Machine Learning for Data Management Systems

Materialized Views

Amol Deshpande
February 21, 2023

Outline

- **Background**
- Prior Work
- Automatic View Generation using DL and RL

Views

- View: Virtual expressions, used for query simplification and access control
- Substituted into the query when used

```
create view physics_fall_2009 as
    select course.course_id, sec_id, building, room_number
    from course, section
    where course.course_id = section.course_id
                and course.dept_name = 'Physics'
                and section.semester = 'Fall'
                and section.year = '2009';
```

```
select course_id
from physics_fall_2009
where building= 'Watson';
```

Materialized Views

- Pre-compute commonly used views for efficiency
  - Need to keep up-to-date as the base tables change
    - Usually done through “incremental” updates
  - Take extra storage space
- Same challenge for any “pre-computed” summary
  - Including many types of “indexes”
- Deciding whether a materialized view can be used for a given query undecidable in general
  - Lot of theoretical work on this problem
  - Especially in the “data integration” context

Materialized Views

- Question 1: Which views to materialize and what types of indexes to build on them (view design or selection)
- Question 2: How to maintain views when base tables are updated (view maintenance)
- Question 3: How and when to use views for a given query (view exploitation)
  - Work dating back to 1985

Outline

- Background
- **Prior Work**
- Automatic View Generation using DL and RL

View Exploitation

- Discussion from: Optimizing Queries with Materialized Views; 1995
- Consider view:
  - Executive(name, dno, sal) with employees with sal > 200k
  - Query: select * from emp where sal > 200k and dno = 0419
  - Index on emp on dno → Use the index
  - No index on emp on dno → Use the view

  - May choose to ignore a materialized view even if the query asks for it
    - Query: select * from Executive where dno = 0419
    - Index on emp on dno → Use the index
- Need to do this in a cost-based manner
  - Have the optimizer enumerate the options and figure out the cost for each of them
  - Can only be done efficiently for simpler views

View Selection

- AutoAdmin (SQL Server 2000)

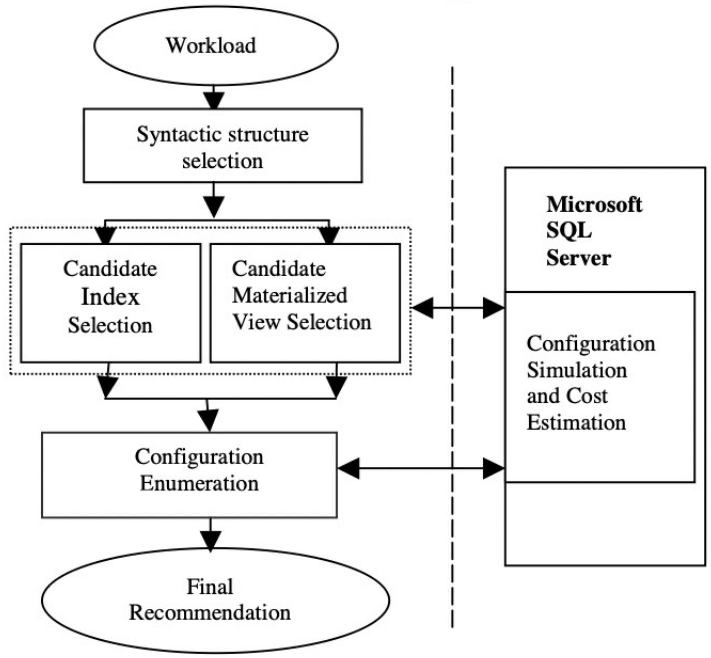


Figure 1. Architecture of Index and Materialized View Selection Tool

BigSubs

- Selecting Subexpressions to Materialize at Datacenter Scale; VLDB 2018
- Slightly different motivation
  - Shared analytics clusters (e.g., in the cloud) with 10000s of jobs
  - Quite a bit of shared computation across them (saved upto 40% of compute cost)
  - Same tasks repeated again and again → optimization will pay off
  - No updates considered
- Another related problem:
  - Multi-query optimization: optimize a group of queries together
  - Little work – considered quite hard, and not well-motivated

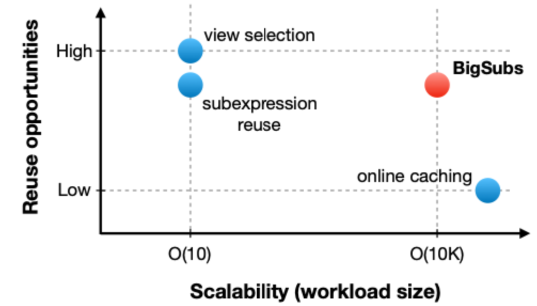


Figure 1: Approaches for computation reuse.

