

Machine Learning for Data Management Systems

SkinnerDB

Amol Deshpande
March 6, 2023

Outline

- Reinforcement Learning and UCB/UCT
- Simplified QP Problem with UCT
- SkinnerDB

Multi-armed Bandit

K arms; unknown sequence of *stochastic* rewards $R_1, R_2, \dots \in [0,1]^K$; $R_t \sim \nu$





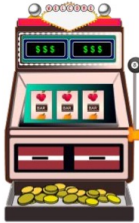




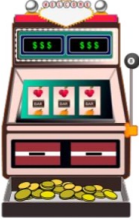


For each round $t = 1, 2, \dots, T$ (assume horizon T is known; will say more later)

- Choose arm $A_t \in [K]$
- Obtain reward R_{t,A_t} and *only* see R_{t,A_t}



Problem was introduced by Robbins (1952).

Multi-armed Bandit

 ?	 ?	 0.2	 ?	Pull arm 3
 0.01	 ?	 ?	 ?	Pull arm 1
 ?	 ?	 0.6	 ?	Pull arm 3

Multi-armed Bandit

K arms; unknown sequence of *stochastic* rewards $R_1, R_2, \dots \in [0,1]^K$; $R_t \sim \nu$

For each round $t = 1, 2, \dots, T$ (assume horizon T is known; will say more later)

- Choose arm $A_t \in [K]$
- Obtain reward R_{t,A_t} and *only* see R_{t,A_t}

Arm i has mean μ_i *which is unknown*.

Goal: Find a policy that minimizes the regret

$$\text{Reg}(T) = \underbrace{T \cdot \mu^*}_{\text{Reward of best arm}} - E \left[\underbrace{\sum_{t \in [T]} R_{t,A_t}}_{\text{Algorithm's reward}} \right] \quad \mu^* = \max_i \mu_i$$

Ideally, we would like that $\text{Reg}(T) = o(T)$.

Multi-armed Bandit

At each time step, we can either:

1. (Exploit) Pull the arm we think is the best one; or
 2. (Explore) Pull an arm we think is suboptimal.
-
1. We do not know which is the best arm so if we keep exploiting, we may keep pulling a suboptimal arm which may incur large regret.
 2. If we explore, we gather information about the arms, but we pull suboptimal arms so may incur large regret again!

Challenge is to tradeoff exploration and exploitation!

Explore-then-commit (ETC)

Perhaps the simplest algorithm that **provably** gets sublinear regret!

Let T_0 be a hyper-parameter and assume $T \geq K \cdot T_0$.

1. Pull each of K arms T_0 times.
2. Compute empirical average $\hat{\mu}_i$ of each arm.
3. Pull arm with largest empirical average for remaining $T - K \cdot T_0$ rounds.

