Conditional Graph Pattern Matching with a Basic Static Analysis

Houari Mahfoud

LRIT Laboratory, Abou-Bekr Belkaid University, Tlemcen, Algeria
houari.mahfoud@univ-tlemcen.dz

Abstract. We propose conditional graph patterns (CGPs) that extend conventional ones with simple counting quantifiers on edges, attributes on nodes, positive and negative predicates. In emerging applications such as social network marketing, CGPs allow to express complex search conditions and to find more sensible information than their traditional counterparts. We show that the CGPs expressivity does not come with a much higher price. Indeed, we propose a matching algorithm that is based on a revised notion of graph simulation and allows to match CGPs over any data graphs in quadratic time, as opposed to the prohibitive solutions based on subgraph isomorphism. We discuss a parallel version of our algorithm that makes it very efficient over real-life data. We investigate the satisfiability and containment problems of CGPs and we show that they are in quadratic time by providing a non-trivial checking algorithm for each one. This paper is the first effort that investigates static analysis of non-conventional graph patterns containing important features that are widely used in practice. An extensive experimental study has been conducted to show effectiveness and efficiency of our results.

Keywords: Isomorphism · Graph simulation · Predicates · Conditional patterns · Parallel matching · Static analysis · Satisfiability · Containment

1 Introduction

Given a data graph $G$ and a graph pattern $Q$, graph pattern matching is to find all subgraphs of $G$ that match $Q$. Matching here is typically expressed in terms of subgraph isomorphism which consists to find all subgraphs of $G$ that are isomorphic to $Q$. However, subgraph isomorphism is an NP-COMPLETE problem [4]. Moreover, the number of matches via subgraph isomorphism may be exponential in the size of the data graph. To reduce this cost, graph simulation [18] and its extensions [5,6,15] have been proposed that allow graph pattern matching to be conducted in polynomial time. Unlike subgraph isomorphism that defines a one-to-one mapping, these extensions map a node of a graph pattern to many nodes of a data graph, which allows overcoming the NP-Completeness aspect. However, many complex graph patterns are needed in practice and which
are not expressible by existing simulation-based models, notably patterns with counting quantifiers (CQs), predicates and negation as we illustrate by the next example.

Example 1. I) Consider the patterns of Fig. 1 where node labels represent type of entities (e.g. Patient, User); edge labels represent type of the relationships that exist between these entities (e.g. is_friend); a label of the form ‘≥k’ specifies quantification on edges (e.g. number of treatments prescribed for a patient). Moreover, dotted-line blocks with ‘+’ (resp. ‘−’) represent positive (resp. negative) predicates that must be satisfied (resp. unsatisfied) by some nodes. P depicts a request that is defined by a nurse identified by “123” in order to return information of all patients that have at least five long-term treatments. Suppose that the hospital wants to impose an access control policy as follows: 1) only nurses of medical staff can access information of patients; 2) a nurse can access only information of patients which are treated by a doctor working with her; and 3) information of top-secret patients (e.g. diplomat patients) cannot be shown to the nurses. To enforce this policy, one may easily rewrite P into a safe one by adding some parts that guarantee (resp. prevent) access to accessible (resp. inaccessible) patient information. The safe version of P is given by the pattern Ps that defines a positive and negative predicate over the node Patient to enforce respectively conditions 1–2 and condition 3. In other words, a patient must not be returned to the nurse if he belongs to the Top-Secret category or is being treated by a doctor which does work with the underlying nurse. The result of Ps over any data graph must be composed by only the entities Patient and Treatment along with their edge, while the remaining parts of P are used only to refine (make safe) this result. II) The pattern Q is useful in social media marketing. It looks for each user x in UK which has at least n friends whose recommend a Sony product; and no friend of x gave a bad rating since 2018 for a Sony product after buying one. The result of Q will be composed by single nodes representing profile of the users satisfying predicates of Q. □
To the best of our knowledge, no existing graph pattern matching model can express the queries \( P_s \) and \( Q \) of the previous example. Thus, we revised the notion of graph simulation to strike a balance between its computational complexity and its ability to deal with more expressive graph patterns.

**Contributions and Road-Map.** (i) We propose conditional simulation that extends dual simulation \([15]\) by supporting simple \( CQs \), positive and negative predicates (Sect. 3). (ii) We show that conditional simulation remains in \( \text{PTIME} \) as earlier extensions of graph simulation \([5,6,14,15]\) by providing a cubic-time computation algorithm (Sect. 4). (iii) We discuss a parallel version of this algorithm that allows efficient matching of \( CGPs \) over large data graphs. (iv) We investigate the problem of satisfiability and containment (Sect. 5) of our graph patterns and we show that these problems are solvable in quadratic time by providing the corresponding checking algorithms. (v) Using a real-life graph, we experimentally verify the performances of our algorithms (Sect. 6).

**Related Work.** We classify previous work as follows.

**Graph Pattern Matching.** Graph pattern matching (GPM) is typically defined in terms of subgraph isomorphism which is an \( \text{NP}-\text{COMPLETE} \) problem \([19]\). Moreover, it is often too restrictive to capture sensible matches \([6]\) as it requires matches from a data graph to have the same topology as a graph pattern, which hinders its applicability in real-life applications such as social network marketing. To overcome these limits, graph simulation \([18]\) has been first adopted for graph pattern matching due to its low computational complexity. GPM via graph simulation has been useful for various applications (e.g. social position detection \([1]\)). However, it preserves only downward mappings which raised a need to extend it in order to find more sensible matches in emerging applications. There has been a host of work on improving the notion of graph simulation (e.g. bounded simulation \([6]\), regular simulation \([5]\), dual simulation \([15]\)). Unfortunately, all existing extensions remain limited in emerging applications that need patterns with complex features, notably counting quantifiers (\( CQs \)), predicates and negation. Among these features, only \( CQs \) have received some attention. The visual tool \( QGraph \) \([12]\) allows annotating edges of a graph pattern with \( CQS \) of the form \([\text{min}, \text{max}]\) that can express different semantics (e.g. negated (\([0, 0]\)) and optional (\([0, 1]\)) edges). The formal matching algorithm of \( QGraph \) (whose complexity is undefined) is a heavy extension of subgraph isomorphism which makes it probably more higher than \( \text{NP}-\text{COMPLETE}\). Authors of \([8]\) proposed \( CQs \) that can express numeric and ratio aggregates of the forms: “\( =p(\%) \)”, “\( \geq p(\%) \)”, and “\( =0 \)” for negation. Their matching algorithm is based on subgraph isomorphism and has been proved to be \( \text{DP}-\text{COMPLETE} \).

