Towards the Scheduling of Vertex-constrained Multi Subgraph Matching Query

Kongzhang Hao, Longbin Lai

The University of New South Wales, Alibaba Group

Abstract

This work studies multi-query optimization for subgraph isomorphism search. Given a data graph G and a set of query graphs $Q = \{q_1, ..., q_n\}$, our aim is to **efficiently** find all subgraphs of G that are isomorphic to one of the query graphs.
We show that the solution in previous work is sub-optimal. We firstly propose a novel method for efficient extraction of useful common subgraphs in polynomial time and a data structure to organise them. To balance memory usage, the algorithm guarantees provable bounds on the cache size and worst-case optimality on intermediate results. We provide strategies to revise the current state-of-the-art join based subgraph matching algorithms to seamlessly utilise the cached results with a series of binary joins and worst-case optimal joins, which along with the common subgraph extraction strategy guarantees that its run time is worst-case optimal. We further propose a parallel scheduling paradigm with maximised computation sharing and bound on makespan. We experimentally verify the efficiency and effectiveness of our solution.

Motivation

Banks and merchants lose billions of dollars every year due to credit card fraud. An effective solution to fraud detection that has been well-studied is subgraph matching, where data graph models credit card transactions and subgraph queries represent fraud patterns. However, recent years have seen a considerable increase in the number of patterns involved in credit-card frauds. This gives rise to a natural problem: how to efficiently solve many subgraph queries in a large-scale transaction network? Obviously, sequentially computing the queries in a row is too costly, as the problem of subgraph isomorphism is already NP-complete and a credit card transaction network is usually in massive-scale.
This highlights the need for multi subgraph query optimisation. Given a data graph G and a set of query graphs $Q = \{q_1, ..., q_n\}$, our aim is to efficiently find all subgraphs of G that are isomorphic to one of the query graphs. Specifically, we want to effectively make use of computation sharing between the queries, so that when there are significant overlaps, the overall processing time is considerably shorter than sequential execution.

Challenges

However, multi subgraph query optimisation, is quite hard. To effectively share the computation between queries, we need to identify query overlaps that are worthwhile to extract, which provides more benefits than overhead. The closest work in VLDB2016 follows a heuristic that picks larger maximum common subgraphs between pairs of queries. However, we prove that the algorithm finds common sub-patterns that are large in size, but computational sub-optimal and highly overlapped as shown in the below two examples. Our experiment additionally verifies this. Secondly, the intermediate results of common sub-patterns being cached are memory-intensive and can easily cause memory thrashing when the data graph is large. Moreover, in order to compute final queries from results of sub-patterns, how can we leverage the state-of-the-art subgraph matching algorithms to maximise the performance. Lastly, when there are more threads provided, how do we schedule the queries in parallel execution to effectively share the computation and minimise query time.

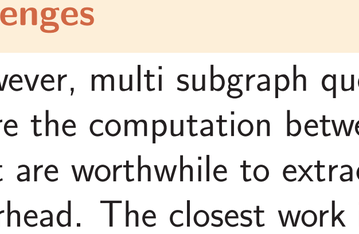


Figure 1: Sub-optimal Example

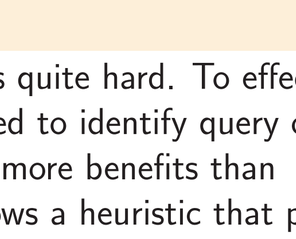


Figure 2: Overlap Example

Contribution

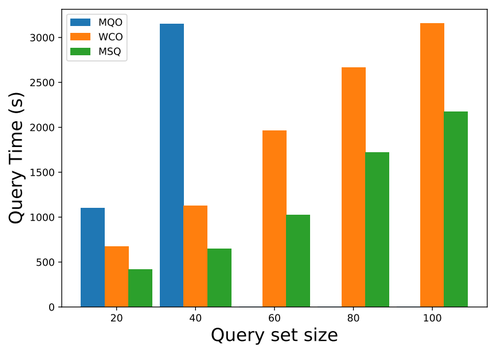
(1) We introduce the concept of AGM-bounded Approximate MCS, which can be efficiently computed in polynomial time and meanwhile filters out graphs that do not share useful common subgraphs.
(2)We propose an optimisation paradigm which not only makes effective use of cost-sharing but also provides worst-case optimal guarantee on intermediate results and theoretical bound on cached storage.
(3) We propose a join optimiser which not only integrates the results of sub-queries, but is also adaptive to the properties of data graph.
(4) For parallel execution, we propose a novel scheduling algorithm which takes into account both computation sharing and load balancing, with bound on its makespan.

