Rectifying Bound

22 August 2018

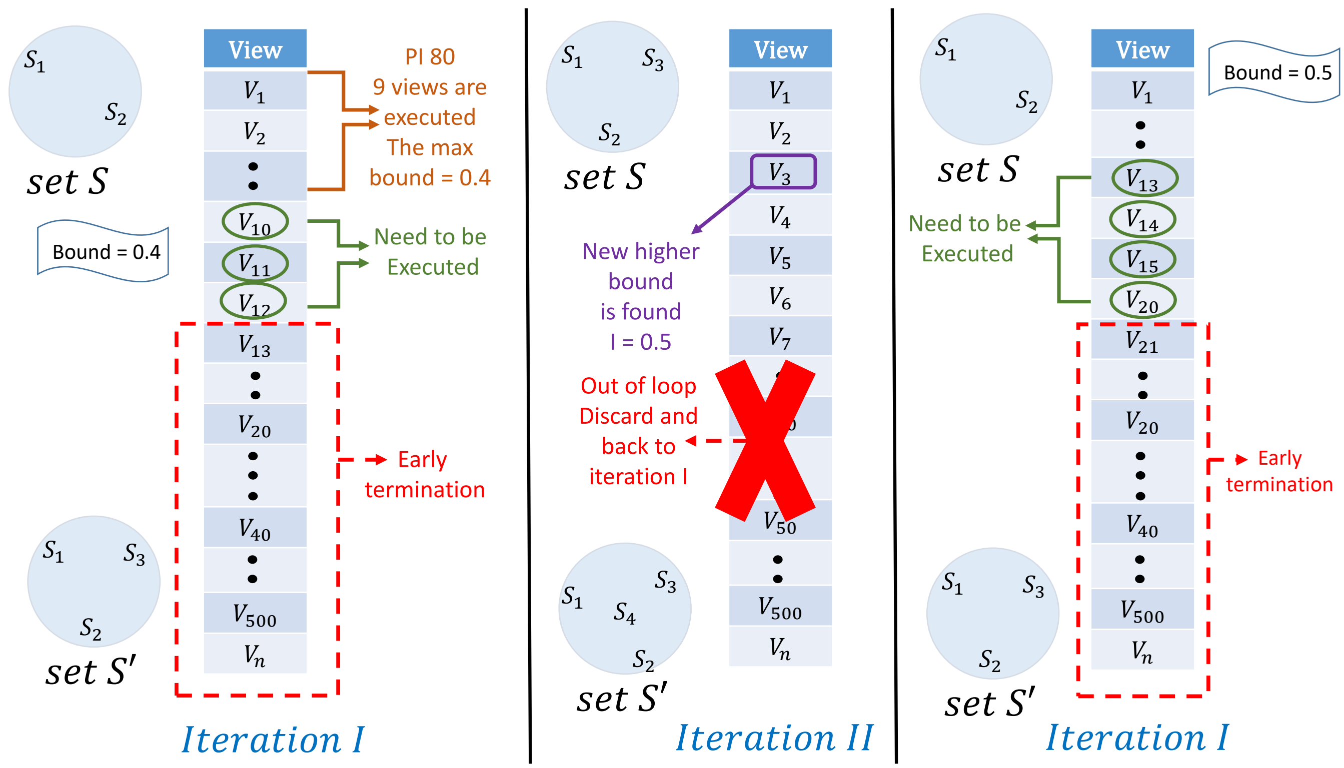

Figure 1: Rectifying maximum bound in DiVE-Greedy-Adaptive while $k = 5$

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, $getMaxPI(L)$ 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.

