Extended Experiment Report

05 October 2018

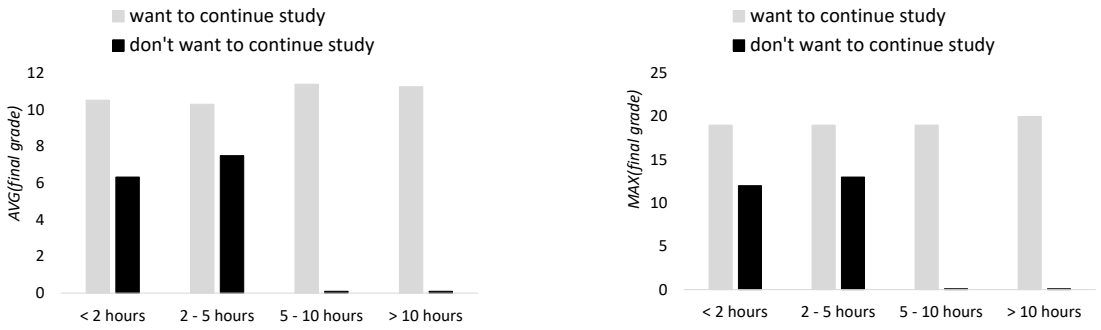

Figure 1: Study time and final grade of students who want to countinue their study to higher education vs. students who do not want to continue their study to higher education

1 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 in this experiments. 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 wants 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 1 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 1 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.

2 Distance Functions

There are several distance functions that can be used to compare two probability distributions as follows:

- Kullback-Leiber (KL) 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, this distance is not bounded as well.
- Kolmogorov Distance has the maximum bound equal to 1. However, this distance generally is used as hypothesis test that need another parameters such as α and confidence interval.

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

Cont.

2.1 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

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

2.2 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:

$\sqrt{\sum (x-y)^2} \leq \sqrt{\sum (x)^2 + \sum (y)^2}$
$\sqrt{\sum (x-y)^2} \leq \sqrt{1+1}$
$\sqrt{\sum (x-y)^2} \leq \sqrt{2}$

3 Max-sum and Max-min diversification

The main objective of diversity is to select a set S from a given set $\mathbf{V}$ in such a way that the diversity among the selected elements in set S is maximized. 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 the objective of MaxMin is to maximize of minimum score of distance among all points 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 2. 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 as shown in Figure 3.

Cont.

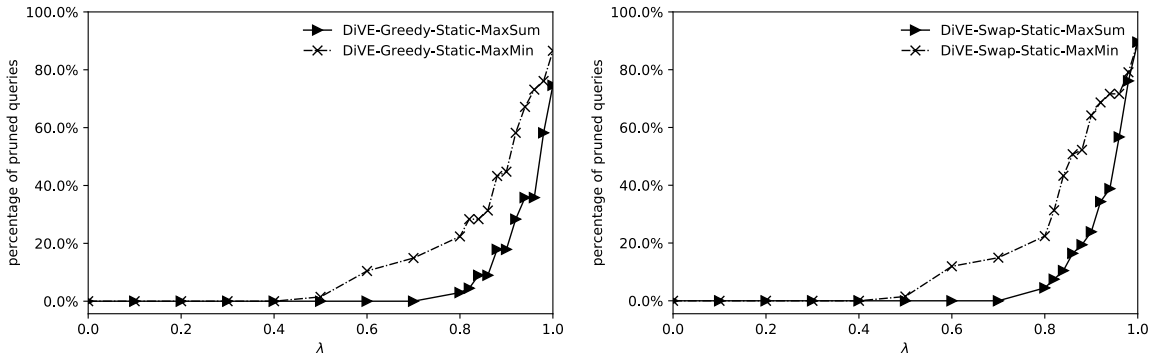

Figure 2: MaxSum vs. MaxMin diversification on Greedy and Swap running on Flights dataset, $k = 5$

