Rectifying Bound

20 September 2018

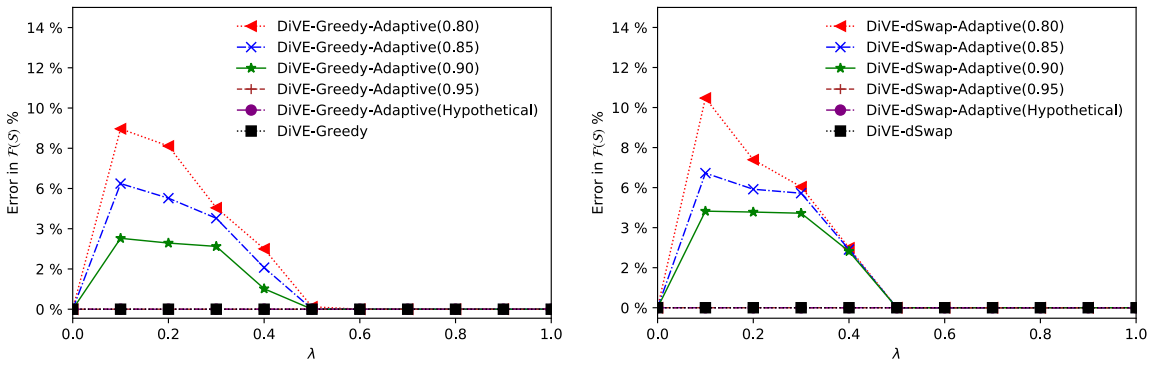

Figure 1: Error $F(S)$ Adaptive Pruning

1 Rectifying wrong upper bound on Adaptive Pruning schemes

In order to reduce costs, our DiVE schemes utilize the importance score bound to do pruning. There are two pruning techniques proposed: 1) static bound approach and 2) adaptive bound approach. In static bound, the theoritical upper bound ($\sqrt{2}$) is used and this bound will not be changed until the end of iteration. Meanwhile, adaptive scheme estimates upper bound based on view queries that have been executed, the bound is updated where there is a higher upper bound found on the next query execution.

In order to know how many samples that need to be executed to estimate the upper bound, samping based on prediction interval is used. Before running the scheme, user needs to define what PI that she wants to use. For instance, while user uses PI 80, it means the current bound will be updated to the maximum importance score which have seen so far of 9 executed views. Generally, PI can be defined as following:

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

Our experiment results show that 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 sample executed views are needed that leads to wrong estimation of upper bound. Figure 1 shows the error in $F(S)$ in different value of PI. The safest way is to use higher PI such as PI95 or PI97 but it needs to execute more views which contradict to our propose to minimizing the cost.

If there is a way to use PI80 without reducing effectiveness, it will definitely very good. In fact, *the goal of our 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 algorithms of our rectifying bound can be seen in algorithm 1 for DiVE-Greedy-Adaptive and 2 for DiVE-dSwap-Adaptive. The detail our rectifying technique is presented below:

1.1 Rectifying upper bound

As mentioned above that static pruning uses theoritical upper bound ($\sqrt{2}$) until the end of iteration and there is no mechanism to update the bound. Hence, the pruning performance is not working optimal due to the theoritical upper bound may very far from the actual upper bound from the dataset (e.g., the actual bound = 0.6). To overcome this issue, adaptive pruning is proposed to estimate the upper bound by executing some sample of views and the upper bound will be updated while in the next execution the higher bound is found. However, adaptive pruning leads to over-prune due to the estimated upper bound may far below the actual bound (e.g., actual bound = 0.6, estimated bound = 0.3). Consequently, it reduces the quality of the recommended views or produces error in $F(S)$. Without rectifying upper bound, the error in $F(S)$ in different value of PI can be seen in Figure 1.

Cont.

