Rectifying Bound

20 September 2018

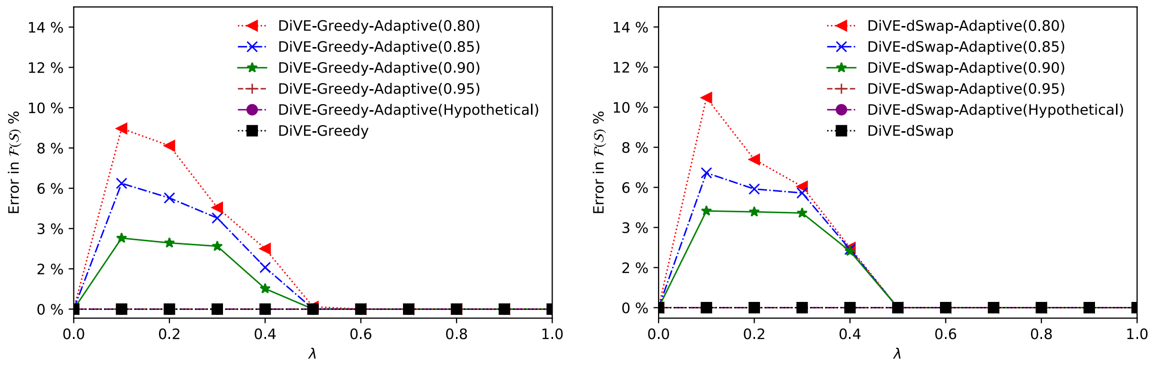


Figure 1: Error $F(S)$ Adaptive Pruning

1 Rectifying wrong upper bound on Adaptive Pruning schemes

In order to reduce cost, 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, theoritical upper bound ($\sqrt{2}$) is used and this bound will not be updated until the end of iteration. Meanwhile, adaptive scheme estimates upper bound based on the importance score have seen so far from the sample executed view queries.

In order to know how many samples that need to be executed to estimate the upper bound, samping based on prediction interval (PI) is used. Before running the scheme, user needs to define what PI that she wants to use. For instance, while user uses $PI80$, it needs 9 executed views to updaate the upper bound to the highest importance score of the executed views. Generally, PI can be defined as following:

- $PI85$: 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 our adaptive pruning schemes have 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 purpose 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 schemes is to minimize 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 Algorithm 2 for DiVE-dSwap-Adaptive. The detail of our rectifying strategy is presented below:

1.1 Rectifying upper bound

As mentioned in the previous section, 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 query 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.

