Online Appendix to: Plan Bouquets: A Fragrant Approach to Robust Query Processing

ANSHUMAN DUTT and JAYANT R. HARITSA, Indian Institute of Science

A. PROOFS FOR RANDOMIZED VARIANTS OF THE PLAN BOUQUET ALGORITHM

A.1. Randomizing Intra-Contour Plan Sequence

LEMMA A.1. *The bouquet algorithm with randomized intra-contour plan sequence provides* $MESO_g = \rho(\frac{r}{r-1} + \frac{r}{2}) + \frac{r}{2}$, *while retaining* $MSO_g = \frac{\rho r^2}{r-1}$.

PROOF. Assume that q_a lies in the *unique* region corresponding to plan P_i^k (ith plan on the kth contour). Note that the randomization strategy only affects the cost incurred due to the *finishing* contour IC_k, since any permutation on the previous contours will fail to complete q_a. Specifically, since the plan sequence is chosen uniformly at random from all possible permutations, q_a will finish with $1, 2, 3, \ldots, n_k$ executions with equal probability of $\frac{1}{n_k}$.

With the above framework, the expected bouquet cost is given by

$$E[c_B(q_a)] = cost(IC_1) + cost(IC_2) + \cdots + cost(IC_{k-1}) + E[cost(IC_k)],$$

leading to

$$E[c_B(q_a)] = (n_1)a + (n_2)ar + (n_3)ar^2 + \cdots + (n_{k-1})ar^{k-2} + \left[\frac{1}{n_k}(1 \times ar^{k-1} + \cdots + n_k \times ar^{k-1})\right].$$

Overestimating every n_i with ρ leads to

$$E[c_B(q_a)] \leq \rho(a + ar + ar^2 + \cdots + ar^{k-2}) + \left[\frac{1}{\rho}(1 \times ar^{k-1} + \cdots + \rho \times ar^{k-1})\right].$$

Dividing by ar^{k-2}, that is, the minimum possible cost in $(IC_{k-1}, IC_k]$ gives

$$E[SubOpt(*, q_a)] \leq \rho\left(\frac{1}{r^{k-2}} + \frac{1}{r^{k-3}} + \cdots + r + 1\right) + \left[\frac{r}{\rho}(1 + 2 + \cdots + \rho)\right].$$

Finally, overestimating the finite geometric series with an infinite series provides

$$E[SubOpt(*, q_a)] < \rho\left(1 + \frac{1}{r} + \frac{1}{r^2} + \cdots \infty \text{ terms}\right) + \left[r \times \left(\frac{\rho + 1}{2}\right)\right],$$

resulting in

$$E[SubOpt(*, q_a)] < \rho\left(\frac{r}{r-1}\right) + \left(\frac{\rho+1}{2}\right)r = \rho\left(\frac{r}{r-1} + \frac{r}{2}\right) + \frac{r}{2}. \quad \square \qquad (13)$$

Clearly, the above analysis and the bound on expected sub-optimality is independent of k. For $r = 2$, we get

$$\text{MESO}_\text{g} = |3\rho + 1|_{r=2} \quad \text{with} \quad \text{MSO}_\text{g} = 4\rho.$$

© 2016 ACM 0362-5915/2016/05-ART11 $15.00
DOI: http://dx.doi.org/10.1145/2901738

The bound on expected sub-optimality can be optimized by minimizing the multiplier for ρ, that is, $M = \frac{r}{r-1} + \frac{r}{2}$, which reaches its minima for $r = \sqrt{2}+1 \approx 2.4$. This leads to

$$\text{MESO}_\text{g} = \rho\left(\frac{r}{r-1} + \frac{r}{2}\right) + \frac{r}{2} = |(1.5+\sqrt{2})\rho + 1.2|_{r=1+\sqrt{2}} = |2.9\rho + 1.2|_{r=2.4}.$$

A.2. Randomizing Contour Placement

LEMMA A.2. *The bouquet algorithm with randomized contour placement provides* $MESO_g = \rho\frac{r}{\ln r}$, *while retaining* $MSO_g = \frac{\rho r^2}{r-1}$.

