Rectifying Bound

20 August 2018

1 Correcting wrong maximum bound in 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 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 estimation of maximum bound as the first, then this bound is updated where there is a higher maximum bound found.

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, it means after 9 views are executed, the current bound will be updated to the maximum importance score which have seen so far. In the algorithm 1 and 2, $getMaxPI(S, X)$ is the function to get the estimated maximum bound from some number of executed views based on PI. 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

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 executed views are needed for PI80. It causes the increasing of probability to have wrong bound. The safest way is to use higher PI such as PI95 or 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 algorithms of our rectifying bound can be seen in algorithm 1 for DiVE-Greedy-Adaptive and 2 for DiVE-dSwap-Adaptive.

The idea behind the rectifying bound algorithm is to keep track: 1) the maximum of $setDist$ score position in L while it gets early termination, 2) the set S (i.e., the initial set S while iteration start) and 3) S' (i.e., the set S in the end of iteration) in each Greedy and Swap iteration. User can determine the *step_back* parameter in case the size of k is large. For instance, while user uses $k = 20$ and *step_back* $= 3$, if there is a higher bound in the 14^{th} iteration, the algorithm will go to 11^{st} iteration and rectify the bound. The L, S, and S' of 11^{st} iteration will be used, the new maximum bound will be used for the remaining views in L (i.e., unexecuted views due to early termination) and if there is a view that can improve the $F(S)$ compared to the previous result then the new result will be used. However, in case of small number of k, such as $k = 5$, there will be default *step_back* which is revert to the first iteration. For instance, Greedy starts with two views in set S, while $k = 5$ means there are only three iterations to generate the result. While in the middle of Greedy running, there is a higher bound then the iterations can be start again from size $S = 2$.

The results of this rectifying bound strategy can be seen in Figure 1 and Figure 2. Figure 1 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 2 there is no effectiveness loss after rectifying bound is implemented. Moreover, the costs comparison of our rectifying algorithm is presented as weel in Figure 3. This Figure shows the total cost of our pruning scheme with and without rectifying bound strategy which running on Flights dataset.

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
$L_{base}, S_{base}, S'_{base} \leftarrow getL(S, X), S, S \cup L[X_1]$
$max_b \leftarrow getMaxPI(S, X)$
$rectify \leftarrow False$

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**
    $S, S' \leftarrow S_{base}, S'_{base}$
    $L \leftarrow L_{base}$
    $i \leftarrow 2$
  else
    $storeTempResult(i, S, S', L, min_d)$
    $S \leftarrow S'$
    $i = i + 1$
return S

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 ← Result set of only diversity
X ← [V\S]
function getL(f,S, X,L):
    for X_i in set X do
        for S_j in set S do
            d ← setDist(X_i, S)
            L.append([S_j, X_i, d])
    L ← sorted_by_d(L)
    return L
F_current, counter ← 0, 0
improve, rectify ← True, False
S_base, L_base ← S, getL(S, X)
max_b ← getMaxPI(S, X)