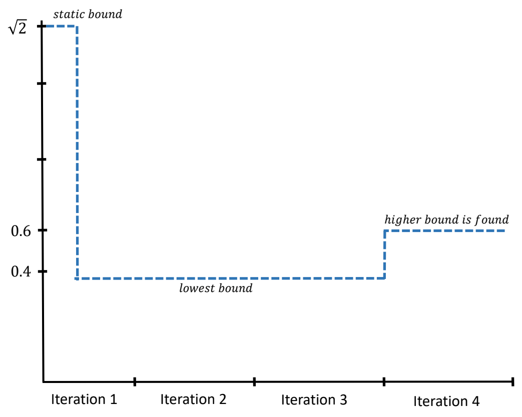


Figure 2: The changing of upper bound in each iteration of DiVE-Greedy-Adaptive

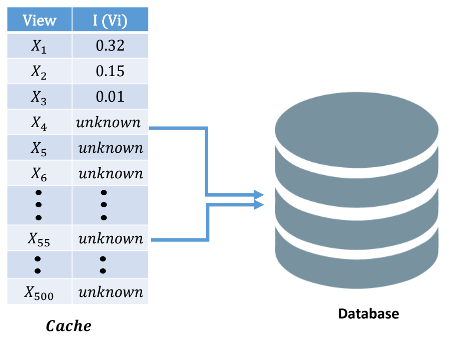


Figure 3: Caching storage of query execution

In order to eliminate error in $F(S)$ and keep the high pruning performance by using low PI (e.g., $PI80$), rectifying upper bound is proposed. The idea behind the rectifying bound is to support backtracking after a higher upper bound is found. 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 with $k = 6$. Similar to the static pruning approach, ($\sqrt{2}$) is used as the upper bound for the first time. In the first iteration, the scheme runs with the upper bound is equal to ($\sqrt{2}$). 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). This upper bound (e.g., 0.4) is used until a new higher bound is found in the next iterations. As shown in Figure 2, there is no importance score of executed views in the first, second, and third iteration that higher than the current upper bound (e.g., 0.4). However, in the last iteration (Iteration 4), a higher upper bound is found (e.g., 0.6). Consequently, the scheme runs with wrong upper bound in the first, second and third iterations, which may leads to over-prune and reduce the effectiveness. Without rectifying, the result are presented directly to users as the final result, even though we know that the scheme uses the wrong bound. Meanwhile, with rectifying strategy, backtracking is supported. The scheme is able to backtrack to the first iteration and uses the correct upper bound for the whole iterations.

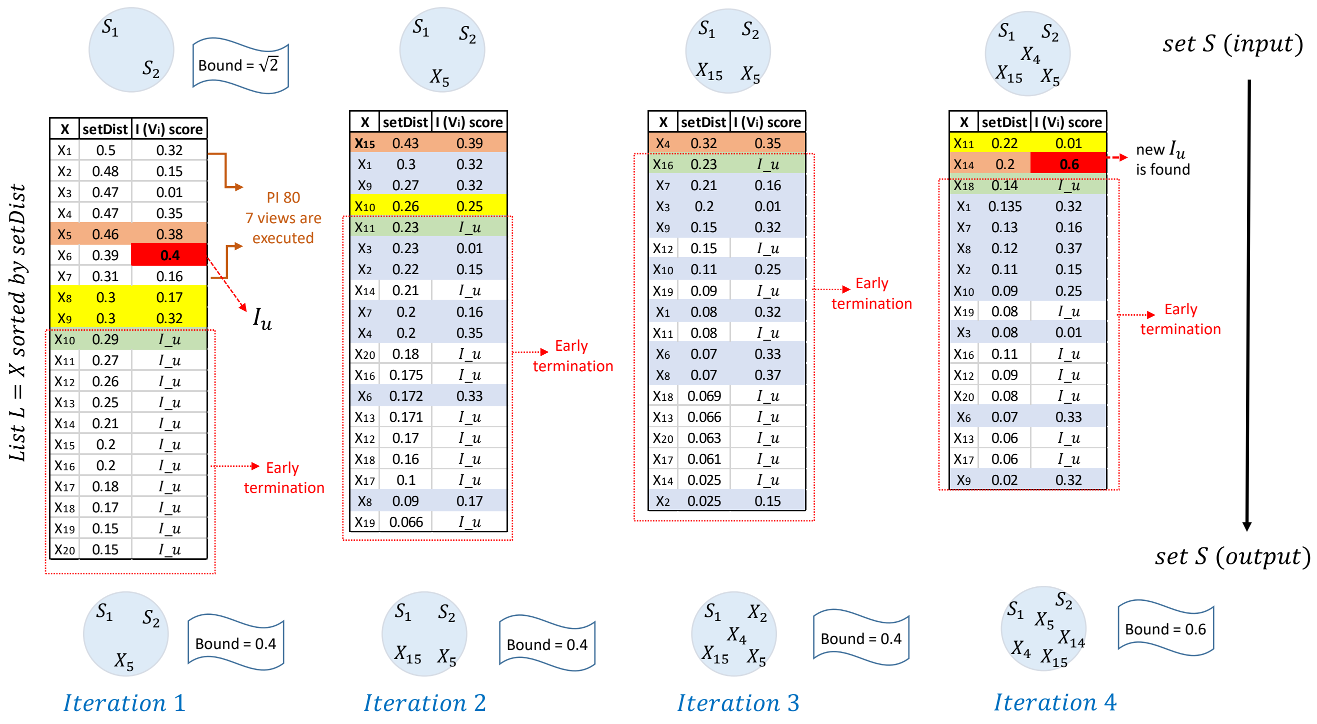
In the next section, we explain the detail of our rectifying strategy that consists of two important techniques: 1) Bookkeeping (keep tracking the result in each iteration) and 2) Caching (utilize cache to reuse the components from the previous iteration).

