Probabilistic Pattern Queries over Complex Probabilistic Graphs

Alfredo Cuzzocrea
ICAR-CNR and University of Calabria
Cosenza, Italy
cuzzocrea@si.deis.unical.it

Paolo Serafino
DEIS Dept. – University of Calabria
Cosenza, Italy
pserafino@deis.unical.it

ABSTRACT
This paper introduces probabilistic pattern queries over complex probabilistic graphs, a theoretical graph model proposed by us recently for dealing with complex probabilistic graph data of modern applications characterized by uncertainty and imprecision. Effective algorithms implementing such queries are also provided.

1. INTRODUCTION
In [1] we have preliminarily introduced the so-called Complex Probabilistic Graphs (CPG) that are probabilistic graphs [8] capable of capturing linked data structures embedding both complex-modeling (e.g., [7]) and uncertainty and imprecision aspects (e.g., [2,3,4,5]). As we discuss in [1], actual graph-like data models, even with probability constructs, are not prone to capture the model requirements drawn by modern application scenarios such as linked web data (e.g., [3]), sensor networks (e.g., [2]), distributed stream systems (e.g., [6]), and so forth. One of the main novelty due to CPG graphs introduced in [1] consists of the innovative idea of associating Probability Density Function (PDF) [10] to nodes, beyond to simple confidence intervals (plus related probability) [10], like in traditional approaches (e.g., [3]). Likewise classical formulations, even in CPG graphs edges are equipped with existence probabilities in order to model the probability by which an edge can be traversed during query evaluation.

A major results of [1] is represented by the proposal for two meaningful classes of graph queries that allow us to extract useful knowledge from CPG graphs in terms of algebra-aware (sub)graphs. These queries are named Zero-memory Membership Probabilistic Query (ZMPQ) and Non-zero-memory Membership Probabilistic Query (NMMPQ), respectively [1], and make use of the well-understood reachability concept of graph models (e.g., [9]), in a probabilistic manner. As we discuss in [1], both ZMPQ and NMMPQ queries are suitable to check the probabilistic membership of a given query PDF (which, for instance, may model an event or a sequence of events) to the target CPG graph by inspecting the PDF associated to its nodes, with both the variants of assuming the availability of a memory (for the case of NMMPQ queries) or not (for the case of ZMPQ queries). With the aim of extending the research results introduced in [1], in this paper we introduce two more novel classes of queries over CPG graphs that extend the previous ones by focusing on query patterns rather than query PDF, via modeling such patterns by means of (conventional) graphs. These queries are named Zero-memory Graph Pattern Probabilistic Query (ZGPPQ) and Non-zero-memory Graph Pattern Probabilistic Query (NGPPQ), respectively. ZGPPQ and NGPPQ make use of ZMPQ and NMMPQ queries as baseline routines. We also provide effective algorithms implementing ZGPPQ and NGPPQ queries, respectively.

2. BASIC REACHABILITY-BASED MEMBERSHIP PROBABILISTIC QUERIES
In this Section, we provide the formal definitions of the basic queries of our theoretical framework, i.e. ZMPQ and NMMPQ queries.

Before introducing such query classes, some background concepts are necessary. Let us focus the attention on these concepts. In the following, given a CPG $G$ and two nodes $s$ and $v$ of $G$, we will be referring to the sequence of nodes $K_{s,v} = \langle v_0, v_1, \ldots, v_{m-1} \rangle$ as a path from node $u$ to node $w$ in $G$ if the following conditions hold: (i) $v_i \in V$ for $i \in \{0,1,\ldots,m-1\}$; (ii) $v_i \neq v_j$ for each $i \neq j$; (iii) $(v_i,v_{i+1}) \in E$ for $i \in \{0,1,\ldots,m-2\}$; (iv) $v_0 = u$; and (v) $v_{m-1} = v$.

Definition 1 introduces the so-called Path Chain Probability (PCP). Furthermore, given a CPG $G$, we denote as $R_{s,v}$ the path in $G$ from $s$ to $v$ having the greatest PCP, and as $\Psi(R_{s,v})$ the PCP of $R_{s,v}$.