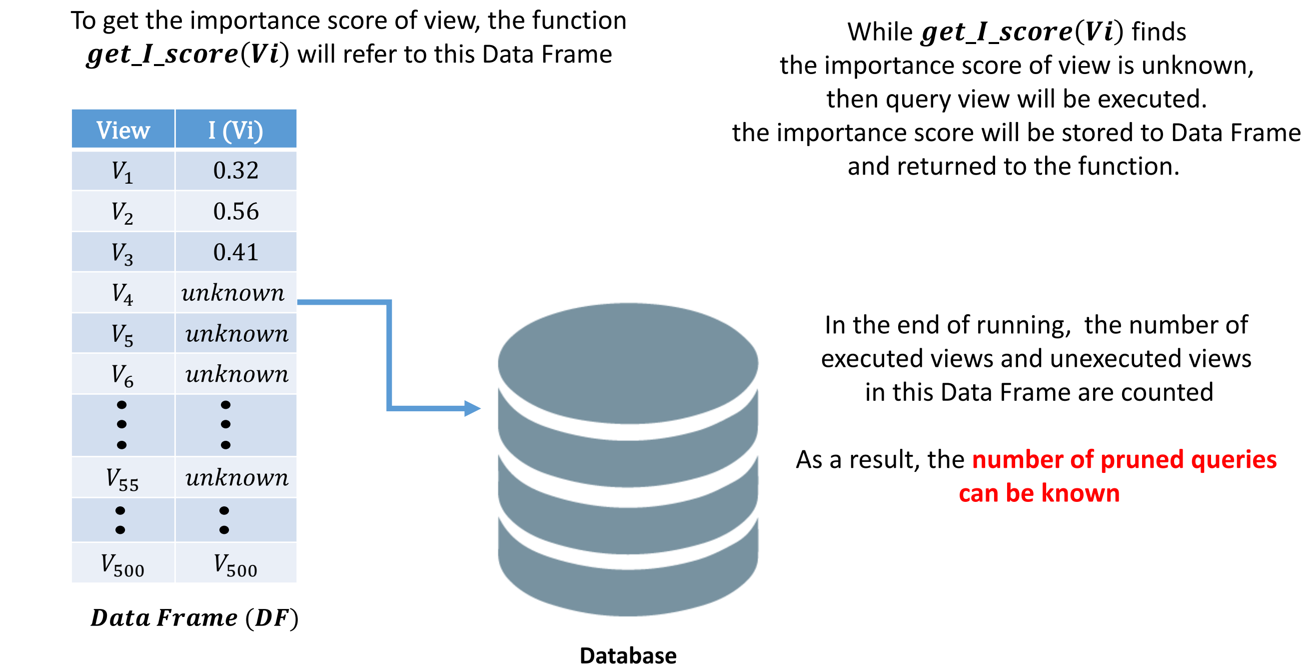

Figure 2: get_I_score function

The idea behind the rectifying bound algorithm is to keep track: 1) the position of maximum *setDist* score 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. For instance, Figure 1 shows the rectifying algorithm of DiVE-Greedy-Adaptive while $k = 5$. Firstly, there are two most distant views in the set S as the initialization. The first iteration of Greedy is to add one most optimal view to the set S, while in the end of the first iteration, size of $S = 3$. As shown in the Figure, some views are executed in advanced (i.e., $V_1 - V_9$) due to PI80 is currently used (Algorithm 1, line 11). The one of view from 9 executed views which has the highest improvement to set S will be added to set S (i.e., $S \cup V_1$), this $F(S \cup V_1)$ will be the current $F(S)$. Moreover, the max_b is obtained from the highest importance score of these executed views, e.g., $max_b = 0.4$.

Notice that up to this step, the current $F(S)$ and max_b are known. The next step is to calculate $maxF(S \cup V_i)$ using actual diversity score ($setDist(V_i, S)$) and upper bound of importance score (max_b). If $maxF(S \cup X_i) > F(S)$, then the X_i can "potentially" improve $F(S)$. However, if $maxF(S \cup X_i) < F(S)$, then X_i can be "pruned". As shown in the *Iteration I*, three views (V_{10}, V_{11}, V_{12}) have $maxF > F$. Hence, these three views need to be executed. Otherwise, views below V_{13} can be ignored (i.e., early termination start on V_{13}). In the end of the *iteration I*, the result set S' will be generated which consists of three views (set $S \cup X_i$). Moreover, in the end of each iteration, S, S', L, max_d, and *iteration count* are stored, max_d is the maximum view while it gets early termination (i.e., V_{13}).

While in the next iteration (i.e., *Iteration II*), there is a higher bound compared to the current max_b, then the current max_b will be updated to the new one and the loop will be stoped. Afterwards, the *Iteration I* will be visited again and before that S, S', L are fetched from the previous result of *iteration I*. As shown in the Figure, after revisit the *iteration I* with the new max_b, there are some views that need to be executed (i.e., $V_{13}, V_{14}, V_{15}, V_{20}$) and the early termination start on V_{21}. If in this revisit, there is a view that can improve the S' from the previous result then $F(S)$ will be updated, otherwise, $F(S)$ is still same and go to the next iteration with the current max_b.

Cont.

