set is divided into two equal parts in *line 17-20* in *Algorithm 4*, which results in $2T(\frac{n}{2})$ time complexity; (3) the time complexity of the merge operation in *line 26-32* in *Algorithm 4* is $O(n(n + m \log d_{max}))$. Therefore, the total time complexity is $O(n^2 + nm \log n \log d_{max})$.

Algorithm 4 Recursive

Input: L, R.
Output: O.

// *defaultlen* is the default length of the grid diagonal, usually set as $+\infty$;
if $L = R$ **then**
  return *null*;
end
if $R-L = 1$ **then**
  if $A[L] \in P$ *and* $A[R] \in Q$ **then**
    call *Algorithm 1* and *Algorithm 2* to calculate the TSP-MC with length len_1 from vertex $A[L]$ to vertex $A[R]$
    update O_1;
    return O_1;
  end
  if $A[L] \in Q$ *and* $A[R] \in P$ **then**
    call *Algorithm 1* and *Algorithm 2* to calculate the TSP-MC with length len_2 from vertex $A[R]$ to vertex $A[L]$;
    update O_1;
    return O_1;
  end
end
mid = L + (R - L) / 2;
construct a vertical line l: $x = mid$;
LPair = Recursive(L, mid);
RPair = Recursive(mid+1, R);
update O;
if O.size() $\geq$ k **then**
  $r = min\{O.peek().len, defaultlen\}$;
end
impose a grid, where each cell is an axis-parallel square with side length $r/\sqrt{2}$ and l is a line in the grid;
for *each cell* c_1 *on the left of* l **do**
  for *each r-neighbor cell* c_2 *on the right of* l **do**
    calculate the TSP-MC of each pair of vertices $< p_1, p_2 > \in c_1(P) \times c_2(Q)$;
    calculate the TSP-MC of each pair of vertices $< p_2, p_1 > \in c_2(P) \times c_1(Q)$;
    update O;
  end
end
return O;

IV. EXPERIMENT

We conduct experiments on 8 real-world D-CPSS datasets to evaluate the effectiveness and efficiency of the proposed divide-and-conquer cloud-based algorithm in answering TMC-k-CPQ. We first list the details of the experiment setting and implementation. Then, we analyze the experimental results.

TABLE 1. Datasets.

Dataset	$\|V\|$	$\|E\|$	$\|d_{avg}\|$	$\|d_{max}\|$	π	$\|T_G\|$
Openflights	12K	67K	11.15	946	20	200
Arxiv	28K	4,597K	327.26	11,134	262	2,337
Elec	7K	104K	29.13	1,167	5	101,012
Enron	87K	1,148K	26.31	38,785	1,074	213,218
Epinions	132K	841K	12.77	3,622	1	939
Opsahl	2K	60K	63.02	1,546	98	58,911
Slash	51K	141K	5.51	3,357	17	89,862
Topology	35K	171K	9.86	5,305	3	32,824

A. EXPERIMENT SETTING AND IMPLEMENTATION

- Datasets: We collect an airport dataset[1] that contains 12,000 airports and 67,000 routes. Then we collect 7 D-CPSS datasets from *konect.uni-koblenz.de*, including the email based temporal graphs (e.g., Enron), the co-authorship based temporal graphs (e.g., Arxiv) and the internet based temporal graphs (e.g., Topology). These temporal graphs have been widely used in the literature to study the temporal path discovery in temporal graphs [31]. The details of these datasets are shown in *Table 1*, where we can see the number of vertices ($|V|$), the temporal edges ($|E|$), the average degree (d_{avg}), the maximum degree (denoted as d_{max}), the maximum number of the temporal edges between two vertices (π) and the number of time points ($|T_{G_A}|$) contained in each temporal graph.
- Parameter Setting: (1) For each temporal graph, we randomly select one-fifth of the vertices as P and one-fifth of the vertices as Q, where P and Q are independent; (2) The number of the returned closest pairs (i.e., k) can affect the execution time of the algorithm, thus k is set to 2, 4, 6, 8, 10 respectively to investigate the running time of the algorithm; (3) The number of constraints (i.e., W) is set to 2, 4, 6, 8, 10 in each dataset to investigate the performance of the algorithm; (4) We randomly set the value of each attribute on an edge to the scope of [0-100] at each time point. The corresponding constraint for each attribute is set to 60, 120, 180, 240 and 300 respectively; (5) The time interval $[t_\alpha, t_\beta]$ can affect the overall execution time. In this experiment, we set five time intervals, $I_5 = [0, |T_G|]$, $I_4 = [0, \frac{4}{5}|T_G|]$, $I_3 = [0, \frac{3}{5}|T_G|]$, $I_2 = [0, \frac{2}{5}|T_G|]$, $I_1 = [0, \frac{1}{5}|T_G|]$, to investigate the effect of different $[t_\alpha, t_\beta]$ on the performance of our algorithm. These settings ensure a high possibility of returning the k-TSP-MC discovery results in an ATG.
- Implementation: (1) Since the algorithms proposed in [31] are the most promising ones to find the shortest temporal path, and the plane sweep [19], [22], [23] is the common algorithm to find k closest pairs, so we combine the algorithms in [31] with the plane sweep technique as the baseline to investigate the performance of the k closest pairs with multiple constraints discovery. The baseline algorithm is simply denoted as BL;