Definition 1 – Path Chain Probability (PCP) – Given a CPG $G = (V,E,s,\Phi,\Psi)$ and the path $K_{s,v}$, the chain probability PCP of $K_{s,v}$, denoted as $\Psi(K_{s,v})$, is defined as follows: $\Psi(K_{s,v}) = \prod_{i=0}^{m-2} \Phi(v_i,v_{i+1})$.

Definition 2 introduces the concept of Zero-memory Membership Probabilistic Reachability (ZMPR). Given a CPG $G$ and a set of nodes $V' \subseteq V$ in $G$, the ZMPR nodes, denoted by $R_{s,\delta}(V')$, are defined as the set of nodes of $G$ that satisfy the ZMPR property, fixed a similarity threshold $\delta$ and an existence probability threshold $\xi$. Intuitively enough, the ZMPR property is an extended probabilistic membership of a set of nodes to a CPG $G$ that is characterized by the fact that it is computed (or, equally, verified) under the no-memory assumption. At a more practical level, this means that, during the evaluation for checking the ZMPR property, at the actual node, the check algorithm does not consider the probabilities associated to already visited nodes, but the probability associated to the actual node only. The ZMPR property is the conceptual basis of the first class of innovative classes of graph queries we introduce in our reachability-based framework, the so-called Zero-memory Membership Probabilistic Query (ZMPQ), which is formally defined by Definition 3.

Definition 2 – Zero-memory Membership Probabilistic Reachability (ZMPR) – Given a CPG $G = (V,E,s,\Phi,\Psi)$, a set of nodes $V' \subseteq V$, an input PDF $\Phi$, a similarity threshold $\delta \in [0,1]$ and an existence probability threshold $\xi \in [0,1]$, the following definitions hold: (i) the set of one-step ZMPR nodes, denoted by
Definition 3 - Zero-memory Membership Probabilistic Query (ZMPQ) - A ZMPQ $Q^0$ on a CPG $G$ is a tuple of kind: $Q^0 = (\Phi, \delta, \xi)$ such that: (i) $\Phi$ is an input PDF; (ii) $\delta \in [0,1]$ is a similarity threshold; (iii) $\xi \in [0,1]$ is a probability threshold over edge probabilities. The answer to $Q^0$ is defined as a sub-graph $G_Q = (V_Q, E_Q, s, \Phi_Q, \Psi_Q)$ of $G$ such that: (i) $V_Q = R_{\Phi,\delta,\xi}(s)$ is the set of ZMPR nodes from the source node $s$ of $G$; (ii) $E_Q \subseteq E \cap (V_Q \times V_Q)$ is the subset of edges in $E$ connecting nodes in $V_Q$, such that, for each $(v_1, v_2) \in E_Q$, $\Psi_Q(v_1, v_2) \geq \xi$; (iii) $\Phi_Q = \Phi|_{V_Q}$ denotes a function obtained by restricting $\Phi$ to the domain $V_Q$; (iv) $\Psi_Q = \Psi|_{E_Q}$ denotes a function obtained by restricting $\Psi$ to the domain $V_Q$.

Definition 4 instead introduces the concept of Non-zero-memory Membership Probabilistic Reachability (NMPR), which complements the ZMPR concept above. Intuitively enough, this novel property extends the previous ZMPR property via admitting that a memory is available during computing (or, equally, verifying) the probability associated to the edges, $w_{v_1v_2}$, is shown. Figure 2 shows an example NMPQ.