Theorem. Let $\Delta_i := \mu^* - \mu_i$ be suboptimality of arm i . Then

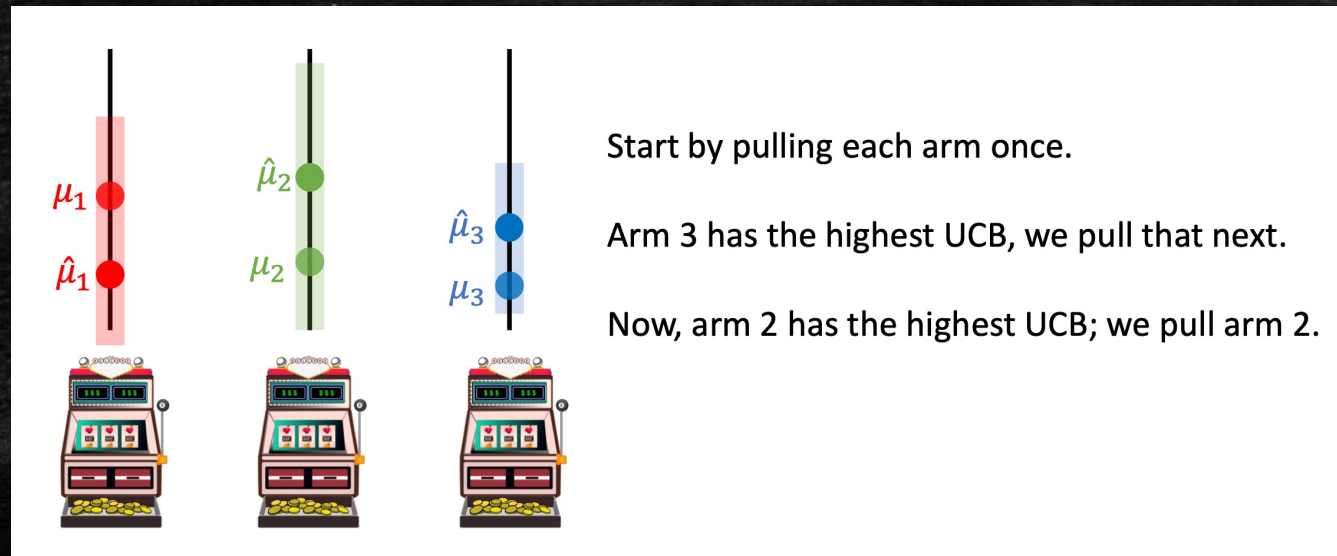
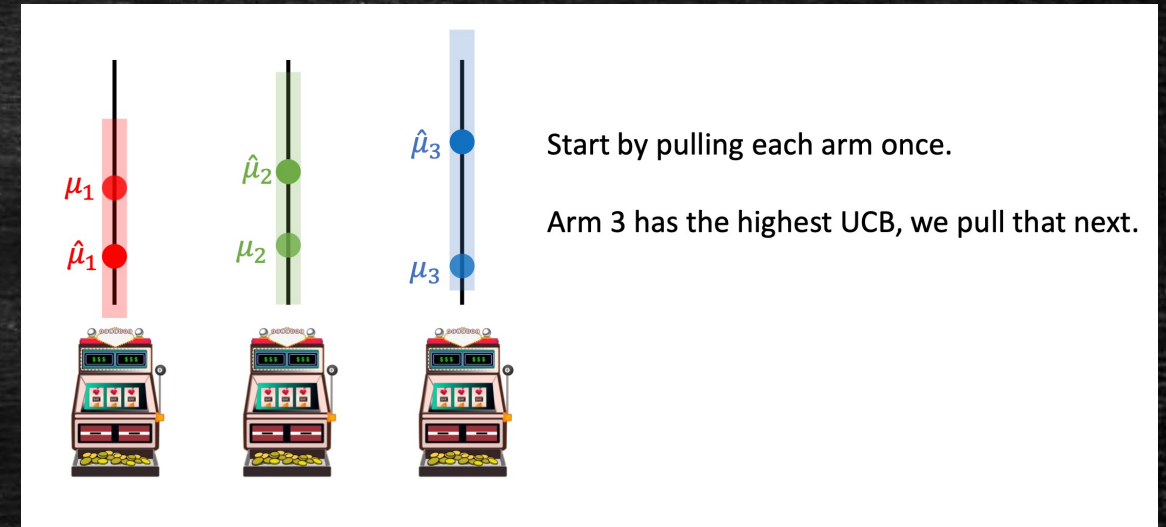
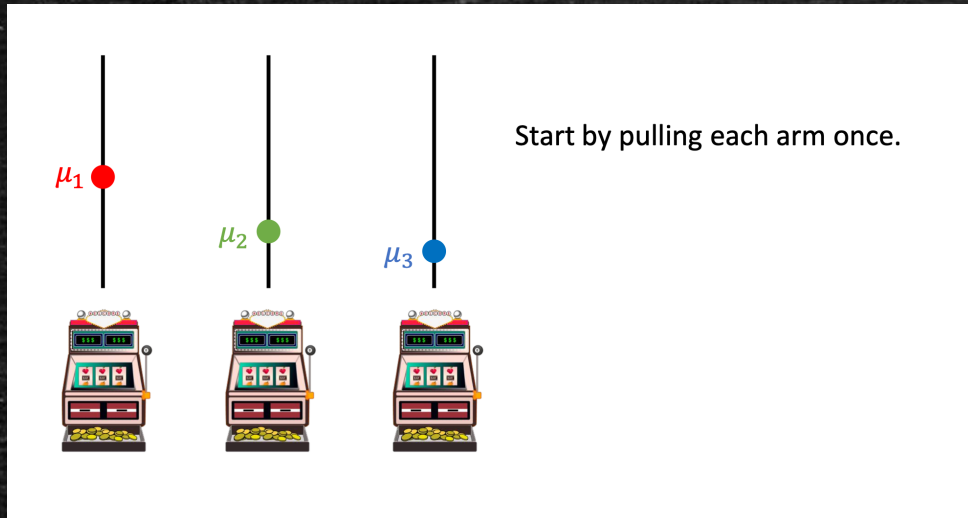
$$\text{Reg}(T) \leq T_0 \sum_{i \in [K]} \Delta_i + (T - K \cdot T_0) \cdot \sum_{i \in [K]} \Delta_i \exp\left(-T_0 \cdot \frac{\Delta_i^2}{C}\right)$$

“Cost of exploration”

Suboptimality of each additional step.