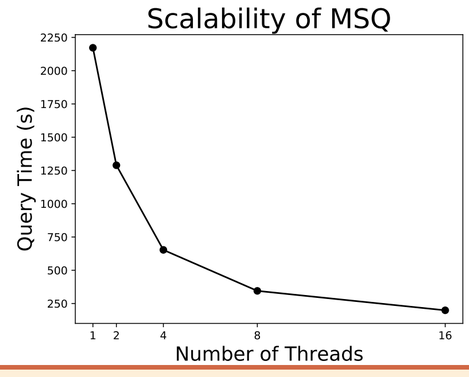
Methodology

1. **AGM-Bounded AMCS**
   - Original MCS problem is NP-Complete. We adopt its basic idea of vertex-at-a-time extension, but remove backtracking and define a heuristic on vertex to be extended. The heuristic prefers to pick vertex that has most connections with prefix and the matched node pair from two input graphs has highest neighborhood similarity. Once a node is chosen to be extended, we check if $AGM(extended\_subgraph) <= Min(AGM(Q_1), AGM(Q_2))$ Algorithm terminates if the above check is not satisfied or no more nodes can be extended.
   - AGM-Bounded AMCS can be computed in *polynomial* time.
2. **Distance-based Clustering**
   - Queries are clustered using a hierarchical clustering strategy, which iteratively selects two units (i.e. either a query or a cluster of queries) with the smallest distance and merges them together. The algorithm terminates when the distance between any two units is larger than clustering factor, or there is only one cluster left.
   - Effectively balance the memory usage (similar to batching).
   - Queries in the same group more likely share helpful overlaps.
   - Time complexity is $\mathbf{0(N^3)}$
3. **Sub-pattern Extraction**
   - **Min-width GHD** generalised hypertree decomposition of a graph whose maximum AGM bound is minimised. A graph can have many min-width GHDs.
   - Iteratively pick a pair of queries with smallest distance, and find the pair of their min-witdth GHDs which have the most components in common. Algorithm terminates when all queries are covered. Components in GHD of a query directly form its sub-patterns.
   - Algorithm guarantees that the results size of any extracted sub-pattern is *worst-case optimal*.
   - Algorithm avoids redundant overlap between sub-patterns.
4. **Memory Control**
   - We only cache intermediate results for sub-patterns that are shared by two or more queries.
   - Trie structure is used to cache intermediate results.
   - The size of intermediates results is bounded by $\mathbf{max}_{c \in Clusters}(|c| - 1)\, \mathbf{max}_{q \in c}\, \mathbf{min}_{d \in GHDs(q)}\, AGM(d)$
5. **Join Optimiser**
   - Some of the sub-patterns are pre-computed and available in the cache.
   - Query is computed from partial results through a plan consisting of a sequence of binary joins and worst-case optimal joins, which is generated based on Intersection cost and HashJoin cost estimation.
   - *Worst-case optimal* is guaranteed at all time.
6. **Parallel Scheduling**
   - When a thread finishes its current job, schedule a READY (i.e. all parents scheduled in any of the threads) task with maximum **comparative advantage** of sharing if placed in this thread (i.e. the query's vertex cover by pre-computed sub-patterns in this thread minus its max vertex cover in any other thread).
   - Makespan is bounded by $\mathbf{(2 - 1/t)}L_{opt}$

Experiment

Dataset LDBC Benchmarking (3,181,724 Nodes, 17,256,033 Edges)
Environment A server with 2 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz (each has 20 cores 40 threads), 500GB memory and 2 TB disk
Analysis MQO, the algorithm proposed in VLDB2016, shows the worst performance and runs out of memory. This is because its sub-optimal strategy can generate too many intermediate results. Our algorithm MSQ shows an obvious improvement over sequential Worst-case Optimal join, which is up to 1.9 times faster. Our algorithm also shows good scalability.





k.hao@unsw.edu.au