In contrast, our first motivation was to use simple \( CQs \) that are less expressive than those of \([8,12]\) but make quantified matching in \( \text{PTIME} \). Moreover, we allow negation of any part of a graph pattern which is more useful then negation of

\[^{1}\] As observed in \([8]\).
simple edges. At the time of writing this paper, we do not know any approach that allows definition and matching of predicates within graph patterns.

**Static Analysis of Graph Patterns.** The satisfiability and containment problems are classical and fundamental problems for any query language. They have been well studied for XPath (e.g., [10, 11]). For graph patterns however, it is striking how little attention has been paid for these problems. The containment problem has been studied in [5] for graph patterns without neither predicates nor negation. The satisfiability problem has been studied in [16] for child relationships only, which raised for a trivial solution. Apart from these works, we are not aware of other ones that investigate static analysis of graph pattern queries.

## 2 Background

### 2.1 Data Graphs and Graph Patterns

A **Data Graph** is a directed graph \( G = (V, E, \mathcal{L}, \mathcal{A}) \) where: 1) \( V \) is a finite set of nodes; 2) \( E \subseteq V \times V \) is a finite set of edges in which \((v, v')\) denotes an edge from node \( v \) to \( v' \); 3) \( \mathcal{L} \) is a function that assigns a label \( \mathcal{L}(v) \) (resp. \( \mathcal{L}(e) \)) to each node \( v \in V \) (resp. edge \( e \in E \)); and 4) for each node \( v \in V \), \( \mathcal{A}(v) \) is a tuple \((A_1 = c_1, \ldots, A_n = c_n)\) where: \( A_i \) is an attribute of \( v \), \( c_i \) is a constant value, \( n \geq 0 \), and \( A_i \neq A_j \) if \( i \neq j \).

Intuitively, for any data node \( v \), the label \( \mathcal{L}(v) \) represents the type of \( v \) (e.g. **Movie**) while the tuple \( \mathcal{A}(v) \) defines its properties (e.g. **title**, **release date**, **running time**). Moreover, since an edge relies two entities (e.g. **Movie** and **Person**), then its label may represent a relationship between them (e.g. **produced by**, **edited by**). Notice that the attributes defined over a node \( v \in V \) may be different for those defined over another node \( w \in V \), even if \( \mathcal{L}(v) = \mathcal{L}(w) \).

A **Subgraph** \( G_s = (V_s, E_s, \mathcal{L}_s, \mathcal{A}_s) \) of \( G = (V, E, \mathcal{L}, \mathcal{A}) \) must satisfy: 1) \( V_s \subseteq V \); 2) \( E_s \subseteq E \); 3) \( \mathcal{L}_s(x) = \mathcal{L}(x) \) for each \( x \in E_s \cup V_s \); and 4) \( \mathcal{A}_s(v) = \mathcal{A}(v) \) for each \( v \in V_s \).

A **Ball** in \( G \) with center \( v \) and radius \( r \) is a subgraph of \( G \), denoted by \( B(G, v, r) \), s.t.: 1) all nodes \( v' \) are in \( B(G, v, r) \) if the number of hops between \( v' \) and \( v \) is at most \( r \); and 2) it has exactly the edges appearing in \( G \) over the same node set.

A **Graph Pattern**\(^2\) is a directed connected graph \( Q = (V, E, \mathcal{L}, \mathcal{A}) \) where: 1) \( V \), \( E \), and \( \mathcal{L} \) are defined as for data graphs; and 2) for each node \( u \in V \), \( \mathcal{A}(u) \) is a predicate defined as a conjunction of atomic formulas of the form “\( A \ op \ c \)” where: \( A \) is an attribute of \( u \), \( c \) is a constant, and \( \{\geq, \leq, =, \neq\} \).

For any pattern node \( u \), we call \( \mathcal{A}(u) \) the **attributes constraints** of \( u \) which specify a search condition: e.g. a movie released in **US** after 2017 (i.e. \( \mathcal{A}(u) = \text{“country=US \land year > 2017”} \)). If \( \mathcal{A}(u) = \emptyset \) then \( \mathcal{L}(u) \) is the only search condition for \( u \) as in [8, 15]. Notice that conflicting constraints over \( u \) (e.g. \( \mathcal{A}(u) = \text{“Age} \geq 25 \land \text{Age} < 20” \)) should be considered by the satisfiability checking.

\(^2\) Inspired from [5, 6].
The **Diameter** of $Q$, written $d_Q$, is the longest distance between all pairs of nodes in $Q$. That is, $d_Q = \max(\text{dist}(v, v'))$ for all pairs $(v, v')$ in $Q$ where: $\text{dist}(v, v')$ is the length of the shortest undirected path from $v$ to $v'$.

### 2.2 Conventional Graph Pattern Matching Revisited

We slightly revise subgraph isomorphism [9], graph simulation [18] and dual simulation [15] to take in account labels on edges and attributes on vertices. We refer next to the data graph $G = (V, E, \mathcal{L}, \mathcal{A})$ and the graph pattern $Q = (V_Q, E_Q, \mathcal{L}_Q, \mathcal{A}_Q)$.

**Subgraph Isomorphism.** A subgraph $G_s = (V_s, E_s, \mathcal{L}_s, \mathcal{A}_s)$ of $G$ matches $Q$ via **subgraph isomorphism** if there exists a bijective function $f: V_Q \to V_s$ s.t.: 1) for each $u \in V_Q$, $\mathcal{L}_Q(u) = \mathcal{L}_s(f(u))$; 2) for each $e = (u, u')$ in $E_Q$, there exists an edge $e' = (f(u), f(u'))$ in $G_s$ with $\mathcal{L}_Q(e) = \mathcal{L}_s(e')$; and 3) for each atomic formula “$A \text{ op } c$” in $\mathcal{A}_Q(u)$, there exists $A = c'$ in $\mathcal{A}_s(f(u))$ where $c'$ op $c$ holds.