while improve = True do
    counter = counter + 1
    if rectify = False then
        L ← getL(S, X)
        S' ← S
    for L_i in L do
        if rectify = True then
            start loop at L[min_d]
        if F(S') < F(S\S_j ∪ X_i, max_b) then
            I ← get_I_score(X_i)
            if F(S') < F(S\S_j ∪ X_i, I) then
                S' ← S\j ∪ X_i
            if I > max_b then
                max_b ← I
                rectify = True
                break(Out of Loop)
            else
                rectify = False
    if rectify == True then
        S, S' ← S_base
        L ← L_base
        counter = 0
        improve ← True
    else
        storeTempResult(counter, S, S', L, min_d)
        if F(S') > F(S) then
            S ← S'
        if F(S) > F_current then
            F_current ← F(S)
            improve ← True
        else
            improve ← False
return S
```

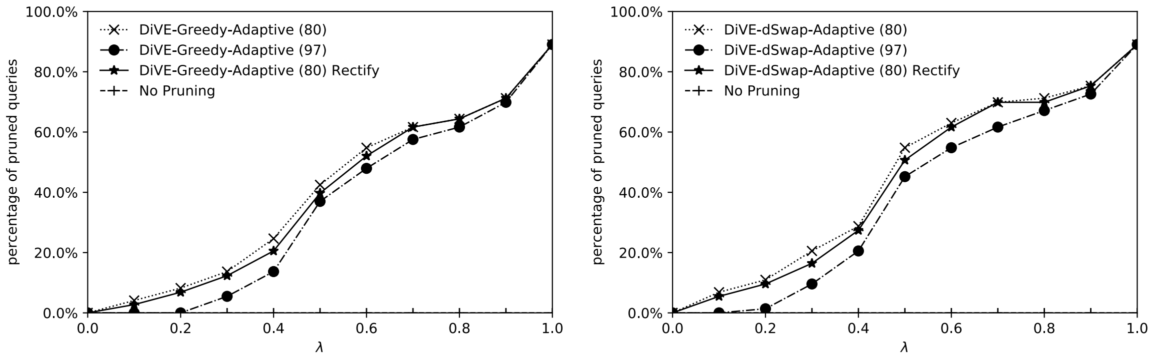

Figure 1: Pruning performance of rectifying bound schemes compared to others

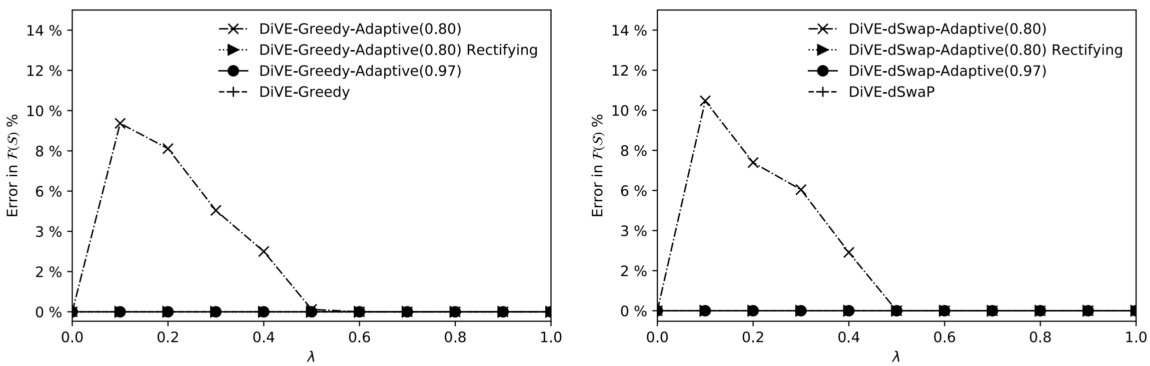

Figure 2: Error $F(S)$ after rectifying bound

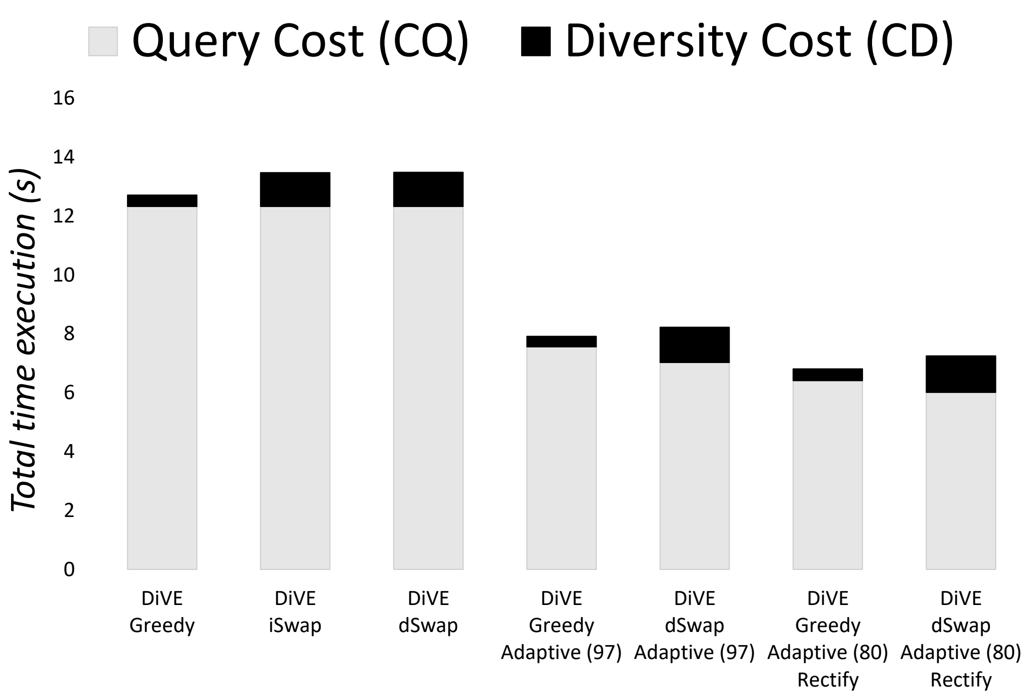

Figure 3: Total costs of schemes running on Flights dataset, $k = 5$, and $\lambda = 0.5$

The End.