Package queries: Efficient and scalable computation of high-order constraints

Neeraj Sharma **Thomas Fang** **Ananya Gupta**

1 Problem Statement

Traditional SQL queries are tailored towards simple constraints where each tuple of the database is evaluated one at a time. However, many real world problems involve higher order constraints which require the resulting set of tuples to satisfy them collectively. Current capabilities are not well suited for this task. Approaches such as directly utilizing integer programming, self-joins, or recursion have their limitations, especially when the problem is scaled up in size. The paper addresses this problem by introducing package queries, a new query model with its own declarative language along with an efficient evaluation approach.

Example of this use-case can be a meal planner where suppose a dietitian wants to decide meal plan of particular calorie range with minimum fat content. So since here we need to consider this as a combinatorial problem, the traditional SQL queries fail for this task.

2 Main Approach

Package queries consists of two main components: a new declarative language and an improved evaluation method.

The new declarative language, called PaQL (Package query language), is a simple and intuitive extension of the traditional SQL language. It's streamlined to be compatible with package query problems. This has the important benefit of easing the burden of the user writing the query. Traditional SQL language becomes lengthy and confusing when it comes to package queries. PaQL does a good job in simplifying the syntax so that query declaration is much more intuitive. For example, it has a SUCH THAT clause where the user can specify the global predicates of the package query. The MAXIMIZE/MINIMIZE clause specifies the objective constraint. Moreover, the paper proves that PaQL has at least the same expressiveness as any integer linear program. It maps a bijection between a general linear optimization problem to the structure of a PaQL query. This nicely ties the new interface it introduced to its improved evaluation method.

The evaluation method is centered around two algorithms DIRECT and SKETCHREFINE. From a high level point of view, DIRECT resembles the traditional straightforward ILP approach while SKETCHREFINE adopts something of a divide and conquer approach. The paper later compares the performance of these two approaches. By setting a baseline with DIRECT, there is basis of comparison for SKETCHREFINE in terms of time and objective performance.

DIRECT is an extension of a traditional integer linear program solver. It first simplifies the query by filtering by base relations which are the typical per tuple constraints. Then it maps the problem to a linear program problem and uses an off-the-shelf solver to get a solution.

The idea of SKETCHREFINE is to break the problem into smaller chunks via partitioning. An offline partitioning strategy divides the data into smaller chunks according to a size constraint τ. If more rigor is desired, each group can also have diameter constraints w_{ij}. Different partitioning methods are possible, but the paper uses k-dimensional quad tree indexing. The data recursively splits into groups until each group satisfies the size and diameter constraints. Then a representative tuple is calculated for each group which is the group's centroid, averaging the tuples in the group over the partitioning attributes. Then SKETCHREFINE has two phases. The SKETCH phase uses an ILP solver to find a solution over the set of representative tuples. Then the REFINE phase replaces one by one each group's representative tuples with actual tuples. The approach is

flexible since the partition can be further clustered if the problem is too big for the ILP to solve and SKETCHREFINE can use backtracking if REFINE phase breaks any constraints.

2.1 Direct Method

This is the basic evaluation method for package queries. SketchRefine is an extension of this method and is basically reused in SketchRefine too. The following are the implementation details-

- **Input** The method takes as input the dataframe which is the entire table, whether the objective is minimization or maximization, constraints for the package query to satisfy, and flag indicating whether or not this is the SketchRefine or not. There is another flag A_0 which indicates whether or not the objective is on tuple count or on other attribute.

- **Output** The method outputs named tuple which contains boolean indicator whether or not the solver failed, list of tuples in the solution, the final objective and the runtime. If the solver fails the status codes and corresponding messages are displayed.

- **Libraries** We used the IBM cplex as the ILP solver for all our methods. We tried using pulp, but the observation was that it took 3-4x more time than what the cplex took, and also pulp was failing on large data sizes, so we used only cplex for solving the ILP problems. For handling the data we have used pandas.

- **Approach** It consists of ILP formulation and ILP evaluation. In the simple direct method, we associate each tuple in the dataframe with an indicator variable which tells whether or not that tuple is included in the solution or not. Then we add all the combinatorial constraints, and finally use cplex to solve the ILP problem. In Sketch phase of Refine we use a slight variation of direct method to include the count constraints of the representative groups explained in detail in section 2.3

2.2 Offline Partitioning

Offline partitioning is a process to break down the whole dataset into similar groups. Authors represent that this way package query solution can be found faster with good error bounds. Paper presents k dimensional quad tree approach which uses size of group τ and diameter constraint of the group as constraint to break the whole dataset into partitions. For partitioning, first all the data is in single group and then recursively we segregate data into partitions until all the partitions satisfy size and diameter constraint. This partitioning process need to be run once and then we can store the partitions and can reuse them further for all the queries. Sketch Refine uses these partitions to make the evaluation of package query faster. Following is the implementation detail:

- **Input**: Dataset with initially all tuples having group id = 1 and τ and partitioning attributes.