**Graph Simulation.** $G$ matches $Q$ via **graph simulation** if there exists a binary match relation $S \subseteq V_Q \times V$ s.t.: 1) for each $(u, v) \in S$, $\mathcal{L}_Q(u) = \mathcal{L}(v)$; 2) for each $(u, v) \in S$ and each atomic formula “$A \text{ op } c$” in $\mathcal{A}_Q(u)$, there exists $A = c'$ in $\mathcal{A}(v)$ where $c'$ op $c$ holds; 3) for each $(u, v) \in S$ and each edge $e_u = (u, u')$ in $E_Q$, there exists an edge $e_v = (v, v')$ in $E$ with: $(u', v') \in S$ and $\mathcal{L}_Q(e_u) = \mathcal{L}(e_v)$; 4) for each node $u \in V_Q$, there exists at least one node $v \in V$ with $(u, v) \in S$.

By condition (3), graph simulation preserves only child relationships.

**Dual Simulation.** $G$ matches $Q$ via **dual simulation** if there exists a binary match relation $S_D \subseteq V_Q \times V$ s.t.: 1) for each $(u, v) \in S$, $\mathcal{L}_Q(u) = \mathcal{L}(v)$; 2) for each $(u, v) \in S$ and each atomic formula “$A \text{ op } c$” in $\mathcal{A}_Q(u)$, there exists $A = c'$ in $\mathcal{A}(v)$ where $c'$ op $c$ holds; 3) for each $(u, v) \in S$ and each edge $e_u = (u, u')$ (resp. $e_u = (u', u)$) in $E_p$, there exists an edge $e_v = (v, v')$ (resp. $e_v = (v', v)$) in $E$ with $(u', v') \in S$ and $\mathcal{L}_Q(e_u) = \mathcal{L}(e_v)$; 4) for each node $u \in V_Q$, there exists at least one node $v \in V$ with $(u, v) \in S$.

Dual simulation enhances graph simulation by preserving parent relationships.

**Match Result.** When $G$ matches $Q$ via subgraph isomorphism, the match result is the set of all subgraphs of $G$ that are isomorphic to $Q$. Moreover, the match result w.r.t. a binary match relation $S \subseteq V_Q \times V$ (i.e. produced by dual simulation or its extensions) is a subgraph $G_s$ of $G$ s.t.: 1) a node $v \in V_s$ if it matches some node of $Q$ via $S$; and 2) an edge $e = (v, v')$ is in $E_s$ if there exists an edge $(u, u') \in E_Q$ s.t. $\{(u, v), (u', v')\} \in S$. 


3 Conditional Graph Patterns

3.1 Definition of CGPs

Contrary to complex CQs proposed in [12] that lead for a prohibitive cost, we use simple but useful CQs in order to strike a balance between the expressivity and matching cost.

Definition 1. A QGP is a connected directed graph \( Q = (V, E, L, A, C) \) where:

1) \( V, E, L, A \) are defined as for conventional graph patterns; 2) for each edge \( e \in E \), \( C(e) \) is a CQ given by an integer \( p \) (\( p \geq 1 \)).

Intuitively, for any data graph \( G \) and any edge \( e = (u, u') \) in \( E \) with \( C(e) = p \), a node \( v \) from \( G \) matches \( u \) if it has at least \( p \) children that match \( u' \), and moreover, these children must be reached from \( v \) via an edge labeled \( L(e) \). As a special case, \( C(e) = 1 \) expresses existential quantification. Contrary to [8], we do not allow CQ of the form \( C(e) = 0 \) since its semantic would be ambiguous with negation. Using QGPs as building blocks, we next define CGPs.

Definition 2. A CGP is a connected directed graph \( Q = (V, E, L, A, C, P) \) where:

1. \((V, E, L, A, C)\) is a QGP called the core of \( Q \) and denoted by \( Q^c \); and
2. For each node \( u \in V \), \( P(u) \) is given by \( l_1 \land \ldots \land l_k \) s.t.: a) \( k \geq 0 \); b) each \( l_i \) is given by \( q_i \) (positive predicate) or \( \text{not}(q_i) \) (negative predicate); c) \( q_i = (V_i, E_i) \) is a QGP satisfying \( V_i \cap V = \{u\} \).

Remark that CGPs extend QGPs with the function \( P \) that allow definition of positive and negative predicates. Intuitively, \( P(u) \) assigns a (possibly empty) conjunction of positive and/or negative predicates to the vertex \( u \) of \( Q^c \). Each positive (resp. negative) predicate specifies a quantified and attributed graph-based condition\(^3\) that must be satisfied (resp. unsatisfied) by each match of \( u \) in a data graph. This kind of predicates give our matching model some expressivity that is not covered by existing approaches [5,6,8,12,15].

The whole semantic of \( Q \) can be stated naturally with the friendly syntax "match \( Q^c \) where \( P(u_1) \land \ldots \land P(u_k) \)" (\( u_i \in V, 1 \leq i \leq |V| \)). This means that a data graph \( G \) matches \( Q \) if it has a subgraph \( G_s \) that satisfies all constraints expressed by \( Q^c \) (child and parent relationships, attribute constraints, and CQs). Moreover, for any node \( u \in Q^c \), each match of \( u \) in \( G_s \) must satisfy (resp. unsatisfy) all positive (resp. negative) predicates given by \( P(u) \). Inspired by well-known conditional languages (e.g. SQL, XPath), the core \( Q^c \) of \( Q \) represents the structure of the match result that will be returned to the user, predicates of \( Q \) are used only during the matching process to refine this match result.