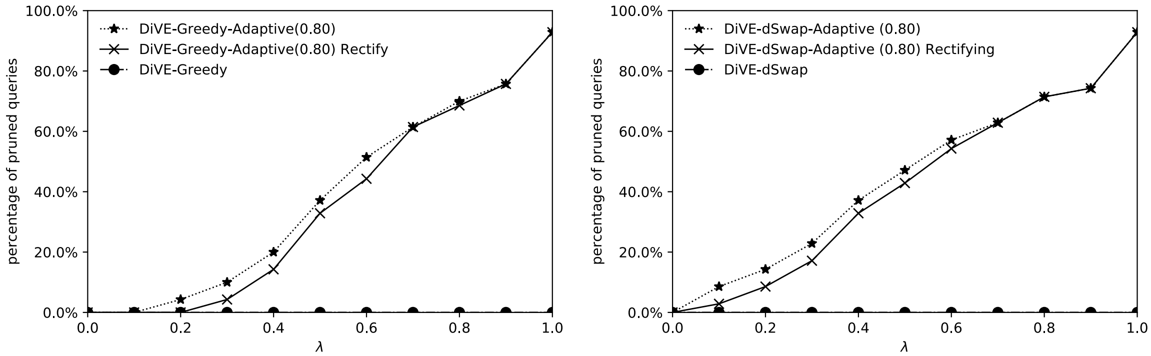

Figure 3: Rectifying vs. No Rectifiying of adaptive scheme (PI80)

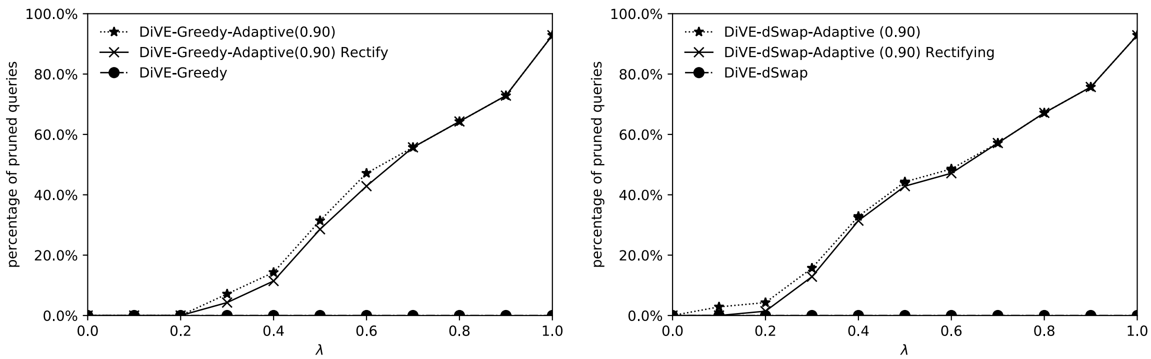

Figure 4: Rectifying vs. No Rectifiying of adaptive scheme (PI90)

In the end of running, the number of executed and unexecuted queries are calculated. This algorithm uses book keeping strategy as shown in Figure 2.

The results of this rectifying bound strategy can be seen in Figure 3 to 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 presented as weel in Figure 7. This Figure shows the total cost of our pruning scheme with and without rectifying bound strategy which running on Flights dataset.

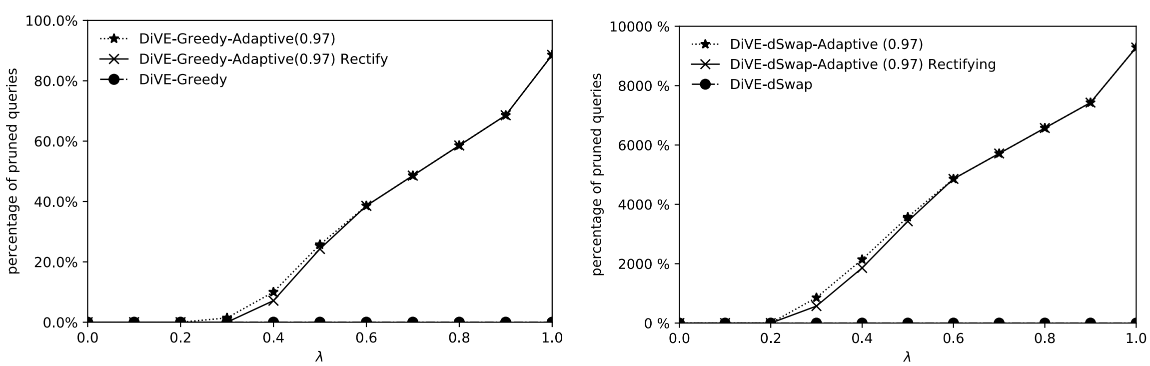

Figure 5: Rectifying vs. No Rectifiying of adaptive scheme (PI97)