- **Recursive Approach**: We run the group by query for each group and see if it satisfies the size constraint. For our implementation we ignore the diameter constraint.

-
  - Base Condition: If group satisfy the size constraint then stop partitioning that group.
  - Recursion: If group size is bigger then size constraint, compute the representative of the group by taking mean. Assign the tuples of that group to new group by comparing partitioning attribute values of those tuples with their representative tuples. Assign new group id to those tuple. [1]

- **Output**: Finally we store the representatives of the groups and whole data with a new column group id. So we have 2 separate csv files one for representatives and the other with whole data and group id.

- **Choosing representatives after partitioning** Finally, for the SketchRefine package, we store the representatives of these groups by using group by query and save min, max and average of partitioning attributes for those groups in representative dataframe.

We experimented with different τ values and will present result of sketch refine for all of these different partition sizes.

[1] https://github.com/neerajsharma9195/paql-project/blob/main/src/partitioning.py

2.3 SketchRefine

In SketchRefine approach, first we sketch a package solution using representative tuples (2.2) and then refines this sketch package by replacing the representative tuples with the original tuples.

Sketch: We use our implementation of direct method over representative tuples with a change that variables are not needed to be binary. They can take integer values as in final package there can be more than one tuples from same representative group.

Refine: Using the sketched solution over the representative tuples, the REFINE procedure iteratively replaces the representative tuples with the tuples from original relation, until no more representative are present in the package. Refine procedure refines the sketch packages one group at a time.

Following are the implementation details:

- **Input** The SketchRefine takes the whole dataframe, representatives obtained from offline partitioning and constraints and objective of the original package query as input.

  The Sketch method only takes the representatives (after offline partitioning of data), and the package query as input. The Refine method takes the entire dataframe, group id on which the current refine is running implying which group representatives have to be replaced with the original tuples, and the package query as input.

- **Output** The SketchRefine method returns the same output as described in section 2

- **Approach** This version of SketchRefine does not include backtracking. Here we simply refine all the groups one by one in a loop and in case of failures return that ILP failed in refine on a particular group. The details of the Sketch and Refine approach are described in detail in the next section 2.3

- **Analysis** The observation was that the SketchRefine without backtracking was returning infeasible solutions very frequently, suggesting the need of the backtracking (following section).

2.4 SketchRefine with Backtracking

This is an extension of the SketchRefine with backtracking support in case of infeasible solutions when refining with a particular order of groups. The failing groups are given a higher priority and the method backtracks and refines in a different order after that. The motivation is to reduce the infeasible solutions.

- **Input Output** of this method are similar to that of the SketchRefine in previous section 2.2. However the Refine method parameters are different. It takes as input the dataframe, representatives, the list of groups, refining package and the package query. Output being the refining package and list of failed groups.

- **Approach**
  - **Sketch** This method uses the direct method with flag sketch True to solve the ILP. It takes as input the representative dataframe and in the direct method there are certain differences in solving the package query. The main change is that in order to accommodate for the upper bound count constraints of the group representatives, we have considered an integer variable to be associated with each tuple instead of an indicator variable. So basically the integer variable tells how many of that group instances or tuples need to be taken from one particular group which is represented by the representative element. At the end of the sketch method, we also add another column to the representative dataframe which contains the solution of this integer variable indicating how many original tuples need to be replaced in the refine method.
  - **Refine with Backtracking** The refine with backtracking method is very similar to the pseudo code in the Algorithm 2 of the PaQL paper. It pushes all the groups left to refine in a priority queue initially in a random order. It then pops elements from the **priority** queue and runs direct method on that particular group refining the group with original tuples.
    This requires the constraints to be **updated** too, as the refining package is be-

ing updated continuously and the constraints should be cognizant of the original tuples being incorporated instead of the representative tuples. So basically for the upper and lower bounds we subtract the already present attribute sum of the remaining of the refining package. And then calls the direct method for only this particular group with upper bounding the maximum number of tuples to the integer variable solution of the representatives.

