Extended Experiment Report

15 July 2018

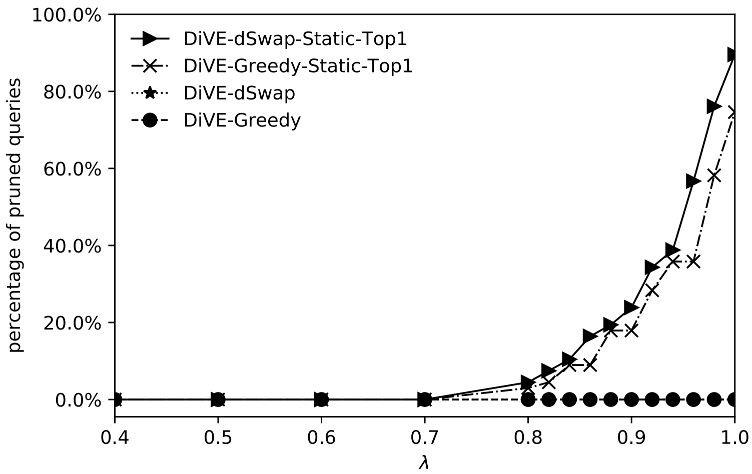

Figure 1: Static bound pruning performance while different value of λ, $k = 5$, MaxSum diversification function and running on Fligths dataset

1 Greedy Static Max-Min vs Greedy Static Top-1

Greedy MaxMin and Greedy Top1 are very similar, both algorithms have same logics. First, as the initialization to get the set S, two most distant views are selected based on diversity. Then, all candidate views in X are sorted based on $setDist$ (distance from view in X to set S). In case of MaxMin pruning algorithm, $Umax$ of all candidate views in X are computed using actual score of diversity and maximum bound of importance score while $Umin$ of all candidate views in X are computed using actual score of diversity and 0 as the importance score. All views in X are pruned while the value of $Umax$ less than the maximum of $Umin$. The view in X is executed one by one from the top, since the actual score of importance has been known, the actual utility score of the view can be calculated. This actual utility score of view will be used as the maximum of $Umin$. Since, more views are executed the higher actual utility will be used as the maximum of $Umin$ and more views can be pruned.

Meanwhile, Top1 algorithm directly uses the total objective function of the set $F(S)$. After all candidate views in X are sorted based on $setDist$, the view is executed one by one start from the top. While the importance score of first executed view has been known, the $F(S \cup X[0])$ can be calculated and this objective function as the current objective function $F(S)$. Then, $maxF(S \cup X[i])$ of all remaining views in X are calculated using actual diversity score and maximum bound of importance score. If $maxF(S \cup X[i]) < F(S)$ then it is guaranteed that the actual objective function to be less than the current objective function $F(S)$ and those views can be pruned. The current objective function $F(S)$ is updated while the next $F(S)$ of executed query has higher score than the current. This experiment use Flights dataset with 5 queries, Figure 1 shows that both algorithms have similar performance.

For the rest, we will use Top1 instead of MaxMin pruning due to we also have Swap technique that implement Top1 approach. The reason why we still need swap is because Greedy starts with small number of views in the initialization. Only two views as the set S. Hence, while calculating the $setDist$ of each views in X, there are a lot of views have same score that it may decrease the chance of pruning. Moreover, there is no guarantee that two most distant views which selected by Greedy as the initialization have high importance score and there is no way to replace those two views. However, Swap has bigger number of views in the initialization (e.g. five views). While calculating the $setDist$ of views in X, the chance of views have same score is lower than Greedy. Additionally, swap also has a replacing mechanism that can replace low quality views in the current set with the better one.

2 Distance Functions

In terms of distance functions, I have been looking for the bounded distance function to compare between two probability distributions, there are several distance functions that I have studied:

Cont.

- Euclidean distance, currently is used and the maximum bound is $\sqrt{2}$.
- Kullback-Leiber (KL) distance, this distance is not bounded, the mathematically proof can be seen below.
- Earth Mover Distance (EMD), this distance is widely used and very good for comparing two probability distributions. Mostly this distance used in computer vision application to compare between two histograms. However, based on my finding, this distance is not bounded as well.
- Kolmogorov Distance has the maximum bound equal to 1. However, mostly example code that I found uses this distance as hypothesis test that need another parameters such as α and confidence interval.