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

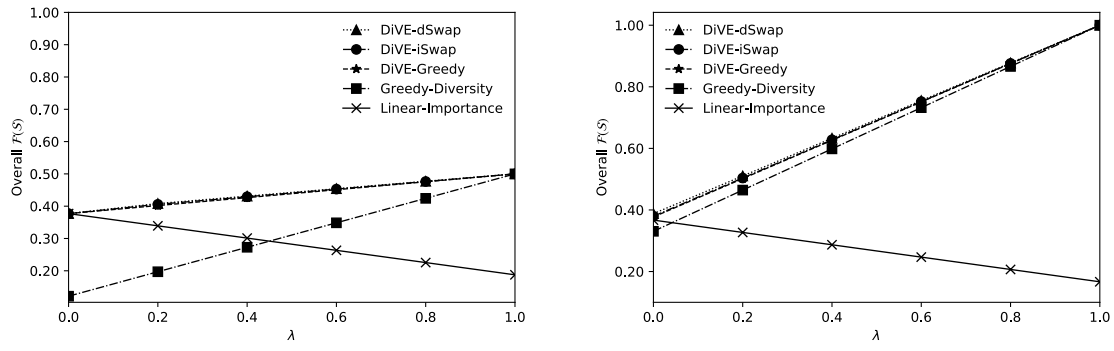

Figure 3: Objective function shape on MaxSum vs. MaxMin diversification while different value of λ, $k = 5$

Cont.

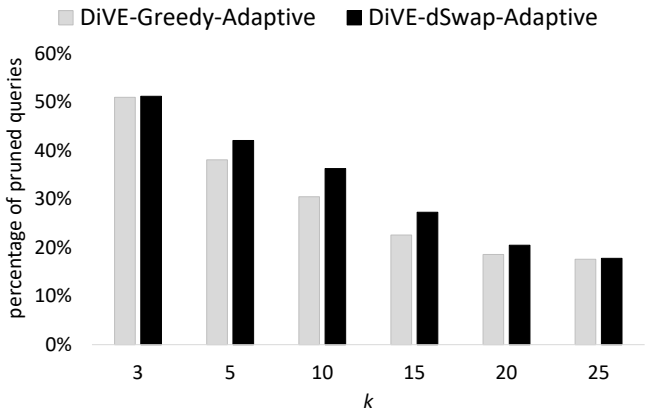

Figure 4: Impact of k to pruning performance, $\lambda = 0.5$, MaxSum diversification, running on Flights dataset

4 Impact of k to pruning performance

To observe the impact of k to pruning performance while the λ is constant, we run DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive on Flights dataset, using 5 queries, $\lambda = 0.5$ and different value of k. Figure 4 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 and less views are pruned.

Moreover, There are some interesting finding from Figure 4. Overall, DiVE-dSwap-Adaptive has better performance compared to DiVE-Greedy-Adaptive in terms of the number of pruned queries. This is because the diversity of $setDist$ score. In case of DiVE-Greedy-Adaptive, no matter how many views inside the set S (i.e., size of k), Greedy always start with two most distant views as the initialization set S. Consequently, there will be many candidate views in X have same $setDist$ score in the first Greedy iteration that may reduce the chance of pruning (i.e., the effect of diversity of $setDist$ score). Meanwhile, DiVE-dSwap-Adaptive has higher number of views in the initialization set S (i.e., in this experiment, we used $k = 5$ as the minimum). Hence, the $setDist$ score on DiVE-dSwap-Adaptive is more diverse than DiVE-Greedy-Adaptive in the first iteration and next iteration as well.

To confirm that, we did experiments that the results can be seen in the Figure 4. Figure 4 shows DiVE-dSwap-Adaptive has better performance compared to DiVE-Greedy-Adaptive. However, DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive have almost same number of pruned queries while the number of k is small such as $k = 3$. While $k = 3$, Greedy has two most distant views in the initialization set S and it only needs one Greedy iteration to generate the recommended views. Meanwhile, Swap has three views as the initialization, it needs to exchange the candidate views in X to set S until the set S has the optimal $F(S)$. In this case, both algorithms will have same condition in terms of $setDist$ calculation (i.e., the distance between each view in X to two views in set S). Hence, the Figure shows both algorithms have same performance while $k = 3$. This results indicate that the diversity of $setDist$ scores affects the performance of pruning.

5 Total cost with and without pruning

To see the performance of our proposed pruning approach, especially compared to schemes without pruning, Figure 5 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.

Cont.

