Modeling and Querying Sensor Networks Using Temporal Graph Databases

Bart Kuijpers[1], Valeria Soliani[1,2], and Alejandro Vaisman[2(✉)]

[1] Hasselt University, Hasselt, Belgium
bart.kuijpers@uhasselt.be
[2] Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina
{vsoliani,avaisman}@itba.edu.ar

Abstract. Transportation networks (e.g., river systems or road networks) equipped with sensors that collect data for several different purposes can be naturally modeled using graph databases. However, since networks can change over time, to represent these changes appropriately, a temporal graph data model is required. In this paper, we show that sensor-equipped transportation networks can be represented and queried using temporal graph databases and query languages. For this, we extend a recently introduced temporal graph data model and its high-level query language T-GQL to support time series in the nodes of the graph. We redefine temporal paths and study and implement a new kind of path, called Flow path. We take the Flanders' river system as a use case.

Keywords: Graph databases · Temporal databases · Sensor networks

1 Introduction and Related Work

A *sensor network* [1] is a collection of sensors that send their data to a central location for storage, viewing and analysis. These data can be used in various application areas, like traffic control and river monitoring. A sensor network through which a flow circulates (e.g., data, water, traffic) is called a *sensor-equipped transportation network*. These networks are rather *stable*, in the sense that the changes over time are minimal and occur occasionally. For example, the direction of the water flow in a river may change due to a flood or a branch may disappear due to long dry weather periods. Sensors attached to transporta-tion networks produce time-series data, a problem studied in [3], where a formal model and a calculus are proposed. In that work, the network is modeled as a property graph (a graph whose nodes an edges are annotated with proper-ties) [2] where nodes are associated with time series (see also [6]), obtained from the sensor measurements. One limitation of the work in [3], is that the model assumes that graphs are not *temporal*, that is, they do not keep track of their history. To address this problem, in this paper we propose to use the temporal graph data model proposed by Debrouvier et al. [4], denoted TGraph, where nodes and edges are labeled with temporal validity intervals telling the period when a node, an edge, or a property exists in the graph. Using this model we

can query the existence or not of a graph object at a certain time instant, the values of a property measured by a sensor, and even the intervals where the sensor was working. In addition, the model comes with a high-level query language called T-GQL. TGraph builds upon three notions of paths: continuous, pairwise continuous, and consecutive. Intuitively, a *continuous path* (CP) is continuously valid during a certain time interval. A *pairwise continuous path* (PCP) is a path where consecutive edges overlap during a certain time interval. Finally *consecutive paths* (CSP) are paths where the temporal intervals between consecutive edges do not overlap (typically used for scheduling).

TGraph accounts mainly for connections between nodes, but do not address nodes associated with time series functions, like it is the case in sensor networks. For this, in this paper we extend TGraph and T-GQL, and redefine the temporal path notions to address queries like *"List the paths between two sensor nodes J and A were all temperature measurements are above a value τ, and the interval I when this occurred,"* which cannot be expressed by static graph models. We take the Flanders' river system as a use case and consider that some nodes, which represent river segments, are equipped with sensors while other ones are not. The model is introduced in Sect. 3. In addition, we redefine the three kinds of paths mentioned above and introduce the notion of *Flow Path* (Sect. 4), also showing how complex queries can be expressed using the extended T-GQL. Section 5 presents the algorithm to compute Flow Paths and describes how T-GQL queries are translated into Cypher using the underlying graph structure. We conclude in Sect. 6.

2 Background and Preliminary Definitions

A *transportation network* $\mathcal{TN}$ is a directed graph (N, E), where N is a finite set of nodes and $E \subseteq N \times N$ is a set of directed edges. Under this definition, we may model networks (e.g., rivers, roads, electrical) in at least two ways: (a) Segments represented by edges that connect two (geographic) points, modeled as nodes (illustrated on the left-hand side of Fig. 1); (b) Segments represented by nodes, and an edge between two nodes A and B indicates that the flow goes in the direction of the edge; we call $\mathsf{FlowsTo}$ the relationship (that is, the edge type) representing that the flow goes from A to B (Fig. 1, center). Following [3] we adopted the latter approach. Adding sensors to this network yields the notion of sensor-equipped transportation network.