Definition 5 - Non-zero-memory Membership Probabilistic Query (NMPQ) - An NMPQ $Q^n$ on a CPG $G$ is a tuple of kind: $Q^n = (\Phi, \delta, \xi)$ such that: (i) $\Phi$ is an input PDF; (ii) $\delta \in [0,1]$ is a similarity threshold; (iii) $\xi \in [0,1]$ is a probability threshold over edge probabilities. The answer to $Q^n$ is defined as a sub-graph $G_Q = (V_Q, E_Q, s, \Phi_Q, \Psi_Q)$ of $G$ such that: (i) $V_Q = \bigcup_n N_{\Phi,\delta,\xi}(V^n)$ is the set of NMPR nodes from the source node $s$ of $G$; (ii) $E_Q \subseteq E \cap (V_Q \times V_Q)$ is the subset of edges in $E$ connecting nodes in $V_Q$, such that, for each $(v_1, v_2) \in E_Q$, $\Psi_Q(v_1, v_2) \geq \xi$; (iii) $\Phi_Q = \Phi|_{V_Q}$ denotes a function obtained by restricting $\Phi$ to the domain $V_Q$; (iv) $\Psi_Q = \Psi|_{E_Q}$ denotes a function obtained by restricting $\Psi$ to the domain $V_Q$.

Figure 1 shows an example ZMPQ query and Figure 2 shows an example NMPQ query, respectively, along with their answers. In particular, Figure 1 shows an example ZMPQ query evaluated on CPG $G$ (upper half of Figure 1), which will be used as example graph in the remaining part of this paper. CPG $G$ is represented according to the following conventions: labels on edges represent the probability associated to the edges, whereas, for each node $v$ of $G$, the associated PDF $\Phi(v)$ is shown. Figure 2 shows instead an example NMPQ along with its answer, evaluated on CPG $G$.

3. Extended Reachability-Based Probabilistic Pattern Queries

Based on the previous classes of graph queries, i.e. ZMPQ and NMPQ queries, we now introduce more complex graph queries that focus on extracting graph patterns rather than algebra-aware (sub)-graphs. These queries are called Zero-memory Graph Pattern Probabilistic Query (ZGPPQ) and Non-zero-memory Graph Pattern Probabilistic Query (NGPPQ), respectively. ZGPPQ and NGPPQ make use of ZMPQ and NMPQ queries as baseline routines, respectively, and they represent other classes of graph queries introduced by our reachability-based framework. Formally, ZGPPQ and NGPPQ queries are introduced by Definition 6 and Definition 7, respectively.

Definition 6 - Zero-memory Graph Pattern Probabilistic Query (ZGPPQ) - A ZGPPQ $Q^n_\Phi$ on a CPG $G$ is a tuple of kind: $Q^n_\Phi = (G_\Phi, \delta)$ such that: (i) $G_\Phi = (V_\Phi, E_\Phi)$ is an ordinary (non-

\[ R^n_{\Phi,\delta,\xi}(V^n) = \{ v \in V, v' \in V^n, (v', v) \in E, \Phi(v, v') \leq \delta, \Psi(v,v') \geq \xi \} \cup V^n \]
Definition 7 – Non-Zero-Memory Graph Pattern Probabilistic Query (NGPQP) – A NGPQP \( Q_p \) on a CPG \( G \) is a tuple of kind: \( Q_p = (G_p, \delta) \) such that: (i) \( G_p = (V_p, E_p, \Phi_p, \Psi_p) \) is an ordinary (non-CPG) undirected query graph; (ii) \( \delta \in [0,1] \) is a similarity threshold. A NGPQP \( Q_p \) returns as output a CPG graph \( G_0 \) that is composed by the answers to appropriately-selected ZMPQ queries on sub-graphs of \( G \) that are isomorphic with \( G_s \).