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 view in L while it gets early termination (e.g., X_{10} Iteration 1). These three important variable are stored to be used while rectifying bound is needed. The detail of bookkeeping technique in our rectifying strategy is described next.

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 (e.g., $setDist$) 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

Cont.



X	setDist	I (V_i) score
X_1	0.5	0.32
X_2	0.48	0.15
X_3	0.47	0.01
X_4	0.47	0.35
X_5	0.46	0.38
X_6	0.39	**0.4**
X_7	0.31	0.16
X_8	0.3	0.17
X_9	0.3	0.32
X_{10}	0.29	I_u
X_{11}	0.27	I_u
X_{12}	0.26	I_u
X_{13}	0.25	I_u
X_{14}	0.21	I_u
X_{15}	0.2	I_u
X_{16}	0.2	I_u
X_{17}	0.18	I_u
X_{18}	0.17	I_u
X_{19}	0.15	I_u
X_{20}	0.15	I_u

X	setDist	I (V_i) score
X_{15}	0.43	0.39
X_1	0.3	0.32
X_9	0.27	0.32
X_{10}	0.26	0.25
X_{11}	0.23	I_u
X_3	0.23	0.01
X_2	0.22	0.15
X_{14}	0.21	I_u
X_7	0.2	0.16
X_4	0.2	0.35
X_{20}	0.18	I_u
X_{16}	0.175	I_u
X_6	0.172	0.33
X_{13}	0.171	I_u
X_{12}	0.17	I_u
X_{18}	0.16	I_u
X_{17}	0.1	I_u
X_8	0.09	0.17
X_{19}	0.066	I_u

X	setDist	I (V_i) score
X_4	0.32	0.35
X_{16}	0.23	I_u
X_7	0.21	0.16
X_3	0.2	0.01
X_9	0.15	0.32
X_{12}	0.15	I_u
X_{10}	0.11	0.25
X_{19}	0.09	I_u
X_1	0.08	0.32
X_{11}	0.08	I_u
X_6	0.07	0.33
X_8	0.07	0.37
X_{18}	0.069	I_u
X_{13}	0.066	I_u
X_{20}	0.063	I_u
X_{17}	0.061	I_u
X_{14}	0.025	I_u
X_2	0.025	0.15

X	setDist	I (V_i) score
X_{11}	0.22	0.01
X_{14}	0.2	**0.6**
X_{18}	0.14	I_u
X_1	0.135	0.32
X_7	0.13	0.16
X_8	0.12	0.37
X_2	0.11	0.15
X_{10}	0.09	0.25
X_{19}	0.08	I_u
X_3	0.08	0.01
X_{16}	0.11	I_u
X_{12}	0.09	I_u
X_{20}	0.08	I_u
X_6	0.07	0.33
X_{13}	0.06	I_u
X_{17}	0.06	I_u
X_9	0.02	0.32

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

and the upper bound 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 in Figure 4, $PI80$ 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 (e.g., $X_1 - X_7$ 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 X_{10}. Hence, X_{10} and all views below of X_{10} can be ignored because $maxU(X_i) < U(H)$. Meanwhile, all views above X_{10} (e.g., X_8, X_9) need to be generated and its actual utility $U(X_i)$ is calculated 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., X_5, orange background color).