When the refine method fails to replace a particular group with the original tuples from the dataset, the method backtracks. The way this happens is by keeping track of the list of failed groups and then increasing the priority of those groups which failed. Thereby forcing the groups with higher priority which failed earlier to be picked up for refining sooner.

2.5 Hybrid Package Query

For some queries in order to achieve feasibility we had to use a variation of representatives. More specifically for all our queries earlier we were using the mean of the attributes as representatives for each group. But this was causing infeasibility issues, because the mean representative was not a good substitute of all the group tuples. So we switched to considering the min and max also for all attributes. The observation was that this led to feasibility for all of the queries.

3 Experimental Results

The experimental results validated the efficacy of the proposed methods. By testing on TPC-H benchmark, it demonstrated that

1. new evaluation strategy performs around an order of magnitude faster than using the ILP solver directly over the problem,

2. new methodology allows the problem to scale up to sizes that solvers can't handle directly,

3. the output packages have objectively good quality, and

4. the flexible offline partitioning approach is robust in its application to different queries.

5. Additionally, another observation was that seeding the solver with good initial guesses can significantly improve performance time.

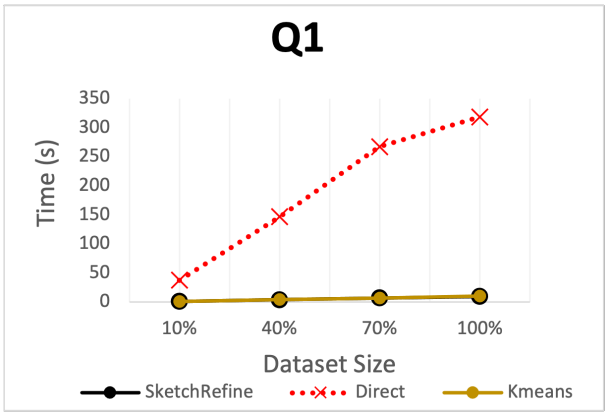
The experiments were held using benchmark data utilized TPC-H (table size of over 6 million rows). The results compared the performances of DIRECT, which showcased using ILP solver directly on entire dataset, verses using SKETCHREFINE evaluation method. The two main metrics were performance time and approximation ratio, which is the ratio of objective value between DIRECT and SKETCHREFINE methods.

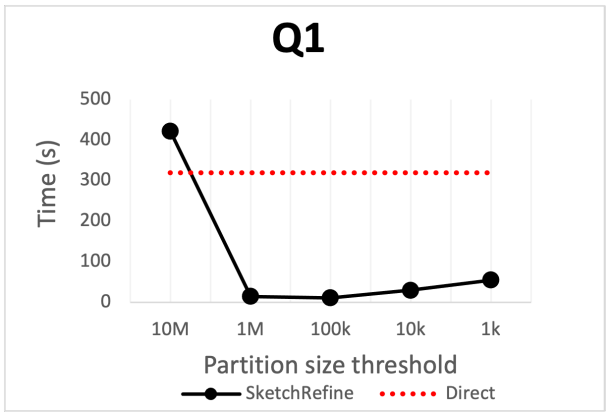
The first experiment examined the scalability of the two methods. Figure 1 show the scalability of the direct method and the sketch refine with quad-tree and k-means partitioning methods. Setting τ at 1 million, the experiment plotted performance time versus dataset size across four different queries for TPC-H dataset. Overall, SKETCHREFINE scaled much better with increasing dataset size, and it performed significantly faster than DIRECT. The mean and median approximation ratio for the objective function for all the queries was also seen to be good as mentioned in the figure, with Q4 giving almost 1.0 approximation ratio.

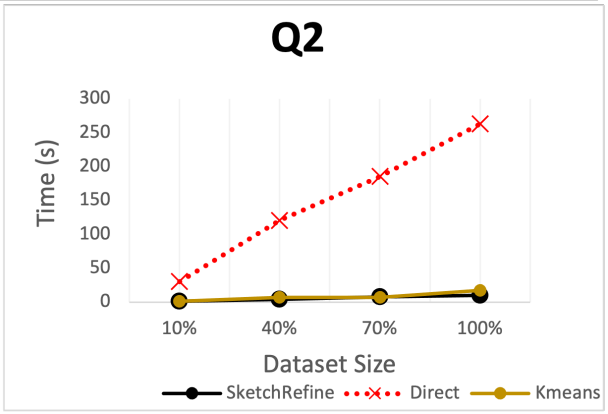
Figure 2 shows the results of the second experiment which compared performance time versus varying size thresholds τ. DIRECT doesn't vary by τ but it was used as a baseline. The results indicated that performance time follows a bit of a U-curve. Initially, when τ is very large, SKETCHREFINE is very similar to just directly using the solver on its few partitions. Then as τ decreases and the number of partitions increase, performance time improves since the solver is applied to more manageable group sizes. Finally, as τ hits a point where there are too many partitions, then performance time actually increases since the SKETCH and REFINE phases are applied to a larger set. The results show that τ has a big impact on performance time but has very little impact on approximation ratio. With proper tuning of τ, SKETCHREFINE performs an order of magnitude faster than DIRECT.

