Pattern Matching on Massive Metadata Graphs at Scale

PhD Student - Tahsin Reza, Advisor - Professor Matei Ripeanu
Department of Electrical and Computer Engineering, University of British Columbia, Vancouver
{treza,matei}@ece.ubc.ca

ABSTRACT

Pattern matching is a powerful graph analysis tool. Unfortunately, existing solutions have limited scalability, support only a limited set of patterns, and/or focus on only a subset of the real-world problems associated with pattern matching. First, we present a new algorithmic pipeline based on graph pruning that: *(i)* enables highly scalable *exact* pattern matching on labeled graphs, *(ii)* supports arbitrary patterns, *(iii)* enables trade-offs between precision and time-to-solution, and *(iv)* supports a set of popular analytics scenarios. We implement our approach on top of HavoqGT and demonstrate its advantages through strong and weak scaling experiments on massive-scale real-world (up to 257B edges) and synthetic (up to 4.4T edges) graphs, respectively, and at scales (1,024 nodes / 36,864 cores) orders of magnitude larger than used in the past for similar problems. Furthermore, we explore avenues to enable approximate matching within the graph pruning model, targeting contemporary and emerging high-impact, real-world applications.

1 PROBLEM OVERVIEW AND PROPOSED SOLUTION APPROACH

Pattern matching in graphs, that is, finding subgraphs that *match* a small *template graph* within a large *background graph*, is fundamental to graph analysis and has wide applications in multiple areas. A *match* can be broadly categorized as either *exact* - i.e., there is a bijective map between the vertices/edges in the template and those in the matching subgraph, or *approximate* - the template and the match are just similar by some defined similarity metric. If the template size is not limited, exact matching is not known to have a polynomial time solution in the general case. Berry et al. [1] introduced the problem of *type-isomorphism*: metadata graphs where vertices and edges are labeled and, in addition to topological constraints, a *match* identifies nodes and edges with the same labels in the template and the background graph. While the labeled version does not reduce the worst-case complexity of the original problem, past experience has demonstrated that label-based matching can be a powerful tool with potential for practical, real-world applications such as social network analysis.

The Challenge. Applications that mine graphs consist of tens of billions of edges are common [2, 3]. However, existing solutions have limited capabilities: most importantly, they do not scale to massive graphs and/or support only a restricted set of search templates. Additionally, the algorithms at the core of the existing techniques are not suitable for today's infrastructures relying on horizontal scalability and share-nothing clusters as most them are inherently sequential and difficult to parallelize. Finally, pattern matching is susceptible to combinatorial explosion of the intermediate or final state: for low selectivity queries, the number of subgraphs partially (or entirely) matching the template can grow exponentially with the number of nodes and edges in the already large background graph, posing serious memory and communication challenges.

A New Approach for Scalable Pattern Matching. We propose a new algorithmic pipeline based on *graph pruning* (Fig. 1 illustrates this idea at a high level). The idea of exploring graph pruning to support pattern matching stems from three key observations: First, the traditionally used *tree-search* techniques generally attempt to enumerate all matches through explicit search. When a search path fails, such an unprofitable path is marked invalid and ignored in the subsequent steps. Similar to applications that use graph pruning or, more generally, input reduction, we observe that it is much cheaper to first focus on eliminating the vertices and edges that do not meet the label and topological requirements of the template. Our experience shows that relatively simple pruning heuristics based on label and vertex neighborhood constraints can significantly prune away much of the background graph.

Second, we observe that a *vertex-centric* formulation for such pruning algorithms exists, and this makes it possible to harness existing high-performance, vertex-centric frameworks (e.g., Giraph, GraphLab, HavoqGT). In our vertex-centric formulation for pruning, a vertex must satisfy two types of constraints: *local* and *non-local*, to possibly be part of a match. *Local constraints* involve only the vertex and its neighborhood: a vertex in an exact match needs to *(i)* match the label of a corresponding vertex in the template, and *(ii)* have edges to vertices labeled as prescribed in the adjacency structure of this corresponding vertex in the template. *Non-local constraints* are topological requirements beyond the immediate neighborhood of a vertex (e.g., that the vertex must be part of a cycle).

Third, we observe that, full match enumeration is not the most efficient avenue to support many high-level graph analysis scenarios. Depending on the final goal of the user, pattern matching problems fall into a number of categories which include: *(a)* determining if a match exists (or not) in the background graph (yes/no answer), *(b)* selecting all the vertices and edges that participate in matches, *(c)* ranking these vertices or edges based on their *centrality with respect to the search template*, i.e., the frequency of their participation in matches, *(d)* counting/estimating the total number of matches (comparable to the well-known triangle counting problem), or *(e)* enumerating all distinct matches in the background graph. The traditional approach is to perform *(e)* and to use the result of the enumeration to answer *(a)* – *(d)*. However, this approach is limited to small background graphs or is dependent on a low number of near and exact matches within the background graph (due to exponential growth of the search state-space). We argue that a pruning-based pipeline is not only a practical solution to *(a)* – *(d)* (and to other pattern-matching-related analytics), when full match enumeration is not the main interest, but also an efficient path towards full match enumeration. We demonstrate that a solution starting from the techniques we develop for pruning, efficiently supports match enumeration for two reasons: First, the pruned graph can be multiple orders of magnitude smaller than the background graph, and existing high-complexity enumeration routines (which otherwise would be intractable following the conventional approach) are now applicable. Second, our pruning techniques collect additional key information to accelerate match enumeration:

for each vertex in the pruned graph, our algorithms build a list of its potential matches in the template.