Since the semantics of ZGPPQ and NGPQP queries is indeed quite complex, we next describe it in terms of operational semantics, i.e. by describing the steps executed by an arbitrary (graph) query engine in order to compute the answers to such queries. For the sake of simplicity, we will refer these queries simply as GPPQ queries (indicating this class both ZGPPQ and NGPQP queries). The entire process of evaluating a GPPQ query \( Q_p \) can be divided in three main steps. The first step consists in extracting a set of candidate isomorphic sub-graphs from the target CPG \( G \), namely \( \Gamma_C = \{ G_0, G_1, ..., G|\Gamma_C|-1 \} \), such that each graph \( G_i \) in \( \Gamma_C \) is isomorphic with the query graph \( G_0 \) of \( Q_p \). In the second step, each candidate graph \( G_i \) \( \in \Gamma_C \), for each node \( v \) in \( V_i \) an MPQ (ZMPQ query for the case of ZGPPQ queries and NMPQ query for the case of NGPQP queries, respectively) query \( Q_{i\ell} = (\Phi_i(v), \delta, \xi_v) \) is evaluated on \( G_i \) such that: (i) \( \Phi_i(v) \) is the PDF of node \( v \) in \( V_i \); (ii) \( \delta \) is the input parameter of \( Q_p \); (iii) \( \xi_v \) is computed as the average existence probability of edges \((v,u)\ in E_i \) (i.e., the average of existence probabilities associated to outgoing edges of node \( v \) in \( V_i \)). The process above is iterated for each graph \( G_i \in \Gamma_C \), hence, finally, evaluating a set of MPQ queries MPQ(\( G_i \)) = \{\( Q_{i0}, Q_{i1}, ..., Q_{i|\Gamma_i|-1} \}\), Answers of queries in MPQ(\( G_i \)) are then combined, thus obtaining in output a single graph \( Y = (V_p, E_p, s, \Phi_p, \Psi_p) \) such that: (i) \( V_p = \bigcap_{i=0}^{\Theta} V_{i} \), such that \( V_{i} \) is the set of nodes of graph \( A_{i0} \); (ii) \( E_p = \bigcup_{i=0}^{\Theta} E_{i} \), such that \( E_{i} \) is the set of edges of graph \( A_{i0} \); (iii) \( s \) is the source node; (iv) \( \Phi_p(v) = \Phi(v) \); (v) \( \Psi_p = \bigcup_{i=0}^{\Theta} \Psi_{i} \). Step ii returns as output the set of graphs \( A(\Gamma_C) = \{y_0, y_1, ..., y_{|\Gamma_C|-1} \} \). Finally, in step iii the final answer \( G_0 \) is obtained by merging graphs in \( A(\Gamma_C) \). The final answer \( G_0 \) to the GPPQ query \( Q_p \) is retrieved in terms of the following tuple: \( G_0 = (V_0, E_0, s, \Phi_0, \Psi_0) \) such that: (i) \( V_0 = \bigcup_{i=0}^{\Theta} V_{i} \); (ii) \( E_0 = \bigcup_{i=0}^{\Theta} E_{i} \); (iii) the source node \( s \); (iv) \( \Phi_0(v) = \Phi(v) \); (v) \( \Psi_0 = \Psi_0 \).

4. MPQ Query Algorithms

Algorithm EvaluateZMPQ (see Figure 3) retrieves the answer to a ZMPQ query \( Q_0 \). In more detail, algorithm EvaluateZMPQ takes as input (i) a CPG \( G \), (ii) a PDF function \( \Phi_0 \), (iii) a similarity threshold \( \delta \), and returns as output a (CPG) graph answer \( G_0 \). The algorithm performs two main steps. The first step is aimed at computing the set of nodes \( V_0 \) of the graph answer \( G_0 \), whereas the second step is devoted to compute the set of edges \( E_0 \) of the graph answer \( G_0 \). We now look inside these steps in more detail.

In the first step, algorithm EvaluateZMPQ performs a visit of the input CPG \( G \) starting from the source node \( s \). During the visit of \( G \), the auxiliary stack data structure \( ST \) is used to keep track of nodes that are yet to be visited. At each iteration of the first step of the algorithm, the current node \( v \) to be processed is extracted from \( ST \). At this point, node \( v \) is put in the list \( VN \), which keeps track of visited nodes, and the value of the distance \( \chi^2 \) between \( \Phi(v) \) and the input PDF \( \Phi \) is computed by means of the auxiliary function ChisSqr. If the resulting distance between the PDF of \( v \) and the input PDF \( \Phi \) is less than \( \delta \), then \( v \) is added to \( V_0 \), otherwise \( v \) is discarded. At this point, adjacent-to-\( v \) nodes are considered for being processed by subsequent iterations, as follows. A node \( u \) that is adjacent to \( v \) is added to stack \( ST \) if both the following conditions are met: (i) \( u \) is not contained in \( VN \) (i.e., it has not already been visited in a previous iteration); (ii) the edge existence probability of edge \((v,u)\ connecting v to u \) is greater than or equal to \( \ell \). The first step ends after that all nodes of CPG \( G \) have been visited.