Definition 1 (Sensor-equipped transportation network ([3])). *Consider a set $\mathbb{T}$ of (possible) time moments and a set $\mathbb{V}$ of (possible) measurement values. A* sensor-equipped transportation network SN, *is a 4-tuple (N, E, S, TS), such that (N, E) is a transportation network, $S \subseteq N$ is a set of sensor-equipped nodes (*sensor nodes*, for short), and $\mathsf{TS} : S \to \mathcal{P}(\mathbb{T} \times \mathbb{V})$ is a (*time-series*) function that maps sensors to a set (or sequence) of time-value pairs, ordered according with their time component (Fig. 1 (right)).* □

In the sequel, we call the networks in Definition 1 *sensor networks*.

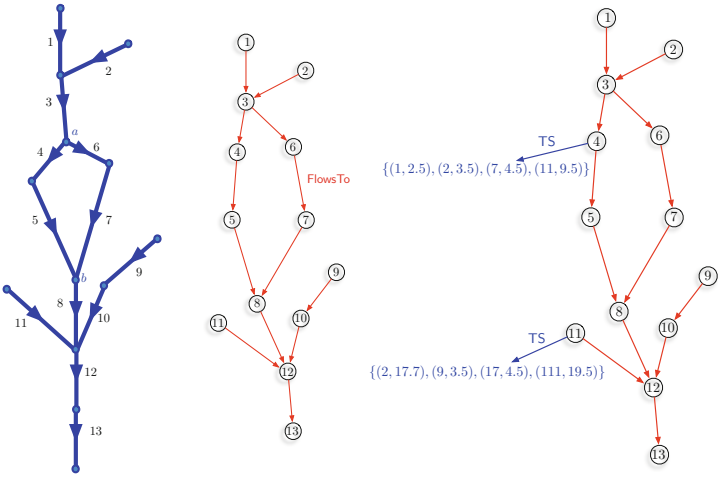

Fig. 1. Left: a physical transportation network for a river system (segments represented as edges); Center: representing segments as nodes; Right: a transportation network with the time series attached to sensor nodes 4 and 11 (segments represented as nodes)

Definition 2 (Temporal property graph [4]). *A temporal property graph is a structure $G(N_o, N_a, N_v, E)$ where G is the name of the graph, E is a set of edges, and N_o, N_a, and N_v are disjoint sets of nodes, called Object nodes, Attribute nodes, and Value nodes, respectively. Object and attribute nodes, as well as edges, are associated with a tuple (*name*, *interval*). The *name* represents the content of the node (or the type of the edge), and the *interval* the time period(s) when the node is (or was) valid. Analogously, value nodes are associated with a (*name*, *interval*) pair. For any node n, the elements in its associated pair are referred to as $n.$*name*, $n.$*interval*, and $n.$*value*. As usual in temporal databases, a special value *Now* tells that the node is valid at the current time. All nodes also have an (non-temporal) identifier denoted* `id`. □

A set of temporal constraints hold in Definition 2, and are intuitively explained next. First, all nodes with the same value associated with the same attribute node must be coalesced. Analogously, all edges with the same name between the same pair of nodes, must be coalesced. For nodes, it holds that: (a) An Object node can only be connected to an attribute node or to another object node; (b) Attribute nodes can only be connected to non-attribute nodes; (c) Value nodes can only receive edges from attribute nodes. Attribute nodes must be connected by only one edge to an object node, and value nodes must only be connected to one attribute node with one edge. Finally, for intervals: (d) The interval of an attribute (value) node must be included in the interval of its associated object (attribute) node; (e) Intervals associated with a value node must be disjoint; (f) The intervals of two edges between the same pair of nodes must be disjoint.

The model described above comes with a high-level query language denoted T-GQL. The language has a slight SQL flavor, although it is also based on Cypher [5], the query language of the Neo4j graph database. The implementation of T-GQL also extends Cypher with a collection of functions that allow handling the different kinds of temporal paths. T-GQL queries are translated into Cypher, hiding all the underlying structures that allow handling a temporal graph.

3 Temporal Graphs for Sensor Networks

The model in Definition 2 must be modified to handle sensor networks: We must distinguish Object nodes that hold a sensor from the ones which do not. We call the former *Segment* nodes. Also a list of time intervals indicates the periods of time where a segment had a working sensor on it. Properties that do not change across time are represented as usual in property graphs. We remark that we work with *categorical* variables. Also, we assume that there is at most one sensor per segment, which measures different variables, instead of many sensors that measure different variables.