The size of a QGP \( q' = (V', E') \) is given by \(|V'| + |E'|\), while the size of a CGP \( Q = (V, E, L, A, C, P) \) is given by \(|V| + |E| + \sum_{u \in V} (\sum_{q' \in P(u)} |q'|) + \)

\(^3\) This is why we call them conditional graph patterns.
∑_{not(q')\in\mathcal{P}(u)} |q'|). In other words, the size of $Q$ is given by the number of vertices and edges that belong to the core $Q^c$, positive and/or negative predicates of $Q$.

We simply denote the CGP $Q$ as $(V, E)$ when it is clear from the context.

**Example 2.** Consider the CGPs of Fig. 2. According to Definition 2 (condition c), remark that each predicate intersects with the core of the original CGP in only one node\(^4\). The QGP $Q_1$ returns all professors ($Pr$), their PhD students ($PhD$), and the articles ($Article$) published by these latter. The QGP $Q_2$ is a special case of $Q_1$ since it requires: 1) each PhD student to have at least two published articles; and 2) each professor to be from $UK$ and between 38 to 45 years old. The CGP $Q_3$ returns pairs of nodes composed by professors and their PhD students providing that each of these student have exactly two published articles. This restriction is imposed by the conjunction of a positive and negative predicate defined over the node $PhD$ of $Q_3$. This means that each PhD node from a data graph is returned by $Q_3$ only if it has a parent node labeled $Pr$ and satisfies each predicate of $Q_3$ (has exactly two children labeled $Article$).

**Fig. 2.** Examples of CGPs.

**Definition 3.** The positive version of a CGP $Q$, denoted by $Q^+$, is a QGP that contains nodes and edges belonging to the core and/or positive predicates of $Q$; by keeping their labels and attributes constraints unchanged.

### 3.2 Conditional Simulation

We introduce conditional simulation that extends dual simulation of Ma et al. [15] with attribute constraints, $CQ$s, positive and negative predicates. Our extension allows matching of large data graphs in $\text{PTIME}$ via more expressive patterns.

**Definition 4.** A data graph $G = (V, E, \mathcal{L}, \mathcal{A})$ matches a CGP $Q = (V_Q, E_Q, \mathcal{L}_Q, \mathcal{A}_Q, \mathcal{C}_Q, \mathcal{P}_Q)$ via conditional simulation, denoted by $Q \preceq G$, if there exists a binary match relation $S^G_Q \subseteq V_Q \times V$ s.t.:\(^4\) This may not reduce the practicability of our approach since many query languages (e.g. XPath, SQL) adopt this syntax of predicates.
1. For each \((u, v)\) \(\in S^G_Q\), \(L_Q(u) = L(v)\).
2. For each \((u, v)\) \(\in S^G_Q\) and each atomic formula “\(A \text{ op } m\)” in \(A_Q(u)\), there exists a tuple “\(A = n\)” in \(A(v)\) where “\(n \text{ op } m\)” holds.
3. For each \((u, v)\) \(\in S^G_Q\) and each \(C_Q(u, u') = n\), \(v\) has at least \(v_1, \ldots, v_n\) children in \(G\) where: \(L(v, v_i) = L_Q(u, u')\) and \((u', v_i)\) \(\in S^G_Q\) for \(i \in [1, n]\).
4. For each \((u, v)\) \(\in S^G_Q\) and each edge \(e = (u', u)\) in \(E_Q\), there exists an edge \((v', v)\) in \(E\) where: \(L(v', v) = L_Q(u', u)\) and \((u', v')\) \(\in S^G_Q\).
5. For each \((u, v)\) \(\in S^G_Q\) and each \(q\) (resp. \(\text{not}(q)\)) in \(P_Q(u)\), there is a ball (resp. no ball) \(B(G, v, d_q)\) s.t.: \(q \prec_C B(G, v, d_q)\) with a match relation \(S\), \((u, v)\) \(\in S\).
6. There is at least one couple \((u, v)\) \(\in S^G_Q\) for each \(u \in V_Q\).

Conditions (1–2) concern node properties: a pattern node \(u\) can be matched by any data node \(v\) that has its label and whose attribute values satisfy attribute constraints of \(u\). Conditions (3) and (4) check respectively the satisfaction of \(CQs\) and parent relationships. Condition (5) specifies that for each match \(v\) of \(u\), the ball centered at \(v\) in \(G\) satisfies (resp. does not satisfy) each positive (resp. negative) predicate defined over \(u\). Finally, condition (6) states that there are matches in \(G\) for any node in \(Q\).

As graph simulation and its counterparts, conditional simulation aims to find the maximum version of \(S^G_Q\), that is all subgraphs of \(G\) that satisfy topology and constraints of \(Q\). Based on this notion, we make the following result\(^5\).

**Proposition 1.** For any CGP \(Q\) and data graph \(G\) with \(Q \prec_C G\), there is a unique maximum match relation \(S^G_Q\) for \(G\) and \(Q\).

We denote by \(M_C(Q, G)\) the match result that corresponds to this maximum match relation \(S^G_Q\). It can be constructed as explained in Sect. 2.2.

**Proposition 2.** For any CGP \(Q = (V_Q, E_Q)\) and data graph \(G = (V, E, A)\) with \(Q \prec_C G\), \(M_C(Q, G)\) is constructed w.r.t the maximum match relation \(S^G_Q\) in \(O(|V_Q||V| + |E_Q||E|)\) time.

\(\square\)

4 An Algorithm for **CGPs** Matching

4.1 Description and Complexity Analysis

Figure 3 shows our CGPs matching algorithm, referred to as \(\text{MATCH}^C\), that inputs a CGP \(Q\) and a data graph \(G\), and returns \(\emptyset\) if \(Q \not\prec_C G\) or the corresponding maximum matching relation \(S^G_Q\) otherwise. Firstly, the match relation \(S^G_Q\) is initialized by matching each node in \(Q\) with all nodes of \(G\) that have its label (line 1). Since \(Q^c \not\prec_C G\) implies \(Q \not\prec_C G\), then \(\text{MATCH}^C\) starts by refining the initial version of \(S^G_Q\) w.r.t \(Q^c\) constraints only (lines 2–3). This step is ensured by the algorithm \(\text{MATCH}^Q\) that is explained below. If the call of

\(^5\) One can prove this result by leveraging that of dual simulation [15].
Fig. 3. Algorithm for CGPs matching.

