SwarmGuide: Towards Multiple-Query Optimization in Graph Databases

Zahid Abul-Basher[§] Nikolay Yakovets[†]
Parke Godfrey[†] Mark Chignell[§]

[§]University of Toronto, Toronto, ON, Canada
`{zahid,chignell}@mie.utoronto.ca`
[†]York University, Toronto, ON, Canada
`{hush,godfrey}@cse.yorku.ca`

1 Introduction

Analytic tasks over graph databases often require sequences of related queries to be evaluated. While there are promising query optimization methods for graph queries [6], these do not optimize globally for sets of queries. For relational, *multiple query optimization* (MQO) has been studied [5]. We propose Swarmguide, a framework for MQO for graph databases.

2 Graph Databases & Queries

Preliminaries. A *graph database* G is a finite, directed, edge-labeled, multigraph defined by $G = \langle N, \Sigma, E\rangle$, where N is a finite set of nodes (vertices), Σ is a set of labels, E is a set of directed, labeled edges, and $E \subseteq N \times \Sigma \times N$. A *path* p in G is defined as a sequence of $n_0 a_0 n_1 \cdots n_{k-1} a_{k-1} n_k$ where $n_i \in N$, $a_i \in \Sigma$, and $\langle n_i, a_i, n_{i+1}\rangle \in E$ for $0 \leq i \leq k$. We call the sequence of edge labels Σ^* of a particular path p the *word*, $\omega(p)$ that p induces.

A *regular* path rp is a path in the graph where $\omega(rp)$ is a word in a given regular language $L(\text{reg})$ (*e.g.*, $\omega(rp) \in L(\text{reg})$). A regular path query (RPQ) [2] is a triple $\langle x, reg, y\rangle$ in which x and y are free variables over the domain of nodes, and reg is a regular expression. An answer of an RPQ is a node-pair $\langle s, t\rangle$ ($s, t \in N$) such that there is a path p in G between s and t and $\omega(p) \in L(\text{reg})$. The *answer set* of an RPQ is the set of all its answers. The RDF data model and SPARQL query language instantiate these concepts. With the introduction of *property paths* in SPARQL 1.1, the query language encompasses RPQs. For our examples in this paper, we adopt SPARQL syntax.

Evaluation of RPQs. Evaluating an RPQ is a two-step procedure. *Path recognition* determines whether the word of path p, $\omega(p)$, is recognized by the regular expression reg (*i.e.*, $\omega(p) \in L(\text{reg})$); *path search* finds pairs of nodes, $s, t \in N$ such that there exists such a path p from node s to node t. There has been recent interest in how to evaluate RPQs efficiently in SPARQL over RDF stores. Yakovets *et al.* [6] demonstrate in their Waveguide framework that there is a rich plan space for such queries, and that different plans for a given query can

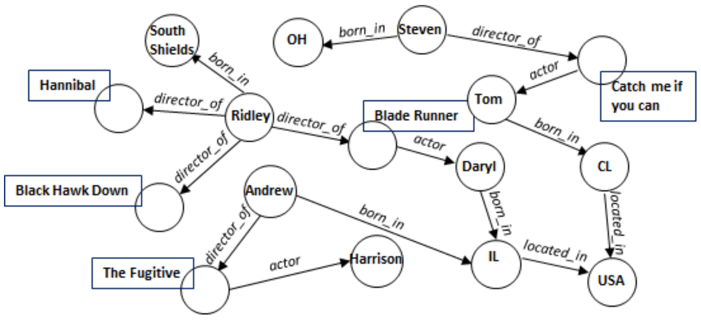

Fig. 1: Example movie graph

differ by orders of magnitude in evaluation cost. A simple cost model for such a plan is how many edges traversals (edge-walks) in the graph are made during the plan's evaluation.

