TASWEET: OPTIMIZING DISJUNCTIVE REGULAR PATH QUERIES IN GRAPH DATABASES

Zahid Abul-Basher
University of Toronto
Toronto, Canada
zahid@mie.utoronto.ca

Nikolay Yakovets
Eindhoven University of Technology
Eindhoven, The Netherlands
hush@tue.nl

Parke Godfrey
York University
Toronto, Canada
godfrey@cse.yorku.ca

Shadi Ghajar-Khosravi
Defence Research and Development Canada
Toronto, Canada
shadi.ghajar@drdc-rddc.gc.ca

Mark H. Chignell
University of Toronto
Toronto, Canada
chignell@mie.utoronto.ca

ABSTRACT

Regular path queries (RPQs) have quickly become a staple to explore graph databases. SPARQL 1.1 includes *property paths*, and so now encompasses RPQs as a fragment. Despite the extreme utility of RPQs, it can be exceedingly difficult for even experts to formulate such queries. It is next to impossible for non-experts to formulate such path queries. As such, several visual query systems (VQSs) have been proposed which simplify the task of constructing path queries by directly manipulating visual objects representing the domain elements. The queries generated by VQSs may, however, have many commonalities that can be exploited to optimize globally. We introduce TASWEET, a framework for optimizing "disjunctive" path queries, which detects the commonalities among the queries to find a globally optimized execution plan over the plan spaces of the constituent RPQs. Our results show savings in edge-walks / time-to-completion of 59%.

1. INTRODUCTION

Military intelligence consists of collecting, analyzing, and extracting intelligence on situations, events, and entities (*e.g.*, people, places, and organizations) from different types of data (*e.g.*, text, video, picture, and signals) produced by a variety of sources (*e.g.*, human, electronic, open source, and sensors) [2]. The goal of the intelligence is to label suspected entities, relationships, and patterns of events. With the emergence of social networks and cyber warfare, graph database systems, especially with visual query facilities, have become a popular choice for military intelligence, as evidenced by the success of products such as GOTHAM by *Palentir*[1] and LINKURIOUS by *Neo4j*.[2]

[1]https://www.palantir.com/palantir-gotham/platform/
[2]https://neo4j.com/developer/guide-data-visualization/

It has become common methodology to search for pairs of nodes such that each pair is connected by a path—a sequence of labeled edges—matching a path expression expressed via a *path query*. For example, an analyst working on a money laundering case might be searching financial transactions data to find individuals who *own* companies that have transferred money to businesses with investments in off-shore companies, *or* individuals who *work* at said companies. Thus, the analyst is looking for potential links from individuals to off-shore companies matching those path specifications, as illustrated in Figure 1.

In path queries, one can specify the path of interest via a *regular expression*, instead of needing to specify the path explicitly. This is known as a *regular path query* (RPQ). Given the usefulness of path queries, SPARQL 1.1 now supports them via *property paths*. While highly expressive, formulating such queries can be exceedingly difficult, even for experts. As such, visual query systems have been proposed (OptiqueVQS is one such system [7]) to formulate queries which are more user-friendly, intuitive, and less expertise-demanding. Queries generated by VQSs may contain similar sub-queries, and so can be further optimized. We address one such optimization problem: optimizing *disjunctive* regular path queries (dRPQs); that is, a disjunction of a set of regular path queries (RPQs). As the answer set of a dRPQ is the *union* of the answer sets of the individual RPQs, one way to evaluate it is to evaluate the RPQs independently. One can often do better, however, by sharing evaluation of commonalities across the RPQs. Thus, the problem is to optimize multiple regular path queries in graph databases.

In a recent work, Yakovets *et al.* [11] demonstrate that there is a rich plan space for RPQs, and that evaluation cost can differ by orders of magnitude from one plan to another. Their WAVEGUIDE framework is able to find the optimal execution plan with respect to cost-based estimation over this plan space for a given RPQ. Of course for us, when evaluating a dRPQ, choosing the "locally" optimal plans for each of the RPQs may not be globally optimal, given commonalities. One might benefit then by choosing alternative plans that take advantage of the commonalities. We introduce TASWEET, a methodology and prototype for this. TASWEET detects the commonalities over the RPQs (of a dRPQ), then uses WAVEGUIDE with knowledge of the commonalities to find an improved global execution plan (by estimated cost) over the plan spaces of the constituent RPQs.