**Algorithm** \( \textup{MATCH}^Q(Q, G) \)

**Input:** A CGP \( Q=(V_Q, E_Q, L_Q, A_Q, C_Q, \mathcal{P}_Q) \) and a data graph \( G=(V, E, L, A) \).

**Output:** The maximum match relation \( S^G_Q \) if \( Q \prec_C G \), and \( \emptyset \) otherwise.

1. \( S^G_Q := \{(u, v) \mid u \in V_Q, v \in V, L_Q(u) = L(v)\}; \) /* Initialize the match relation \( S^G_Q \) */
2. \( S^G_Q := \textup{Match}^Q(Q^c, G, S^G_Q); \) /* Refine \( S^G_Q \) w.r.t the core of \( Q \) */
3. if \( (S^G_Q = \emptyset) \) then return \( \emptyset; \) /* \( Q^c \not\prec_C G \) */
4. for each \( (u, v) \in S^G_Q \) and each \( q \in \mathcal{P}_Q(u) \) do /* Whether \( v \) satisfies \( q \) */
5. if \( (\text{not}(q \prec_C B(G, v, d_q) \text{ with } S \text{ and } (u, v) \in S)) \) then \( S^G_Q := S^G_Q \setminus \{(u, v)\}; \) end if;
6. end for
7. for each \( (u, v) \in S^G_Q \) and each \( q \in \mathcal{P}_Q(u) \) do /* Whether \( v \) does not satisfy \( q \) */
8. if \( (q \prec_C B(G, v, d_q) \text{ with } S \text{ and } (u, v) \in S) \) then \( S^G_Q := S^G_Q \setminus \{(u, v)\}; \) end if;
9. end for
10. \( S^G_Q := \textup{Match}^Q(Q^c, G, S^G_Q); \)
11. return \( S^G_Q \).

**Algorithm** \( \textup{MATCH}^Q(Q^c, G, S) \)

**Input:** A CGP \( Q=(V^c, E^c, L^c, A^c, C^c) \), a data graph \( G=(V, E, L, A) \), and an initial match relation \( S \) to refine.

**Output:** A refined version of \( S \) that ensures \( Q^c \prec_C G \), and \( \emptyset \) if this is not possible.

1. for each \( (u, v) \in S \) and each \( "A op m" \in A^c(u) \) do
2. if \( \left( \exists \text{ "A = m" } \in A(v) \text{ s.t } (m \text{ op } n) \text{ holds} \right) \) then \( S := S \setminus \{(u, v)\}; \) end if;
3. end for
4. do
5. for each \( (u, v) \in S \) and each \( e_u = (u', u) \in E^c \) do
6. if \( \left( \exists \text{ } e_{u'} = (v', v) \in E : L^c(e_u) = L(e_{u'}), (u', v') \in S \right) \) then \( S := S \setminus \{(u, v)\}; \) end if;
7. end for
8. for each \( (u, v) \in S \) and each \( (u, u') \in E^c \) do
9. if \( (G^c(u, u') < C^c(u, u')) \) then \( S := S \setminus \{(u, v)\}; \) end if;
10. end for
12. while there are changes in \( S \);
13. if \( \left( \forall u \in V^c, \exists v \in V : (u, v) \in S \right) \) then return \( S \); else return \( \emptyset \);
The algorithm MATCH\(^Q\) of Fig. 3 extends dual simulation of Ma et al. [15] by attribute constraints, labeled edges, and specially by CQs. The algorithm checks whether \(Q^c \prec_c G\) for any data graph \(G\) and QGP \(Q^c\) in input. This is done by checking conditions (1 – 4 + 6) of Definition 2. Given a match relation \(S\) (defined by algorithm MATCH\(^C\)) that maps nodes of \(Q^c\) to nodes of \(G\), the goal is to eliminate all incorrect matches from \(S\) as follows. A couple \((u, v)\) is deleted from \(S\) if:

1. an atomic formulas “\(A \text{ op } m\)” (e.g. “\(\text{age} \geq 35\)”) is defined over \(u\) in \(Q^c\) but there is no tuple “\(A = n\)” over \(v\) in \(G\) that holds this formulas (e.g. “\(\text{age} = n\)” with \(n < 35\)) [lines 1–5];
2. there exists an edge \((u', u)\) in \(E^c\) with label \(l\) but \(v\) has no parent node \(v'\) in \(G\) that matches \(u'\) and reaches \(v\) with an edge labeled \(l\) (lines 7–9); iii) \(u\) must have at least \(n\) instances of the child \(u'\) in \(Q^c\) (i.e. \(C^c(u, u') = n\)) but the number of children of \(v\) in \(G\) that match \(u'\) (i.e. size of the set \(C(v, u, u')\)) is lesser than \(n\) (lines 10–13). The two latter cases are repeated (lines 6–14) until there is no incorrect match in \(S\). Finally, if each node in \(Q^c\) gets some matches from \(S\) then \(Q^c \prec_c G\) and the refined version of \(S\) that ensures this matching is returned; otherwise \(Q^c \not\prec_c G\) and \(\emptyset\) is returned.

**Lemma 1.** For any QGP \(Q\), any data graph \(G\) and any match relation \(S \subseteq V_Q \times V\), algorithm MATCH\(^Q\) takes at most \(O(|Q||G|)\) time to refine \(S\). □

**Theorem 1.** For any CQP \(Q\) and any data graph \(G\), it takes \(O(|Q||G|)\) time to determine whether \(Q \prec_c G\) and to compute the maximum match relation \(S^G_Q\). □

### 4.2 Parallel Version

To allow efficient matching of CGPs over large data graphs, we discuss here a parallel version of our algorithm MATCH\(^C\), referred to as PMATCH\(^C\) (not shown here due to page limit). The intuition behind this version is as follows: rather to evaluate the core and predicates of a given CGP \(C\) in a sequential manner, the goal is to match each of them over the underlying data graph \(G\) via separated threads. If the data graph does not match the core and/or at least one positive predicate of \(C\) then we return null. Otherwise, we combine the match relations produced by the different threads to compute the final match result.

