A Visual Language for Querying and Updating Graphs

H. Blau  N. Immerman  D. Jensen

{blau, immerman, jensen}@cs.umass.edu

Department of Computer Science, University of Massachusetts, Amherst, MA 01003-9264

Abstract

QGRAPH is a new visual language for querying and updating graph databases. In QGRAPH the user can draw a query consisting of some vertices and edges with specified relations between their attributes. The response will be the collection of all subgraphs of the database that have the desired pattern. QGRAPH is very useful for knowledge discovery. QGRAPH has a powerful and elegant counting feature that enables the user to easily specify how many of certain objects and links should exist in order for a subgraph to match a query. [...]
Figure 1: Graphical data fragment from a movie database

Figure 2: Find all Person, ActorIn, Movie subgraphs

2.1 Conditions

The query in Figure 2 finds all subgraphs with an ActorIn link between a Person and a Movie. The type restrictions are expressed by conditions on the two vertices and one edge of the query. In this example only one attribute is tested in each condition; in general a condition can be any boolean combination of restrictions on attribute values.

A, B, and X are unique labels assigned to each vertex and edge in the query. We use letters at the beginning of the alphabet for vertices, and those from the end of the alphabet for edges. The labels have no intrinsic meaning and do not indicate anything about the type of object or link that would match the labeled element. Where desired, type restrictions are enforced with conditions on vertices and edges.

For the sample database of Figure 1, this query produces 8 matches (Figure 3). Unlike the SELECT statement in SQL, a QGRAPH query does not specify which attributes of matching objects and links should be included in the result. Evaluating a QGRAPH query returns a collection of all the matching subgraphs from the database. The user can examine any subgraph in the resulting collection, and any object or link in that subgraph, with the user interface. All the object and link attributes, not just those mentioned in the query conditions, are available for inspection.

2.2 Numeric annotations

To group the actors together for each movie, we add a numeric annotation to the Person vertex (Figure 4). Executing this query against the database produces 4 matches (Figure 5), one for each movie, compared with 8 matches for the matching subgraphs. A query with only match vertices and edges serves to identify and display a collection of subgraphs. To match the query, a subgraph must have the correct structure and satisfy all the boolean conditions and constraints. A query with both match and update vertices and edges can be used for attribute calculation and for structural modification of the database. The query processor first finds the matching subgraphs using the query’s match elements, then makes changes to those subgraphs as indicated by the query’s update elements.

Figure 1 is an example of the graph databases for which we designed QGRAPH. Our database consists of objects, binary links, and attributes that record features of the object or link. An object or link can have zero or more attributes. All attributes are set-valued. For example, a person can have multiple names. The figure shows a fragment from a database about movies. The labels on the objects indicate their name, and the labels on links indicate their type. Not shown in the figure are other attributes of objects, such as the year a movie was released or the location of a studio. Similarly, links could have attributes, such as the salary an actor received for starring in a given movie.
query without the numeric annotation (Figure 3). A numeric annotation can be specified on a vertex or an edge of a QGRAPH query. (We will see in Section 2.6 that a subquery can also have a numeric annotation.) A numeric annotation takes one of three forms. An **unbounded range** \([i,\ldots]\) on a vertex (or edge) means at least \(i\) instances of the annotated object (or link) must be present in any matching database fragment. A **bounded range** \([i..j]\) means at least \(i\) and no more than \(j\) instances are required for a match. An **exact annotation** \([i]\) means exactly \(i\) instances are required. \(i\) can be any integer \(\geq 0\); \(j\) can be any integer \(> i\). If the lower end of the possible range is 0, the annotated structure is optional in any matching database fragment. (The annotation \([0..j]\) is not allowed because it would be ambiguous between \([0..j]\) and \([1..j]\).) The annotation \([0]\) on a vertex (or edge) indicates negation: to match the query, a database fragment must not contain the corresponding object (or link). To be well-formed, a query must remain a connected graph when any optional or negated structures (annotations \([0]\), \([0,\ldots]\), or \([0..\ldots]\)) are removed. To avoid ambiguities of interpretation, only one of any two adjacent vertices can be annotated.

A numeric annotation serves two purposes in a query. It **groups together into one match repeated isomorphic substructures that would otherwise create multiple matches for the query** (compare Figures 3 and 5). It places limits on how many such structures can occur in matching portions of the database. To group the substructures without limiting their number, we use the annotation \([1,\ldots]\) (as in Figure 4). There is no mechanism in QGRAPH to limit the number of matching substructures without grouping them together. If we changed the annotation on the vertex A in

---

**Figure 3:** Matches for query in Figure 2

**Figure 4:** For each movie, find all its actors

**Figure 5:** Matches for query in Figure 4
Figure 6: Mysteries with fewer than 3 female actors and no Oscar awards

Figure 7: Movies nominated for Best Picture in 1997 that did not win

Figure 4 to be [1..2] instead of [1..], then the subgraph on the right-hand side of 5 would no longer be a match for the query. The subgraphs on the left-hand side would still be matches.

The edge $X$ of Figure 4 also has an annotation [1..]. An edge incident to an annotated vertex must itself be annotated. The annotation on the vertex takes precedence over the annotation on the edge. We first find all the actors for a specific movie, then for each of those actors we find all the ActorIn links that connect the actor to the movie. For example, an actor who played multiple roles in a particular movie might have multiple ActorIn links to the same Movie object. The annotation [1..] groups all these links into a single match. To avoid clutter in the following examples, we have omitted the annotation [1..] from the edges adjacent to annotated vertices. The annotation [1..] is implicit unless some other annotation is specified on the edge.

The query of Figure 6 selects mystery movies that never received an Oscar and have fewer than three female actors. A movie that has won no awards at all, or has won awards that are not Oscars, could match this query. The movie Sleuth (1972) is a match. Sleuth had only one female actor (Eve Channing) and won no Oscars, although it did win an Edgar Allan Poe Award and a New York Film Critics Circle Award. If we wanted only movies that have won no awards at all, we would drop the conjunct AwardType = Oscar from the condition on node C.

A negated element (annotation [0]) does not show up in the results of a query, because a subgraph matches the query only if it has no object (or link) matching the negated vertex (or edge). For the query of Figure 6, no Award objects or Awarded links would appear in the results. Person objects and ActorIn links would appear only in matches for movies that had exactly one or two female actors, such as Sleuth. They would not appear in matches for movies that had no female actors.

The query of Figure 7 selects movies that were nominated for the Best Picture Oscar in 1997 but did not win. This query illustrates a numeric annotation on a link. The movies As Good as It Gets, The Full Monty, Good Will Hunting, and L.A. Confidential match this query.