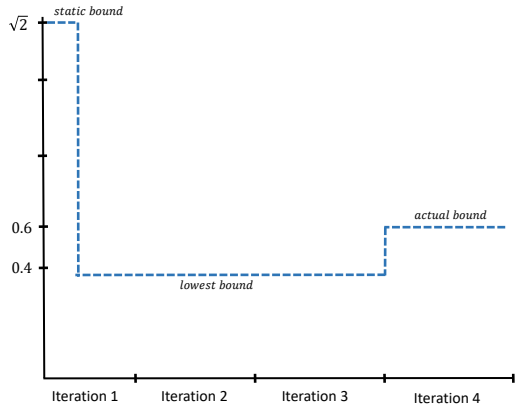

Figure 2: The changing of upper bound in each iteration of DiVE-Greedy-Adaptive

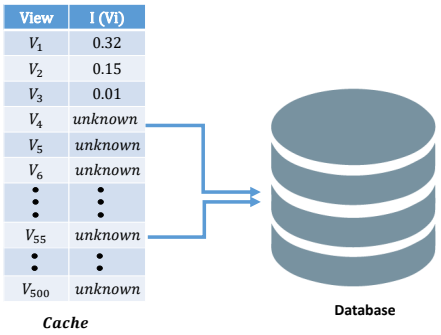

Figure 3: Caching storage of query execution

In order to eliminate error in $F(S)$ and keep the high pruning performance, rectifying upper bound is proposed. The idea behind the rectifying bound technique is to support backtracking. For instance, Figure 2 shows the changing of upper bound in each iteration of DiVE-Greedy-Adaptive. Let assume that user defines $k = 6$, as size of the initialization of set S equal to 2, there will be four times iterations to recommend set S which $k = 6$. Similar to the static pruning approach, as the initialization, $(\sqrt{2})$ is used as the upper bound. The scheme runs with the upper bound is equal to $(\sqrt{2})$ in the first iteration. While certain number sample of views are executed and have been fullfilled the PI condition (e.g., PI80 needs 9 samples of executed views), the upper bound will be updated to the maximum importance score of views have seen so far (e.g., 0.4). As shown in Figure 2 there are no importance score of executed views in the first, second, and third iteration that higher than the current upper bound (0.4). However, in the last iteration (iteration 4), the higher bound is found (e.g., 0.6). In this example, it shows that in the second and third iteration, the scheme runs with wrong upper bound which may leads to over-prune and reduce the effectiveness. Without rectifying, the result are presented to users as the final result. Meanwhile, with rectifying strategy the backtracking is supported. The scheme is able to backtrack to the first iteration and uses the correct upper bound for the next iterations.

In the next section, we explain the detail of our rectifying strategy that consists of two important techniques: 1) Bookkeeping (keep tracking the result in each iteration) and 2) Caching (utilize cache to optimize re-using strategy).

1.1.1 Bookkeeping Technique

In order to support rectifying bound, bookkeeping technique is used. The idea behind this technique is to keep track: 1) the set S as the input in each iteration and 2) list L in each iteration and 3) the position of maximum $setDist$ score in L while it gets early termination. For instance, Figure 4 shows the rectifying algorithm of DiVE-Greedy-Adaptive while $k = 6$. Firstly, there are two most distant views in the set S as the initialization and $\sqrt{2}$ as the upper bound. In each iteration, Greedy add one most optimal view to the set S, the iteration will stop while $k = 6$. The detail of bookkeeping technique in our rectifying strategy as follows.

A list L of all views in X is created such that each X_i is assigned an importance equal to the upper bound I_u and L is sorted based on the diversity score of each view X_i to the current set S as shown in Figure 4. The goal for DiVE-Greedy is to find the view with the highest utility $U(H)$. As described in utility function equation, such utility score $U(X_i)$ is a weighted sum of two measures: 1) the importance score of X_i (i.e., $I(X_i)$), and 2) the distance of X_i from S (i.e., $setDist\,(X_i, S)$). Then, the highest utility $U(H)$ is initialized to a default value of 0.0, and the list L is traversed in order. For each visited view X_i, the upper bound on the utility achieved by X_i (i.e., $maxU(X_i)$) is computed using its actual diversity score and the upper bound

Cont.