BigSubs Example

- S1 more common, but less benefit
- S2 has high storage cost
- If S2 is picked, s1 doesn't benefit Q3 or Q4

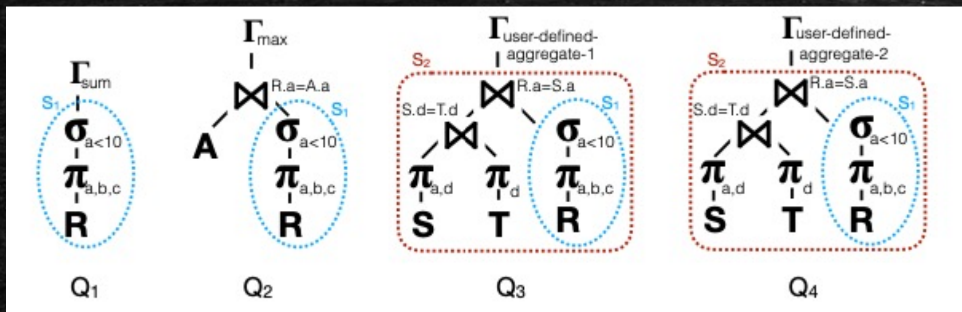


Figure 3: Illustrating the subexpression selection problem.

BigSubs Approach

- Key assumption: Focus on the query plans generated by the optimizer
  - i.e., don't try to reoptimize the query given a materialized view – prior work did this
  - This provide better statistics (assuming the same query was ran before)
- Choose candidate subexpressions
  - Any subexpression of any query is a valid option
- Utility of a subexpression, or a set of them, for a query

DEFINITION 2 (UTILITY OF SUBEXPRESSION). *Let q_i be a query and s_j one of its candidate subexpressions. We define the utility u_{ij} of s_j for q_i to be:*

$$u_{ij} = C_{\mathcal{D}}(s_j) - C_{acc}(s_j) \tag{2}$$

DEFINITION 3 (UTILITY OF SUBEXPRESSION SET). *Let q_i be a query and S be a set of candidate subexpressions. Let R_i^{max} be the rewriting that leads to the highest cost reduction for q_i. We define the utility $U_S(q_i)$ of S for q_i to be:*

$$U_S(q_i) = \sum_{s_j \in R_i^{max}} u_{ij} \tag{3}$$

BigSubs Approach

- Interacting subexpressions captured as a matrix

DEFINITION 4 (INTERACTING SUBEXPRESSIONS). *Two candidate subexpressions s_1, s_2 for query q are* interacting, *if the tree corresponding to the logical plan of one is a subtree of the other.*

- Problem formulation,

$$\underset{\mathcal{S} \subseteq \mathbb{S}}{\operatorname{argmax}} \sum_{i=1}^{n} U_{\mathcal{S}}(q_i), \text{ with } B_{\mathcal{S}} \leq B_{max}$$

- As an Integer Linear Program:

$$\begin{aligned}
\text{maximize} \ & \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij} \cdot y_{ij} \\
\text{s.t.} \ & \sum_{j=1}^{m} b_j \cdot z_j \leq B_{max} \\
& y_{ik} + \frac{1}{m} \sum_{\substack{j=1 \\ j \neq k}}^{m} y_{ij} \cdot x_{jk} \leq 1 \quad \forall i \in [1,n], k \in [1,m] \\
& y_{ij} \leq z_j \quad \forall i \in [1,n], j \in [1,m]
\end{aligned}$$

BigSubs Approach

- ILP can't be solved at scale
- Instead cast it as a bipartite graph labelling problem, and use a greedy heuristic on that

