Extended Experiment Report

August 2018

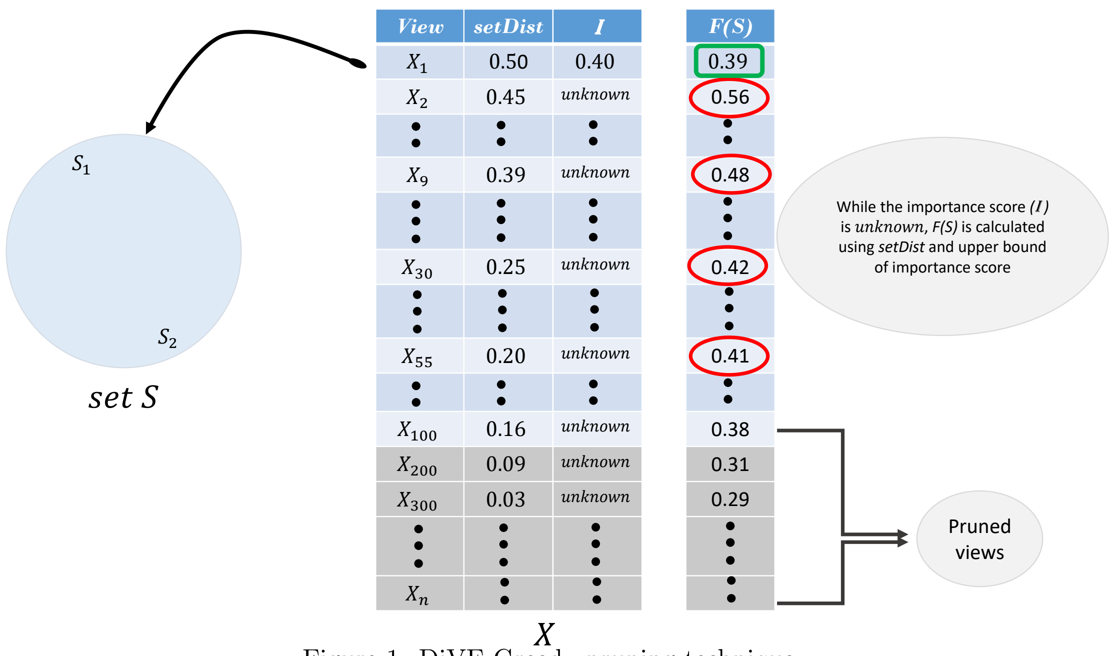

View
setDist
I
F(S)
X_1 0.50 0.40 0.39
X_2 0.45 unknown 0.56
X_9 0.39 unknown 0.48
X_{30} 0.25 unknown 0.42
X_{55} 0.20 unknown 0.41
X_{100} 0.16 unknown 0.38
X_{200} 0.09 unknown 0.31
X_{300} 0.03 unknown 0.29
X_n
S_1
S_2
set S
X
While the importance score (I) is unknown, F(S) is calculated using setDist and upper bound of importance score
Pruned views

Figure 1: DiVE-Greedy pruning technique

1 Pruning performance of DiVE-Greedy and DiVE-dSwap

Note: *Greedy MaxMin and Greedy Top-1 are same, both algorithms have same logic. For the rest of this report, two proposed schemes DiVE-Greedy and DiVE-dSwap will be discussed and both algorithms use Top-1 strategy.*

1.1 DiVE-Greedy

Similar to the classical Greedy Construction, DiVE-Greedy initializes the set S with the two most distant views. Then, DiVE-Greedy iteratively selects new views to be added to S. Particularly, in each iteration a view is selected from the set of remaining views in X and is added to S. The criterion for the view that can be selected is to improve the overal hybrid objective function $F(S)$. Without pruning mechanism, that requires iteration of all candidate views in X to find the best view that can improve $F(S)$ most. While in that process, executing query of each view is needed. To avoid such expensive processing, pruning scheme is proposed.

Our proposed pruning technique is based on the observation that the utility score of each view $U(V_i)$ is a weighted sum of two measures; 1) the importance score of the view (i.e., $I(V_i)$), and 2) the distance of the view from S (i.e., $setDist(V_i, S)$). We note that the computation of $setDist(V_i, S)$ is a CPU-bound (i.e., the execution time is determined by the speed of CPU). To the contrary, computing the importance score of a view $I(V_i)$ is an expensive operation that requires executing two queries (i.e., target and reference data) of V_i (i.e., I/O-bound, where the execution time is determined by the disk speed that more slowly compared to CPU-bound). Our pruning technique works by leveraging the diversity score and importance score that allows for pruning low-utility views without incurring the high cost for evaluating their importance as described next.