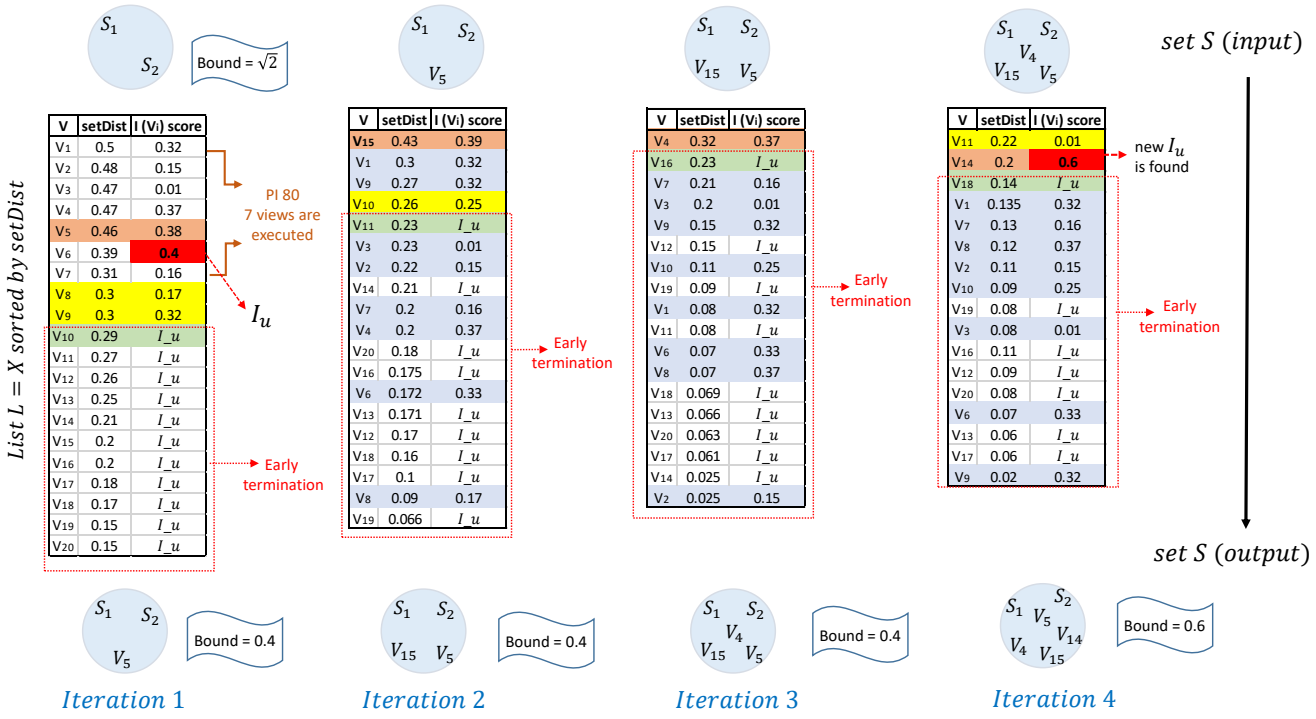

Figure 4: Rectifying upper bound in DiVE-Greedy-Adaptive while $k = 6$

on its importance. If $maxU(X_i) > U(H)$, then X_i is generated and its actual utility $U(X_i)$ is calculated. Accordingly, if $U(X_i) > U(H)$, then $U(H)$ is set to be equal to $U(X_i)$. However, if $maxU(X_i) < U(H)$, then early termination is reached.

Let start from the first iteration, for instance, $PI80$ is used which needs 9 sample executed views. There are two views in set S, it needs 7 more views to be executed to update the upper bound I_u as shown in Figure 4 (e.g., $V_1 - V_7$ in Iteration 1). Let assume that from the executed views, 0.4 is the higest importance score, then I_u is updated from $\sqrt{2}$ to 0.4. After I_u is updated to 0.4, early termination is reached on V_{10}. Hence, V_{10} and all views below of V_{10} can be ignored. Meanwhile, all views above V_{10} (e.g., V_8, V_9) need to be executed because these views satisfy the condition $maxU(X_i) > U(H)$. As shown in this Figure, the highest importance score of all executed views still 0.4 and there are no more views in Iteration 1 that need to be executed. Finally, view with the highest utility $U(H)$ is added to set S (e.g., V_5).

In the next iteration, the size of set S is increased because one view is added in each iteration. A new list L is created and $setDist$ score is recalculated. Similar to the first iteration, list L is sorted based on $setDist$ score. Figure 4 shows that in the second and third iterations, there are no importance score of executed views that higher than 0.4. Hence, there is no updating the upper bound in the second and third iterations. However, the higher upper bound is found in forth iteration (e.g., 0.6). In this condition, we realize that the scheme used wrong upper bound in the previous iterations that may lead to over-prune and reduce the quality of recommended views.

In order to fix the wrong bound in the previous iterations, bookkeeping startegy is used in this approach. In each iteration, list L, set S (S as the input), and view which early termination is started (e.g., $V_{10}, V_{11}, V_{16}, V_{18}$ in Iteration 1,2,3,4 respectively) are stored. The position of view when early termination started is important because it needs to be evaluated using the correct upper bound.

According to the example in Figure 4, the higher upper bound is found in the forth iteration. Hence, upper bound which used in the previous iterations is wrong. In order to fix it, the scheme is able to go back and start from the previous iteration using L and S of the previous iteration which are stored before. This

Cont.