[1] openflights.org/data

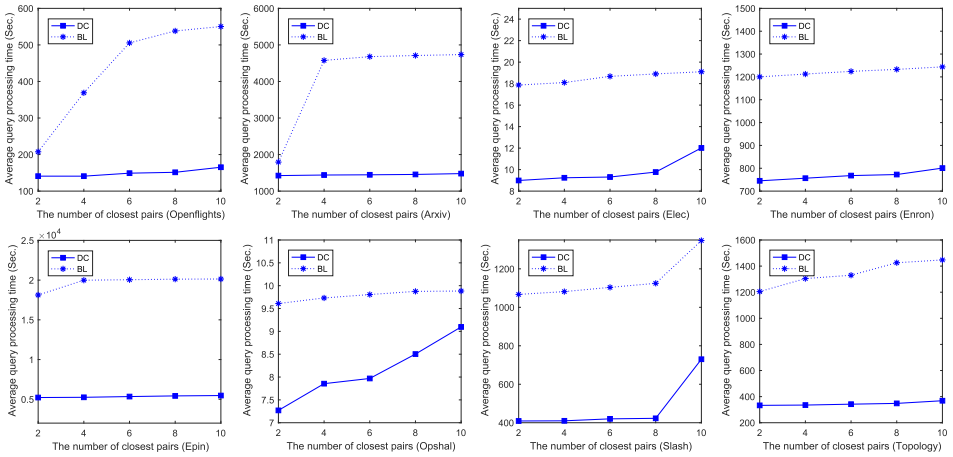


FIGURE 4. The average execution time based on different number of closest pairs.

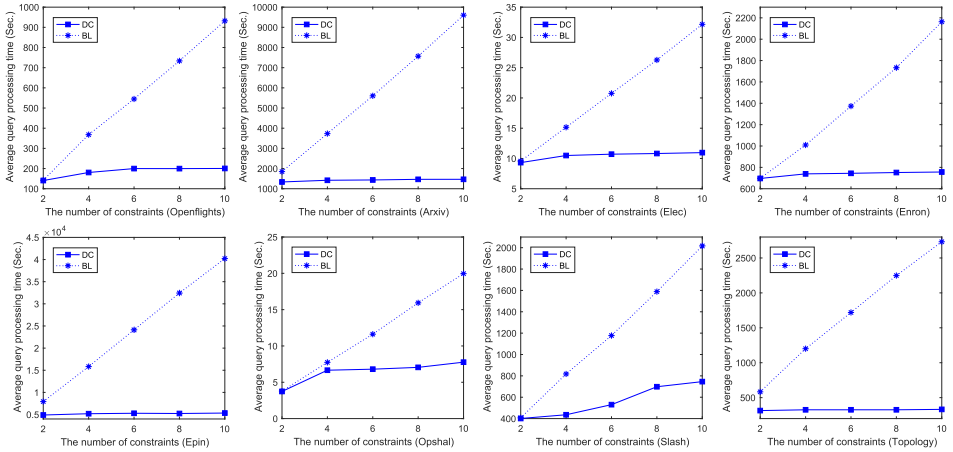


FIGURE 5. The average execution time based on different number of constraints.

(2) We implement our proposed divide-and-conquer cloud-based algorithm (which is simply denoted as DC) and compare the execution time and the quality of the results of the temporal k closest pairs under multiple constraints.

All the algorithms are implemented using Java and run on a server with Intel 2.6GHz CPU, 256GB RAM under the Linux operating system. All the experimental results are averaged based on five independent runs.

B. EXPERIMENTAL RESULTS AND ANALYSIS

In the experiments, we compare our DC with the state-of-the-art algorithm in solving the k closest pairs problem with constraints. The performance is measured by the execution time and the average distance of k closest pairs for TPC-k-CPQ.