Definition 3 (Sensor Network Temporal graph). *A* Sensor Network Temporal Graph (SNGraph) *is a structure* $G(N_s, N_a, N_v, E)$ *where* G *is the name of the graph,* E *a set of edges, and* N_s, N_a, *and* N_v *sets of nodes, denoted Segment, Attribute, and Value nodes, respectively. Nodes are associated with a tuple (*name, interval*), but in Segment nodes this tuple exists only if the segment contains (or ever contained) a sensor. In this case,* name = Sensor*, and* interval *represents the periods when a sensor worked. They may also have properties that do not change over the time (called static). An Attribute node represents a variable measured by the sensors, its* name *property is the name of such variable, and* interval *is its lifespan. A Value node is associated with an Attribute node, its* name *property contains the (categorical) values registered by the sensors, and* interval *the period when the measure was valid. The* name *property of the edges between Segment nodes represents the flow between two segments, and* interval *is the validity period of the edge. All nodes have a static identifier denoted* id*.* □

Temporal constraints in the TGraph model also hold for the model in Definition 3 (we omit them here). We use the Flemish river system in Belgium as a case study. Figure 2 shows a part of the Meuse river modeled as an SNGraph. There are five Segment nodes, three of which have sensors (the shaded ones, with id = *120*, id = *345*, and id = *1200*), thus, name = Sensor. The static property riverName in Segment nodes contains the river's name. The Segment with id = *345* had a sensor between times 20 and 80 and measured two variables: Temperature and pH, thus, there are two Attribute nodes connected to it, one for each variable, with intervals [25–80] and [20–80], respectively. Note that time intervals in Attribute nodes are included in the interval of the Segment node, i.e., they satisfy the temporal constraints. There are two Value nodes for the Temperature Attribute node, such that between instants 20 and 25 the temperature was *Low*, between instants 25 and 27 it was *High*, and between instants 27 and 80 it went down to *Low* again. Finally, FlowsTo is the edge type.

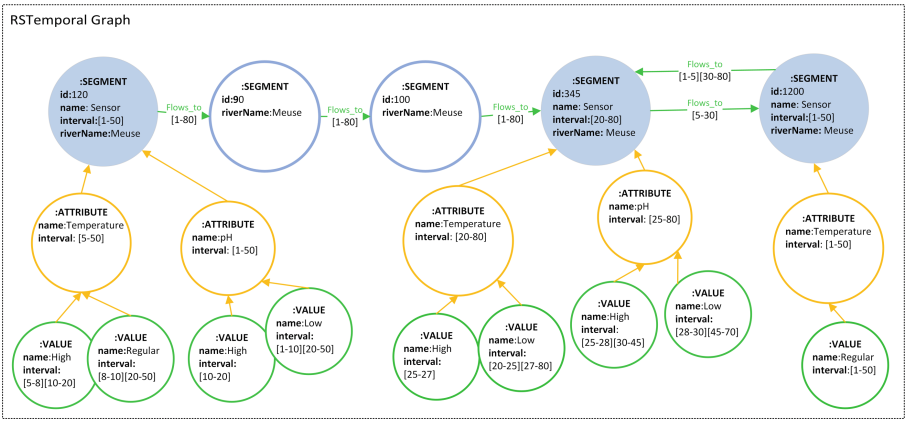

Fig. 2. A temporal graph for a river sensor network (sensor nodes are shaded).

4 Temporal Paths in Sensor Networks

We now redefine the path notions in [4], according with the model in Sect. 3.