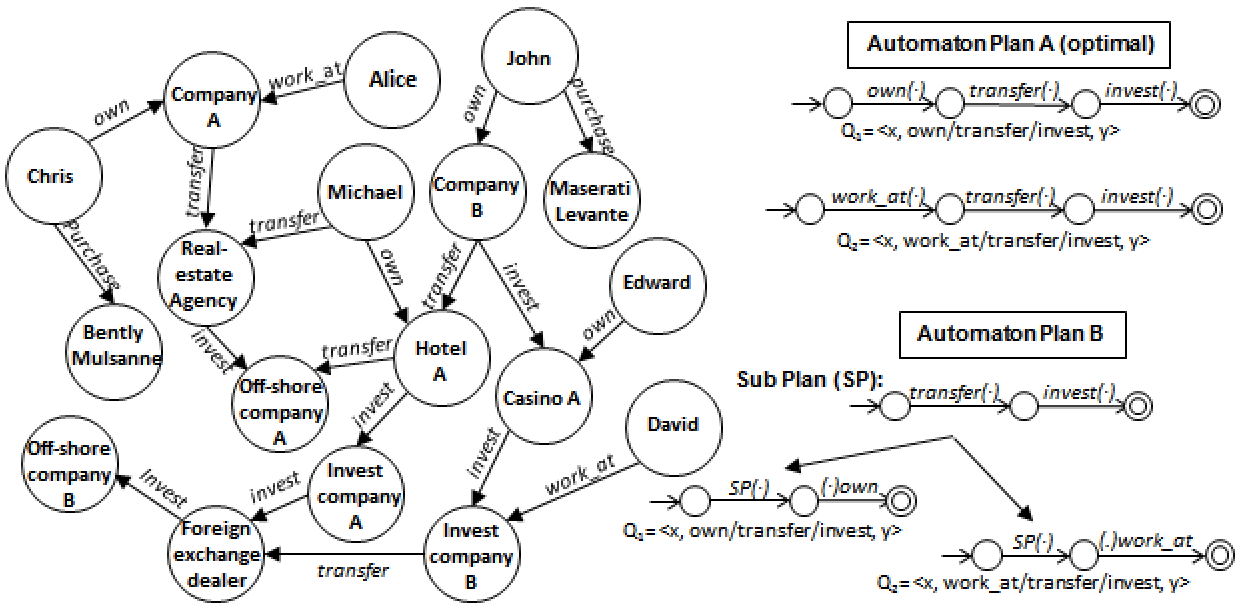

Figure 1: A graph example in military intelligence with two queries Q_1 and Q_2 and their two automaton plans.

2. METHODOLOGY

A regular path query (RPQ) [4] over a graph G is a triple $\langle x, reg, y\rangle$ where x and y are free variables over the nodes N of G, and *reg* is a regular expression. An answer of an RPQ is a node-pair $\langle s, t\rangle$ (with $s, t \in N$) such that there is a sequence of edge labels $a_0a_1 \cdots a_k$ in G, called a path p, between s and t, and the path p matches the given *reg*. Thus, the *answer set* of an RPQ contains all its answers. The RDF data model and SPARQL query language represent these concepts. With the introduction of *property paths* in SPARQL 1.1, the query language contains RPQs. Herein, we use SPARQL syntax in explanations.

2.1 Evaluation of RPQs

An initial approach to evaluating RPQs was introduced in G+ [4]. There, a *product* finite automaton (FA) is constructed that can be used to navigate and match paths to the regular expression simultaneously. (This approach is nicknamed the FA approach in [11].) More recently, there has been renewed interest in how to evaluate RPQs efficiently in SPARQL over RDF stores [1]. Much focus has been on expanding the relational algebra to leverage relational query optimization and evaluation. It suffices to add an additional operator for transitive closure (called "α"). (This approach is nicknamed as α-RA in [11].) Virtuoso is a relational system that has been extended in this way to support SPARQL and RDF.