Exp-1 (Efficiency): This set of experiments is to investigate the efficiency of our DC by comparing the query processing time of DC and BL based on different datasets and different parameters.

Result: From Fig.4 to Fig.7, we can see that (1) when k increases, the query processing time of both algorithms increases; (2) with the increase of the number of constraints, the execution time of both DC and BL increases in all datasets; (3) with the increase of the values of the constraints, the execution time of BL and DC both increases in all datasets; (4) with the increase of the time interval, the execution time of both DC and BL increases; (5) in all the cases, DC consumes less execution time than BL. The detailed experimental results are listed in Table 2. Statistically, the average execution time of DC is 56.63% less than that of BL.

Analysis: The result illustrates that (1) For DC, with the increase of k, more vertices will be contained in each cell of the grid, due to the increase of the side length of the grid which is equal to the k^{th} shortest distance of the current k closest pairs result. Therefore, we need to spend time calculating the distances of more vertex pairs; For BL, with the increase of k, the pruning strength of the plane sweep technology will be smaller, so the running time of the algorithm will increase; (2) For DC, as the number of constraints increases,

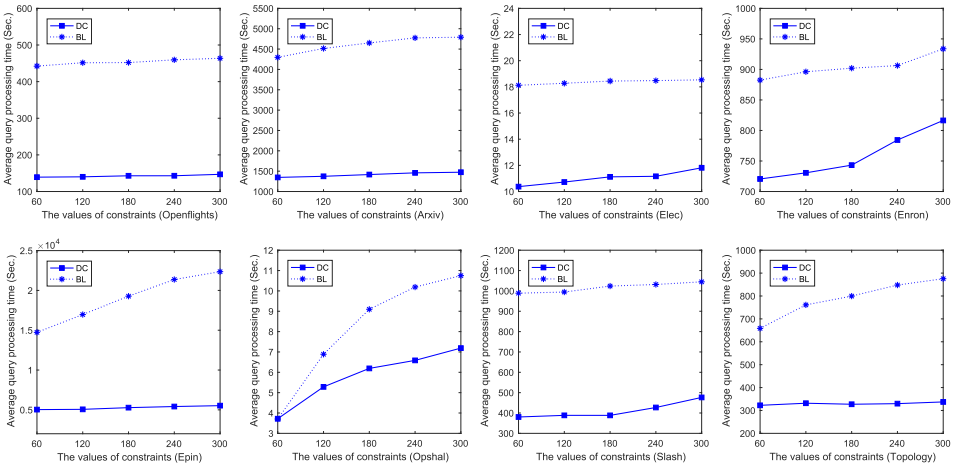


FIGURE 6. The average execution time based on different values of constraints.

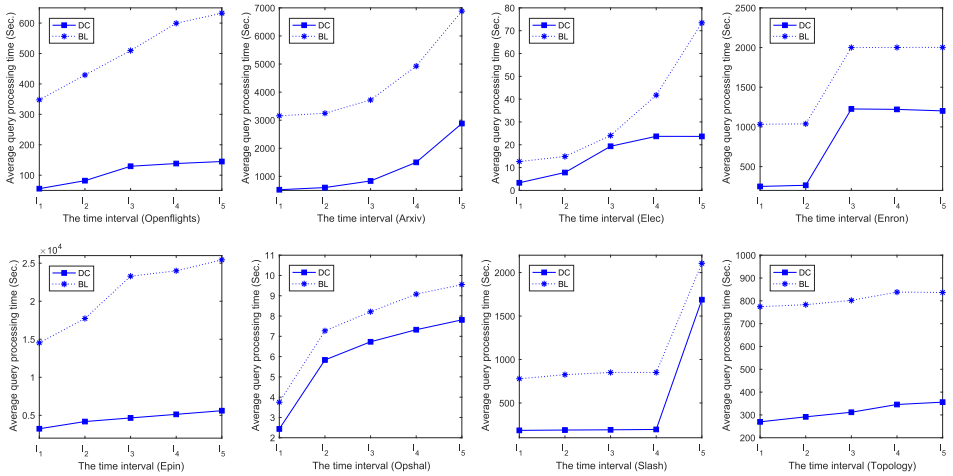


FIGURE 7. The average execution time based on different time intervals.

TABLE 2. Comparison.