<table>
<thead>
<tr>
<th>Algorithm EvaluateZMPQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Graph ( G ), ( \Phi_0 ), ( \delta ), ( \xi )</td>
</tr>
<tr>
<td>Output: Graph ( G_0 )</td>
</tr>
<tr>
<td>BEGIN</td>
</tr>
<tr>
<td>( ST := \emptyset ); ( VN := \emptyset ); ( V_0 := \emptyset ); ( E_0 := \emptyset );</td>
</tr>
<tr>
<td>Push((ST,v));</td>
</tr>
<tr>
<td>WHILE ((</td>
</tr>
<tr>
<td>( v := \text{Pop}(ST) );</td>
</tr>
<tr>
<td>Add((VN,v));</td>
</tr>
<tr>
<td>IF ChisSqr((GetPhi(v), \Phi_0) &lt;= \delta) DO</td>
</tr>
<tr>
<td>Add((V_0,v));</td>
</tr>
<tr>
<td>DONE</td>
</tr>
<tr>
<td>FOREACH ((u \in \text{AdjacentNodes}(v))) DO</td>
</tr>
<tr>
<td>IF ((\text{Contains}(VN,u) \text{ AND } \text{GetPsi}(u,v) &gt;= X_1)) DO</td>
</tr>
<tr>
<td>Push((ST,u));</td>
</tr>
<tr>
<td>DONE</td>
</tr>
<tr>
<td>DONE</td>
</tr>
<tr>
<td>E := getEdges((G));</td>
</tr>
<tr>
<td>FOREACH ((u,v) \in E) DO</td>
</tr>
<tr>
<td>IF ((\text{ChisSqr}(\text{GetPhi}(v), \Phi_0) &lt;= \delta \text{ AND } \text{Contains}(V_0,u) \text{ AND } \text{Contains}(V_0,v))) DO</td>
</tr>
<tr>
<td>Add((E_0, (u,v)));</td>
</tr>
<tr>
<td>DONE</td>
</tr>
<tr>
<td>DONE</td>
</tr>
<tr>
<td>RETURN ( G_0 = \text{BuildGraph}(V_0, E_0) )</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

Figure 3. Algorithm EvaluateZMPQ.
Algorithm \textsc{EvaluateNMPQ} (see Figure 4) instead, retrieves the graph answer $G_{Q}$ to an NMPQ query $Q^{0}$. The structure of algorithm \textsc{EvaluateNMPQ} is quite similar to the one of algorithm \textsc{EvaluateZMPQ}. Particularly, algorithm \textsc{EvaluateNMPQ} takes as input a CPG graph $G$, a PDF $\Phi$, a similarity threshold $\delta$, and an edge existence probability threshold $\xi$. Much like algorithm \textsc{EvaluateZMPQ}, algorithm \textsc{EvaluateNMPQ} is composed of two main steps such that the first step is aimed at computing the set of nodes of the graph answer $G_{Q}$, namely $V_{Q}$, whereas the second step is targeted at computing the set of edges of the graph answer $G_{Q}$, namely $E_{Q}$. Furthermore, algorithm \textsc{EvaluateNMPQ} makes use of the following auxiliary data structures: (i) stack $ST$, which keeps track of nodes that are yet to be visited; (ii) list $VN$, which keeps track of visited nodes; (iii) associative array PCP, such that each entry PCP[$v$] records the value of the path having the greatest PCP value from the source node $s$ to node $v$. The key difference between algorithm \textsc{EvaluateZMPQ} and algorithm \textsc{EvaluateNMPQ} consists in the criterion according to which a node $u$ that is adjacent to the current node $v$ is added to stack $ST$. Indeed, algorithm \textsc{EvaluateNMPQ} computes the current PCP value (see Section 2) of a node $v$ and stores this value in the associative array PCP, so that, at any time, PCP[$v$] stores the current best PCP value of node $v$. In more detail, at each iteration of the algorithm, nodes $v$ are added to stack $ST$ if both the following conditions are met: (i) $v$ has not yet been visited; (ii) the current PCP of $v$ is higher than the input threshold $\xi$ (e.g., the latter criterion is checked by means of the value stored in the associative array PCP). In the second step, algorithm \textsc{EvaluateNMPQ} builds the set $E_{Q}$ of edges of the graph answer $G_{Q}$ by inspecting each edge of $G$ and retaining in $E_{Q}$ just those edges $(u, v)$ for which the following conditions are met: (i) $u$ belongs to $V_{Q}$, namely $u \in V_{Q}$; (ii) $v$ belongs to $V_{Q}$, namely $v \in V_{Q}$; (iii) edge $(u, v)$ belongs to a path from the source node $s$ whose PCP is higher than the input threshold $\xi$. After the second step, algorithm \textsc{EvaluateNMPQ} terminates returning as output the graph answer $G_{Q}$.