Note: The term $\Delta_i \exp\left(-T_0 \cdot \frac{\Delta_i^2}{4}\right)$ is small when T_0 is large.

Upper Confidence Bound (UCB)



Upper Confidence Bound (UCB)

Let $\delta \in (0,1)$ be a hyper-parameter.

- Pull each of K arms once.
- For $t = K + 1, K + 2, \dots, T$
 1. Let $N_i(t)$ be number of times arm i was pulled so far and $\hat{\mu}_i(t)$ be empirical average.
 2. Let $UCB_i(t) = \hat{\mu}_i(t) + \sqrt{2 \log \left(\frac{1}{\delta} \right) / N_i(t)}$
 3. Play arm in $\arg \max UCB_i(t)$.

Claim. Fix an arm i . Then with probability at least $1 - 2\delta$, we have

$$|\mu_i - \hat{\mu}_i(t)| \leq \sqrt{2 \log \left(\frac{1}{\delta} \right) / N_i(t)}$$

Upper Confidence Bound (UCB)

Theorem. Let $\Delta_i := \mu^* - \mu_i$ be suboptimality of arm i . If we choose $\delta \sim 1/T^2$:

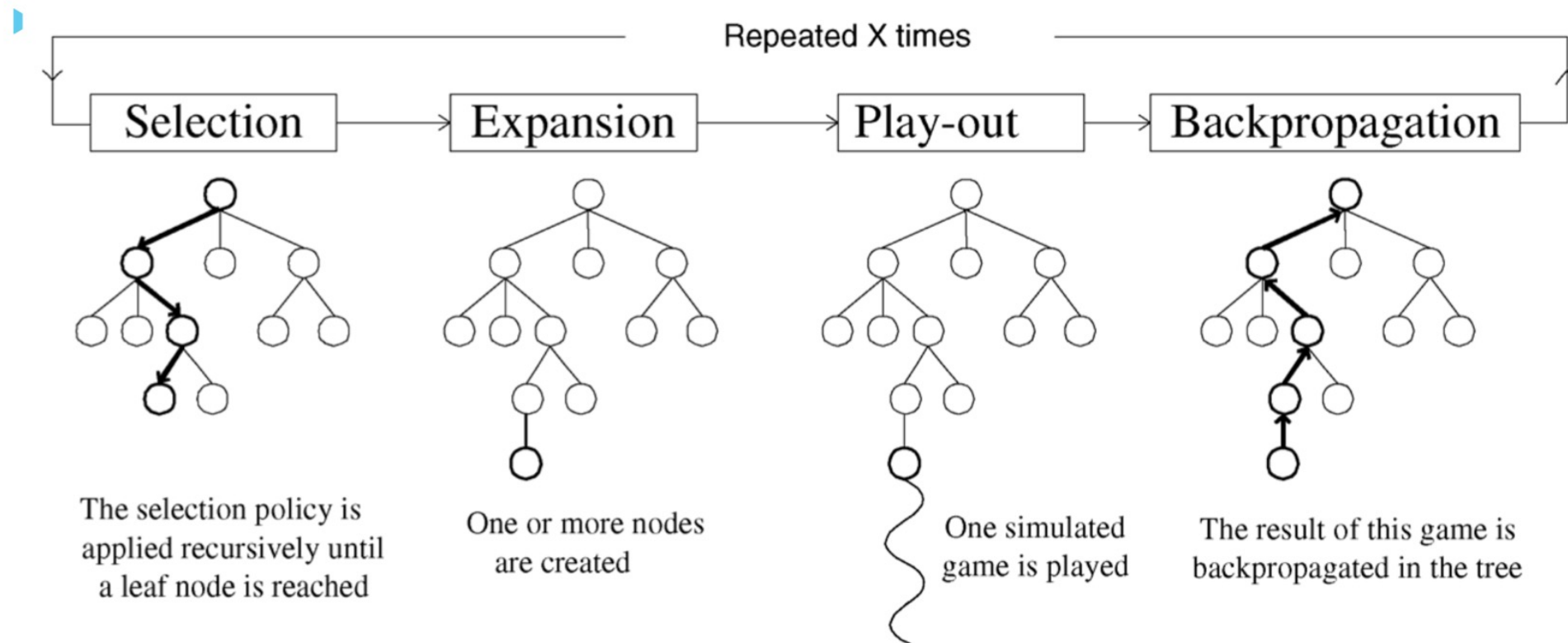
$$\text{Reg}(T) \leq C \sum_{i \in [K]} \Delta_i + \sum_{i: \Delta_i > 0} \frac{C \log(T)}{\Delta_i}$$

Always have to pay.