Dataset	average query processing time	average distance
Openflights	69.76% less	29.59% less
Arxiv	69.93% less	79.56% less
Elec	47.67% less	48.89% less
Enron	37.03% less	59.99% less
Epinions	75.43% less	82.85% less
Opsahl	28.18% less	78.84% less
Slash	56.44% less	31.42% less
Topology	68.63% less	79.65% less

the previously found vertex pairs satisfying the constraints do not necessarily satisfy the current W constraints. Therefore, the distance between the k^{th} shortest vertex pairs that satisfy the current W constraints may be longer, which makes the grid side length larger, leading to more vertex pairs calculation. In addition, the time consumption caused by the aggregation of the path in the forward pass of the *Two-Pass* algorithm increases; For BL, firstly the pruning strength will be smaller; Secondly, when we find a feasible path between two vertices, BL needs to search the temporal edges between two vertices W times to find the shortest path that satisfies W constraints. Thus the execution time increases with the increase of W; (3) For DC, as the values of the constraints increase, more temporal paths will satisfy the requirements in the temporal graph. Then, more time will be needed to calculate the TSP-MC by *Two-Pass* and find the k closest pairs with multiple constraints; For BL, the reason is similar to DC; (4) With the increase of the time interval, more temporal edges are included in the search. Thus both DC and BL consume more execution time to search these temporal edges; (5) The execution time of DC is always less than that of BL, there are two reasons. First, the *Two-Pass* algorithm in DC to find TSP-MC only need to scan the attributed temporal graph twice, while the path finding algorithm in BL needs to scan the temporal graph W times; Second, the grid technology enables a lot of pruning to speed up the query process.

Exp-2 (Effectiveness): In order to investigate the effectiveness of our DC, we compare the average length of the obtained k closest pairs of DC and BL based on different datasets and different parameters.

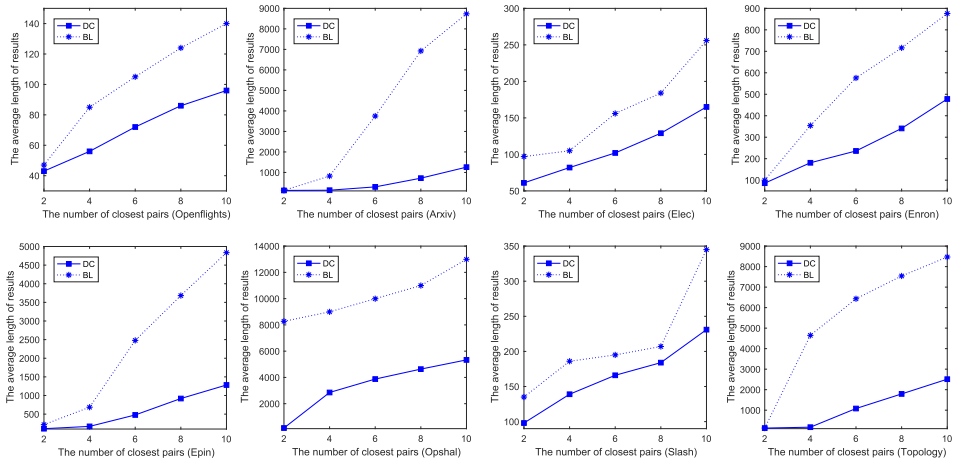


FIGURE 8. The average k closest pairs length based on different number of closest pairs.

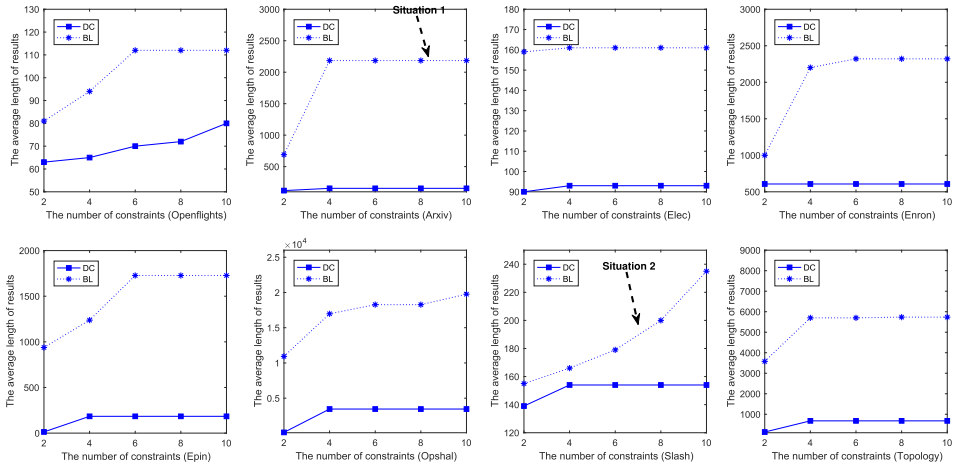


FIGURE 9. The average k closest pairs length based on different number of constraints.