Algorithm 1: *DiVE* Greedy Pruning Rectifying

```
Input: Set of views V and result set Size k
Output: Result set S ≤ V, size S = k
S ← two most distant views
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([X_i, d])
    L ← sorted_by_d(L)
    return L
max_b ← getMaxPI(L)
rectify ← False

while i < k do
    if rectify = False then
        L ← getL(S, X)
        S' ← S ∪ L[X_1]
    for L_i in L do
        if rectify = True then
            start loop at L[min_d]
        if F (S') < F (S ∪ X_i, max_b) then
            I ← get_I_score(X_i)
            if F (S') < F (S ∪ X_i, I) then
                S' ← S ∪ X_i
            if I > max_b then
                max_b ← I
                rectify = True
                break(Out of Loop)
            else
                rectify = False
    if rectify == True then
        G ← fetchTempResult(i − 2)
        S, S' ← G[S], G[S']
        L ← G[L]
        i = i − 2
    else
        storeTempResult(i, S, S', L, min_d)
        S ← 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
max_b ← getMaxPI(L)

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
        G ← fetchTempResult(counter = 1)
        S, S' ← G[S], G[S']
        L ← G[L]
        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
```

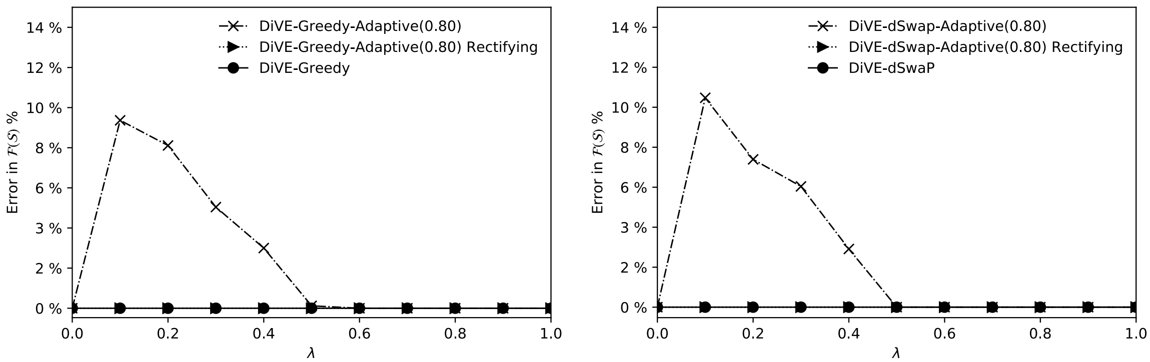

Figure 6: Error $F(S)$ after rectifying bound

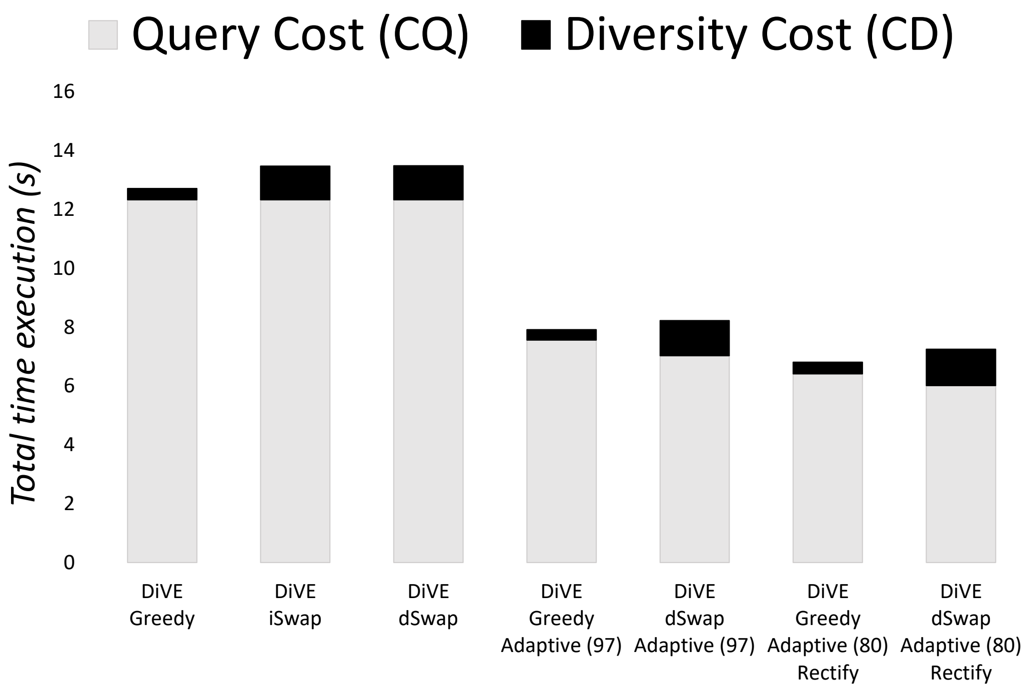

Figure 7: Total costs of schemes running on Flights dataset, $k = 5$, and $\lambda = 0.5$

The End.