For instance, Figure 1 shows how DiVE-Greedy pruning works. After the set S are initialized by two most distant views, X is sorted based on the distance score of candidate views in X to S (i.e., $setDist(X_i, S)$). The highest score will be on the top (i.e., the highest score of $setDist$ in this example is 0.50). Notice that up to this point there is no query execution, only diversity computation cost (i.e., $setDist(X_i, S)$) is needed. However, to calculate the $F(S)$, the importance score of view need to be known by executing the query view. To get the importance score of the view, the query view is executed one by one starts from the top. While first view X_1 is executed and its importance score has been known (i.e., the importance score of X_1 is 0.40), the hybrid objective function can be calculated. The $F(S)$ is the objective function of set S while X_1 is

Cont.

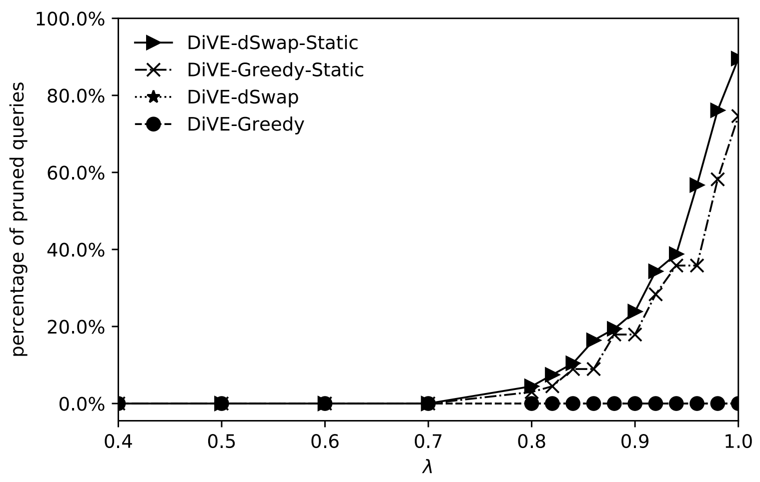


Figure 2: Static bound pruning performance while different value of λ, $k = 5$, MaxSum diversification function and running on Fligths dataset

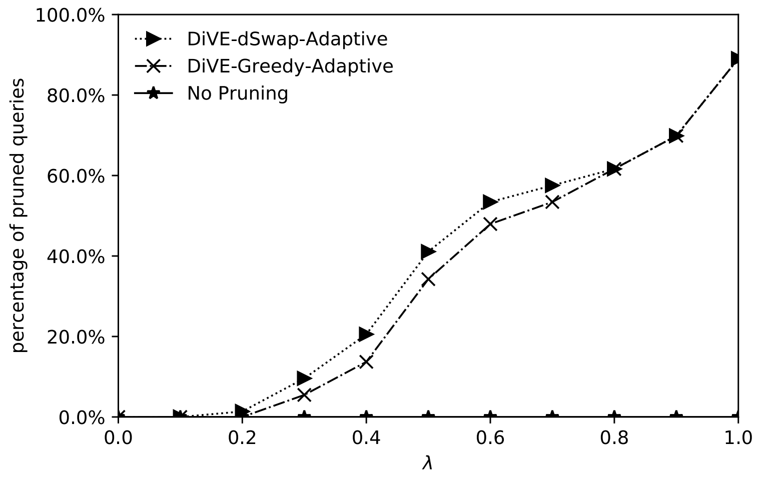


Figure 3: Adaptive bound pruning performance using different value of λ, $k = 5$, $PI = 0.97$, MaxSum diversification, running on Flights dataset

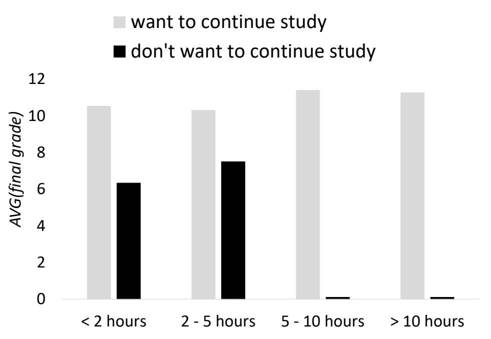
added to the set S (i.e., $F(S \cup X_1)$). This objective function will be the current objective function $F(S)$ (i.e., 0.39). Then, $maxF(S \cup X_i)$ of all remaining views in X are calculated using actual diversity score and upper bound of importance score.