2 RESEARCH CONTRIBUTIONS

2.1 Exact Matching - Work Accomplished

In our preliminary study [4], we focus on evaluating the feasibility and effectiveness of the proposed approach over two directions. On the one side, we are interested in understanding how far graph pruning can go. To this end, we focus on an *exact* matching scenario, design algorithms for aggressive vertex pruning, and prove that for some template subclasses (*acyclic* or *edge-monocyclic* and with unique labels), the resulting pruned list is a complete enumeration of all the vertices that participate in a match (proof available in [4]). For more general templates, the pruned list is a superset.

Our recent work [5] (appeared at this year's Supercomputing conference, SC'18) capitalizes on our preliminary study [4] that highlighted the effectiveness of pruning (yet in the context of a restricted set of templates) and presents a pattern matching solution that is: *generic* - no restrictions on the set of patterns supported, *precise* - no false positives, *offers 100% recall* - retrieving all matches, *efficient* - low generated network traffic, and *scalable* - able to process graphs with up to trillions of edges on tens of thousands of cores (as demonstrated). In particular, we make the following contributions:

(i) Novel Asynchronous Algorithms. We have developed asynchronous vertex-centric algorithms that are able to prune the background graph to a precise, enumeration of all vertices and edges that participate in a match for *arbitrary* templates. Importantly, the algorithms map well on a distributed asynchronous graph processing platform, thus enabling scalability and high-performance. For a given template, we propose heuristics to generate the constraints that are later used for pruning.

(ii) Optimized Distributed Implementation. We offer an efficient implementation of these algorithms on the top of HavoqGT, an open-source asynchronous graph processing framework. The prototype includes two key optimizations that dramatically reduce the generated traffic: aggressive edge elimination, and what we call *work aggregation*, a technique that skips duplicate checks in non-local constraint checking, thus preventing possible combinatorial explosion. Additionally, our pruning implementation collects potential matching information: not only it prunes away all vertices and edges that do not participate in any match, but, for each of the vertices that remain, it collects their mappings to the search template. We use this information to accelerate match enumeration.

(iii) Proof of Feasibility. We demonstrate the applicability of this solution by experimenting on real-world and synthetic datasets orders of magnitude larger than prior work. We evaluate scalability through two experiments: first, a strong scaling experiment using real-world datasets, including the largest openly available webgraph whose undirected version has over 257 billion edges; second, a weak scaling experiment using synthetic, R-MAT graphs of up to 4.4 trillion edges, on up to 1,024 compute nodes (36,864 cores). We demonstrate support for search patterns representative of practical queries in both relatively low selectivity and *needle in the haystack* scenarios, and, to stress our system, consider patterns containing only the highest-frequency vertex labels (each with 100K to 10B instances). We show that our technique prunes the graph by orders of magnitude, which, combined with using the intermediary state generated by pruning, makes match enumeration feasible on graphs with trillions of edges.

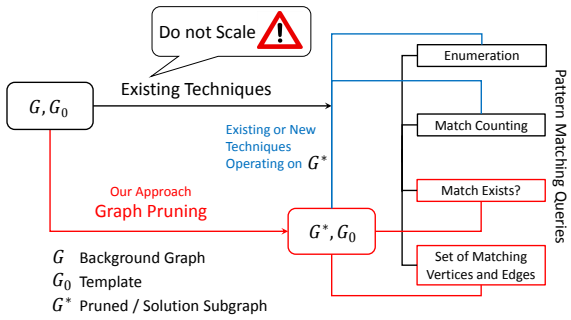

Figure 1: A high-level illustration of our *graph pruning* approach, showing the key steps towards answering various pattern matching queries. For some queries, pruning itself leads to the answer (depicted using red color outline).

(iv) Exploring Trade-offs, and the Impact of Strategic Design Choices and Optimizations. Our approach has the added flexibility that search can be stopped early, leading to the ability to trade faster time to an approximate solution (or even precise solution, yet without 100% precision guarantees) for precision (rate of false positives in the pruned graph) and precision guarantees. We also explore the impact of each optimization used: the cumulative impact of these optimizations is a multiple orders of magnitude of runtime reduction, bringing pattern matching on massive metadata graphs in the realm of possible graph analytics.

Please refer to [4, 5] for literature review, empirical comparison with a recent work QFrag and demonstration of practical use cases (using Reddit and IMDB datasets) of our technique to support rich pattern mining.

2.2 Approximate Matching - Work in Progress

Currently, we are working on enabling *approximate* matching within the *graph pruning* model. The importance and appeal of approximate matching is attributed to its practical relevance: the acquired data could be noisy, resulting in the graph being somewhat diïňĂerent from the ideal model. Similarly, a user may only infer the exact search pattern a priori. In such scenarios, the ability to identifying 'close' or approximate matches is the most practical.

To date, we have developed an initial design for approximate matching following the *graph edit distance* approach that could be seamlessly implemented on the top of the existing distributed graph pruning infrastructure.

REFERENCES

[1] J. W. Berry, B. Hendrickson, S. Kahan, and P. Konecny. 2007. Software and Algorithms for Graph Queries on Multithreaded Architectures. In *2007 IEEE International Parallel and Distributed Processing Symposium*. 1–14.

[2] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One Trillion Edges: Graph Processing at Facebook-scale. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1804–1815.

[3] Seth A. Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. 2014. Information Network or Social Network?: The Structure of the Twitter Follow Graph. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion)*. 493–498.

[4] T. Reza, C. Klymko, M. Ripeanu, G. Sanders, and R. Pearce. 2017. Towards Practical and Robust Labeled Pattern Matching in Trillion-Edge Graphs. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–12.

[5] T. Reza, M. Ripeanu, N. Tripoul, G. Sanders, and R. Pearce. 2018. PruneJuice: Pruning Trillion-edge Graphs to a Precise Pattern-Matching Solution. In *Proceedings of The IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. ACM, New York, NY, USA.