PROOF. For starters, assume that the contours are placed as per the deterministic bouquet algorithm. Now, consider a q_a that lies infinitesimally above IC_{k-1}—its worst-case suboptimality is $\frac{\rho r^2}{r-1}$, as per Theorem 3.4. However, when the entire geometric sequence is shifted left by a random multiplicative factor $\frac{1}{r^X}$, where X is a uniform random variable $\in [0, 1)$, this suboptimality becomes a decreasing function of the amount of shift. Specifically, the expected sub-optimality is

$$E[SubOpt(*, q_a)] < E\left[\frac{1}{r^X}\left(\frac{\rho r^2}{r-1}\right)\right] = \frac{\rho r^2}{r-1}\left(\int_{X=0}^{1} r^{-X} dX\right) = \frac{\rho r^2}{r-1} \times \left(\frac{r-1}{r\ln r}\right) = \frac{\rho r}{\ln r}. \quad (14)$$

It is easy to show that a similar analysis holds for an arbitrary q_a lying between the original deterministic contours IC_{k-1} and IC_k. The only difference is that instead of a continuously decreasing sub-optimality function of the shift, we get the following behavior: The sub-optimality initially decreases as the IC_k contour moves from its original position towards q_a, reaching a minimum when q_a is present on the contour. Then, there is a sudden discontinuous increase in sub-optimality when the contour crosses q_a, because q_a is now covered by a new contour that is r times its optimal cost. As the shift continues, the new covering contour begins to move closer to q_a, again resulting in a decreasing function. In expectation, this behavior is the same as that shown for the specific case above.

Caveat: An implicit assumption in the above analysis is that the value of ρ is not increased by the randomization (although the individual contour plan densities may have been altered due to the changed contour locations). □

Using the above randomization scheme with $r = 2$ results in an MESO_g of 2.89ρ. This already-low value can be improved even further by setting $r = e$, corresponding to the minima of the $(\frac{r}{\ln r})$ expression. Specifically, we obtain $\text{MESO}_\text{g} = \rho e \approx 2.72\rho$, the corresponding MSO_g being 4.3ρ.

A.3. Using the Randomization Strategies in Tandem

We now move on to deriving MESO_g when *both* randomization strategies are applied to the plan bouquet algorithm in tandem. Specifically, the contour placement randomization is applied first, and then, for each resulting contour, the execution order randomization is applied. For this combined algorithm, the following theorem gives an upper bound on the worst-case expected sub-optimality.

THEOREM A.3. *Given a query* Q *on a multi-dimensional* ESS, *the bouquet execution algorithm with contour placement randomization followed by intra-contour plan sequence randomization provides* $MESO_g = \rho\frac{(r+1)}{2\ln r} + \frac{(r-1)}{2\ln r}$, *while retaining* $MSO_g = \frac{\rho r^2}{r-1}$.

PROOF. The intra-contour randomization performance for a given placement of contours (Lemma A.1), remains essentially the same for each instantiated value of the contour placement random variable (Lemma A.2). Hence, we calculate the expected sub-optimality for a given location q_a, by leveraging Equation (13) across different random contour placements:

$$E[SubOpt(*, q_a)] \leq E\left[\frac{1}{r^X}\left[\rho\left(\frac{r}{r-1}+\frac{r}{2}\right)+\frac{r}{2}\right]\right]$$

$$E[SubOpt(*, q_a)] \leq \left[\rho\left(\frac{r}{r-1}+\frac{r}{2}\right)+\frac{r}{2}\right]\times E\left[\frac{1}{r^X}\right]$$

$$E[SubOpt(*, q_a)] \leq \left[\rho\left(\frac{r}{r-1}+\frac{r}{2}\right)+\frac{r}{2}\right]\times \frac{r-1}{r\ln r}$$

$$E[SubOpt(*, q_a)] \leq \left[\frac{r(r+1)\rho}{2(r-1)}+\frac{r}{2}\right]\times \frac{r-1}{r\ln r}$$

$$E[SubOpt(*, q_a)] \leq \rho\frac{r+1}{2\ln r}+\frac{r-1}{2\ln r} \quad \square \tag{15}$$