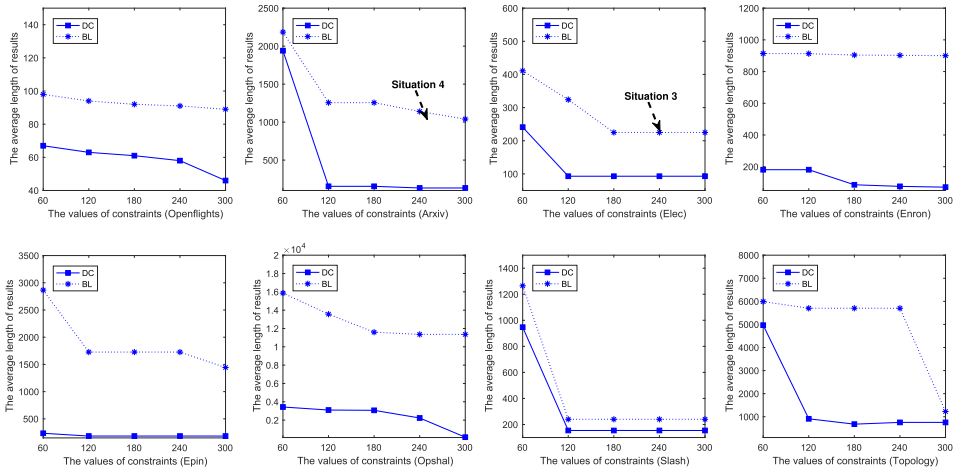


FIGURE 10. The average k closest pairs length based on different values of constraints.

Result: Fig.8, Fig.9, Fig.10 and Fig.11 depict the average path length of the obtained k closest pairs by DC and BL under different parameters. We can see that (1) when k increases, the average length of the results obtained by two

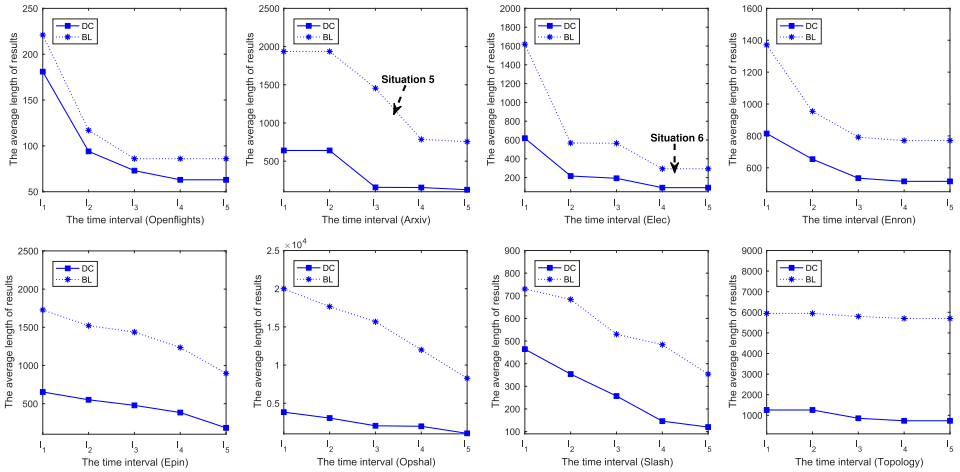


FIGURE 11. The average k closest pairs length based on different time intervals.

algorithms both increases; (2) with the increase of the number of constraints, the average length of the results increases first. Then, in some cases (e.g., *situation 1*), the average length of the obtained k closest pairs remains the same; in other cases (e.g., *situation 2*), the average length of the results still increases with the increase of W; (3) with the increase of the values of the constraints, the average length of results obtained by BL and DC decreases first, and then in some cases (e.g., *situation 3*), the average length of the results keeps the same; in other cases (e.g., *situation 4*), the average length of the results still decreases; (4) with the increase of the time interval, the average length of results of DC and BL decreases in some cases (e.g., *situation 5*), and in other cases the average length of the results remains the same (e.g., *situation 6*); (5) in all the cases, the average path length of the k closest pairs obtained by DC is shorter than that obtained by BL. The detailed experimental results are listed in Table 2. Statistically, the average length of results obtained by DC is 61.35% less than that obtained by BL.