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 upper bound in the second and third iterations. However, a 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., $X_{10}, X_{11}, X_{16}, X_{18}$ in Iteration 1,2,3,4 respectively) are stored. The position of view when early termination started is important because all views which terminated need to be re-evaluated while a new upper bound is found. Some views which previously have a condition $maxU(X_i) < U(H)$ may change to $maxU(X_i) > U(H)$ after using a new upper bound. While a new higher upper bound is found such as in Figure 4 (Iteration 4),

Cont.

the scheme will update the upper bound and backtrack to the previous iterations which have wrong bound to re-evaluate the result. The list L, set S and other components which stored in each iteration are used in this process. Our bookkeeping technique can be seen in Algorithm 1 for DiVE-Greedy-Adaptive (e.g., line 30 - 34) and Algorithm 2 (e.g., line 32 - 37) for DiVE-dSwap-Adaptive.

This rectifying strategy seems promising, however, rectifying bound needs a lot of forward and backward looping that may leads to high cost. To overcome this issue, we propose caching technique which can minimize the cost as presented in the next section.

1.1.2 Caching Technique

As explained in the previous section, our rectifying strategy uses bookkeeping technique by storing S, L, and the early termination position in each iteration. These stored variables are used while backtracking is needed to re-evaluate the wrong bound and eliminate error in $F(S)$. As shown in Figure 4, there is no higher upper bound in the second and third iteration. However, while reach to the last iteration, a new higher upper bound is found. Hence, scheme have to backtrack to the first iteration with the new higher upper bound, then use the previous variables which are stored and re-evaluate the result using new upper bound.

The example above is only one possible scenario, it is possible in each iteration a higher upper bound is found. If new higher upper bound is found in each iteration, it needs a lot of forward and backward looping. For instance, in the last iteration a new upper bound is found in 4_{th} view, then the scheme backtrack with the new upper bound to the first iteration and go to the next iterations. Then, after reach the last iteration, a new higher upper bound is found again (e.g., 7_{th} view), then the scheme go back to the first iteration again. This condition leads to high cost process and inefficient.

In order to minimize cost when rectify the upper bound, we proposed cache management technique. The main goal of our caching management is to reuse components which already generated in the previous iteration and to avoid double query view execution. Our cache management can be seen in Figure 3. Our cache is the table that consists of all views and its importance score. In each iteration, to get the importance score of view, the scheme will search in the cache table first. If the importance score can be found in the cache then it is returned to the scheme. The query view is executed to DB while the importance score is not available in the cache table (e.g., *unknown*). Our caching technique force the scheme to re-use components from the previous iterations and it can avoid double query execution. For instance, let see in Figure 4, the importance score of views in turquoise color area (e.g., X_1, X_9, X_3, X_2 in Iteration 2, etc) are from the cache table (e.g., reuse from the previous iterations). This cache management is able to minimize the cost significantly.

2 Experiment Results

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

Cont.

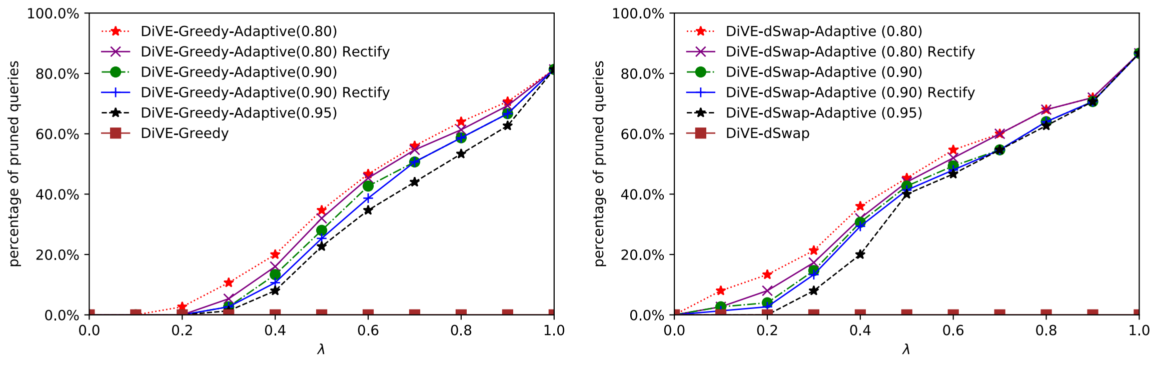


Figure 5: Rectifying bound DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive

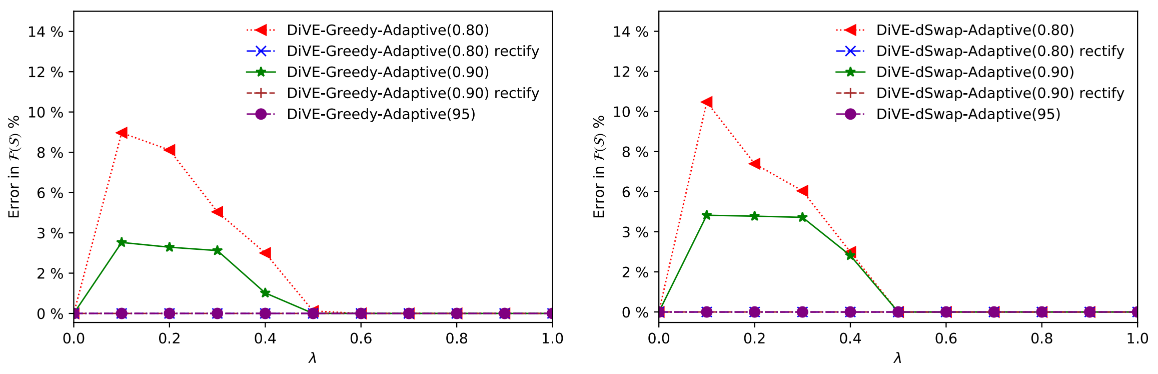


Figure 6: Error in $F(S)$ DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive after rectifying upper bound

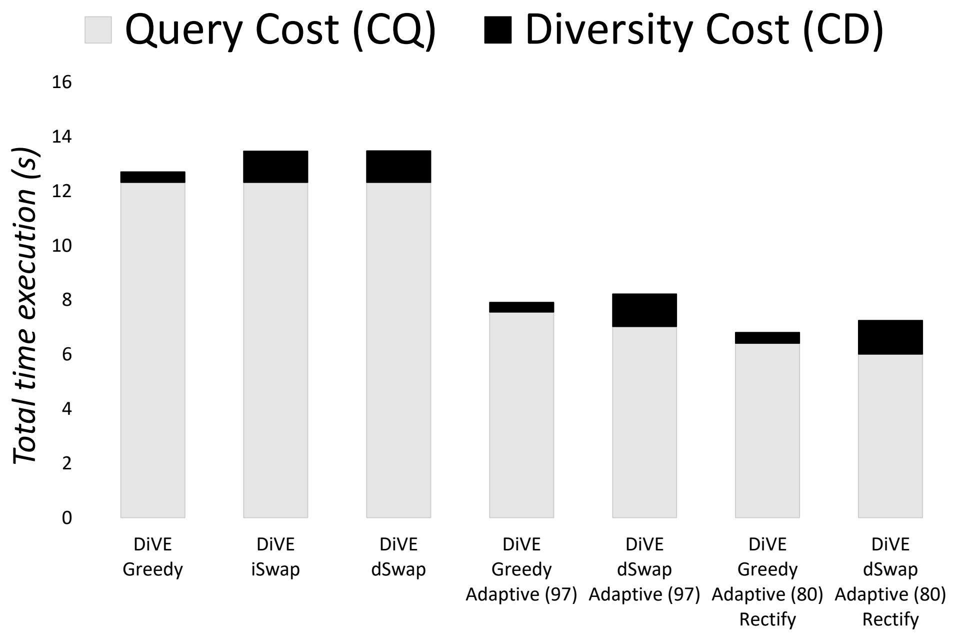


Figure 7: Total costs of schemes running on Flights dataset, $k = 5$, and $\lambda = 0.5$

Cont.

Algorithm 1: *DiVE* Greedy Pruning Rectifying

```
Input: Set of views V and result set $S$ize 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 $S$ize k
Output: Result set $S \leq V$, size S = k
S ← 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.