The WAVEGUIDE approach generalizes over both the FA and α-RA approaches to provide a very rich plan space for RPQs [11]. While the plan space is exponential (with respect to the length of the regular expression), WAVEGUIDE offers a cost-based optimization via polynomial-time enumeration. In WAVEGUIDE, path search is conducted *simultaneously* while recognizing the path expressions. Its input is a graph database G and a *waveplan* (WP) P_Q which *guides* a number of search *wavefronts* that explore the graph. A *wavefront* is a part of the plan that evaluates breadth-first during the evaluation. Here, the graph exploration is driven by an iterative search strategy which is inspired by the semi-naïve bottom-up procedure used in the evaluation of linear recursive expressions that is based on a *fixpoint*. The key concept is to expand the search wavefronts continuously until there are no new answers; i.e., we reach a fixpoint. Any search wavefront is guided by an *automaton* in the plan, based on a finite state machine FA.

Consider query plans for Q_1, $\langle x, own/transfer/invest, y\rangle$, as in Figure 1. *Plan A* captures the notion of "pipelining". It employs a WP corresponding to a single FA that directly encodes a recognizer for the query's regular expression. Whereas Plan B captures the notion of "materialization". It 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. Note that in the second subplan automaton, we used a prepend transition $((\cdot)own)$ over the previous state. The cost for Plan A is 9 edge-walks whereas for Plan B it is 11 edge-walks. In addition to these two plans, there is also a reverse "pipelining" plan (not shown in Figure 1) which is evaluated "backwards", retrieving all pairs connected by edge label *invest* (across such labeled edges in reverse), then prepending with those connected by *transfer*, and finally by *own*, with the plan cost of 12.

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

2.2 Evaluation of Disjunctive RPQs

Instead of evaluating each RPQ individually, one may benefit by sharing the results of common sub-expressions. Consider the two queries as in Figure 1: Q_1, $\langle x, own/transfer/invest, y\rangle$; and Q_2, $\langle x, work\_at/transfer/invest, y\rangle$. The (locally) optimal automaton plans ("Plan A") do not share any common subplan for Q_1 and Q_2. If we chose sub-optimal automaton plans ("Plan B"), however, then we can share a common sub-plan automaton ($transfer/invest$) between the evaluations of Q_1 and Q_2.

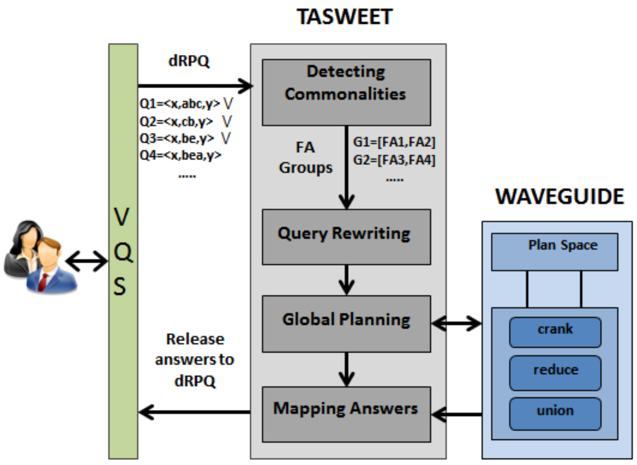

Figure 2: TASWEET Framework

3. THE FRAMEWORK

Our TASWEET framework, shown in Figure 2, takes a dRPQ $\mathcal{D} = Q_1 \vee ... \vee Q_n$ that consists of n RPQs as input over a graph G. Its evaluation is a two-step process: identifying common sub-expressions among queries; and then searching for a global optimal plan such that its cost is less than the total cost of the locally optimal plans of the corresponding RPQs. Figure 2 shows the overall TASWEET architecture.

3.1 Detecting Commonalities

The problem of finding common sub-automata resembles finding the maximum common subgraphs in graphs. The problem is more difficult here, however, as we need to find the largest common subgraphs (sub-automata) for multiple graphs (FAs). We begin our discussion by converting the FA equivalence problem into a graph isomorphism problem. We then extend the concept to find the equivalent sub-automata in FAs using sub-graph isomorphism techniques.

3.1.1 *Finite Automata Isomorphism*

Deterministic Finite Automata (DFA). A deterministic finite automaton M is a *5-tuple* language acceptor machine, $(S, \sum, \delta, q_0, F)$ where it consists of a set of states (S), a set of input symbols called the alphabet ($\sum$) and a transition function ($\delta : S \times \sum \rightarrow S$). There is one state that is marked as an initial state ($q_0 \in S$) and also there is a set of states marked as accept states ($F \subseteq S$). Two DFAs are said to be equivalent if they accept the same language.