For $r = 2$, we obtain

$$|3\rho+1|_{r=2} \times 0.72 = |2.16\rho+0.72|_{r=2}.$$

Further, this guarantee bound can be improved by minimizing the $\frac{r+1}{\ln r}$ multiplier of ρ—the multiplier reaches its minimum value for $r = 3.6$, leading to

$$\text{MESO}_\text{g} = |1.8\rho+1|_{r=3.6}.$$

B. ALGORITHMIC DETAILS FOR EXECUTION COVERING ENHANCEMENT

B.1. Algorithms for Covering Sequence Identification

Clearly, the basic bouquet algorithm provides a simple-to-implement solution where no skipping of executions take place. On the other hand, the exhaustive algorithm enumerates all candidate covering sequences but incurs a computational effort that is exponential in the number of executions present in the bouquet sequence. To bridge this gulf, we describe here a polynomial-time algorithm, CSI, that attempts to improve the MSO$_\text{g}$ of the bouquet sequence with the following idea—"find covering executions for plans on the MSO causing contour and its predecessors."

For instance, the MSO$_\text{g}$ of 24.5 in the example of Figure 11(b) is caused at IC_4 due to the aggregate impact of $\omega(CS^1)$ through $\omega(CS^4)$. This MSO$_\text{g}$ can be improved by finding the covering sequences individually for the contours IC_1 through IC_4. For this purpose, we describe below a subroutine of CSI that finds the covering sequence for a contour IC_k while trying to minimize its effective plan density—employing this routine brings the MSO$_\text{g}$ in Figure 11(b) down to 14.5.

Minimizing $\omega(CS^k)$. The problem of minimizing $\omega(CS^k)$ can be abstracted as an adaptation of the *Red Blue Weighted Dominating Set* [Fomin and Stepanov 2007] problem; a sample adaption corresponding to our running example is shown in Figure 24(a).

Here, the blue set contains nodes from contour IC_k, and the red set contains nodes from the contours IC_k through $IC_{k'}$, where $cost(IC_{k'}) \leq |IC_k| \times cost(IC_k)$. Apart from the (covering) edges borrowed from the Hasse diagram (i.e., IC_{k+1} to IC_k), it also includes

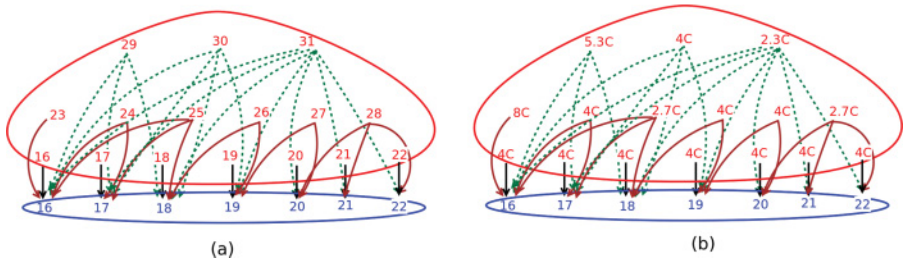

Fig. 24. Adapting red-blue weighted domination for solving *minimize(w(*CS^4*))*.

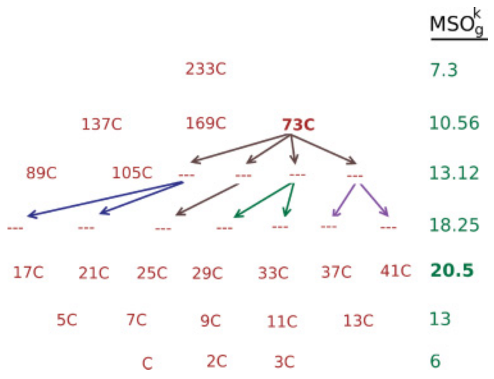

Fig. 25. Solution for $minimize(w(CS^4))$ using red-blue domination.

transitive edges for farther contours (IC_{k+2} onwards) and identity edges from red version to blue version of IC_k nodes. Finally, the weight of any red node is given by the corresponding value of $\omega(E_i)$.