Until now, I am not sure if there is a bounded distance function that can be used for our work except the Euclidean distance.

2.1 Euclidean Maximum bound

For the general case, Euclidean distance d is defined as following: $d = \sqrt{\sum(x-y)^2}$, where $\sum(x-y)^2 = \sum x^2 + \sum y^2 - 2\sum xy$. Given that in probability vectors all values are nonnegative, d is max when the last term is zero, then $d = \sum x^2 + \sum y^2$. The theoretical maximum is attained when $2\sum xy = 0$ and $\sqrt{2}$ as the maximum theoretical bound can be proven as following:
$\sqrt{\sum(x-y)^2} \leq \sqrt{\sum(x)^2 + \sum(y)^2}$
$\sqrt{\sum(x-y)^2} \leq \sqrt{1+1}$
$\sqrt{\sum(x-y)^2} \leq \sqrt{2}$

2.2 Maximum bound of Kullback-Leibler (KL) distance

For distributions which do not have the same support, KL divergence is not bounded. Look at the definition: $KL(P||Q) = \int_{-\infty}^{\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right) dx$

If P and Q have not the same support, there exists some point x' where $p(x') \neq 0$ and $q(x') = 0$, making KL go to infinity. Even both distributions have the same support, when one distribution has a much fatter tail than the other. Then:

$$KL(P||Q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x$$

when

$$p(x) = \overbrace{\frac{1}{\pi}\frac{1}{1+x^2}}^{\text{Cauchy density}} \qquad q(x) = \overbrace{\frac{1}{\sqrt{2\pi}}\exp\{-x^2/2\}}^{\text{Normal density}}$$

then

$$KL(P||Q) = \int \frac{1}{\pi}\frac{1}{1+x^2}\log p(x)\,\mathrm{d}x + \int \frac{1}{\pi}\frac{1}{1+x^2}[\log(2\pi)/2 + x^2/2]\,\mathrm{d}x$$

and

$$\int \frac{1}{\pi}\frac{1}{1+x^2}x^2/2\,\mathrm{d}x = +\infty$$

In conclusion, Kullback-Leibler (KL) is not bounded. For instance, when I implement KL in my code. In some case while the bin does not has its pair in the reference (which means 0), the result are two posibilities: error devided by 0 or log 0 which is undefined.

Cont.

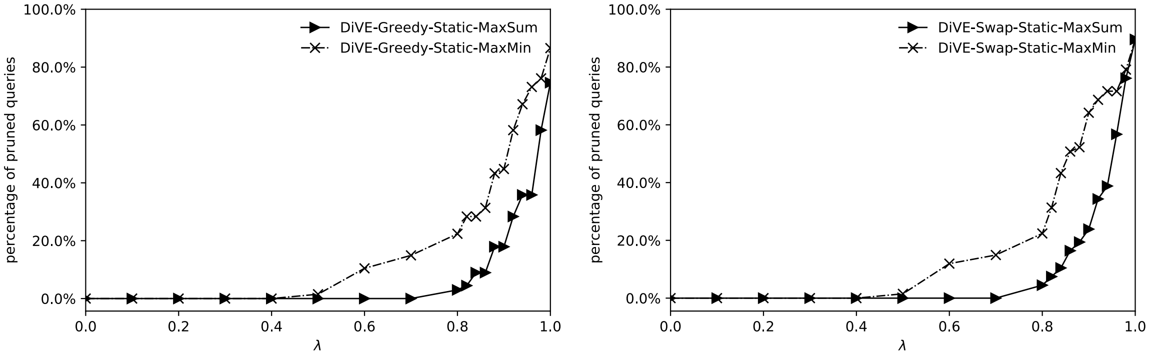

Figure 2: MaxSum vs. MaxMin diversification on Greedy and Swap running on Flights dataset, $k = 5$

Table 1: Example setDist score

v in X	MaxSum	MaxMin
v_1	0.5	1.0
v_2	0.472222222	0.833333333
v_3	0.444444444	0.666666667
v_4	0.4166666667	0.5
v_5	0.3899999999	0.333333333
v_6	0.3611111111	0.166666667
v_7	0.3333333333	0.166666667

3 Max-sum and Max-min diversification

Current results that we have are using MaxSum diversification. Based on my hypothesis and experiments that MaxMin Diversification can improve the pruning performance. The results while using MaxSum vs. MaxMin on Greedy Static Top1 and Swap Static Top 1 can be seen in Figure 2.