Analysis: The result illustrates that (1) As k increases, k closest pairs are needed to be found, so the average length of the results increases; (2) As the number of constraints increases, the previously found vertex pairs that satisfy the requirements do not necessarily satisfy the current W constraints. Therefore, some pairs of vertices with longer distances that satisfy the current multiple constraints will generate, resulting in the increase of the average distance. In some cases, this trend has been maintained. In other cases, previously found pairs of vertices that satisfy the constraints always satisfy the existing W constraints, so the average distance of the results keeps the same; (3) As the values of the constraints increase, it means that the constraints on the attributes between vertex pairs are relaxed. Therefore, some vertex pairs that did not meet the high requirements will now meet the requirements after relaxation. Thus, in some cases, the average path length in the result set will reduce. And in other cases, the previously found result set still satisfies the current constraints, so the average length of the result set is unchanged; (4) With the increase of the time interval, more temporal edges are included in the search. In some cases, some pairs of vertices with shorter distances that satisfy the constraints will appear, so that the average distance decrease. In other cases, not appear, so the average distance keeps the same; (5) BL investigates each constraint separately when determining whether a vertex pair meets the requirements, which cannot find a path even if there is a feasible temporal path between two vertices. Nevertheless, the proposed *Two-Pass* can always find a feasible temporal path if one exists. Therefore, the results obtained by DC are always better than those obtained by BL.

V. RELATED WORK

The existing k-closest pairs queries are mainly based on static graphs, and the similarity join is very similar to the k-closest pairs query, so we will briefly introduce the related research methods from these two aspects.

A. SIMILARITY JOIN

Given two datasets, similarity join refers to the problem of finding pairs of objects between two datasets within a certain distance ϵ, which involves a large number of applications, such as data cleaning, multimedia databases and geographical information systems. Jacox and Samet [8] propose a method called Quickjoin, which uses a modified ball partition to recursively divide and conquer an object set into subsets based on the distance to the pivot, and prunes unpromising vertex pairs. Fredriksson and Braithwaite [5] improve the Quickjoin algorithm in three ways. Paredes and Reyes [15] solve metric similarity join and k-CP retrieval based on a new metric index, coined List of Twin Clusters (LTC), which maintains two lists of overlapping clusters to index both object sets jointly instead of the natural approach of indexing one or both sets independently. Pearson and Silva [16] present an algorithm that significantly extends eD-Index [4]

to support generic similarity join queries over two relations using only the individual indices. Sarma *et al.*, [20] propose a general framework to compute similarity joins in MapReduce based on metric distance functions, and the framework partitions data space based on the distribution of underlying data. Wang *et al.*, [30] propose a parallel framework to solve similarity join in metric space. The core of the proposed approach is to intelligently route the data into independent worksets processed by independent MapReduce workers, so that no pair will be evaluated twice.

B. K-CLOSEST PAIRS FINDING

The nested loop method can be used to answer k-CP queries, but this naive approach requires plenty of distance calculations. Such high search cost results in poor scalability. Kurasawa *et al.*, [10] present a new divide-and-conquer cloud-based k-CP search method in metric space, called Adaptive Multi-Partitioning (AMP) which uses a sparse-region-first strategy to iteratively divide a region. Corral *et al.*, [3] propose five different algorithms to solve the problem of k-CP, four of which are recursive and one is iterative. Yang *et al.*, [35] propose an index structure called bRdnn, which uses information about the nearest neighbors to help prune the search paths more effectively. Shin *et al.*, [22], [23] present an efficient algorithm based on a spatial index, which uses bidirectional vertex expansion and plane-sweeping techniques to quickly prune distant pairs, and further optimizes the plane-sweeping by novel strategies for selecting a sweeping axis and direction. Roumelis *et al.*, [19] utilize two improvements that can be applied to the plane-sweep algorithm and propose a new algorithm that can minimize distance computation. Ahmadi and Nascimento *et al.*, [1] propose two different approaches for processing k-CPQs. The first approach applies a top-down traversal paradigm using a best-first search strategy, and the second approach looks for the k-closest pairs by traversing the G*-tree in a bottom-up manner. Xue and Li *et al.*, [33] first study the stochastic closest pair problem under uncertainty. Then, they [34] revisit the Range Closest Pair problem, which aims to preprocess the dataset into a data structure so that when a query range X is specified, the closest-pair in the dataset cap X can be reported efficiently. Besides, they put forward some new data structures for various query types, including quadrants, strips, rectangles and halfplanes. Zhou *et al.*, [38] propose a simple and efficient algorithm to solve the closest pair problem in k-dimensional space. It sorts these points according to the one-dimension value and only calculates the distances of pairs which may be closer than the current closest pair.

However, the traditional methods of k closest pairs discovery in graphs do not consider temporal information, nor the attributes on edges and the corresponding constraints on the attributes. In many D-CPSS based applications, it is popular and significant to explore k closest pairs in temporal graphs, but there is no effective and efficient way to solve this problem.