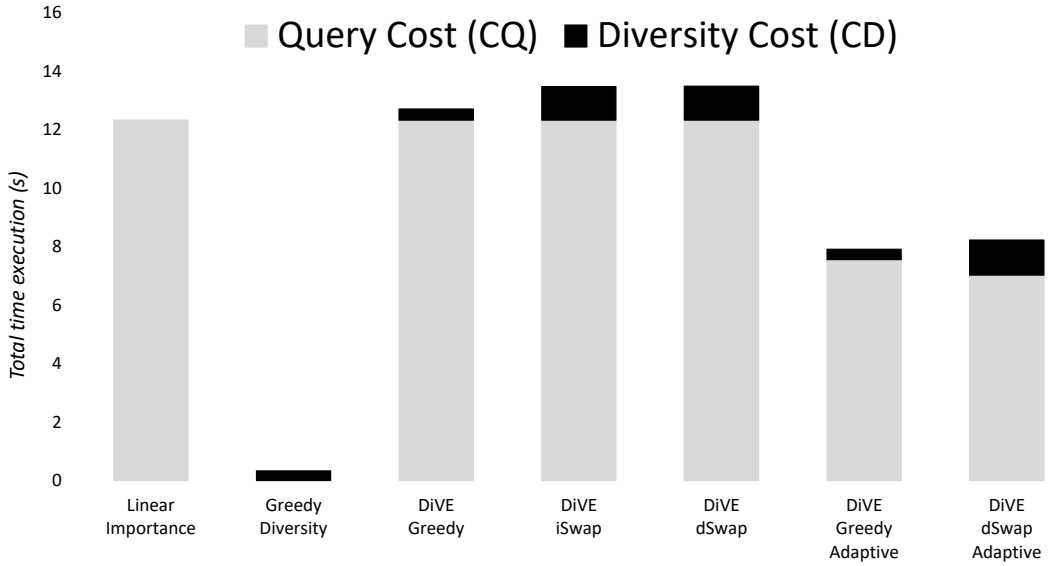

Figure 5: Total costs of schemes running on Flights dataset, MaxSum diversification, $k = 5$, $PI = 0.97$, and $\lambda = 0.5$

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

7 Correcting wrong maximum bound in adaptive pruning scheme

As explained in the first section, DiVE schemes utilize the importance score bound to do pruning. There are two techniques proposed: 1) static bound and 2) adaptive bound. In the static bound, the theoritical maximum bound ($\sqrt{2}$) is used and this bound will not be changed until the end of running. Meanwhile, adaptive used the theoritical maximum bound ($\sqrt{2}$) as the first, then this bound is updated while the maximum bound of actual importace score of views has been known.

In order to know when the bound should be updated, 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 means after 9 views are executed, then the current bound will be updated to the maximum importance score which have seen so far. Generally, 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

Cont.

Algorithm 1: *DiVE* Greedy Pruning Rectifying

Input: Set of views V and result set Size k
Output: Result set $S \geq V$, size S = k

$S \leftarrow$ two most distant views
$X \leftarrow [V \backslash S]$
function `getL(`f,S, X,L`):`
    for X_i *in set* X **do**
        for S_j *in set* S **do**
            $d \leftarrow setDist\,(X_i, S)$
            $X' \leftarrow [X_i, d]$
            $L.append(X')$
    $L \leftarrow sorted\_by\_d(L)$
    return
$maxI, max_b \leftarrow 0, \sqrt{2}$
$rectify \leftarrow False$
$S_{rectify}, L_{rectify} \leftarrow S, getL(S, X)$

while $i < k$ **do**
    if $max_b == \sqrt{2}$ **then**
        $samples \leftarrow get\_samples(PI)$
        $maxI\_S \leftarrow get\_maxI(S)$
        $maxI\_samples \leftarrow get\_maxI(samples)$
        if $maxI\_S > maxI$ **then**
            $maxI \leftarrow maxI\_S$
        if $maxI\_samples > maxI$ **then**
            $maxI \leftarrow maxI\_samples$
        $max_b \leftarrow maxI$
    $S' \leftarrow S$
    $L \leftarrow getL(S, X)$
    for L_i *in* L **do**
        if $F\,(S') < F\,(S \cup X_i, max_b)$ **then**
            $L'.append(L_i)$
        $I \leftarrow get\_I\_score(L'_i)$
        if $F\,(S') < F\,(S \cup X_i, I)$ **then**
            $S' \leftarrow S \cup X_i$
        if $I > max_b$ **then**
            $max_b \leftarrow I$
            $rectify = True$
            $break(Out\ of\ Loop)$
        else
            $rectify = False$
    if *rectify* $==$ *True* **then**
        $S, S' \leftarrow S_{rectify}$
        $L \leftarrow L_{rectify}$
        $i \leftarrow len(S)$
    else
        if $F\,(S') > F\,(S)$ **then**
            $S \leftarrow S'$
        $i = i + 1$
$returnS$

Cont.