MaxSum uses average score of diversity of the set S which is by computing the total sum of all distances then dividing by $k * (k - 1)$ while MaxMin uses the maximum of minimum score of distance in the set S. Hence, the range diversity score from those both approaches are different. For instance, there are three views in set Z which each view is different with others. The maximum score of distance between two views is 1 and the minimum is 0. Using MaxSum method the diversity score of set Z will be $(1 + 1 + 1)/(3 * (3 - 1)) = 0.5$ whereas diversity score of MaxMin is 1 because the minimum distance in the set Z is 1.

The example variance of *setDist* score using Flights dataset between MaxSum diversification and MaxMin diversification can be seen in the Table 1. In this experiment, I selected two most distant views as the initial set S and then calculate the *setDist* of all views in X. For instance, the highest score of *setDist* is $v1$, where on MaxSum the maximum sore is 0.5 and on MaxMin the maximum score is 1. This Table is just an example, in the real data there are many views have same score. In this Table, I only want to show the distributions of *setDist* score and the different range of *setDist* score between MaxSum and MaxMin.

Due to this different diversity score, MaxMin diversification can improve the pruning performance as shown in the Figure 3, **in fact, I also found MaxMin diversification is able to start pruning while $\lambda = 0.4$ in some query subsets**. However, this MaxMin makes unbalance between the importance score and diversity score. The maximum diversity can be equal to 1 while the value of importance score is lower than that. This thing makes the shape of objective function unbalance. Please see Figure 3 below for more details.

Cont.

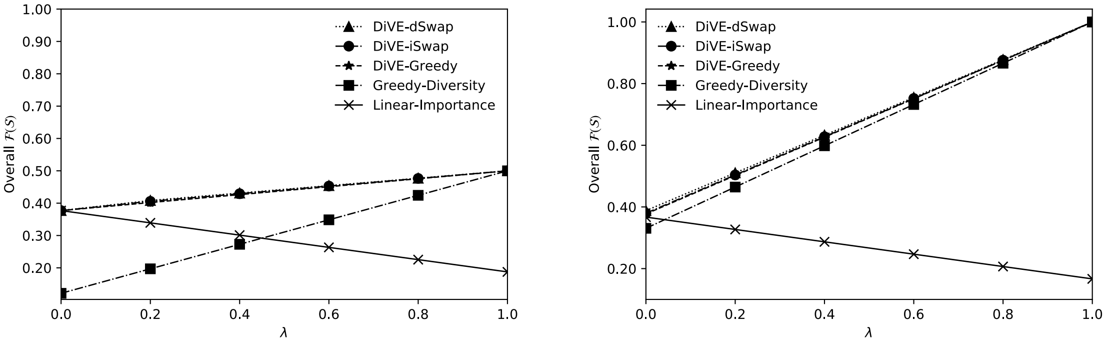

Figure 3: Objective function shape on MaxSum vs. MaxMin diversification while different value of λ, $k = 5$