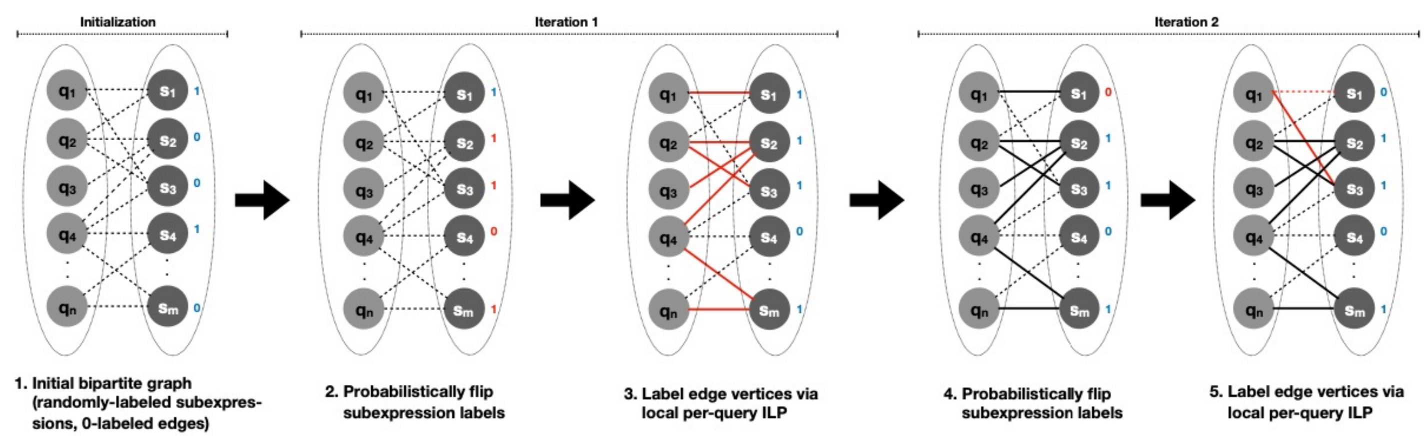
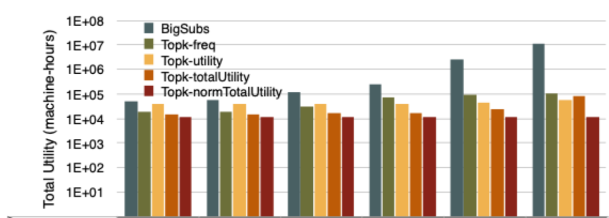


Figure 4: Illustrating first two iterations of subexpression selection via bipartite graph labeling. We assume each subexpression has storage footprint $b_j=1$ and the total budget is $B_{max}=3$. Subexpression labels are shown next to the vertices. For the edges, we use solid lines when label is 1 and dashed ones otherwise. At each iteration, we mark with red the labels whose value changed.

BigSubs Approach



Heuristic \ Data size	10GB	100GB	1TB	10TB	100TB	1PB
Topk-freq	2.69	2.95	4.22	3.47	29.57	112.92
Topk-utility	1.30	1.43	3.19	6.46	60.72	207.95
Topk-totalUtility	3.38	3.72	7.52	15.30	108.50	144.38
Topk-normTotalUtility	4.42	4.86	10.88	21.46	221.51	939.43

Figure 10: Total utility (machine-hour savings) on production *Workload1* using different selection methods and cost budgets. Since the graph shows utility (i.e., savings), higher is better. The table shows the improvement factor of BIGSUBS when compared to the other schemes (higher factor is better).

Outline

- Background
- Prior Work
- **Automatic View Generation using DL and RL**

Automatic View Generation using DL/RL

- Same setting as BigSubs
  - Select subexpressions/views to materialize for a workload
  - Ignore updates