Continuous Path in Sensor Networks. Many queries of interest can be answered using the model of Definition 3. For example: "*Starting from a segment, obtain all the paths and their corresponding time intervals T_i such that the temperature in the path has been simultaneously High for all nodes in the path during T_i*". The original "Continuous Path" notion only accounts for the connections between nodes in a temporal graph, so it must be modified to compute a path restricted to a certain value of a variable measured by the sensors. The upper part of Fig. 3 shows, for each sensor node in a river, the temperatures registered during a certain period. Sensor nodes are denoted by a filled red square. Measures categorized as *High* (higher than 10) for the variable Temperature are denoted in red boxes over the registered measurement. The interval [10:30–11:00), where the value is *High* for all sensors, is framed in the figure. The lower part of the figure depicts the SNGraph, showing also non-sensor nodes. In the definitions next, the following notation is used: (a) An edge e between two nodes n_a and n_b is denoted $e\{n_a, n_b\}$; (b) An Attribute node is denoted $n_a\{n\}$ where n is the Object node connected to n_a; (c) A Value node is denoted $n_v\{n_a\}$ where n_a is the Attribute node connected to n_v.

Definition 4 (SN Continuous Path). *Let X be a variable that can take n possible values $x_1, x_2, \ldots, x_n$ during a certain time interval. Consider also an SNGraph G and a function $f(X)$. An* SN continuous path for $f(X)$ (SNCP) *with interval T from node s_1 to node s_k, traversing edges of type R, is a structure $P(S, R, f(X), T)$, where S is a sequence of k nodes $(s_1, \ldots, s_k)$, such that $s_i \in N_s$, $s_i.\mathsf{name} = Sensor$, and T is an interval such that $\exists(a \in N_a, v \in N_v, v\{a\{s_i\}\})$ $(a.\mathit{name} = X$, $v.\mathit{name} = f(X)$, $T = \bigcap_{i=1,k} v_i.\mathit{interval} =$ and $T \neq \emptyset)$. Between a pair (s_i, s_{i+1}) of sensor nodes, a path $e_1(s_i, n_1, R)$, $e_2(n_1, n_2, R)$, $\ldots$, $e_k(n_m, s_{i+1}, R)$ can exist, where $n_p \in N_s$ is a segment node with no sensor.* □

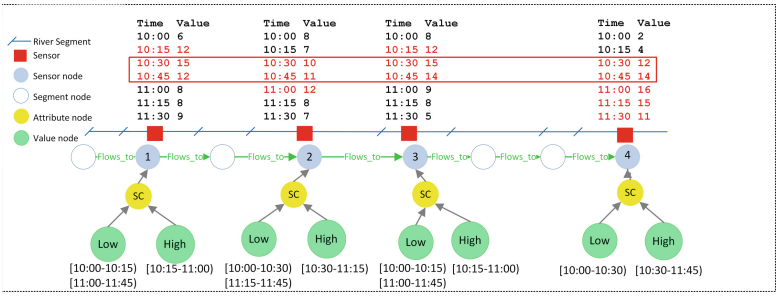

Fig. 3. SN Continuous Path with Temperature = *High* in [10:30–11:00].

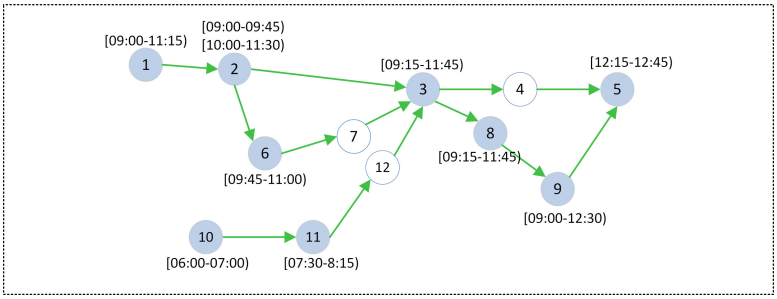

Fig. 4. Simplified SNGraph showing nodes with Temperature = *High*.

Example 1. Figure 4 depicts a simplified SNGraph where attributes and value nodes are not shown (R here is FlowsTo). The intervals tell when a *High* value of variable Temperature occurred. Filled nodes represent segments with a sensor and non-filled ones are non-sensor nodes. A query asking for all SNCPs between nodes 1 and 9, with *High* temperature values between 09:00 and 12:00, with a number of sensors between 5 and 7, returns (we use a concise notation, omitting the variable and the edge type): $Path_1$ = [(1, 2, 3, 8, 9), [09:15–09:45]]; $Path_2$ = [(1, 2, 3, 8, 9), [10:00–11:15]); $Path_3$ = [(1, 2, 6, 7, 3, 8, 9), [10:00–11:00]). □

Pairwise Continuous Path in Sensor Networks. Requiring a path to be valid throughout a time interval is a strong condition. A weaker notion of temporal path that asks for paths where there is an intersection in the intervals of every pair of consecutive sensor nodes may suffice. This is shown in Fig. 5. There is no SNCP with Temperature = *High* that involves the four sensors but the value of Temperature of the first pair was *High* during the interval [10:30–11:00), for the next two segments during [11:15–11:45), and for the last two pairs during [11:30–12:00). That means, although there is no SNCP between the four sensors, there is a consecutive chain of pairwise temporal relationships between them, denoted an *SN pairwise continuous path* (SNPCP). We omit the formal definition here.