Minimum Automata. A DFA M is minimal if there is no other DFA N that is equivalent to M with fewer states than M. A DFA M is minimal if (i) all its states can be reached from the initial state q_0 and (ii) no two equivalent states exist. (Two states q_1 and q_2 are equivalent if for all $x \in \sum^*$, $\delta(q_1, x) \in F$ *iff* $\delta(q_2, x) \in F$.) All minimal DFAs for a language L are isomorphic; *i.e.*, they have identical transitions with the same number of states (by the Myhill-Nerode Theorem [5]). By corollary, when joining two minimum DFAs, the structure of the new DFA remains the same. Therefore, we can use the techniques of detecting the maximum common edge subgraph (MCES) to identify the common sub-automata among minimum DFAs; hence, the commonalities among RPQs.

3.1.2 *Maximum Common Subautomaton*

Most solutions of MCES problem only consider non-labeled edges and nodes in undirected graphs. We adopt the solution from [6] 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 correspond to MCESs (therefore, common sub-automata).

Constructing Linegraphs. The linegraph $L(G)$ of a graph G is a directed graph where each edge (with label) in original G becomes a node (with the same edge label) in $L(G)$. Two nodes in $L(G)$ are connected with an edge if their corresponding edges in original G share a common node. In the original G, this common node can be an incoming or an outgoing node for two connected edges. That is, it can have four different possibilities depending on the directions of connected edges: source-destination, destination-source, source-source, and destination-destination. Therefore, during our linegraph construction, we store the directions of edges of the common node as edge labels in $L(G)$. Before the construction of linegraphs, we removed all the self-loops in the automaton by creating additional transitions with an additional state. This process is necessary for the next step, clique detection. (Most algorithms for detecting cliques only work with non self-loop graphs.)

Constructing Product Graph. The product graph $L(G_p)$ of two linegraphs, $L(G_1)$ and $L(G_2)$, is constructed as follows [8]. The nodes N_p in $L(G_p)$ are node-pairs defined in the Cartesian product of N_1 of $L(G_1)$ and N_2 of $L(G_2)$. In constructing N_p, we only consider node-pairs that have the same node label of the corresponding linegraphs $L(G_1)$ and $L(G_2)$. The two node-pairs in N_p have an edge in $L(G_p)$ if either *(i)* the same edges exist between the corresponding nodes in the original linegraphs (called a strong connection), or *(ii)* no edges exist between the corresponding nodes (called a weak connection). To reduce the size of the product graph, we remove all nodes with non-common labels among the linegraphs and then build the product graph recursively.

Maximal Cliques in Product Graph. We used the Bron-Kerbosch Algorithm [3] to find all the maximal cliques in the product graph. A maximal clique with a tree that covers strong connections in the product graph corresponds to a maximal common sub-automaton within a group of FAs.

Query Rewriting. A regular expression *reg* can be recognized by several FAs. For example, although the two FAs for the two regular expressions $reg_1 = (xy)^*xz$ and $reg_2 = x(yx)^*z$ are same but the plan space may differ, depending on the chosen FA. In this step, we rewrite FAs based on their shared common sub-automata.

3.2 Global Optimization

In WAVEGUIDE, finding the optimal plan for an RPQ is done by dynamic programming, analogous to join enumeration in SYSTEM R. It works bottom-up to construct a tree which represents the plan. At each level, an optimal plan is selected according to a cost objective; for here, the estimated number of edge-walks. This is calculated based on statistics of the graph (*e.g.*, *n-gram* distributions of the edge labels). In TASWEET, after detecting the common sub-automata, the locally optimal plans for each of them are found by WAVEGUIDE. These plans for the common sub-automata then serve as "black-box" views during a second pass with WAVEGUIDE to plan each RPQ. The cost of each black-box shared plan is treated as its actual estimated cost divided by the number of queries which share that common sub-automaton. While this is a simplistic, greedy approach, it suffices well for the proof of concept for our methodology. (Immediate future work is to improve this integration of the TASWEET frontend and the WAVEGUIDE backend to provide guarantees on the global optimality.)