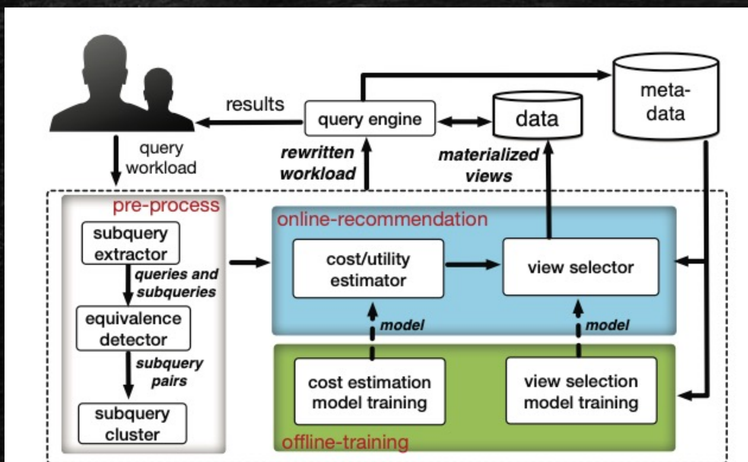


Fig. 3. The overview of system framework

Definition 7 (MVS problem): Given a query workload Q and a set of its possible subqueries S_Q, we select subqueries $S \subseteq S_Q$ to build materialized views V_S and then select views $V_S^q \subseteq V_S$ for each query $q \in Q$ as follows:

$$\arg\max_{S \in S_Q, V_S^q \in V_S} \sum_{q \in Q} \sum_{v_s \in V_S^q} \mathcal{B}_{q,v_s} - \sum_{v_s \in V_S} \mathcal{O}_{v_s}$$

$s.t.$ s_i, s_j are not overlapping, $\forall q \in Q, i, j \in [1, |V_S^q|]$.

Example

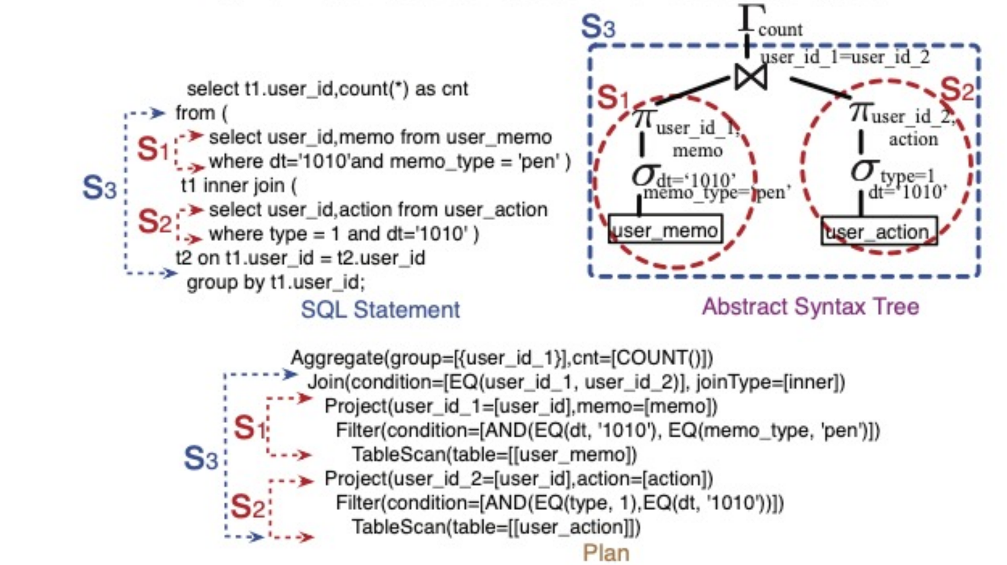


Fig. 2. An example of a query and its subqueries

Automatic View Generation using DL/RL

- Problem 1: How to estimate the cost of a query given a view
  - BigSubs assumed statistics
  - This work uses a cost estimator using DL

- Problem 2: Finding the best set of views to materialize
  - BigSubs approach has some convergence issues
  - Use Reinforcement Learning instead

Utility Estimation

- Features: a combination of operator information and table information
- Build a wide linear model in parallel to a deep model