VI. CONCLUDING REMARKS

In this paper, we have proposed a new exploration model of Temporal Multiple Constraints k Closest Pairs (TMC-k-CP), which is the basis of many D-CPSS based applications. Then, we have proposed a new temporal objective function to dynamically study whether the temporal path satisfies multiple constraints, and we have proposed a two-pass approximation algorithm that uses two steps to find the Temporal Shortest Path with Multiple Constraints (TSP-MC). Finally, we have proposed an algorithm based on divide-and-conquer to quickly find k temporal closest pairs with multiple constraints. The time complexity of our proposed algorithm is $O(n^2 + nm \log n \log d_{max})$. In addition, experiments on eight real-world D-CPSS datasets demonstrate the superiority of our proposed method in terms of effectiveness and efficiency.

REFERENCES

[1] E. Ahmadi and M. A. Nascimento, "K-closest pairs queries in road networks," in *Proc. 17th IEEE Int. Conf. Mobile Data Manage. (MDM)*, Jun. 2016, pp. 232–241.
[2] X. Chi, C. Yan, H. Wang, W. Rafique, and L. Qi, "Amplified locality-sensitive hashing-based recommender systems with privacy protection," Feb. 2020.
[3] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 189–200, 2000.
[4] V. Dohnal, C. Gennaro, and P. Zezula, "Similarity join in metric spaces using ed-index," in *Proc. Int. Conf. Database Expert Syst. Appl.*, Prague, Czech Republic, Sep. 2003, pp. 484–493.
[5] K. Fredriksson and B. Braithwaite, *Quicker Similarity Joins in Metric Spaces*. Berlin, Germany: Springer, 2013.
[6] Y. Gao, L. Chen, X. Li, B. Yao, and G. Chen, "Efficient k-closest pair queries in general metric spaces," *VLDB J.*, vol. 24, no. 3, pp. 415–439, 2015.
[7] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Operations Res.*, vol. 17, no. 1, pp. 36–42, Feb. 1992.
[8] E. H. Jacox and H. Samet, "Metric space similarity joins," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 1–38, Jun. 2008.
[9] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, no. 1, pp. 95–116, 1984.
[10] H. Kurasawa, A. Takasu, and J. Adachi, "Finding the k-closest pairs in metric spaces," in *Proc. 1st Workshop New Trends Similarity Search (NTSS)*, 2011.
[11] G. Liu, Y. Wang, and M. A. Orgun, "Trust transitivity in complex social networks," in *Proc. 21st AAAI Conf. Artif. Intell.*, 2011, pp. 1222–1229.
[12] G. Liu, Y. Wang, and M. A. Orgun, "Optimal social trust path selection in complex social networks," in *Proc. 24th AAAI Conf. Artif. Intell.*, Atlanta, GA, USA, Jul. 2010, pp. 1575–1576.
[13] G. Liu, Y. Wang, M. A. Orgun, and E.-P. Lim, "Finding the optimal social trust path for the selection of trustworthy service providers in complex social networks," *IEEE Trans. Services Comput.*, vol. 6, no. 2, pp. 152–167, Apr. 2013.
[14] G. Liu, K. Zheng, Y. Wang, M. A. Orgun, A. Liu, L. Zhao, and X. Zhou, "Multi-constrained graph pattern matching in large-scale contextual social graphs," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 351–362.
[15] R. Paredes and N. Reyes, "Solving similarity joins and range queries in metric spaces with the list of twin clusters," *J. Discrete Algorithms*, vol. 7, no. 1, pp. 18–35, Mar. 2009.
[16] S. S. Pearson and Y. N. Silva, "Index-based R-S similarity joins," in *Proc. SISAP*. Springer, 2014.
[17] F. Piccialli, G. Casolla, S. Cuomo, F. Giampaolo, and V. S. di Cola, "Decision making in IoT environment through unsupervised learning," *IEEE Intell. Syst.*, vol. 35, no. 1, pp. 27–35, Jan. 2020.
[18] F. Piccialli, S. Cuomo, V. S. D. Cola, and G. Casolla, "A machine learning approach for IoT cultural data," *J. Ambient Intell. Hum. Comput.*, to be published.