As stated in [15], the parallel scalability is an important property that each parallel algorithm must verify. This guarantees that the more processors are used, the less time is taken by the parallel algorithm. We practically prove this property over large data graph in Sect. 6. Even thought PMATCH\(^C\) is quite simple, its allows to improve sequential matching time on average 39.48%–60.74%.

### 5 Static Analysis

We introduce first the notion of matching between QGPs that plays an important role in static analysis tasks as satisfiability and containment checking.
5.1 Pattern-Only Matching

The notion of Pattern-only Matching (PoM) is to check for two graph patterns $Q$ and $R$, whether $Q$ matches $R$. This means that there is at least one sub-pattern of $Q$ that satisfies all constraints of $R$.

**Definition 5.** Given two QGPs $Q = (V_Q, E_Q, L_Q, A_Q, C_Q)$ and $R = (V_R, E_R, L_R, A_R, C_R)$. For any nodes $u \in V_R$ and $w \in V_Q$, we say that $A_Q(w)$ matches $A_R(u)$, written $A_Q(w) \sim A_R(u)$, if the constraints defined over any attribute $A$ in $A_R(u)$ are satisfied by those defined over $A$ in $A_Q(w)$. □

Intuitively, $A_Q(w) \sim A_R(u)$ if for any data graph $G$ and any node $v$ in $G$, if $v$ satisfies $A_Q(w)$ then it also satisfies $A_R(u)$. For instance, the constraint “age > 25” satisfies the constraint “age ≠ 20”, but the inverse does not hold.

**Definition 6.** Consider the QGPs $Q = (V_Q, E_Q, L_Q, A_Q, C_Q)$ and $R = (V_R, E_R, L_R, A_R, C_R)$. We say that $Q$ matches $R$, denoted by $R \prec Q$, if there exists a binary match relation $S \subseteq V_R \times V_Q$ s.t.:

1. for each $(u, w) \in S$: $L_Q(w) = L_R(u)$ and $A_Q(w) \sim A_R(u)$.
2. for each $(u, w) \in S$ and each edge $e_u = (u', u)$ in $E_R$, there exists an edge $e_w = (w', w)$ in $E_Q$ with: $(u', w') \in S$ and $L_Q(e_w) = L_R(e_u)$.
3. for each $(u, w) \in S$ and each edge $e_u = (u, u') \in E_R$, there is an edge $e_w = (w, w') \in E_Q$ s.t.: $(u', w') \in S$, $L_R(e_u) = L_Q(e_w)$, $C_Q(e_w) \geq C_R(e_u)$.
4. For each node $u \in V_R$, there exists a node $w \in V_Q$ with $(u, w) \in S$. □

Intuitively, $R \prec Q$ if for any data graph $G$ that matches $Q$, $G$ matches $R$ too.

**Definition 7.** Given two QGPs $Q = (V_Q, E_Q, L_Q, A_Q, C_Q)$ and $R = (V_R, E_R, L_R, A_R, C_R)$ s.t. $R \prec Q$ with the maximum match relation $S$. We say that an edge $e_w = (w, w')$ in $Q$ is covered by $S$ if there exists an edge $e_u = (u, u')$ in $R$ s.t.: $(u, w)$ and $(u', w')$ are in $S$, and $L_Q(e_w) = L_R(e_u)$. Moreover, we denote by $S^Q$ the set of all edges of $Q$ that are covered by $S$. □

**Example 3.** Consider $Q_1$ and $Q_2$ of Fig.2. As $Q_2$ makes more restrictions over nodes and edges of $Q_1$, any data graph $G$ that matches $Q_2$ will match $Q_1$ too, i.e. $Q_1 \prec Q_2$. We can see that $Q_2 \not\prec Q_1$ since there exists a data graph $G$ that matches $Q_1$ but not $Q_2$. Consider a QGP $Q_5$ given by $Team \xrightarrow{\text{member} \geq 1} Pr \xrightarrow{\text{supervise} \geq 2} PhD \xrightarrow{\text{publish} \geq 2} Article$. Remark that $Q_1 \prec Q_5$ even if $Q_5$ has nodes (resp. edges) that match no nodes (resp. edges) in $Q_1$. If $S$ is the maximum match relation that ensures $Q_1 \prec Q_5$, then $S^{Q_5}$ is given by $\{Pr \xrightarrow{\text{supervise} \geq 2} PhD_2, PhD \xrightarrow{\text{publish} \geq 2} Article\}$ while $Pr \xrightarrow{\text{member} \geq 1} Team$ is not covered by $S$. □

**Lemma 2.** For any QGPs $Q$ and $R$, it takes at most $O(|Q||R|)$ time to check whether $R \prec Q$ and to determine the corresponding maximum match relation. □
5.2 Satisfiability of CGPs

The satisfiability problem discussed in this section is to determine, given a CGP $Q$, whether or not there exists a data graph $G$ that matches $Q$ via conditional simulation, i.e. $Q \prec_C G$. Contrary to existing graph pattern models [2,5,6,15], the problem is more intriguing in our case due to the presence of predicates.

Example 4. Given the CGP $Q_4$ of Fig. 2, it is clear that no data graph $G$ can match $Q_4$ since each subgraph of $G$ that matches the positive predicate of $Q_4$ mismatches its negative predicate and vice versa. 

\[ \square \]

Theorem 2. The satisfiability of any CGP $Q$ can be checked in $O(|Q|^2)$ time. \[ \square \]

Given a CGP $Q = (V,E,L,A,C,P)$. One of the unsatisfiability cases is due to the presence of conflicting attribute constraints over the vertices of $Q$ (e.g. “Age > 25 & Age < 20”). Moreover, $Q$ is unsatisfiable if there exist conflicts between a negative predicate $\text{not}(q) \in P(u)$ and the constraints defined over $u$ in $Q^+$. This can be checked in quadratic time thanks to the notion of PoM (Lemma 2).