Algorithm \textsc{EvaluateNMPQ}

\begin{verbatim}
Input: Graph $G$, $\Phi$, $\delta$, $\xi$
Output: Graph $G_{Q}$
BEGIN
$ST := \emptyset$; $VN := \emptyset$; $V_{Q} := \emptyset$; $E_{Q} := \emptyset$; $PCP := \emptyset$;
Push(ST, s);
WHILE ($\text{Size}(ST) != 0$) DO
  $v := \text{Pop}(ST)$;
  Add(VN, $v$);
  IF (ChisSqr(GetPhi($v$), $\Phi$) <= $\delta$ AND Contains(VQ, $v$) AND Contains(VQ, $\text{GetPsi}(VQ, v)$) AND $PCP[v] * \text{GetPsi}(VQ, v) > \xi$) DO
    Add(EQ, (u, v));
  IF (ChisSqr(GetPhi($u$), $\Phi$) <= $\delta$ AND Contains(VQ, $u$) AND Contains(VQ, $\text{GetPsi}(VQ, u)$) AND $PCP[u] * \text{GetPsi}(VQ, u) > \xi$) DO
    Add(EQ, (u, v));
FOREACH (u in AdjacentNodes(v)) DO
DONE
RETURN $G_{Q} = \text{BuildGraph}(V_{Q}, E_{Q})$;
END
\end{verbatim}

5. \textsc{GPPQ QUERY ALGORITHMS}

Algorithm \textsc{EvaluateZGPPQ} (see Figure 5) retrieves the answer to a ZGPPQ query $Q^{0}$. In more detail, algorithm \textsc{EvaluateZGPPQ} takes as input (i) a CPG graph $G$, (ii) a query graph $G_{Q}$, (iii) a similarity threshold $\delta$, and returns as output a (CPG) graph answer $G_{Q}$. The algorithm is composed by three main steps. In the first step, the algorithm computes the set of subgraphs of the CPG $G$ that are isomorphic with graph $G_{Q}$ (see Section 3). This is accomplished by means of procedure \textsc{ComputesIsomorphism}, which takes as input a CPG graph $G$, a query graph $G_{Q}$ and returns as output the set of all possible subgraphs of $G$ that are isomorphic with $G_{Q}$. These subgraphs are then stored in the array $G_{2}$. In the second step, each subgraph $G_{1}$ of $G$ contained in $G_{2}$ is processed.

In particular, for each node $v$ of a subgraph $G_{1}$, algorithm \textsc{EvaluateZGPPQ} first computes the average edge existence probability $\xi'_{v}$ of all edges incident on $v$. Afterwards, a ZMPQ $Q^{0} = (\Phi(v), \delta, \xi')$ is evaluated over $G_{1}$ and the graph answer $A_{1}$ is stored in the auxiliary data structure $Y$. After that, all graph answers collected in $Y$ are combined yielding a single graph answer $Y_{I}$ for each isomorphic subgraph $G_{1}$ in $G_{2}$. In particular $Y_{1}$ is obtained from sub-graphs in $Y$ as follows. The set of nodes on $Y_{1}$ is obtained as the intersection among sets of nodes of graphs contained in $Y$, whereas the sets of edges of $Y_{1}$ is obtained as the intersection among the sets of edges of graphs contained in $Y$. Graph $Y_{1}$ is finally added to the auxiliary data structure $A_{C}$.