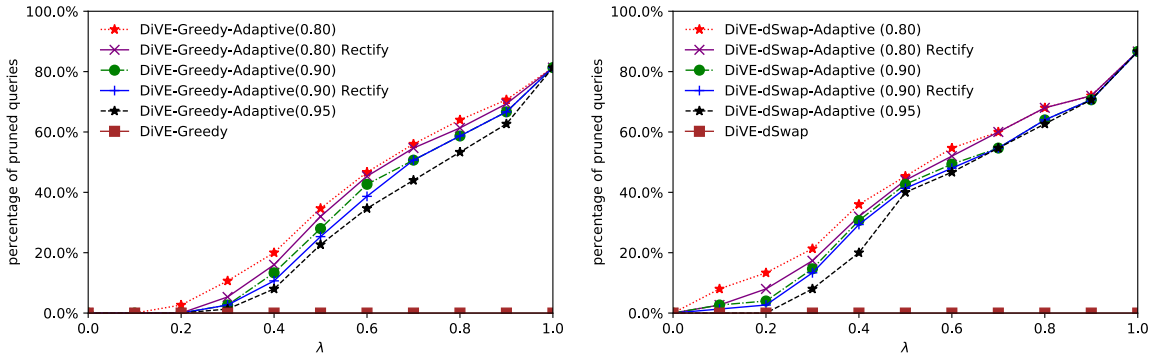

Figure 5: Rectifying bound DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive

bookeeping technique can be seen in Algorithm 1 for DiVE-Greedy-Adaptive (e.g., line 30 - 34) and 2 (e.g., line 32 - 37) for DiVE-dSwap-Adaptive.

Rectifying bound needs forward and backward looping that may leads to higher costs. To overcome this issue, we propose Caching mechanism which can reduce the cost by reusing strategy as presented in the next section.

1.1.2 Caching Technique

The results of this rectifying bound strategy can be seen in Figure 5. The results are average from five queries. These Figures show 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 6 there is no effectiveness loss after rectifying bound is implemented especially for PI80. Moreover, the costs comparison of our rectifying algorithm is also presented in Figure 7. This Figure shows the total cost of our pruning scheme with and without rectifying bound strategy which running on Flights dataset.

The results of this rectifying bound strategy can be seen in Figure 5. The results are average from five queries. These Figures show 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 6 there is no effectiveness loss after rectifying bound is implemented especially for PI80. Moreover, the costs comparison of our rectifying algorithm is also presented in Figure 7. This Figure shows the total cost of our pruning scheme with and without rectifying bound strategy which running on Flights dataset.

2 Experiment Results

The results of this rectifying bound strategy can be seen in Figure 5. The results are average from five queries. These Figures show 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 6 there is no effectiveness loss after rectifying bound is implemented especially for PI80. Moreover, the costs comparison of our rectifying algorithm is also presented in Figure 7. This Figure shows the total cost of our pruning scheme with and without rectifying bound strategy which running on Flights dataset.

Cont.

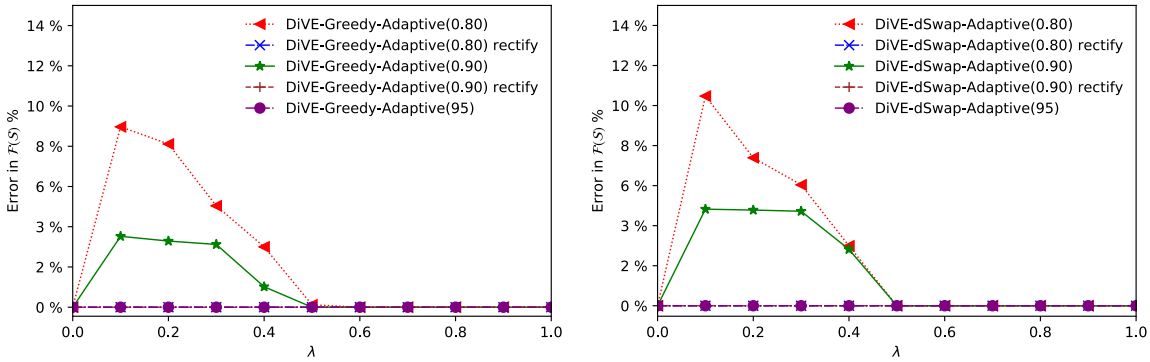

Figure 6: Error in $F(S)$ DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive after rectifying upper bound