Algorithm 2: *DiVE* dSwap Pruning Rectifying

Input: Set of views V and result set Size k
Output: Result set $S \geq V$, size S = k

$S \leftarrow$ Result set of only diversity
$X \leftarrow [V \backslash S]$
function `getL(`f,S, X,L`):`
    for X_i *in set* X **do**
        for S_j *in set* S **do**
            $d \leftarrow setDist\,(X_i, S \backslash S_j)$
            $X' \leftarrow [S_j, X_i, d]$
            $L.append(X')$
    $L \leftarrow sorted\_by\_d(L)$
    return
$F_{current}, maxI, max_b \leftarrow 0, 0, \sqrt{2}$
$improve, rectify \leftarrow True, False$
$S_{rectify}, L_{rectify} \leftarrow S, getL(S, X)$

while *improve = True* **do**
    if $max_b == \sqrt{2}$ **then**
        $samples \leftarrow get\_samples(PI)$
        $maxI\_S \leftarrow get\_maxI(S)$
        $maxI\_samples \leftarrow get\_maxI(samples)$
        if $maxI\_S > maxI$ **then**
            $maxI \leftarrow maxI\_S$
        if $maxI\_samples > maxI$ **then**
            $maxI \leftarrow maxI\_samples$
        $max_b \leftarrow maxI$
    $S' \leftarrow S$
    $L \leftarrow getL(S, X)$
    for L_i *in* L **do**
        if $F\,(S') < F\,(S \backslash S_j \cup X_i, max_b)$ **then**
            $L'.append(L_i)$
        $I \leftarrow get\_I\_score(L'_i)$
        if $F\,(S') < F\,(S \backslash S_j \cup X_i, I)$ **then**
            $S' \leftarrow S \backslash j \cup X_i$
        if $I > max_b$ **then**
            $max_b \leftarrow I$
            $rectify = True$
            $break(Out\ of\ Loop)$
        else
            $rectify = False$
    if *rectify == True* **then**
        $S, S' \leftarrow S_{rectify}$
        $L \leftarrow L_{rectify}$
        $improve \leftarrow True$
    else
        if $F\,(S') > F\,(S)$ **then**
            $S \leftarrow S'$
        if $F\,(S) > F_{current}$ **then**
            $F_{current} \leftarrow F\,(S)$
            $improve \leftarrow True$
        else
            $improve \leftarrow False$
$returnS$

Cont.

