Adaptive Query Processing with the Crowd: Joins and the Joinable Filter

Han Maw Aung, Cienn Givens, Amber Kampen, Rebecca Qin, Beth Trushkowsky
Harvey Mudd College
Computer Science Department
beth@cs.hmc.edu

Introduction

Integrating crowdsourcing into database systems enables us to process database queries that rely on information not contained in the database at the time the query is issued. To process crowd-powered queries, we ask workers to perform tasks such as labeling, ranking, and verifying; we then use their responses to evaluate the query.

Our current work deals with filter queries, which return items that satisfy a set of given constraints, or filters. A crowd-powered filter query has the crowd check if items satisfy the constraints. We build off of work done in Dynamic Filter, a crowd-powered query processing algorithm that adaptively orders filters on a list of items to reduce total work, measured in time taken by the crowd (Lan et al. 2017).

We pursue further efficiency in crowd-powered filter queries by investigating cases where *joins* may be used to replace filters. A join is a database operation which takes in two sets of items, pairs items from different sets, then filters the pairs based on which pairs fulfill a criterion called the *join condition*. The output is the set of remaining pairs.

In our work, joins are relevant when considering filters with a particular form which we call *joinable filters*. As an example of this form, consider filtering a set of hotels based on whether they are within two miles of a café with outdoor seating. The primary items (those being filtered) are hotels. Whether or not they are within two miles of a café is the join condition, because it joins the primary list with another set of items (the secondary items). A café with outdoor seating passes the constraint on the secondary items, called the *secondary filter*. A primary item passes a joinable filter if, by the join condition, it matches a secondary item that passes the secondary filter.

We can replace a joinable filter, a single yes/no task for a given primary item, with an equivalent sequence of operations that make up a join and secondary filter, which we call a join path. In breaking down the joinable filter, we can increase the efficiency of the query by reducing redundant work. For example, if multiple primary items match a secondary item that passes the secondary filter, the joinable filter would have workers find and evaluate the same secondary item multiple times through different primary items. Our approach would only evaluate the secondary filter once and save that information to leverage for the remainder of the query. Our initial findings show that join paths can complete certain queries faster than joinable filters.

Related Work

Our work addresses crowd-powered joins, where the join condition is evaluated by the crowd. Part of our approach to efficient crowd-powered joins involves the idea of a pre-join filter (Marcus et al. 2011). A pre-join filter asks the crowd to categorize each item from the two input lists before the join condition is tested on pairs; two items in the same category have a much higher likelihood of matching by the join condition (in our cases, to the extent that two items from different categories cannot match) (Mitsuishi et al. 2013). For example: if the lists were hotels and cafés, each could be categorized with the city they are located in.

There is existing work on crowd-powered joins e.g. (Li et al. 2017) (Mitsuishi et al. 2013), but to the best of our knowledge, ours is the first work that optimizes crowd-powered joins specifically for the purpose of improving filter queries. The inclusion of the secondary filter as well as the ability to finish the query before the join is complete (when the joinable filter is done) distinguish our work.

Join Paths

Join paths are sequences of operations that can replace joinable filters. Our work explores two classes of join paths.

Item-wise Joins The item-wise join has two variants: one on primary items and one on secondary items. Item-wise joins on primary items ask the crowd to produce every item in the secondary set that matches a primary item by the join condition and then to evaluate the secondary filter for the items found. Since secondary filters are evaluated only once for each secondary item, item-wise joins can avoid doing redundant work that would be done by the joinable filter.

Item-wise joins on secondary items differ in that the crowd is asked to pair primary items with a given secondary item. By performing the join this way, the item-wise join can evaluate secondary filters at the beginning of the query path

Presented in the Work in Progress and Demo track, HCOMP 2019.

to avoid finding matches for secondary items that do not pass the secondary filter. This join path is possible when we start the query with the secondary list.

However, normally we do not have the secondary list at the start of the query. We can use the crowd to compile the secondary list by finding matches with primary items; we use statistical techniques to predict when we have found all secondary items (Trushkowsky et al. 2013). Compiling the secondary list also opens up other join paths which use operations that require it.

Pre-join Filter Join One such path is the pre-join filter join, which labels items before the join. Items with different labels cannot satisfy the join condition, which significantly reduces the number of pairs that need to be evaluated. This path first asks workers to label primary and secondary items in order to categorize them. Workers are then given pairs of items with the same label and asked to evaluate whether they match by the join condition. Finally, they are asked to evaluate the secondary filter until the join path is done.

Initial Findings

We attempt to find if and when join paths are faster than the joinable filter. To collect results, we run simulations of join paths and compare total worker time to the time it takes to evaluate the joinable filter.