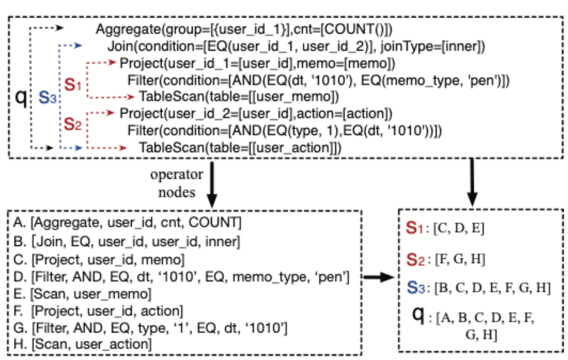


Fig. 4. Feature Extraction from Plans

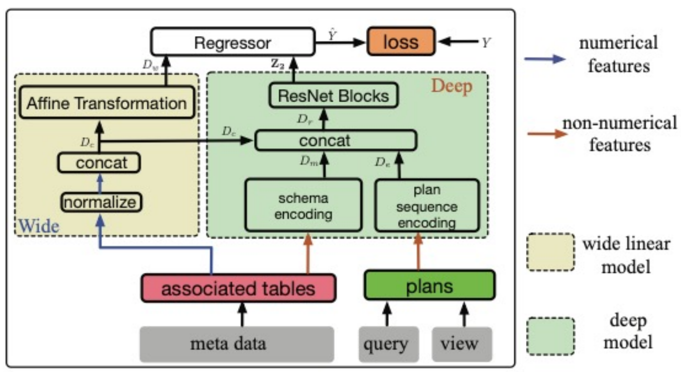


Fig. 5. Wide-Deep Model

Utility Estimation

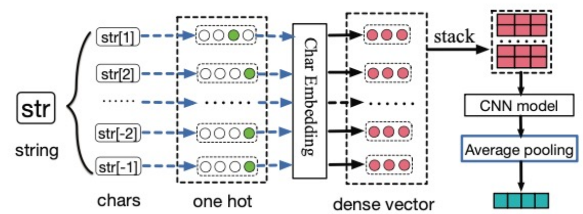
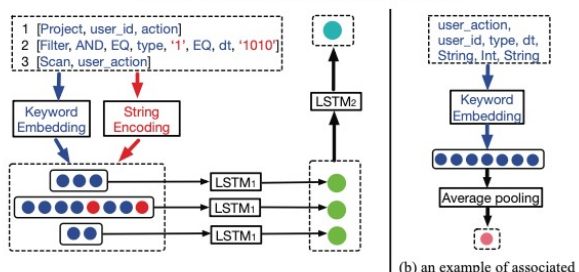


Fig. 6. The overview of String Encoding



(a) an example of query/view plan sequence encoding

(b) an example of associated table schema encoding

Fig. 7. Non-numerical feature encoding examples

Algorithm 1: Wide-Deep Model Training

Input: training features ($X = \{(q_1, v_1, t_1), \cdots\}$), training targets ($Y = \{\mathcal{A}_{\beta,\gamma}(q_1|v_1), \cdots\}$), table schema encoding part $\mathcal{M}_m$, query/view plan encoding part $\mathcal{M}_e$, wide part $\mathcal{M}_w$, deep part $\mathcal{M}_d$, regressor part $\mathcal{M}_r$, learning rate lr, training epochs I, batch size b_s.

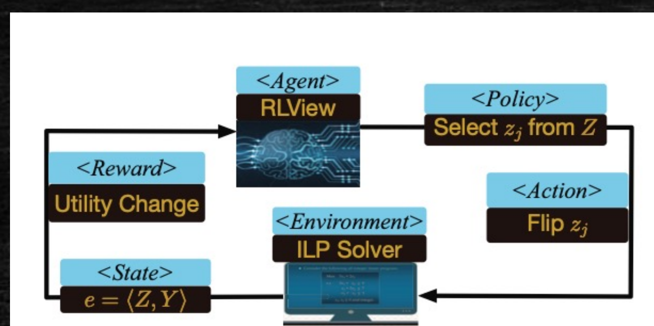
Output: parameters $\theta_m, \theta_e, \theta_w, \theta_d, \theta_r$ for $\mathcal{M}_m$, $\mathcal{M}_e$, $\mathcal{M}_w$, $\mathcal{M}_d$ and $\mathcal{M}_r$

```
extract numerical features X^m from {t_1, t_2, ...};
extract non-numerical features X^n from {(q_1, v_1, t_1), ...};
for j ← 1 ... I do
    training iterations I' = ⌊|Y| / b_s⌋;
    shuffle(X^m, X^n, Y);
    for i ← 1 ... I' do
        X_i^m, X_i^n, Y_i ← sample b_s data from X^m, X^n, Y;
        Normalize and concatenate X_i^m into D_c;
        D_m = M_m(input table schema in X_i^n);
        D_e = M_e(query/view plans in X_i^n);
        D_r = concat(D_c, D_m, D_e);
        Ŷ_i ← M_r(M_w(D_c), M_d(D_r));
        loss_i ← MSE(Y_i, Ŷ_i);
        θ_m, θ_e, θ_w, θ_d, θ_r ← AdamOpt(loss_i, lr);
return θ_m, θ_e, θ_w, θ_d, θ_r;
```

- Data to train the model?