With the above modeling, the problem of minimizing $\omega(CS^k)$ is the same as finding the minimum-weight subset of red nodes that can dominate all the nodes in the blue set. Now, since red-blue weighted domination is known to be a NP-hard problem, and also equivalent to the Minimum-weight Set-Cover problem [Fomin and Stepanov 2007], we have utilized an adaptation of the *greedy weighted set-cover* algorithm to ensure efficiency—the greedy criterion is the minimum weight per newly covered blue node. The pseudocode for the resulting subroutine is shown in Algorithm 3.

For the example formulation in Figure 24(a), the weight per covered blue node for each of the red nodes is shown in Figure 24(b). Here we find that E_{31} is the greedy red choice, and it is the solution since there are no more blue nodes to be covered. The greedy subroutine has a runtime complexity of $V_{blue}log(V_{red})$ and approximation factor of $log(V_{blue}) + 1$. The covering sequence obtained for IC_4 with this approach is shown in Figure 25.

The CSI Algorithm. CSI begins by identifying the contour that causes the MSO$_g$, denoted IC_{k^*}, and then applies the minimization subroutine on contours IC_1 through IC_{k^*} in sequence. After this pass, it is possible that the "culprit" MSO$_g$ contour has now shifted to a new contour $IC_{k^{**}} > IC_{k^*}$—if so, another minimization pass is carried out for contours IC_{k^*} through $IC_{k^{**}}$. Otherwise, the algorithm is concluded since MSO$_g$ cannot be improved further.

The motivation for processing contours in increasing cost order during each minimization pass is that the covering nodes identified for contour IC_k may also cover nodes

ALGORITHM 3: Minimize $\omega(CS^k)$ Subroutine

```
Input: $V_{blue} = IC_k, V_{red} = (IC_k \cup \ldots \cup IC_{k'}), \omega : BS \rightarrow \mathbb{R}$
Output: $V_{cov}$
// $V_{cov}$ is the set of covering executions (subset of $V_{red}$)
$V_{cov} = \{\}$ ;
// $V_{dom}$ is the set of executions covered by $V_{cov}$ (subset of $V_{red}$)
$V_{dom} = \{\}$;
while $V_{dom} \subset V_{blue}$ do
    $v_{sel} = \phi$ ;
    $f(v_{sel}) = \infty$ ;
    for $v \in V_{red} \setminus V_{cov}$ do
        $f(v) = \frac{\omega(v)}{dom(v) \setminus V_{dom}}$ ;
        if $f(v_{sel}) > f(v)$ then
            $v_{sel} = v$;
        end
    end
    $V_{dom} = V_{dom} \cup dom(v_{sel})$;
    $V_{cov} = V_{cov} \cup \{v_{sel}\}$;
end
```

from contours IC_{k+1} and beyond, thereby reducing the number of uncovered nodes for higher contours.

B.2. Efficiently Constructing Hasse Diagram of Executions

In principle, the CSI routine needs to determine the cover relation among *all pairs* of executions in BS. However, the explicit checking of these pairs can be reduced by leveraging the following inference tests:

(1) If both executions are from the same contour, reject the pair as a cover relation cannot exist between them.
(2) If one of the executions is $E_{terminal}$, then accept the pair since the cover exists, by definition of $E_{terminal}$.
(3) If the executions lie more than one contour apart, then explicit evaluation is required only if a transitive cover relation is non-existent.

Even with the above pruning, the processing required for the remaining pairs may still turn out to be computationally significant, since the explicit test for a pair (E_i, E_j) is equivalent to establishing whether $R(E_j)$ is a subset of $R(E_i)$. We therefore present next an alternative procedure to determine the cover relation $E_i \succeq_{cover} E_j$, which does not entail this subset check.