4. RESULTS & DISCUSSIONS

We provide a preliminary benchmark of our TASWEET framework by considering the optimization of a dRPQ $\mathcal{D}$ which models a typical military intelligence workload: $\mathcal{D}$ disjoins seven realistic, related RPQs—shown in Table 1—over the encyclopedic dataset YAGO2s [9].

Q	
1	?p isMarriedTo/livesIn/isLocatedIn+/dealsWith+ Argentina
2	?p hasChild/livesIn/isLocatedIn+/dealsWith+ Japan
3	?p influences/livesIn/isLocatedIn+/dealsWith+ Sweden
4	?p livesIn/isLocatedIn+/dealsWith+ United_States
5	?p hasSuccessor/livesIn/isLocatedIn+/dealsWith+ India
6	?p hasPredecessor/livesIn/isLocatedIn+/dealsWith+ Germany
7	?p hasAcademicAdvisor/livesIn/isLocatedIn+/dealsWith+ Netherlands

Table 1: Queries used in a military-intelligence dRPQ $\mathcal{D}$.

Query $\mathcal{D}$ finds people-country pairs such that the people related to locations or organizations which have connections with given countries. TASWEET correctly identified a shared sub-plan across all the RPQs in $\mathcal{D}$, "livesIn/isLocatedIn+/-dealsWith+". Table 2 shows the difference in the number of edge walks between executing plans in isolation by WAVEGUIDE, and as a dRPQ by TASWEET. (Note that fewer edge walks performed result in proportionally faster execution.)

Q	Tasweet	WaveGuide	Δ
1	260	39827	+39373
2	2288	24525	+22178
3	226	33397	+33046
4	4563	269495	+264932
5	2258	40924	+38379
6	4572	42261	+37610
7	4608	7185	+3827
Shared	269895	-	-269895
Total	288670	457614	+169450 (+59%)

Table 2: Deltas in number of edge walks.

None of the (locally) optimal plans for the RPQs of $\mathcal{D}$ evaluate the shared expression (as a sub-plan). Thus, its evaluation is extra work. (This is Shared in Table 2.) But as its materialization can be used by each of the RPQs, it significantly speeds up evaluation of $\mathcal{D}$ overall. Execution of the locally optimal plans cost 59% more edge walks over the execution over the globally optimal plans that materialize and share their common sub-expression. This is excellent proof of concept for our methodology.

For future work, we have clear objectives. First is to develop more robust, efficient means for global optimization. Our second objective is to extend TASWEET to cluster RPQs that arrive together in an application to maximize the commonalities per group. This project is a small step in a grander project to optimize well graph queries. Disjunctive RPQ are a logical step between single RPQs and *conjunctive* RPQs (cRPQs). Thus, our third objective is to extend the TASWEET / WAVEGUIDE framework for optimizing cRPQs.

5. CONCLUSIONS

We have presented a proof of concept to show that by exploiting commonalities across RPQs which are meant to be evaluated "in batch" with their answer sets to be unioned—thus, a disjunction of RPQs (a dRPQ)—that their evaluation can be globally optimized significantly beyond locally optimized evaluation of each independently. To do this is challenging. First, the commonalities must be efficiently found. We have developed how to do this, which we overviewed above. Second is how to *push down* reasoning about them into the cost-based optimizer (*e.g.*, WAVEGUIDE). Our initial experimentation demonstrated a significant improvement of 59%.

6. REFERENCES

[1] W3c: Resource description framework (rdf). http://www.w3.org/TR/rdf-concepts/,2004.

[2] S. G. Hutchins, P. Pirolli, and S. Card. Use of critical analysis method to conduct a cognitive task analysis of intelligence analysts. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 47, pages 478–482. SAGE Publications, 2003.

[3] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1):1–30, 2001.

[4] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[5] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[6] 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.

[7] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. Optiquevqs: towards an ontology-based visual query system for big data. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, pages 119–126. ACM, 2013.

[8] P. Vismara and B. Valery. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 358–368. Springer, 2008.

[9] YAGO2s: A high-quality knowledge base. http://yago-knowledge.org/resource/. Max Planck Institut Informatik.

[10] N. Yakovets, P. Godfrey, and J. Gryz. WAVEGUIDE: evaluating SPARQL property path queries. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 525–528, 2015.

[11] 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.