Solving the ILP

- Model as a reinforcement learning problem
- Action: Select or de-select a subexpression to materialize
- Reward: The (additional) utility of the resulting state
- Use Deep Q-Learning Network to solve



$Z = \{z_j\}$: z_j is a 0/1 variable indicating whether to materialize the subquery s_j

$Y = \{y_{ij}\}$: y_{ij} is a 0/1 variable indicating whether to use the view v_{s_j} for the query q_i

Fig. 8. The MDP Framework of the ILP Problem

Results

- One a few different synthetic-ish datasets

TABLE V
END-TO-END RESULTS (**O&B**: `Optimizer` + `BigSub`, **O&R**: `Optimizer` + `RLView`, **W&B**: `W-D` + `BigSub`, **W&R**:`W-D` + `RLView`)

Data	JOB			P1			P2		
	$\#q$	c_q($)	l_q(s)	$\#q$	c_q($)	l_q(s)	$\#q$	c_q($)	l_q(s)
	226	15.39	571.16	832	91.27	7.07k	5378	558.19	49.9K
	$\#(q\|v)$	$\#m$	o_m($)	$\#(q\|v)$	$\#m$	o_m($)	$\#(q\|v)$	$\#m$	o_m($)
O&B	182	24	1.04	231	24	1.04	1307	162	34.49
O&R	164	19	0.85	233	24	0.70	1361	156	30.25
W&B	140	17	0.46	224	21	0.79	1345	174	40.79
W&R	148	18	0.53	250	26	0.93	1300	144	22.96
	$b_{q\|v}$($)	$l'_q(s)$	r_c(%)	$b_{q\|v}$($)	$l'_q(s)$	r_c(%)	$b_{q\|v}$($)	$l'_q(s)$	r_c(%)
O&B	2.48	479.12	9.36	8.75	6.04k	8.45	71.86	30.36k	6.69
O&R	2.44	480.61	11.70	8.90	6.02k	8.98	75.31	30.25k	8.07
W&B	2.04	495.47	10.27	8.76	6.02k	8.73	83.20	30.70k	7.60
W&R	2.38	482.86	**12.02**	9.32	5.98k	**9.19**	72.15	30.75k	**8.81**

Table V shows the end-to-end results. At first, we report the number ($\#q$), the cost (c_q) and the latency (l_q) of raw queries. Then, for each method, we report the number ($\#m$) and the overhead (o_m) of materialized views, the number ($\#(q|v)$) and the benefit ($b_{q|v}$) of rewriting queries, and the latency (l'_q) of the rewritten workload. At last, we report the associated ratio (r_c), which is computed as $r_c = \frac{b_{q|v} - o_m}{c_q}$. In conclusion, we can find some observations as follows: (1) Our system outperforms other methods. For `JOB`, **W&R** can save 12.02% cost while **O&B** only save 9.36% cost, so our system improves the performance by $\frac{12.02-9.36}{9.36} \times 100\% = 28.4\%$. Similarly, the improvement for `P1` and `P2` are $\frac{9.19-8.45}{8.45} \times 100\% = 8.8\%$ and $\frac{8.81-6.69}{6.69} \times 100\% = 31.7\%$, respectively. (2) The more accurate the cost model, the better the solution of the view selection model. For example, **W&B** and **W&R** save more cost than **O&B** and **O&R**, respectively. (3) `RLView` is more robust than `BigSub`. Taking `JOB` for example, the ratio r_c of `BigSub` is decreased by $10.27\% - 9.36\% = 0.91\%$ while `RLView` is only decreased by $12.02\% - 11.70\% = 0.32\%$. (4) Building more materialized views doesn't mean saving more cost. In the example of `JOB`, **O&B** gets the most benefit, but it saves the least cost because of the heavy overhead of views.