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 these views can be pruned (i.e., all views below X_{100} will be pruned). It cannot be denied because the real importance score is always below the upper bound. Meanwhile, if $maxF(S \cup X_i) > F(S)$ such $X_2 - X_{55}$ (i.e., read circle), these views need to be executed start from the top. If on the next query view execution, there is a view that can improve $F(S)$ compared to the current $F(S)$ the currect $F(S)$ will be updated. This technique is able to prune many low-quality views as shown in Figure 2.

1.2 DiVE-dSwap

As shown in the Figure 2, DiVE-Greedy has a good pruning performance even it uses static bound. It will have better performance while adaptive bound is used as shown in Figure 3. Although DiVE-Greedy is able to prune queries, Greedy starts with small number of views (i.e., two most distant views) in the initialization set S that these views cannot be replaced. Hence, this technique may reducing the quality of recommended

Cont.



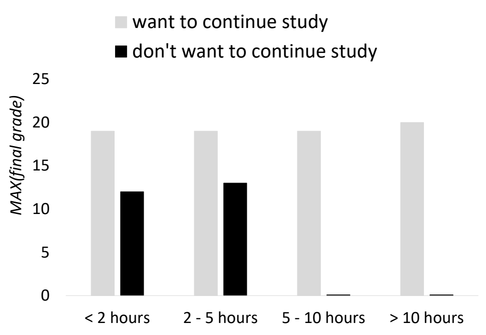


Figure 4: Study time and final grade of students who want to countinue their study to higher education vs. students who do not want to continue their study to higher education

views and pruning performance. For instance, 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. Moreover, due to our context-driven only consist of three components (i.e., Attribute, Measure, and Aggregate Function), in the first Greedy iteration many views have same $setDist(X_i, S)$ score because only two views are in the set S. This issue may decrease the chance of pruning.

In order to overcome these issues, DiVE-dSwap is proposed as well. DiVE-dSwap technique has a replacing mechanism that can replace low quality views in the current set S with the better one. In addition, DiVE-dSwap has bigger number of views in the initialization of set S ($|S| = k$). Meanwhile, higher number of views in the initialization of set S can decrease the chance of views in X have same $setDist$ score.

In this experiment, we used Flights dataset with 5 queries, Figure 2 shows the pruning performance of DiVE-Greedy and DiVE-dSwap while static bound is used and Figure 3 while adaptive bound is used.

2 New Dataset

The student performance dataset is used in this experiment as the motivation example dataset. This dataset can be downloaded from Kaggle [1] and UCI ML website [2]. This dataset has 28 attributes, 5 measures, and four aggregate functions are used. The data were obtained in a survey of students math and Portuguese language courses in secondary school. It contains a lot of interesting social, gender and study information about students. In this experiment, only math dataset is used.

For instance, the analyst want to compare between students who want to continue their study to the higher education and students who do not want to continue their study to higher education. Figure 4 shows two views that have the highest importance score: 1) A = studytime, M = final grade, and F = AVG, the importance score is 0.812831582477977; 2) A = studytime, M = final grade, and F = MAX, the importance score is 0.797221012363291. This Figure shows that without diversity, recommended views may suffer from redundancy.

Figure 4 also shows that students who want to continue to higher education relatively have better final grade compared to students who do not want to continue to higher education. Interestingly, some students who want to continue their education, they spend more time to study (5 to 10 hours, even more than 10 hours) per week. To contrary, students who do not want to continue their study, all of them only spend less than 5 hours a week for study.

[1]https://www.kaggle.com/uciml/student-alcohol-consumption
[2]https://archive.ics.uci.edu/ml/datasets/student+performance

Cont.

3 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:

- Kullback-Leiber (KL) distance, this distance is not bounded, the mathematically proof can be seen below.
- Euclidean distance, currently is used and the maximum bound is $\sqrt{2}$.
- 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.

3.1 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.

3.2 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 $d = \sum x^2 + \sum y^2$. All values are between 0 and 1 (sum up to 1), $\sum x = \sum y = 1$. The theoretical maximum is attained when $2\sum xy = 0$ and $\sqrt{2}$ as the maximum theoretical bound can be proven as following:
$\sqrt{\sum(x-y)^2} \leq \sqrt{\sum(x)^2 + \sum(y)^2}$
$\sqrt{\sum(x-y)^2} \leq \sqrt{1+1}$
$\sqrt{\sum(x-y)^2} \leq \sqrt{2}$

Cont.