Analogously to the above, we define an *SN Consecutive Path* (SNCP) as paths composed of sensor nodes such that, for every pair of consecutive sensors,

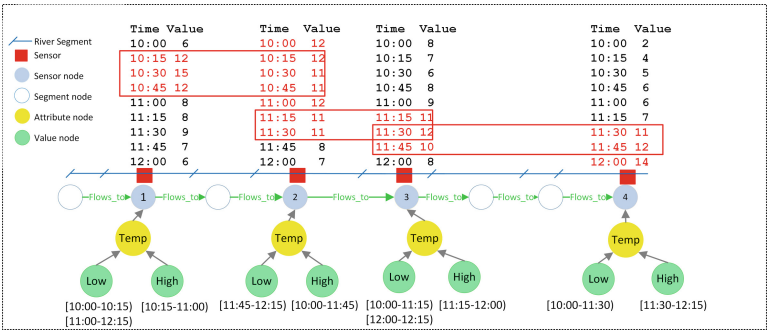

Fig. 5. SN Pairwise Continuous Path with Temperature = *High*.

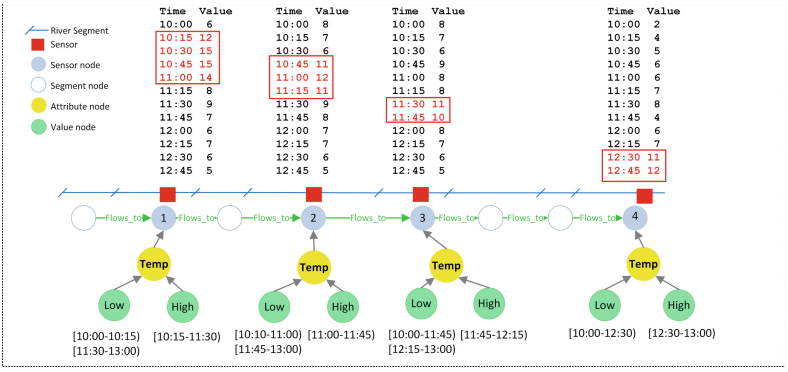

Fig. 6. Flow Path with Temperature = *High*.

the value of a function $f(X)$ is the same and the interval of the second period starts after the first one has finished. We omit the definition here, since SNCPs are included in the Flow Paths defined next.

Flow Paths in Sensor Networks. Sometimes, considering the paths above separately does not suffice to capture the characteristics of the flow. This is the case of an event that is detected by a sensor and may still be happening when is detected by the next sensor. Representing this situation requires a mixture of continuous and consecutive paths. Figure 6 depicts a *High* value of Temperature detected in one sensor earlier than the first time it is detected in the next one. Thus, the measurements overlap in the first pair of sensors but not in the other pairs. We call these paths as *Flow Paths*.

Definition 5 (Flow Path). *Let X be a temporal variable that can take n possible values $x_1, x_2, \ldots, x_n$ during a certain time interval, an SNGraph G, and a function $f(X)$. An SN Flow Path for $f(X)$ (SNFP) traversing edges of type R in G, is a structure $[S, R, f(X), T]$, where S is is a sequence of pairs $(s_1, [t_{s_1}, t_{e_1}]) \ldots, (s_k, [t_{e_k}, t_{s_k}]))$ and s_i is the i-th sensor node in S, for*