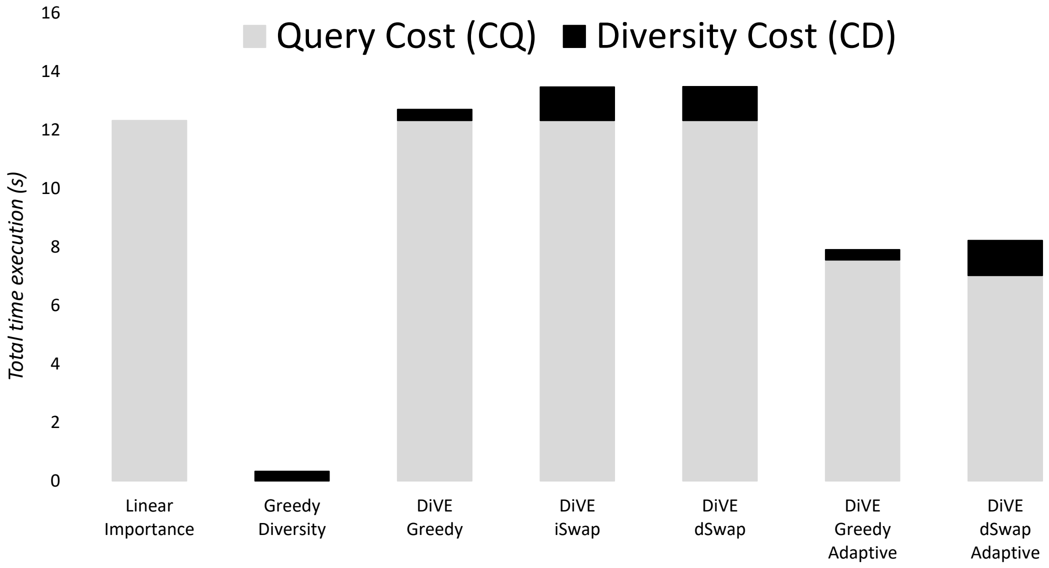

Figure 4: Total costs of schemes running on Flights dataset, MaxSum diversification, $k = 5$, $PI = 0.97$, and $\lambda = 0.5$

4 Total cost with and without pruning

To see the performance of our proposed pruning approach, especially compared to schemes without pruning, Figure 4 shows the total cost running on Flights dataset. As shown in the Figure, DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive are able to reduce query cost. This experiment using adaptive algorithm while $PI = 0.97$ and this result is the avarage from five queries on Flights dataset.

5 Impact of k to pruning performance

Before observing the impact of k to pruning performance, due to the Greedy algorithm has been changed which is using top1 approach, as the result, Greedy and Swap have close performance while using static bound as well as the adaptive bound. The performance of Greedy vs Swap while using static bound can be seen in the Figure 1 whereas the performance while using adaptive bound can be seen in the Figure 5. Using adaptive maximum bound both DiVE-Greedy and DiVE-dSwap are able to prune queries start from $\lambda = 0.2$, and with higher λ more queries are pruned.

Cont.

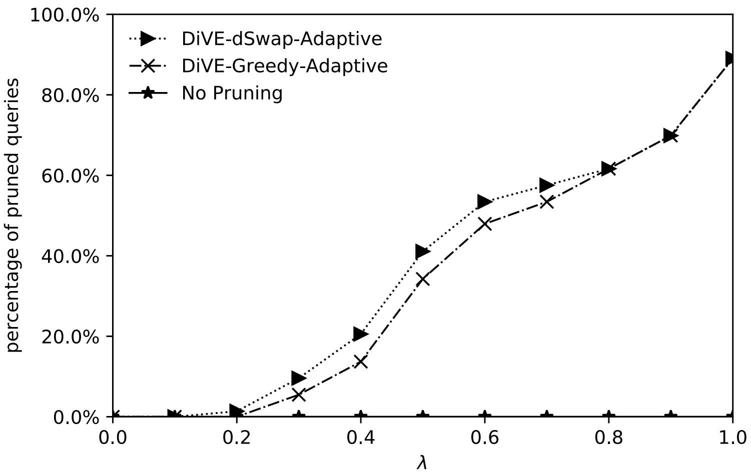

Figure 5: Adaptive bound pruning performance using different value of λ, $k = 5$, $PI = 0.97$, MaxSum diversification, running on Flights dataset

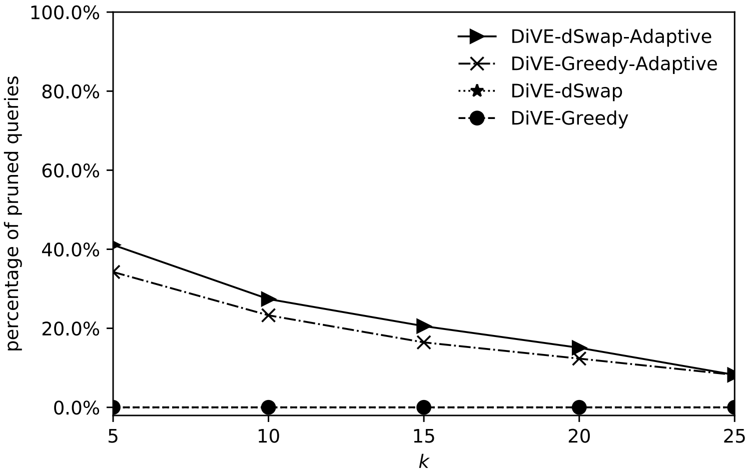

Figure 6: Impact of k to pruning performance, $\lambda = 0.5$, MaxSum diversification, running on Flights dataset

The next experiment is to observe the impact of k to pruning performance while the λ is constant. Figure 6 shows the impact of k to pruning performance. This experiment running on Flights dataset, using 5 queries and $\lambda = 0.5$. The pruning performance is decreasing while k is increasing.