5.3 Containment of CGPs

As widely defined in the literature [11,13], the containment problem is to determine whether all nodes and edges returned by a query $Q$ are also returned by the query $R$. We revisit this traditional definition via conditional simulation as follows.

Definition 8. A CGP $Q$ is contained in a CGP $R$ via conditional simulation, written $Q \subseteq_C R$, if for any data graph $G$: $M_C(Q,G) \subseteq M_C(R,G)$. 

The static checking of $Q \subseteq_C R$ may help decreasing matching time over a large data graph $G$ as follows. If $R \not\prec_C G$ then we deduce statically that $Q \not\prec_C G$, and if $Q \prec_C G$ then we do not need to test $R$ anymore since $R \prec_C G$ holds automatically. In general, if $Q \subseteq_C R$ then we can consider $M_C(Q,G)$ as an upper-evaluation of $R$ over $G$; and $M_C(R,G)$ as an over-evaluation of $Q$ over $G$.

We give next necessary and sufficient conditions for the containment of CGPs.
Lemma 3. For any CGPs $Q = (V_Q, E_Q, L_Q, A_Q, C_Q, P_Q)$ and $R = (V_R, E_R, L_R, A_R, C_R, P_R)$, $Q \subseteq C R$ iff:

1. $R^+ \prec Q^+$ with a match relation $S^{Q}_{R} \subseteq V^{+}_{R} \times V^{+}_{Q}$;
2. For each $(u, w) \in S^{Q}_{R}$ and not $(q_u) \in P_R(u)$, there exists not $(q_w) \in P_Q(w)$: $q_w \prec q_u$ with the match relation $S$ and $(u, w) \in S$.
3. Each edge in $Q$ matches at least one edge in $R$ via $S^{Q}_{R}$.  

Example 5. Consider the data graph $G$ and the CGPs $Q_i$ of Fig. 4 ($i \in [1, 5]$). Based on Lemma 3, one can deduce that: $Q_2 \subseteq C Q_1$ (all conditions satisfied), $Q_4 \subseteq C Q_1$ (all conditions satisfied), $Q_5 \not \subseteq C Q_1$ (condition 1 unsatisfied), $Q_3 \not \subseteq C Q_5$ (condition 2 unsatisfied), $Q_1 \not \subseteq C Q_3$ (condition 3 unsatisfied).

Theorem 3. For CGPs $Q$ and $R$, $Q \subseteq C R$ is checked in $O(|Q||R|)$ time.

6 Experimental Study

We conducted extensive experiments to evaluate the (1) effectiveness of $QGPM$ and $CGPM$; (2) effectiveness of parallel $CGPM$; (3) scalability of algorithms $Match^Q$, $Match^C$ and $PMatch^C$, and (4) parallel scalability of $PMatch^C$.
Experimental Settings. We used the Amazon real-life network dataset, that is a product co-purchasing graph with 548K nodes and 1.78M edges. Each node has attributes such as ID, Title, Rating, and Group, and an edge from product u to w indicates that people who buy u would also buy w with a high probability.

Using Java language, we implemented the algorithms (1) Match\(Q\); (2) Match\(C\); and (3) PMatch\(C\), the parallel version of Match\(C\) that detects the number of processors and dispatches core and predicates of the CGP over them. Moreover, a data graph extractor has been implemented to extract an Amazon subgraph data with a fixed number of vertices.

All the experiments were run on a Ubuntu desktop machine with an Intel Core i5-8250u CPU, 8GB memory, and 256GB SSD storage. Each experiment was run 5 times and the average is reported.

Experimental Results. We present next our findings.

Experiment 1. We first evaluated the performances of quantified and conditional GPM using an Amazon data graph that has 10000 nodes and 37854 edges.

Varying \(|V_Q|\) and \(|E_Q|\) (Match\(Q\)): we generated 8 QGPs with sizes \((|V_Q|, |E_Q|, |C_Q|)\) ranging from \((4, 4, 4)\) to \((10, 10, 20)\). As shown in Fig. 5 (a), we remark that: (i) the larger \(|V_Q|\) is, the longer time is taken by algorithm Match\(Q\), as expected; (ii) \(|V_Q|\) influences on matching time more than \(|E_Q|\).

Varying \(|C_Q|\) (Match\(Q\)): fixing \(|V_Q| = 10\) and \(|E_Q| = 20\) (a very large QGP), we varied the number of CQs from 0 (query with only existential quantifications) to 20. As shown in Fig. 5(c), the number of CQs influences slightly in matching time which is consistent with the fact that our quantified simulation does not increase cost of dual simulation (Lemma 1).

Varying \(|A_Q|\) (Match\(Q\)): using the same QGP of size \((10, 20)\), we varied the number of attribute constraints from 0 to 20 by adding incrementally two attribute constraints (e.g. “Group = Book”, “Rating ≥ 3”) to each vertex of the pattern. As remarked in Fig. 5(c), attribute constraints may reduce slightly the matching time taken by Match\(Q\). This is due to the fact that attribute constraints reduce the size of the initial match relation (lines 1–5 of Fig. 3) which leads for slight (considerable) reduction of matching time. For instance, the average of vertex matches ranges from 2946 (\(|A_Q| = 0\)) to 1814 (\(|A_Q| = 20\)).

Varying \(|P^+_Q|\) and \(|P^-_Q|\) (Match\(C\)): we defined two CGPs \(C_1\) and \(C_2\) with cores of size \((|V_Q|, |E_Q|, |C_Q|, |A_Q|) = (5, 10, 10, 5)\). We varied the number of positive (resp. negative) predicates from 0 to 10 in \(C_1\) (resp. \(C_2\)) by adding incrementally two predicates of sizes \((3, 2, 2, 2)\) to each vertex of \(C_1\) (resp. \(C_2\)). The results are reported in Fig. 5(b). As expected, each predicate requires to be evaluated over the underlying data graph which increases the matching time of the whole CGPs. Match\(C\) gives good performance for a reasonable number of predicates: it takes around 7 seconds for three positive (resp. negative) predicates since the whole

\(^{6}\) http://snap.stanford.edu/data/index.html.
size\(^7\) of \(C_1\) (resp. \(C_2\)) becomes \((11, 16, 16)\), which is a little bit large compared to real-life patterns. Contrary to what one may deduce from the figure, positive and negative predicates require theoretically the same computational time.