This turns out to mean the following:

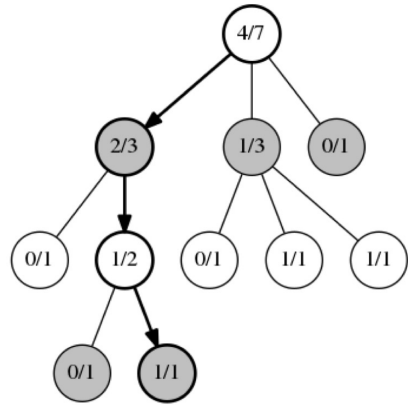
If $\Delta_i > 0$, we only pull arm i roughly $\Delta_i^{-2} \log(T)$ times incurring regret Δ_i each time.

Monte Carlo Tree Search



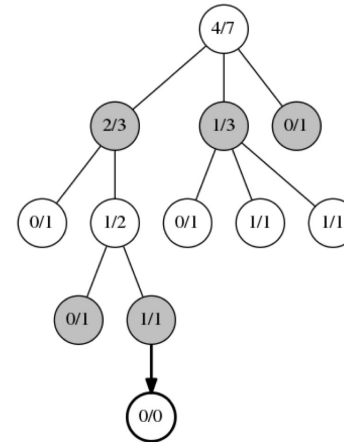
MCTS with UCB1

1) Selection



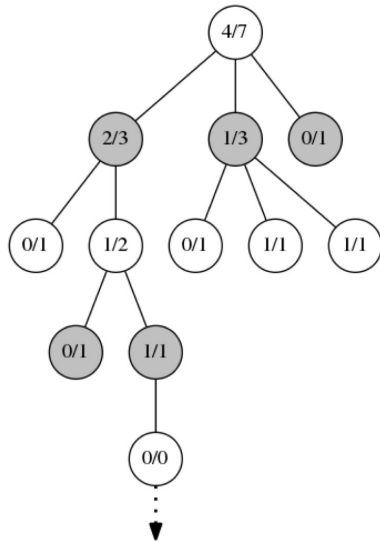
Selection policy is applied recursively until a leaf node is reached

2) Expansion



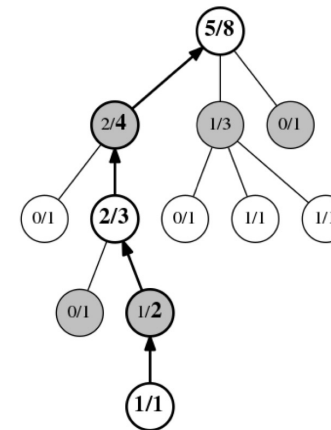
One or more nodes are created

3) Simulation



One simulated game is played

4) Backpropagation



Result is backpropagated up the tree

Outline

- Reinforcement Learning and UCB/UCT
- Simplified QP Problem with UCT
- SkinnerDB

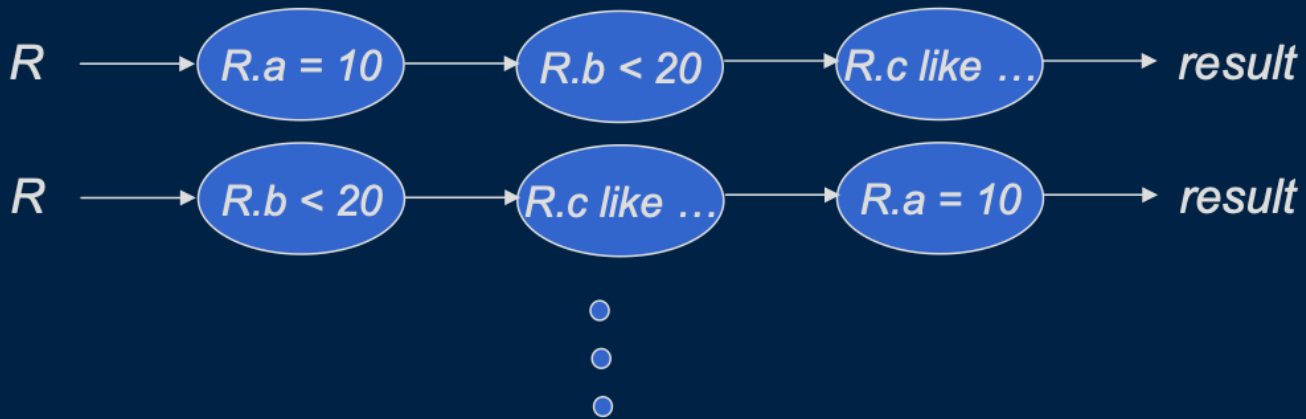
Simplified QP Problem

Find a *single order of the selections* to be used for *all tuples*

Query

```
select * from R
where R.a = 10 and R.b < 20
and R.c like '%name%';
```