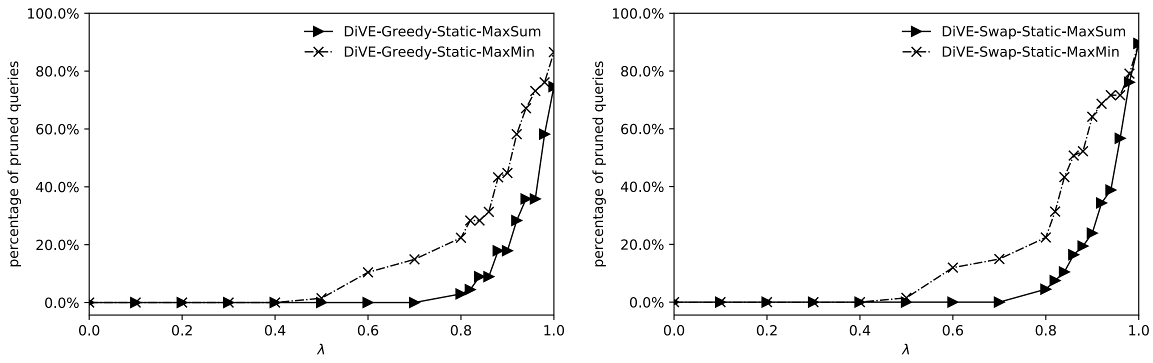


Figure 5: MaxSum vs. MaxMin diversification on Greedy and Swap running on Flights dataset, $k = 5$

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

4 Max-sum and Max-min diversification

All experiments that have been done are using MaxSum diversification as the diversity function. Whereas, there is another function of diversification which is MaxMin diversification. In this section, both diversification functions will be discussed,

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 5. 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 6 below for more details.

Cont.

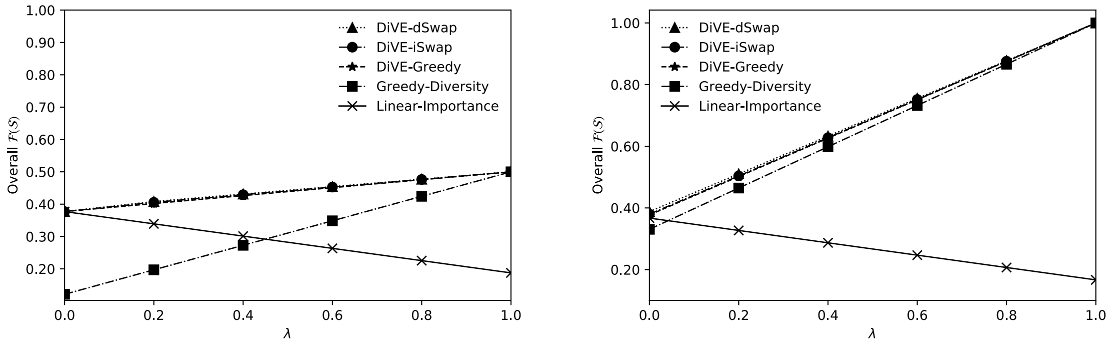


Figure 6: Objective function shape on MaxSum vs. MaxMin diversification while different value of λ, $k = 5$

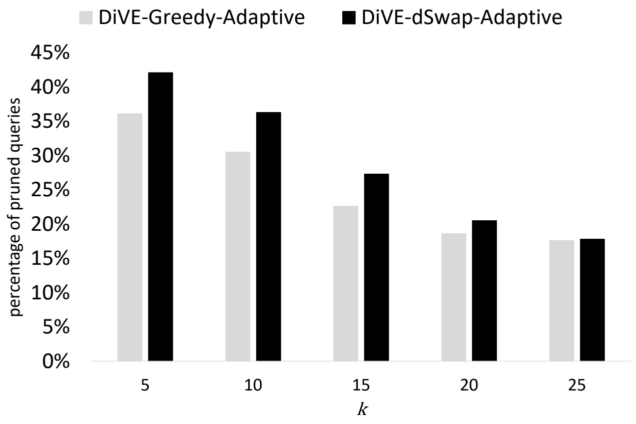


Figure 7: Impact of k to pruning performance, $\lambda = 0.5$, MaxSum diversification, running on Flights dataset

5 Impact of k to pruning performance

It has been discussed in the first section that one of the weekness of DiVE-Greedy is the initialization set. DiVE-Greedy uses only two views as the initialization of set S. This condition may reduce the chance of pruning due to many views in X will have same *setDist* score in the first Greedy iteration. In order to overcome this issue, DiVE-dSwap is proposed. DiVE-dSwap uses swap technique that has bigger number of views in the initialization set S, which is the size of set $S = k$. In this experiment, we use $k = 5$ as the minimum.