$1 \leq i \leq k$, *and* $\exists(a \in N_a, v \in N_v, v\{a\{s_i\}\})$ $(a.\mathsf{name} = X, v.\mathsf{name} = f(X)$, $[t_{s_i}, t_{e_i}] = v.\mathsf{interval}$ *and* $T = \bigcup_{i=1,k} v_i.\mathsf{interval})$. *For every pair* $(s_i, [t_{s_i}, t_{e_i}])$, $(s_{i+1}, [t_{s_{i+1}}, t_{e_{i+1}}])$, $t_{s_{i+1}} > t_{s_i}$ *holds. Between a pair of sensor nodes* (s_i, s_{i+1}), *a path* $e(s_i, n_{i1}, R), e(n_{i_1}, n_{i_2}, R) \ldots e(n_{i_m}, s_{i+1}, R)$ *can exist, where* $n_{i_p} \in N_s$ *is a segment node with no sensor.* □

Example 2. In Fig. 4, a query asking for all Flow Paths starting at node 2 such that the temperature was *High* between 09:00 and 13:00, with a minimum of 3 sensors, returns one SNFP (with Sensor nodes 2, 3, and 5): $Path_1$ = [(2, 3, 4, 5, {[09:00–9:45], [09:15–11:45], [12:15–12:45]}] (node 4 is a non-sensor one). Intervals overlap in segments 2 and 3 but not in segments 3 and 5. □

We extend T-GQL to address the new temporal paths (note the keywords `Variable` and `Value` below). An example of an SNCP query is: "Maximal time intervals (and the paths where they occurred) when temperature was *High* simultaneously, between '2022-03-10 05:00' and '2022-03-10 16:00', starting from the sensor located at segment 3. The number of sensors in the returned path must be between 3 and 5." The T-GQL expression for this query reads:

```
SELECT paths, interval MATCH (s1:Sensor), (s2:Sensor),
paths = SNCP((s1)-[:FlowsTo*3..5]-> (s2),
        '2022-03-10 05:00', '2022-03-10 16:00')
WHERE Variable = 'Temperature' AND Value = 'High'
        AND s1.id = 3;
```

5 Computing the Paths

The T-GQL language is implemented extending Cypher with a collection of procedures stored in the database's `Plugins` folder. T-GQL queries are translated into Cypher, and there is one procedure for each one of the temporal paths previously defined. Algorithm 1 describes the computation of the Flow Paths (Definition 5), the only one we include here for space reasons.

The algorithm receives a temporal graph G, the source and, optionally, the destination nodes (s and d, respectively), a variable X, a function f, a time interval I_q and a δ value that limits the time gaps between sensors. It returns a set of nodes S. To compute the solution, Algorithm 1 builds a transformed graph G_t, whose nodes contain either the interval when $f(X)$ was valid (if they are sensor nodes) or the interval of the previous sensor in the temporal graph, and the edges indicate the nodes reachable from that position. The nodes n in G_t have six attributes: a reference to the node in the original graph ($n.noderef$), a flag telling whether the node is a sensor or a non-sensor one ($isSensor$), a time interval when $f(X)$ was valid ($interval$), the number of sensors in the path $nbrOfSensors$), the number of sensors of a path that passes through that node ($length$), and a reference to the previous node in G_t, in order to allow rebuilding the paths after running the algorithm ($previous$).

After initialization, the algorithm adds the initially transformed graph node to a queue. This (sensor) node is a six-tuple that contains s, the interval time

Algorithm 1. Compute the Flow paths

Input: A graph G, a source node s, a destination node d, a variable X, a function $f(X)$, the maximum number of sensors in the path ns (optional), a query interval I_q and δ a period of time.
Output: A set with the solutions S.

```
Initialize the transformed graph G_t and Q (a queue of G_t nodes)
if (s is sensor node) then
    cInterval = f(s.X) nodes)
    if (cInterval ∩ I_q ≠ ∅) then
        Q.enqueue((s, cInterval, true, 1, 1, null))
        while not Q.isEmpty do
            curr = Q.dequeue()
            for (curr.node, interval, dest) ∈ G.edgesFrom(curr.node) do
                if not(G_t.containsNode(dest.id)) then
                    if (dest.isSensor() and dest.measures(X)) then
                        dInterval = f(dest.X)
                        if (dInterval ∩ I_q == ∅) then
                            S.add(curr)
                            continue
                        end if
                        if cInterval.start < dInterval.start then
                            newNode=(dest, dInterval, true, curr.nbrOfSensors + 1, curr.length + 1, curr)
                            if (dest == d) or (curr.nbrOfSensors == ns) then
                                S.add(newNode)
                            end if
                        end if
                    else
                        newNode=(dest,curr.interval, false, curr.nbrOfSensors,curr.length+1, curr)
                    end if
                    Q.insert(newNode)
                end if
            end for
        end while
        return S
    end if
end if
```