**Experiment 2.** This set of experiments evaluated the performances of PMATCH\(^C\) compared to MATCH\(^C\). Using the same data set as **Experiment 1**, we generated a CGP with a core of size \((|V_Q|, |E_Q|, |C_Q|, |A_Q|) = (3, 6, 6, 3)\). We varied the number of predicates from 0 to 10 by incorporating fairly 5 positive predicates and 5 negative predicates to different vertices of the core. As shown in Fig. 5(d), the running time of PMATCH\(^C\) increases very slightly contrary to MATCH\(^C\). This is due to the fact that our machine has 4 cores with hyper-threading technology that allows to execute 8 threads at the same time, hence, the increasing number of predicates influences slightly at the matching time. Remark that PMATCH\(^C\) outperforms MATCH\(^C\) by 2.7 times, a significant improvement.

**Experiment 3.** We evaluated the scalability of algorithms MATCH\(^Q\), MATCH\(^C\) and PMATCH\(^C\). We generated a CGP \(C\) with a core of size \((3, 4)\), a positive and a negative predicates of size \((3, 4)\) too. This gives a whole graph pattern of a large size, \((7, 16)\). We generated ten data graphs whose sizes varied from \((10^4, 37854)\) to \((10^5, 391545)\). For algorithm MATCH\(^Q\), we compute only the time needed to evaluate the core of \(C\). As shown in Fig. 5(e), (1) time of PMATCH\(^C\) is close to that of MATCH\(^Q\) since the core and predicates of \(C\) have the same size and are matched in parallel; (2) PMATCH\(^C\) scales well with large data graph and is on average 39.48\%–60.74\% times faster than MATCH\(^C\).

**Experiment 4.** We evaluated the parallel scalability of algorithm PMATCH\(^C\). We generated a CGP \(C\) with a core of size \((3, 6)\), three positive and three negative predicates of size \((3, 2)\). This gives a whole graph pattern of size \((15, 18)\). Next, we computed the running times of PMATCH\(^C\) by varying the number of CPU cores from 1 to 4. In the second part of this experiment, we disabled the hyper-threading technology to see its impact in matching time. As shown in Fig. 5(f): (1) PMATCH\(^C\) is parallel scalable since the more cores are used, the less time it takes to match \(C\) over the data graph, (2) enabling hyper-threading allows to increase slightly matching time of CGPs. This suggests to use more CPU cores in order to efficiently match CGPs over large data graphs.

### 7 Conclusion

We have proposed *conditional graph patterns* (CGPs) that extend traditional graph patterns with edge quantifications of the form “\(\geq k\)”, attribute constraints on nodes, positive and negative predicates. Despite their increased expressivity, CGPs do not make our lives much harder which is proved by our matching algorithm that runs in quadratic time. In the second part of this paper, we have studied satisfiability and containment of CGPs that are fundamental problems

\(^7\) Number of vertices and edges that belong to the core and/or predicates.
of any query language. We have shown that these problems are decidable in low \textit{ptime} by providing two checking algorithms. Our experimental results have verified the effectiveness, efficiency and scalability of our algorithms, using real-life data. As future work, we will investigate some strategies to speed up matching of CGPs such \textit{data graphs distribution} [3] and \textit{incremental matching} [7].

References

1. Brynielsson, J., Högberg, J., Kaati, L., Mårtenson, C., Svenson, P.: Detecting social positions using simulation. In: ASONAM
2. Cao, Y., Fan, W., Huai, J., Huang, R.: Making pattern queries bounded in big graphs. In: ICDE
3. Cong, G., Fan, W., Kementsietsidis, A.: Distributed query evaluation with performance guarantees. In: SIGMOD
4. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell., 1367–1372 (2004)
5. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y.: Adding regular expressions to graph reachability and pattern queries. In: ICDE (2011)
6. Fan, W., Li, J., Ma, S., Tang, N., Yinghui, W., Yinpeng, W.: From intractable to polynomial time. In: VLDB Endowment, Graph Pattern Matching (2010)
7. Fan, W., Wang, X., Wu, Y.: Incremental graph pattern matching. ACM Trans. Database Syst., 18:1–18:47 (2013)
8. Fan, W., Wu, Y., Xu, J.: Adding counting quantifiers to graph patterns. In: SIGMOD (2016)
9. Gallagher, B.: Matching structure and semantics: a survey on graph-based pattern matching. In: The AAAI Fall Symposium Series
10. Geerts, F., Fan, W.: Satisfiability of X path queries with sibling axes. In: DBPL
11. Genevès, P., Layaïda, N.: A system for the static analysis of X path. ACM Trans. Inf. Syst. 24(4), 475–502 (2006)
12. Blau, D.J.H., Immerman, N.: A visual language for querying and updating graphs. University of Massachusetts, Technical report (2002)
13. Kostylev, E.V., Reutter, J.L., Vrgoc, D.: Containment of queries for graphs with data. J. Comput. Syst. Sci. 92, 65–91 (2018)
14. Liu, G., et al.: Multi-constrained graph pattern matching in large-scale contextual social graphs. In: ICDE (2015)
15. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Strong simulation: capturing topology in graph pattern matching. ACM Trans. Database Syst. 39(1), 1–46 (2014)
16. Ma, S., Li, J., Hu, C., Liu, X., Huai, J.: Graph pattern matching for dynamic team formation. CoRR, abs/1801.01012 (2018)
17. Mahfoud, H.: Graph pattern matching with counting quantifiers and label-repetition constraints. Clust. Comput. 23(3), 1529–1553 (2019). https://doi.org/10.1007/s10586-019-02977-3
18. Milner, R.: Communication and Concurrency. Prentice-Hall Inc., Upper Saddle River (1989)
19. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM 23(1), 31–42 (1976)