Consider $Q_1 = \langle x, director\_of/actor/born\_in, y\rangle$ over the movie graph in Figure 1. This query can be evaluated in different ways, with different edge-walk costs. We can evaluate it in forward-direction by retrieving all node-pairs connected by edge label $director\_of$ then appending with those linked by $actor$ and then by $born\_in$. We can also evaluate it in the backward direction by retrieving all node-pairs connected by edge label $born\_in$ then prepending with those connected by $actor$ and then by $director\_of$. A third way is from the middle by retrieving all node-pairs connected by edge label $actor$ then appending with those connected by $born\_in$ and then prepending by $director\_of$. Note that, the edge-walk cost for the first is 10, in the second is 9, and in the third is 7.

In Waveguide, path search is performed efficiently while *simultaneously* recognizing the path expressions. Waveguide's input is a graph database G and a *waveplan* (WP) P_Q which *guides* a number of search *wavefronts* that explore the given graph. The term *wavefront* is introduced to refer to a part of the plan that evaluates breadth-first during the evaluation. This graph exploration, driven by an iterative search procedure, is inspired by the semi-naïve bottom-up strategy used in the evaluation of linear recursive expressions based on a *fixpoint*.

The key idea is to expand repeatedly the search wavefronts until no new answers are produced; i.e., we reach a fixpoint. Each search wavefront is guided by an *automaton* in the plan, a finite state machine based on an NFA.

Consider query plans for $Q_1 = \langle x, director\_of/actor/born\_in, y\rangle$ as in Figure 2. *Plan A* employs a WP corresponding to a single FA that directly encodes a recognizer for the query's regular expression. This plan captures the notion of "pipelining". Whereas Plan B employs a WP that consists of two subplan automata, the first subplan (SP) is used as a view for transition over states in the second plan. This captures the notion of "materialization". Note that in the second subplan automaton, we used a prepend transition ($\cdot director\_of$) over the previous state.

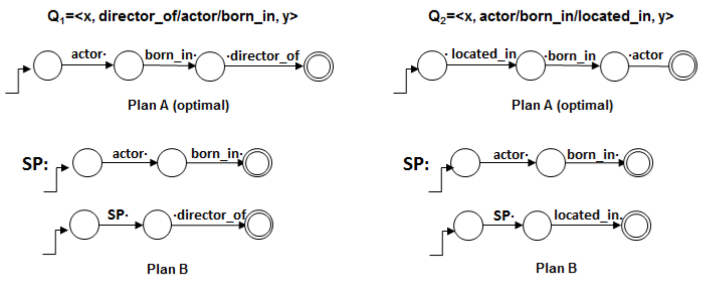

Fig. 2: A batch of two queries

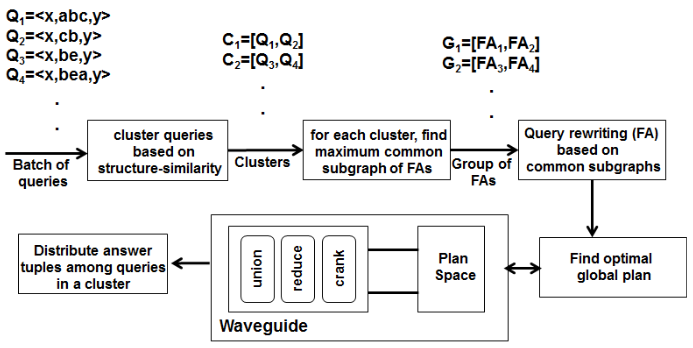
Q₁=<x,abc,y>
Q₂=<x,cb,y>
Q₃=<x,be,y>
Q₄=<x,bea,y>
Batch of queries
cluster queries based on structure-similarity
C₁=[Q₁,Q₂]
C₂=[Q₃,Q₄]
Clusters
for each cluster, find maximum common subgraph of FAs
G₁=[FA₁,FA₂]
G₂=[FA₃,FA₄]
Group of FAs
Query rewriting (FA) based on common subgraphs
Find optimal global plan
union
reduce
crank
Plan Space
Waveguide
Distribute answer tuples among queries in a cluster

Fig. 3: SWARMGUIDE Framework

The WAVEGUIDE prototype implements guided graph search as procedural SQL on top of PostgreSQL. However, any type of back-end physical graph database model (*e.g.*, triple store or adjacency lists) can be used.