When the second step of algorithm \textsc{EvaluateZGPPQ} is completed, $A_{C}$ contains a list of sub-graphs, one for each isomorphic sub-graph $G_{1}$ in $G_{2}$. Finally, in the third step, all graphs collected in list $A_{C}$ are combined into a single graph answer $G_{Q}$ that is finally returned as output. Graph answer $G_{Q}$ is obtained from graphs in $A_{C}$ as follows. The set of nodes of $G_{Q}$ is obtained as the union among sets of nodes of graphs in $A_{C}$, whereas the set of edges of $G_{Q}$ is obtained as the intersection among the sets of edges of graphs in $A_{C}$.

Algorithm \textsc{EvaluateNGPPQ} (see Figure 6), instead, retrieves the answer to a NGPPQ query $Q^{0}$. In more detail, algorithm \textsc{EvaluateNGPPQ} takes as input (i) a CPG graph $G$, (ii) a query graph $G_{Q}$, (iii) a similarity threshold $\delta$, and returns as output a (CPG) graph answer $G_{Q}$. Algorithm \textsc{EvaluateNGPPQ} is very
similar to algorithm EvaluateZGPPQ but with the remarkable difference that, in the second step, NMPQ queries are executed instead of ZMPQ queries.

Algorithm EvaluateZGPPQ

Input: Graph G, Graph G₀, δ
Output: Graph G₀
BEGIN
Γ := ComputeIsomorphism(G, G₀)
Answer := Ø
ATC := Ø
FOREACH (Gᵢ in Γ) DO
  FOREACH (v in GetNodes(Gᵢ)) DO
    ξ := Ø
    FOREACH (u in AdjacentNodes(v)) DO
      ξ := ξ / GetPsi(u, v)
    DONE
    ξ := ξ / AdjacentNodesNumber(v)
    γᵢ := EvaluateZMPQ(GetPhi(v), δ, ξ)
    Add(ATCᵢ, γᵢ)
  DONE
  Eᵢ := E
  Vᵢ := V
  FOREACH (γᵢ in ATCᵢ) DO
    Eᵢ := SetIntersection(Eᵢ, GetNodes(γᵢ));
    Vᵢ := SetIntersection(Vᵢ, GetEdges(γᵢ));
  DONE
  Add(ATCᵢ, BuildGraph(Eᵢ, Vᵢ));
  DONE
E₀ := Ø;
V₀ := Ø;
FOREACH (G in ATC) DO
  E₀ := SetUnion(E₀, GetNodes(G));
  V₀ := SetUnion(V₀, GetEdges(G));
DONE
RETURN G₀ = BuildGraph(V₀, E₀);
END

Figure 5. Algorithm EvaluateZGPPQ.

6. CONCLUSIONS AND FUTURE WORK

With the idea of complementing previous research results we proposed in [1], where the CPG theoretical model has been introduced, in this paper we have further extended [1] via providing two novel classes of probabilistic pattern queries over CPG graphs that extend basic queries proposed in [1]. We have also provided effective algorithms implementing these novel classes of queries. Future work is mainly devoted to experimentally assessing the performance of our framework against both synthetic and real-life complex probabilistic graph data sets, and to incorporate into our framework advanced query classes such as queries such as aggregation and top-k queries.

Algorithm EvaluateNGPPQ

Input: Graph G, Graph G₀, δ
Output: Graph G₀
BEGIN
Γ := ComputeIsomorphism(G, G₀)
Answer := Ø
ATC := Ø
FOREACH (Gᵢ in Γ) DO
  FOREACH (v in GetNodes(Gᵢ)) DO
    ξ := Ø
    FOREACH (u in AdjacentNodes(v)) DO
      Add(Aᵢ, BuildGraph(Eᵢ, Vᵢ));
      RETURN
DONE
END

Figure 6. Algorithm EvaluateNGPPQ.

7. REFERENCES