[19] G. Roumelis, M. Vassilakopoulos, A. Corral, and Y. Manolopoulos, "A new plane-sweep algorithm for the K-closest-Pairs query," in *Proc. Int. Conf. Current Trends Theory Pract. Inform.*, 2014, pp. 478–490.
[20] A. D. Sarma, Y. He, and S. Chaudhuri, "Clusterjoin: A similarity joins framework using map-reduce," *Proc. VLDB Endowment*, vol. 7, no. 12, pp. 1059–1070, 2014.
[21] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "Timecrunch: Interpretable dynamic graph summarization," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1055–1064.
[22] H. Shin, B. Moon, and S. Lee, "Adaptive multi-stage distance join processing," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 343–354, Jun. 2000.
[23] H. Shin, B. Moon, and S. Lee, "Adaptive and incremental processing for distance join queries," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 6, pp. 1561–1578, Nov. 2003.
[24] G. Sun, G. Liu, Y. Wang, M. A. Orgun, and X. Zhou, "Incremental graph pattern based node matching," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 281–292.
[25] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 967–982.
[26] X. Wang, L. T. Yang, L. Kuang, X. Liu, Q. Zhang, and M. J. Deen, "A tensor-based big-data-driven routing recommendation approach for heterogeneous networks," *IEEE Netw.*, vol. 33, no. 1, pp. 64–69 Jan./Feb. 2019.
[27] X. Wang, L. T. Yang, Y. Wang, X. Liu, Q. Zhang, and M. J. Deen, "A distributed tensor-train decomposition method for Cyber-Physical-Social services," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 4, pp. 1–15, Oct. 2019.
[28] X. Wang, L. T. Yang, H. Li, M. Lin, J. Han, and B. O. Apduhan, "NQA," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 4, pp. 1–21, Jul. 2019.
[29] X. Wang, L. T. Yang, X. Xie, J. Jin, and M. J. Deen, "A cloud-edge computing framework for Cyber-Physical-Social services," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 80–85, Nov. 2017.
[30] Y. Wang, A. Metwally, and S. Parthasarathy, "Scalable all-pairs similarity search in metric spaces," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2013.
[31] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 721–732, May 2014.
[32] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet Things J.*, Sep. 26, 2019, early access, doi: 10.1109/JIOT.2019.2944007.
[33] J. Xue and Y. Li, "Stochastic closest-pair problem and most-likely nearest-neighbor search in tree spaces," in *Proc. Workshop Algorithms Data Struct.*, 2017, pp. 569–580.
[34] J. Xue, Y. Li, S. Rahul, and R. Janardan, "New bounds for range closest-pair problems," in *Proc. 34th Int. Symp. Comput. Geometry (SoCG)*, vol. 99, 2018, p. 73.
[35] C. Yang and K. I. Lin, "An index structure for improving closest pairs and related join queries in spatial databases," in *Proc. Int. Database Eng. Appl. Symp.*, Jul. 2002, pp. 140–149.
[36] J. Zhou, X. S. Hu, Y. Ma, J. Sun, T. Wei, and S. Hu, "Improving availability of multicore real-time systems suffering both permanent and transient faults," *IEEE Trans. Comput.*, vol. 68, no. 12, pp. 1785–1801, Dec. 2019.
[37] J. Zhou, J. Sun, P. Cong, Z. Liu, X. Zhou, T. Wei, and S. Hu, "Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT," *IEEE Trans. Services Comput.*, Dec. 31, 2019, early access, doi: 10.1109/TSC.2019.2963301.
[38] Y. Zhou and H. Yu, "An efficient comparison-based deterministic algorithm to solve the closest pair problem," in *Proc. 8th Int. Conf. Intell. Comput. Technol. Autom. (ICICTA)*, Jun. 2015, pp. 145–148.

JUNWEN LU (Member, IEEE) received the master's degree in computer science from Qingdao University, China, in 2008. He is currently a Research Scientist with the Engineering Research Center for Software Testing and Evaluation of Fujian Province, Xiamen University of Technology, China. His research interests include data management, trust computing, software testing, and social networks.

GUANFENG LIU (Member, IEEE) received the Ph.D. degree in computer science from Macquarie University, Australia, in 2013. He is currently an Assistant Professor with the Department of Computing, Macquarie University. He has published more than 80 articles on international journals and conferences papers, including the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, ACM TWEB, ACM TDS, ICDE, WWW, IJCAI, AAAI, and CIKM. He has been actively involved in the research community. His research interests include graph data management, recommender systems, trust management, and social networks. He was the Guest Editors of two special issues in the World Wide Web Journal. He also serviced as the General Chair or PC Co-Chair for AWC2019, AWC2020, and EAI CloudCom2019, and as a Program Committee Member for several high-reputable conferences, like KDD, VLDB, WWW, IJCAI, AAAI, and CIKM.

XIANMEI HUA is currently a Senior Data Scientist with Xiamen LiHan Information Technology Service Company, Ltd., China. Her research interests include data analysis, data mining, and machine learning.