3 SWARMGUIDE: The Framework

Queries running in a batch may have common subexpressions among them. Instead of running each query individually, one could benefit from sharing the results of common subexpressions [5]. Consider two queries, $Q_1 = \langle x, director\_of/actor/born\_in, y\rangle$ and $Q_2 = \langle x, actor/born\_in/located\_in, y\rangle$ as in Figure 2. Clearly, the optimal automaton plans (Plan A) do not share any common subplan for Q_1 and Q_2. However, if we chose suboptimal plans (Plan B) then we could have a common sub-plan automaton ($actor/born\_in$). This common subplan could be shared among Q_1 and Q_2 without re-evaluating it.

Evaluation of MRPQs. The evaluation of multiple regular path queries (MRPQs) is two-step: identifying common subexpressions among queries; and searching for a global optimal plan such that its cost is less than the summed cost of the local optimal plans in the batch. We present our framework SWARMGUIDE, a generic framework for optimizing *multiple regular path queries (MRPQs)* in graph databases. Figure 3 shows SWARMGUIDE architecture.

Clustering Queries. Clustering queries is a preprocessing step to finding the commonalities among the RPQs in the batch. These can be detected by finding the isomorphic subgraphs of their corresponding finite automata (FAs). This process is known to be NP-hard, in general [1]. Therefore, we use heuristics to group only queries that can have common sub-automata, and then do the hard task of identifying the common sub-automata within each group.
Finding Common Sub-automata. Finding common sub-automaton resembles finding the maximum common subgraph among graphs. The problem becomes more difficult here as we need to find the largest common subgraphs (sub-automata) for multiple graphs (FAs). Most existing solutions only consider non-labeled edges and nodes in undirected graphs. We adopt the solution from the maximal common-edge subgraph (MCES) problem [3] to detect common sub-automata. This has three steps: transforming labeled-graphs into the equivalent line-graphs; producing a product graph from the line-graphs; and detecting the maximal cliques in the product graph, which corresponds to MCESs (therefore, common sub-automata).
Query Rewriting. A regular expression reg can be recognized equivalently by many FAs. For instance, the two FAs for two regular expressions $reg_1 = (ab)^*ac$ and $reg_2 = a(ba)^*c$ are equivalent; however, the plan space may differ, depending on the chosen FA. In this step, we rewrite FAs according to their shared common sub-automata.
Global Optimization. A local optimal plan for a given query may not be the best plan when optimizing a batch of queries. The goal of the global optimization is then to search local plans of queries to find a global plan by choosing one local plan per query in a cluster of $RPQs$. The cost of this global plan should be less than, or equal to, the total cost of local optimal plans of queries in the batch. In [4], the authors propose cost-based heuristic algorithms for this; we likewise adapt their algorithms for here.

Other Considerations. A single FA plan is often decomposed into several subplans as views. When planning globally, one has to check if views from other queries' plans can be shared. Intermediate node-pairs—along with their states in FA—are cached to avoid unbounded computation over cyclic graphs. When evaluating a global plan in MRPQs, these local caches need to be maintained globally so subplans shared among queries can be leveraged. The global cache also can be used to obtain useful statistics about the graph to obtain a more accurate global optimal plan and choosing the initial nodes for search exploration.

4 Conclusions

Graph database analytic applications are rapidly on the rise. These raise new and arduous challenges. We can apply many of the lessons learned from the relational domain—*e.g.*, pipelining, materialized views, cost-based optimization, query answering by views, and multiple query optimization—to these challenges to great effect. However, these techniques have to be carefully re-imagined to be effective for graph databases, their queries, and applications.

References

1. W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for SPARQL. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 666–677. IEEE, 2012.
2. A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
3. J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16(7):521–533, 2002.
4. P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM, 2000.
5. T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.
6. N. Yakovets, P. Godfrey, and J. Gryz. Query planning for evaluating SPARQL property paths. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–15, San Francisco, CA, USA, June 2016.