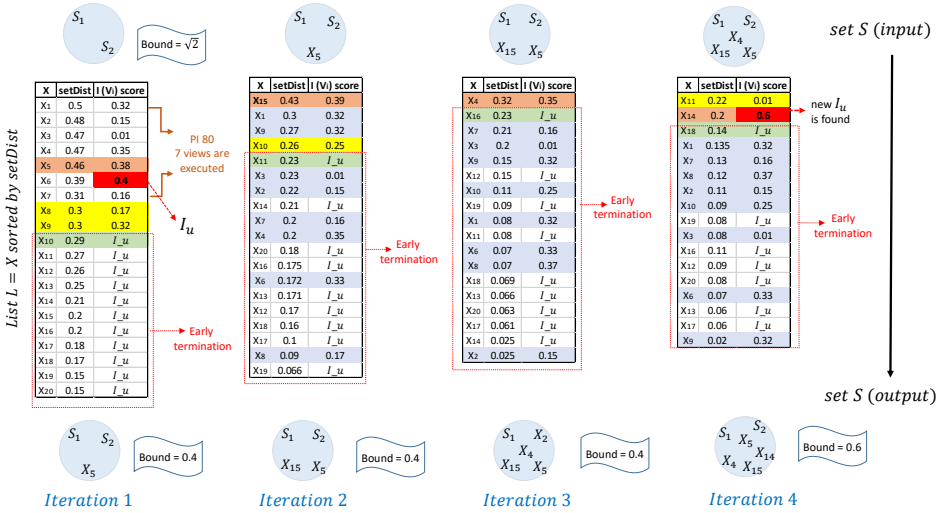

Figure 6: Rectifying flows

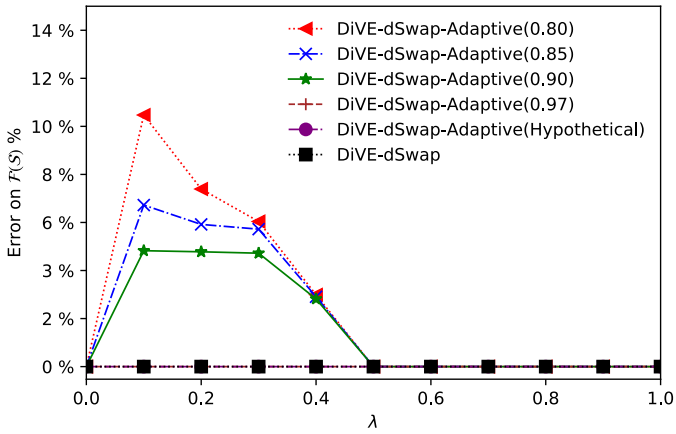

Figure 7: Impact of Adaptive pruning on $F(S)$

Adaptive scheme has the best pruning performance while PI80 is used. However, it reduces the effectiveness of recommended views due to only small number of executed views are needed for PI80 which increase probability to have wrong bound. It can be shown in Figure 7, the error $F(S)$ on Adaptive Pruning. As shown in the Figure, the safest way is to use higher PI such as PI97. However, if there is a way to keep using PI80 without reducing effectiveness, it will definitely be very good. In fact, the goal of pruning scheme is to minimize query view execution (i.e., use low PI) without reducing the quality of recommended views.

In order to overcome this issue, rectifying bound of adaptive pruning is proposed. The algorithm of rectifying bound strategy is quite simple as shown in Figure 6. For instance, there are V_n of number of views in X and user uses PI80. First, the theoritical maximum bound ($\sqrt{2}$) is used such as in the static approach. In this step, some low-quality views may be pruned. Afterward, As shown in Figure 6, nine query views are executed in advanced then the estimated maximum bound is obtained from nine executed views and the views in the initialization set S. The blue dash line shows the first time of maximum bound is changed from the theoritical bound to the estimated maximum bound (i.e., maximum importance score of views have seen so far). In this step, many low-quality views are pruned. The current version of adaptive pruning scheme will consider these pruned views as the final pruned views, however, rectifying bound mechanism will bring back these pruned views while higher importance score is found in the next iteration.

Cont.

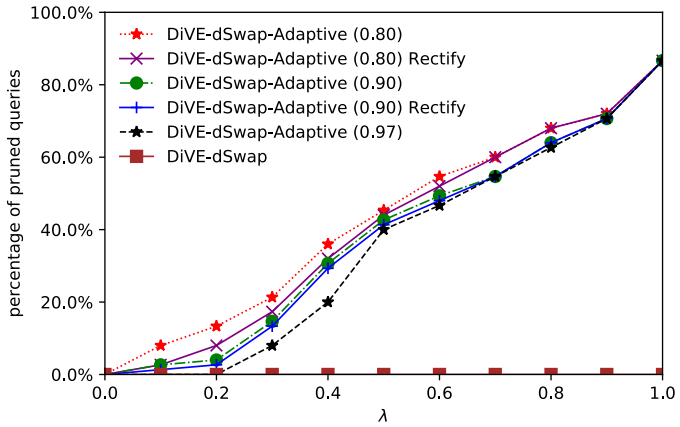

Figure 8: DiVE-dSwap-Adaptive with and without rectifying

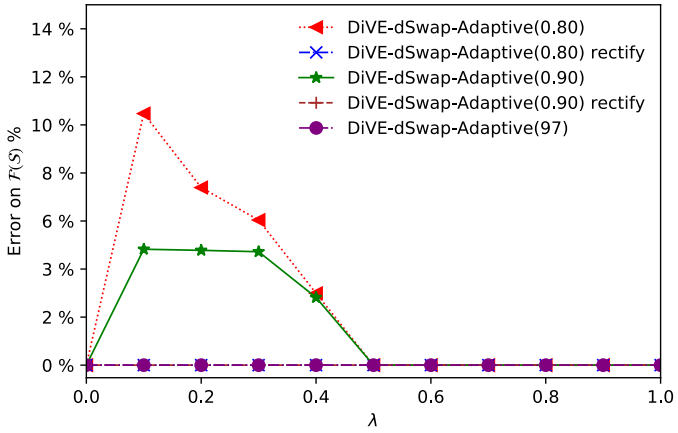

Figure 9: Impact of Adaptive pruning on $F(S)$ with and without rectifying

As shown in the example, the higher importance score are found (i.e., V_{30},V_{55},V_{100}). All pruned views from the previous steps will be returned and evaluated again while the schemes find higher importance score in the next execution. Then, the maximum bound will be updated to the highest importance score. The scheme will repeat this step until no importance score that higher than the current maximum bound.

Figure 8 shows the performance of adaptive pruning scheme with rectifying bound strategy compared to without rectifying bound strategy. The pruning peformance after applying rectifying bound strategy quite close to without rectifying bound strategy. Meanwhile, as shown in Figure 9 there is no effectiveness loss after rectifying bound is implemented.

8 Query sharing computation

Beside using pruning, SeeDB used query sharing optimizaation to reduce the query executions. To do query sharing optimization, SeeDB used techniques such as: 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.

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

Cont.

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.

The End.