Cover Determination Procedure

Let $L(E_i)$ denote the contour locations for E_i. We use MAX_i to represent the ESS location whose value on each dimension corresponds to the maximum selectivity coordinate on that dimension across all contour locations of E_i. That is,

$$s_d(MAX_i) = \max_{l \in L(E_i)} s_d(l) \text{ for } 1 \leq d \leq D.$$

Similarly, we denote with MIN_i the ESS location corresponding to the minimum selectivity coordinate for each dimension across all contour locations of E_i. That is,

$$s_d(MIN_i) = \min_{l \in L(E_i)} s_d(l) \text{ for } 1 \leq d \leq D.$$

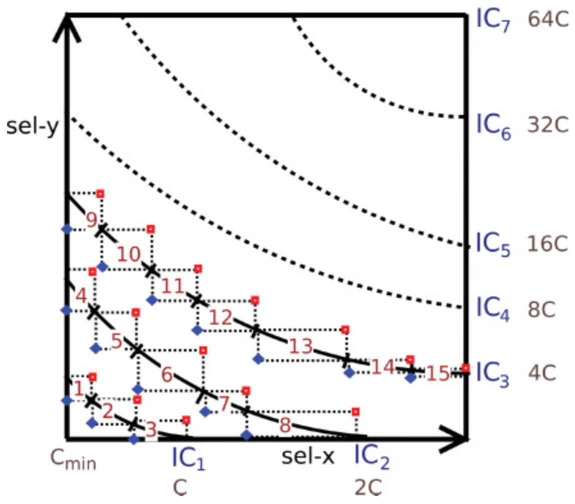

Fig. 26. MAX_i and MIN_i locations for executions on contours IC_1 to IC_3.

To make these notions concrete, the MAX_i and MIN_i locations for the first three contours of the example ESS are shown as *red* squares (■) and *blue* diamonds (◆), respectively, in Figure 26.

To determine the cover existence between E_i and E_j, we first employ two heuristic checks that may not be decisive in all cases but are quite efficient to evaluate. Their efficiency stems from their usage of only MAX_i's and MIN_i's, which are easily computable through a single scan of the $L(E_i)$ locations.

Product Check 1: The necessary and sufficient condition that $MIN_i \succeq_{product} MAX_j$ for $E_i \succeq_{cover} E_j$ helps to efficiently identify true positives. For example, it can quickly determine that in Figure 26, the candidate execution pair (E_7, E_3) satisfies the cover relation.

Product Check 2: The necessary condition that $MAX_i \succeq_{product} MAX_j$ for $E_i \succeq_{cover} E_j$, helps to quickly reject true negatives. For instance, (E_5, E_3) can be quickly rejected using this criterion.

In the event that the above two checks are not conclusive, we employ a final check that is decisive in all scenarios but is comparatively expensive since it requires, for each location in set $L(E_i)$, processing the *entire set* of locations in $L(E_j)$.

Product Check 3: The condition $L(E_i) \succeq_{product} L(E_j)$ is sufficient to decide whether $E_i \succeq_{cover} E_j$. Here, $L(E_i) \succeq_{product} L(E_j)$ is satisfied if there exists a location $l^s \in L(E_i)$ for each $l^t \in L(E_j)$, such that $l^s \succeq_{product} l^t$.

While certainly more expensive than Checks 1 and 2, note that Check 3 is expected to be relatively more efficient than the direct region-subset check, since typically $|L(E_i)| \ll |R(E_i)|$.

To highlight the potency of the above evaluation procedure, consider, for instance, the neighboring contours IC_2 and IC_3 in Figure 26. Here, there are 35 execution pairs arising from the contours, and among them, 6 pairs are identified as true positives (Check 1), 21 pairs are rejected as true negatives (Check 2), leaving only 8 pairs for the final comparison (Check 3).