when $f(X)$ was valid for s, 1 as the number of sensors, 1 as the length of the path so far, and *null* as the reference to the previous node. An element *curr* is iteratively picked from the queue until the queue is empty. There is a node n_i in the temporal graph associated with *curr*. For each edge outgoing from n_i in G, there is a *dest* node associated with it. If the *dest* node is not in G_t and it is a sensor node, we obtain the interval time *dInterval* that corresponds to the times when $f(X)$ is valid for *dest*, and check that $dInterval \cap I_q \neq \emptyset$. We also check that the start time of *dInterval* is greater than the start time of *curr*. In that case, the path is expanded creating a sensor node *newNode* (the flag is true) whose interval is set as *dInterval*. If $dest = d$ or if we have reached the maximum number of sensors, *newNode* is added to S. In case *dest* is not a sensor node, the path is expanded with this node as a segment, and the interval will correspond to the previous sensor in that path (*cInterval*). When Q is emptied, the set of nodes in G_t is returned, and the algorithm reconstructs the paths following the link to the previous node until there is no such node.

Consider the following query, which computes the FPs between 06:00 and *Now* with at least five nodes.

```
SELECT paths MATCH (s1:Sensor), (s2:Sensor),
paths = SNFP((s1)-[:FlowsTo*]->(s2),'06:00','Now',5)
WHERE Variable = 'Temperature' AND Value='High' AND s1.id=10;
```

The query translated into Cypher using the underlying temporal graph structure (Fig. 2) is shown next. The SNFPs are computed using Algorithm 1.

```
MATCH (o1:Segment{name:'Sensor'}),(o2:Segment{name:'Sensor'})
WHERE o1.id = 10 AND o2.id = 8
```

```
CALL consecutive.flowSensor(o1,null,5,null,
 'Temperature','=','High', {edgesLabel:'FlowsTo',
 nodesLabel:'Segment',
 attributeLabel:'Temperature', valueLabel:'High',
 between:'06:00-Now', direction:'outgoing'})
YIELD path as internal_p1, intervals as internal_i1
WITH {path:internal_p1,intervals:internal_i1} as p
RETURN p.path as 'path', p.interval as  'intervals'
```

6 Conclusion and Future Work

We have shown how temporal graphs and temporal graph query languages can be used to model and query sensor networks. We have extended previous work in temporal graphs that allow supporting sensor networks. We have also extended the different notions of temporal paths, and added and implemented the notion of Flow path, that captures a wide variety of scenarios. The proposal has been implemented over the Neo4j graph database as a proof of concept, and our next step is to implement and test this model over large sensor network graphs using optimization techniques that we are developing, like temporal indices.

Acknowledgements. Valeria Soliani and Alejandro Vaisman were partially supported by Project PICT 2017-1054, from the Argentinian Scientific Agency.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Commun. Mag. **40**(8), 102–114 (2002)
2. Angles, R.: The property graph database model. In: Proceedings of AMW, CEUR Workshop Proceedings, Cali, Colombia, 21–25 May 2018, vol. 2100. CEUR-WS.org (2018)
3. Bollen, E., Hendrix, R., Kuijpers, B., Vaisman, A.A.: Time-series-based queries on stable transportation networks equipped with sensors. ISPRS Int. J. Geo Inf. **10**(8), 531 (2021)
4. Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., Vaisman, A.: A model and query language for temporal graph databases. VLDB J. **30**(5), 825–858 (2021). https://doi.org/10.1007/s00778-021-00675-4
5. Francis, N., et al.: Cypher: an evolving query language for property graphs. In: Proceedings of SIGMOD, Houston, TX, USA, 10–15 June 2018, pp. 1433–1445. ACM (2018)
6. Zhang, S., Yao, Y., Hu, J., Zhao, Y., Li, S., Hu, J.: Deep autoencoder neural networks for short-term traffic congestion prediction of transportation networks. Sensors **19**(10), 2229 (2019)