Experimental Setup

In our experiments, we run each join path using simulated worker responses. We control parameter settings, including *secondary filter selectivity* and *join condition selectivity*. These selectivities refer to the proportion of secondary items that pass the secondary filter and pairs that pass the join condition, respectively. With these settings, our simulation program generates ground truth for every operation. Simulated crowd workers are assigned tasks based on the join path being simulated and give responses based on the ground truth, occasionally giving false answers, as we expect real crowd workers would. We use the same consensus approach as in (Trushkowsky et al. 2013).

In a single simulation, we record the number of tasks issued for each operation, which we use along with the time for each operation to calculate a join path's *cost*, the total amount of worker time spent evaluating the path. Cost can be hard to characterize for an arbitrary query because the time it takes to complete each operation depends on the query being evaluated. For our initial results, presented in the next section, we use one set of operation costs and note that join paths for other queries might perform differently based on the operation costs relevant to that query. For the following results, we estimate that finding a single match for an item and a secondary filter operation both cost one unit of time.

For comparison to the join path, we provide cost estimates for the joinable filter. The upper bound estimate assumes that workers first find a secondary item for a given primary item, decide if the secondary item passes the secondary filter, and repeat until the primary item is evaluated. The lower bound assumes that finding pairs and evaluating secondary filters can be done simultaneously, so it takes half the time of the upper bound estimate. These bounds account for worker heuristics that could reduce total work by applying operations over many items at the same time.

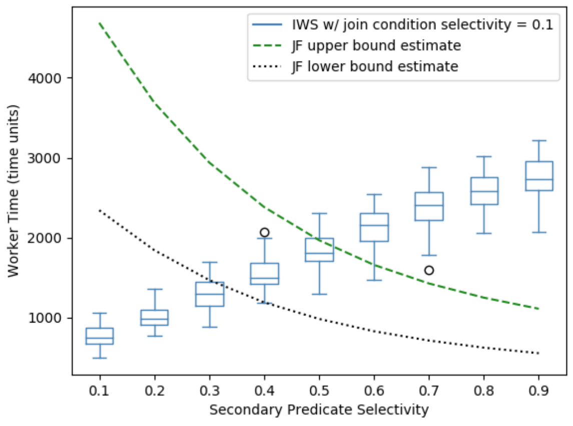

Figure 1: Item-wise on Secondary Items (IWS) vs Joinable Filter (JF). Lower selectivity values are more selective.

Results

Figure 1 shows results from running simulations of an item-wise join path on secondary items (including first enumerating the secondary list) as compared to upper and lower estimates of the joinable filter. The join condition selectivity is fixed at 0.1 and the secondary filter selectivity varies. The box plots are distributions over 50 simulations, each run with 100 primary and 50 secondary items. We can see that when the secondary filter is more selective, the item-wise join on secondary items path is faster than the joinable filter. This effect is expected because fewer secondary items passing the secondary filter means that there are fewer secondary items for which workers must find matches. Therefore, it is possible for a join path to perform significantly better than the joinable filter under favorable circumstances. This graph serves as an example where one of our join paths outperforms joinable filters; future work includes exploring other conditions which may favor join paths.

Conclusions and Future Work

Our initial findings suggest that there are potential cost savings in crowdsourced query processing by replacing joinable filters with joins.

Future work includes using information we have gathered about different join paths and when they perform best to construct an adaptive algorithm that will choose the optimal path for a given query based on its selectivities and other characteristics. To do so effectively, we must continue to characterize the space of join paths by running more simulations with varied parameters. Additionally, next steps include integrating the adaptive join algorithm into Dynamic Filter and testing join paths with real crowd workers.

Acknowledgements

This work was supported by NSF Grant No. 1657259.

References

Lan, D.; Reed, K.; Shin, A.; and Trushkowsky, B. 2017. Dynamic filter: Adaptive query processing with the crowd. In *HCOMP*.

Li, G.; Chai, C.; Fan, J.; Weng, X.; Li, J.; Zheng, Y.; Li, Y.; Yu, X.; Zhang, X.; and Yuan, H. 2017. Cdb: Optimizing queries with crowd-based selections and joins. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, 1463–1478. New York, NY, USA: ACM.

Marcus, A.; Wu, E.; Karger, D.; Madden, S.; and Miller, R. 2011. Human-powered sorts and joins. *Proc. VLDB Endow.* 5(1):13–24.

Mitsuishi, T.; Morishima, A.; Shinagawa, N.; and Aoki, H. 2013. Efficient evaluation of human-powered joins with crowdsourced join pre-filters. In *ICUIMC*, 7:1–7:6. New York, NY, USA: ACM.

Trushkowsky, B.; Kraska, T.; Franklin, M. J.; and Sarkar, P. 2013. Crowdsourced enumeration queries. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 673–684.