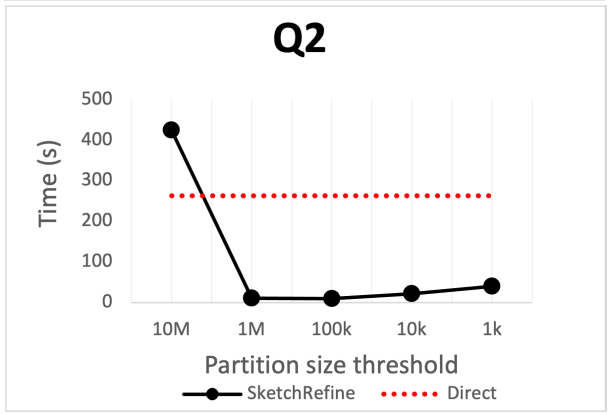
4 Limitations

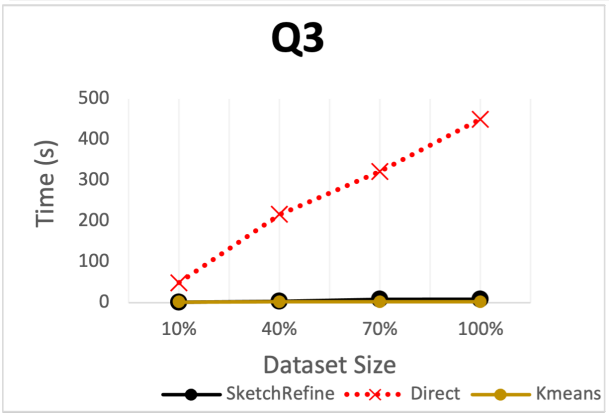
There are some limitations to the paper. The paper focused its study on linear aggregate functions for global predicates and objective constraint. Although PaQL could theoretically support a broader