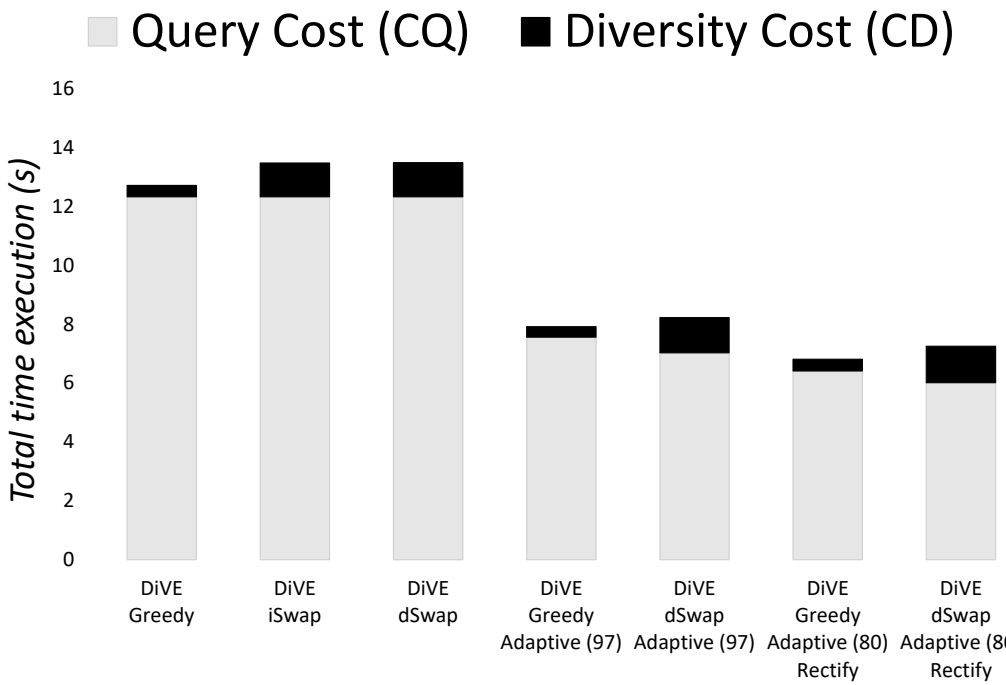

Figure 7: Total costs of schemes running on Flights dataset, $k = 5$, and $\lambda = 0.5$

Cont.

Algorithm 1: *DiVE* Greedy Pruning Rectifying

Input: Set of views V and result set Size k
Output: Result set $S \leq V$, 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**
      $d \leftarrow setDist\,(X_i, S)$
      $L.append([X_i, d])$
  $L \leftarrow sorted\_by\_d(L)$
  return L
$max_b \leftarrow getMaxPI(L)$
$rectify \leftarrow False$

while $i < k$ **do**
  if $rectify = False$ **then**
    $L \leftarrow getL(S, X)$
    $S' \leftarrow S \cup L[X_1]$
  for L_i *in* L **do**
    if $rectify = True$ **then**
      start loop at $L[min_d]$
    if $F\,(S') < F\,(S \cup X_i, max_b)$ **then**
      $I \leftarrow get\_I\_score(X_i)$
      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**
    $G \leftarrow fetchTempResult(i-2)$
    $S, S' \leftarrow G[S], G[S']$
    $L \leftarrow G[L]$
    $i = i - 2$
  else
    $storeTempResult(i, S, S', L, min_d)$
    $S \leftarrow S'$
    $i = i + 1$
return S

Cont.

Algorithm 2: *DiVE* dSwap Pruning Rectifying

Input: Set of views V and result set Size k
Output: Result set $S \leq V$, 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**
      $d \leftarrow setDist\,(X_i, S)$
      $L.append([S_j, X_i, d])$
  $L \leftarrow sorted\_by\_d(L)$
  return L
$F_{current}, counter \leftarrow 0, 0$
$improve, rectify \leftarrow True, False$
$max_b \leftarrow getMaxPI(L)$

while $improve = True$ **do**
  $counter = counter + 1$
  if $rectify = False$ **then**
    $L \leftarrow getL(S, X)$
    $S' \leftarrow S$
  for L_i *in* L **do**
    if $rectify = True$ **then**
      start loop at $L[min_d]$
    if $F\,(S') < F\,(S \backslash S_j \cup X_i, max_b)$ **then**
      $I \leftarrow get\_I\_score(X_i)$
      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**
    $G \leftarrow fetchTempResult(counter = 1)$
    $S, S' \leftarrow G[S], G[S']$
    $L \leftarrow G[L]$
    $counter = 0$
    $improve \leftarrow True$
  else
    $storeTempResult(counter, S, S', L, min_d)$
    if $F\,(S') > F\,(S)$ **then**
      $S \leftarrow S'$
    if $F\,(S) > F_{current}$ **then**
      $F_{current} \leftarrow F\,(S)$
      $improve \leftarrow True$
    else
      $improve \leftarrow False$
return S

The End.