6 DiVE-Greedy and DiVE-Swap complexity

The costs of Greedy Construction algorithm has two components which are the query execution cost C_Q that computing the importance score of view and the diversity cost C_D that computing set distance of each view from the views already in S. The complexity of query execution cost is $O(n)$ as the content of each view is generated only once. The diversity cost C_D is $O(kn)$ where k is the size of subset of views S and n is the number of all possible views.

Meanwhile, The costs of Swap algorithm is also depend on the query execution time C_Q of all possible views and the diversity computation C_D. The query cost C_Q is executed only once but the cost is high due to it needs I/O cost. However, the complexity of diversity computation C_D is $O\left(k^2 n\right)$ and the number of distance computation depends on the number of iterations of the swap and the number of views in X. In the worst case, swap algorithm can perform $O\left(k^n\right)$.

Cont.

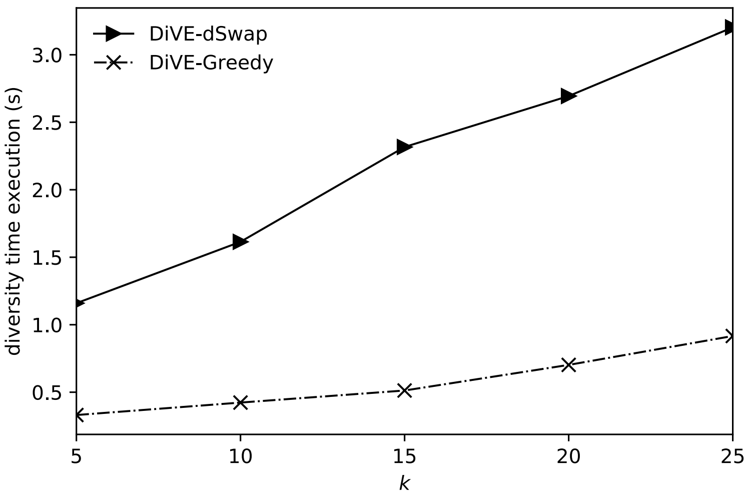

Figure 7: Impact of k to diversity computation running on Flights dataset

The different of total cost between Greedy and Swap approach can be seen in Figure 4. Meanwhile, the impact of k to the distance computation time while running on Flights dataset can bee seen in Figure 7.

7 Proposed algorithm's Figure

We proposed DiVE scheme using two kinds of popular heuristic approach which are Greedy and Swap techniques. Figure 8 shows our proposed scheme technique. In case of Greedy, the set S is initialized by two most distant views. In each iteration, the best view in X that can maximize objective function $F(S)$ will be added to the set S. Meanwhile, for the Swap case, the set S is initialized by the most diverse k views and in each iteration, all candidates views in X will be interchanged to set S and view which can improve the current objective function $F(S)$ will replace a view in set S.

The *setDist* score is used to sort all candidate views in X, the highest score means that views is more diverse to set S. The current bound is utilized to compute the estimate utility of each candidate view. First, the theoritical maximum bound ($\sqrt{2}$) is used and this bound is updated while the actual importace score of view has been known. To avoid the wrong current bound, samping based on prediction interval is used. Before running the program, user needs to defined what PI that she wants to use. For instance, while users set PI to 80 then after 9 views is executed the current bound will be updated to the maximum bound which have seen so far. The most used PI can be defined as following:

- PI80: need to execute 9 views
- PI85: need to execute 12 views
- PI90: need to executes 20 views
- PI95: need to executes 40 views
- PI97: need to executes 60 views

8 Query sharing computation

Beside using pruning, SeeDB used query sharing optimizaation to reduce the query executions as well. To do query sharing optimization, SeeDB used combine multiple aggregates, combine multiple GROUP BY, and combine target and reference view query. However, pruning and query sharing seem two kinds of orthogonal optimizations. Hence, SeeDB proposed *phased execution framework* which was by dividing dataset to phases. For instance, if we have 100,000 records in our dataset and we use 10 phases, the $i = 4^{th}$ processes records 30,001 - 40,000.