To observe the impact of k to pruning performance while the λ is constant. In this experiment, we run DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive on Flights dataset, using 5 queries, $\lambda = 0.5$ and different number of k. Figure 7 shows the result of both schemes.

Overall, the pruning performance is decreasing while k is increasing. This result follows the expectation, while k increase, the number of iteration will be increased. Increasing the number of iterations means more views need to be executed.

Coba kalau k = 2 di Greedy dan Swap bedane opo jal? hehe

Cont.

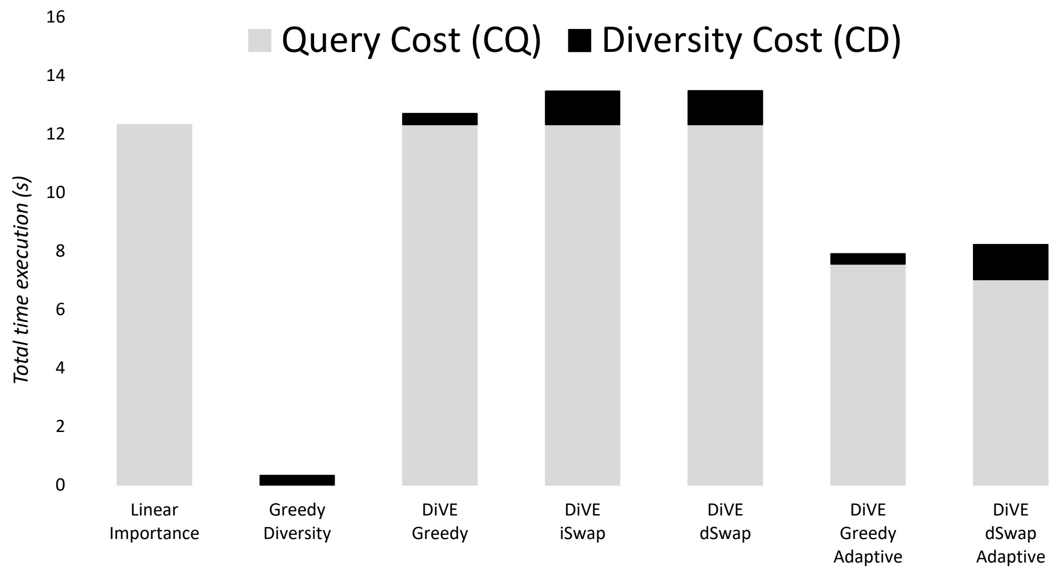


Figure 8: Total costs of schemes running on Flights dataset, MaxSum diversification, $k = 5$, $PI = 0.97$, and $\lambda = 0.5$

6 Total cost with and without pruning

To see the performance of our proposed pruning approach, especially compared to schemes without pruning, Figure 8 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.

7 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 $O\left(k^n\right)$.

The different of total cost between Greedy and Swap approach can be seen in Figure 8. Meanwhile, the impact of k to the distance computation time while running on Flights dataset can bee seen in Figure 9.

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.

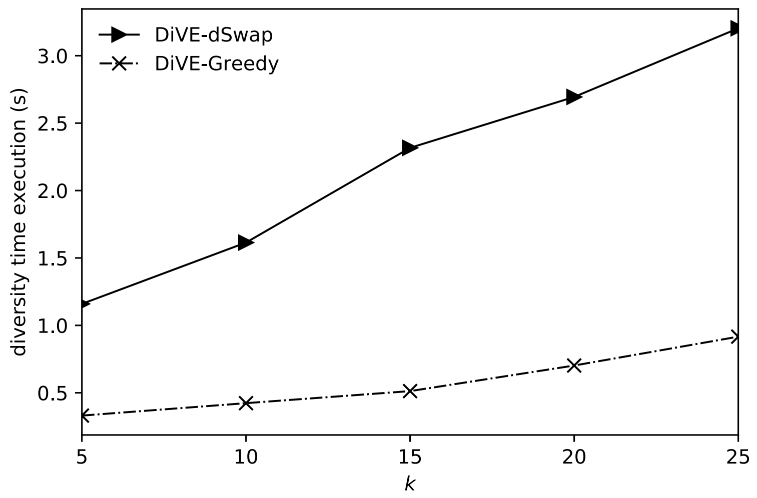


Figure 9: Impact of k to diversity computation running on Flights dataset

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.

The End.