Rectifying Bound

20 August 2018

1 Correcting wrong maximum bound in adaptive pruning scheme

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 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 estimation of maximum bound as the first, then this bound is updated while 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 means after 9 views are executed, then 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 which increase 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 results of this rectifying bound strategy can be seen in Figure 2 and Figure 3. Figure 2 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 3 there is no effectiveness loss after rectifying bound is implemented.

2 Total cost with and without Rectifying mechanism

In the previous report, there is Figure that shows the performance of our proposed pruning approach, especially compared to schemes without pruning. In this report, Figure 1 shows the total cost of our pruning scheme with and without rectifying bound strategy running on Flights dataset.

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
L_base, S_base, S'_base ← getL(S, X), S, S ∪ L[X_1]
max_b ← getMaxPI(S, X)
rectify, step_back ← False, 3

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
        if step_back < i − 2 then
            G ← fetchTempResult(i − step_back)
            S, S' ← G[S], G[S']
            L ← G[L]
            i = i − step_back
        else
            S, S' ← S_base, S'_base
            L ← L_base
            i ← 2
    else
        storeTempResult(i, S, S', L, min_d)
        S ← 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 ≤ 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, step_back ← 0, 0, 3
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
        if step_back < counter then
            G ← fetchTempResult(counter − step_back)
            S, S' ← G[S], G[S']
            L ← G[L]
        else
            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
```

Cont.

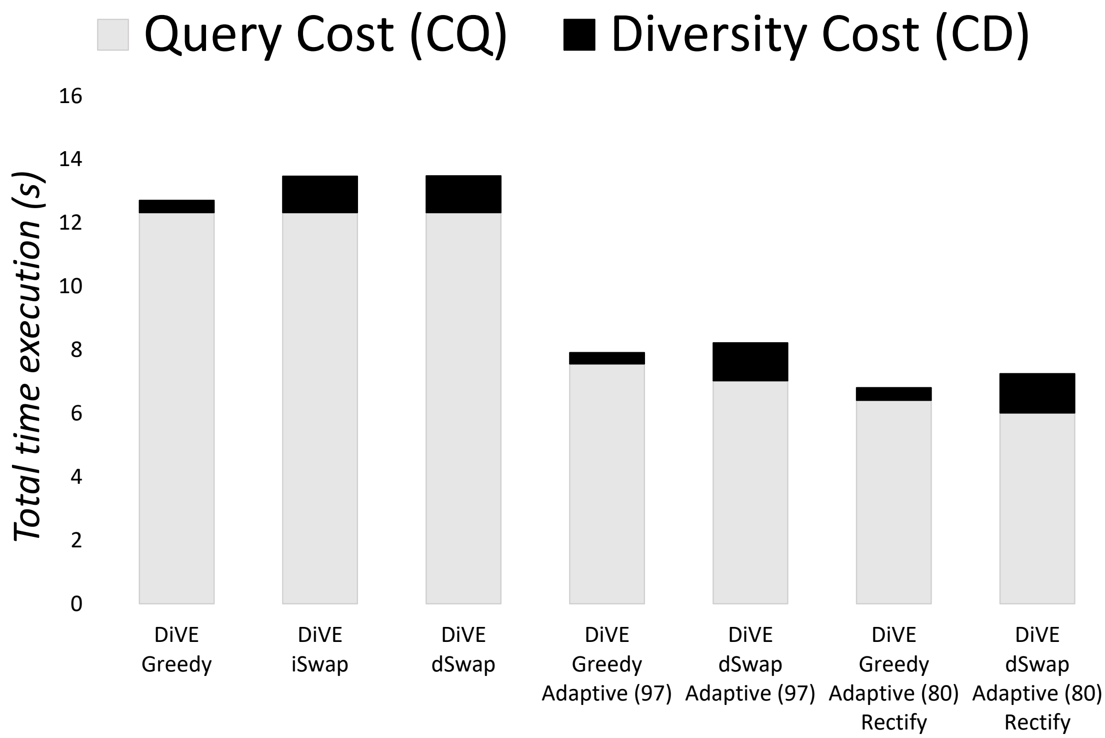

Figure 1: Total costs of schemes running on Flights dataset, $k = 5$, and $\lambda = 0.5$

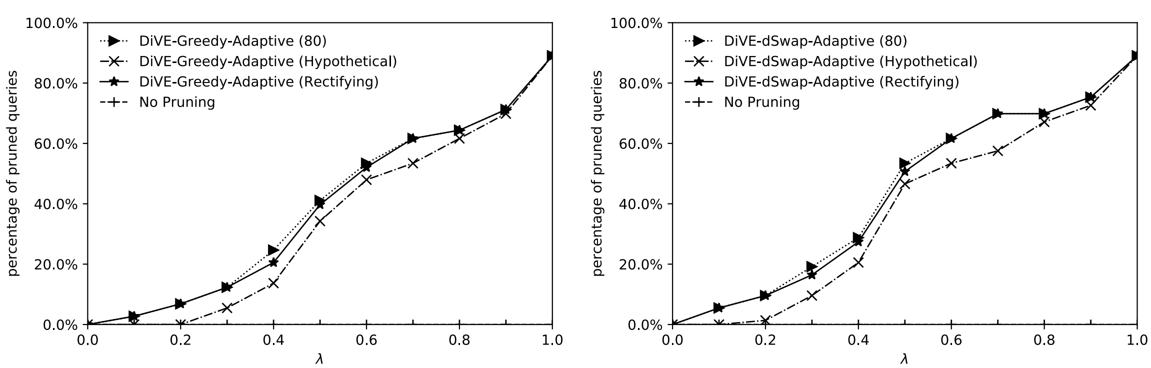

Figure 2: Pruning performance of rectifying bound schemes compared to others

Cont.

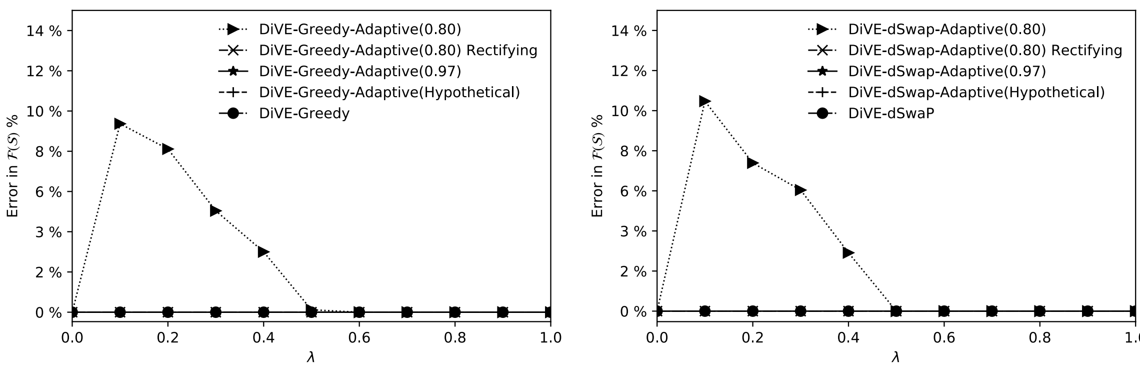

Figure 3: Error $F(S)$ after rectifying bound

The End.