Cont.

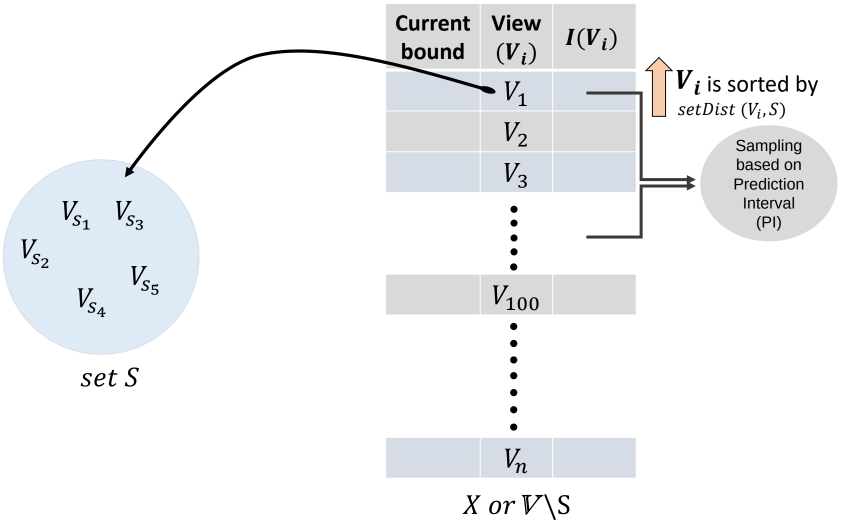
Current bound
View (V_i)
$I(V_i)$
V_1
V_2
V_3
V_{100}
V_n
V_i is sorted by $setDist\ (V_i, S)$
Sampling based on Prediction Interval (PI)
V_{s_1} V_{s_3} V_{s_2} V_{s_5} V_{s_4}
set S
X or $\mathbb{V}\backslash S$

Figure 8: Utilizing *setDist* score to sort candidate views and update current maximum bound to optimize the pruning performance

In fact, SeeDB used partial result of each aggregate view on the fractions and used it to estimate the quality of each view and prune low-quality views in each phase. The query sharing optimization is used to minimize scans on the fraction of dataset in each phase whereas the pruning-based optimization is processed in the end of each phase.

If we see the SeeDB proposed approach, they divided the data to many phases and in the begining of phase, the query sharing computation is applied in the fraction and use that partial result to estimate the quality of view which still under consideration and in the end, the pruning-based optimizataion is applied. Due to of this, I do not think that this query sharing approach is applicable to our work.

In case of our work, we sorted the remaining views based on *setDist* score and execute the query view one by one start from the top and we need to know the importance score of executed view in order to get the current $F(S)$ and to update the maximum bound. We also assume that by sorting the views based on *setDist* the chance view is similar to others view in the above and below position are low. It means that to do query sharing execution by combining multiple aggregate or GROUP BY is difficult. There is no possibility to collect all views with same aggregate function or GROUP BY then execute in the same time due to we need to know the importance score of each view one by one.

9 Correcting wrong maximum bound in adaptive pruning scheme

I developed a program that able to store all bound changed and pruned views in each bound changed. This program is not correcting the bound and bring back pruned views directly while the program realizes that the maximum bound is wrong. That is because while that approach is used, the program may face never ending looping because it repeats the previous wrong step while the maximum bound is wrong. The worst case happens while the view which has the maximum bound is in the below position of list X, if the direct fixing approach is applied then the program always repeat previous step which has wrong maximum bound.

In order to overcome this issue, this program runs until the end without caring the wrong maximum bound. Then, instead of showing the result to the user in the end of running, this program will re-run using the maximum bound that is founded in the first run which definitly the correct maximum bound. However, this approach seems running the program twice and it makes double execution time.

10 New Datasets

- Not yet

The End.