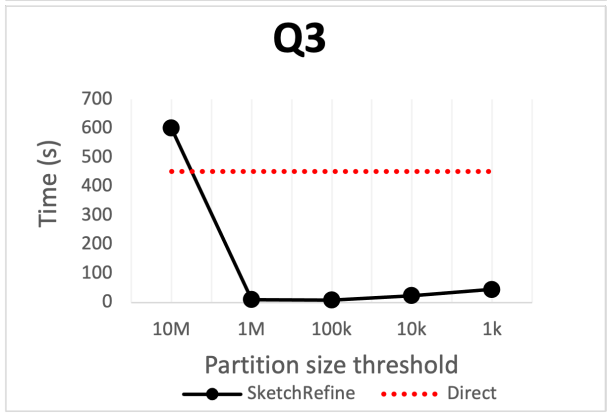


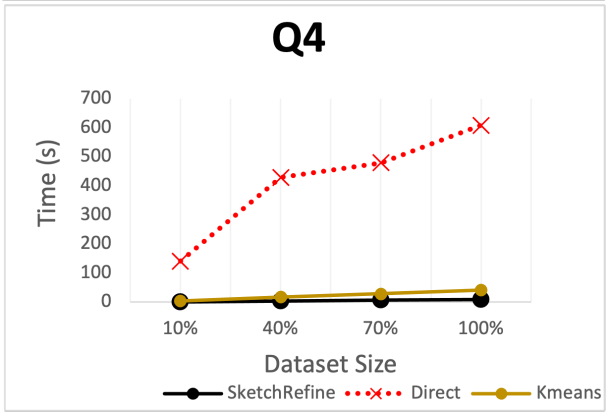












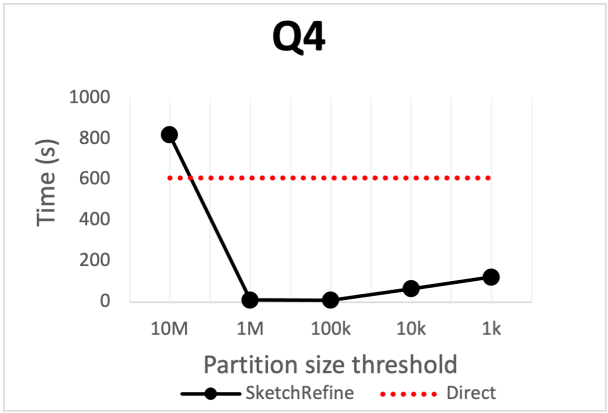


Figure 1: (a) Q1 Datasize vs Time - Approximation Ratios **(for K quad: mean=1.24, median=1.20) (for K Means - mean: 1.12, median: 1.12)** (b) Q2 Datasize vs Time - Approximation Ratios **(for K quad: mean=1.04, median=1.04) (for K Means - mean: 1.99, median: 1.02)** (c) Q3 Datasize vs Time - Approximation Ratios **(for K quad: mean=1.50, median=1.50) (for K Means - mean: 1.38, median: 1.5)** (d) Q4 Datasize vs Time - Approximation Ratios **(for K quad: mean=1.0, median=1.0) (for K Means - mean: 1.0, median: 1.0)**

Figure 2: (a) Q1 Part size with Approximation Ratio - **mean: 1.15, median: 1.13** (b) Q2 Part size with Approximation Ratio - **mean: 1.8, median: 1.04** (c) Q3 Part size with Approximation Ratio - **mean: 1.4, median: 1.5** (d) Q4 Part size with Approximation Ratio - **mean: 1.0, median: 1.0**

range of functions, the application of those functions extended beyond the scope of the paper. Additionally, the paper only focuses on the subset of package queries with an objective constraint. The objective constraint simplifies the output of the package query to a single set of tuples. Without it, the result would be a set of sets of tuples which would require further considerations in implementation and constructing a representation. Another limitation is SKETCHREFINE may suffer from false in infeasibility, which happens when the algorithm reports a feasible query to be infeasible, though the probability is bounded but it might occur.

5 Extension and Results

Besides implementing the basic algorithms and reproducing results[2], for our extension we compared the performance of SKETCHREFINE using two different partitioning methodologies. We implemented both k-means clustering and k-dimensional quad trees and have analyzed their performance.

5.1 Analysis: K-quad tree based partitioning

K-quad tree based partitioning was the main partitioning technique used in the papers. We implemented this method which recursively splits up clusters until every cluster satisfies the size threshold τ. We picked either min, max, or mean for representative tuples depending on the query's constraints. Running the offline partitioning did take a fairly significant amount of time (around 2-4 hours per partition). We were able to expedite the process by building off previous partitions to create new ones. The benefit of offline partitioning is that creating the partitions is just a one time cost and running future queries will be much faster compared to Direct.

5.2 Analysis: K Means clustering based partitioning

We also implemented K means clustering, using the scikit library, and ran SketchRefine with the resulting partitioning output. As our experimental results demonstrate, the runtime and approximation ratio of SketchRefine is very similar to that of K-quad tree indexing. Both have extremely good run time and very reasonable approximation ratios. The main difference between the two methods lies in the flexibility of partitioning design and also memory/computational constraints. K means does not allow for partition size limits. Moreover, there are scaling issues with selecting the number of cluster sizes. We found that at higher cluster sizes such as 500 or 1000, partition creating process would still be running even after 24 hours, indicating computational constraint. We chose to use 100 clusters. Creating this partition took only around 15 minutes. Thus, we see that there are benefits and drawbacks of K means compared to K Quad tree indexing. K means cannot partition in memory as finely K quad tree indexing or have as much flexibility in partition design, but generating the partitions is faster and produces reasonably good results.

6 Summary

We were able to reproduce the results of the PaQL paper, implementing Direct and SketchRefine with backtracking methods. Our experimental results validated the results of the paper, showing similar trends over TPC-H dataset. We implemented offline partitioning using both K dimensional quad tree and K means clustering as part of our extension. Overall we can conclude that the new evaluation approach using SketchRefine scales significantly better with dataset size and runs much faster.

7 Team Contributions

The work was split evenly among the three of us. We collaborated throughout the project, brainstorming and planning together. At a more detailed level, Thomas took the lead with Direct, Neeraj with partitioning, and Ananya with backtracking.

[2] https://github.com/neerajsharma9195/paql-project