Query plans considered



$3! = 6$ distinct
plans possible

Applying UCT

- Learn based on prior executions
- Reward only provided for a permutation
 - However, reward calculation requires executing on a sample of data
 - Reward = # tuples processed in a fixed amount of time
- Overall:
 - Choose a permutation using UCT algorithm
 - Process using that permutation for a fixed amount of time
 - Use the #input tuples processed as the reward

Outline

- Reinforcement Learning and UCB/UCT
- Simplified QP Problem with UCT
- SkinnerDB

Overview

- Learns (near-)optimal query plans on the fly during query execution
- No reliance on data statistics or simplifying cost and cardinality models
- Uses UCT to balance between exploitation and exploration
- No generalization errors across queries and no need for a-priori information

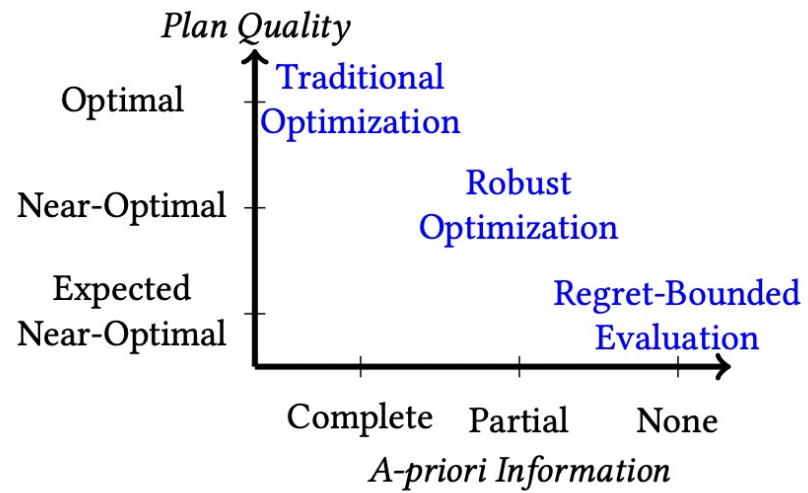


Figure 1: Tradeoffs between a-priori information and guarantees on plan quality in query evaluation.

SkinnerDB Architecture

- Pre-processor: filters base tables and partitions tuples for joins
- Join Processor: executes joins in small time slices with learning optimizer, join executor, progress tracker, and reward calculator
- Post-processor: performs grouping, aggregation, and sorting on join results

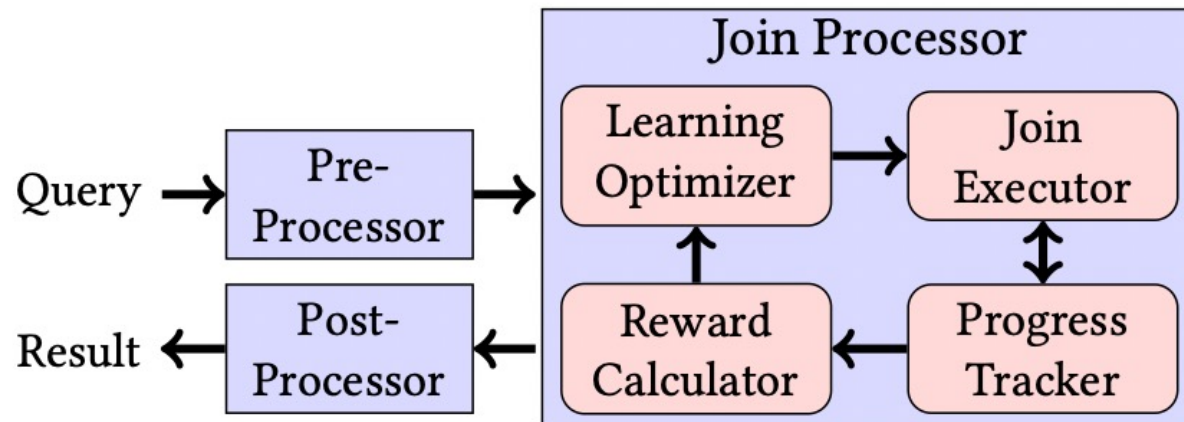


Figure 2: Primary components of SkinnerDB.

SkinnerDB: Search Tree

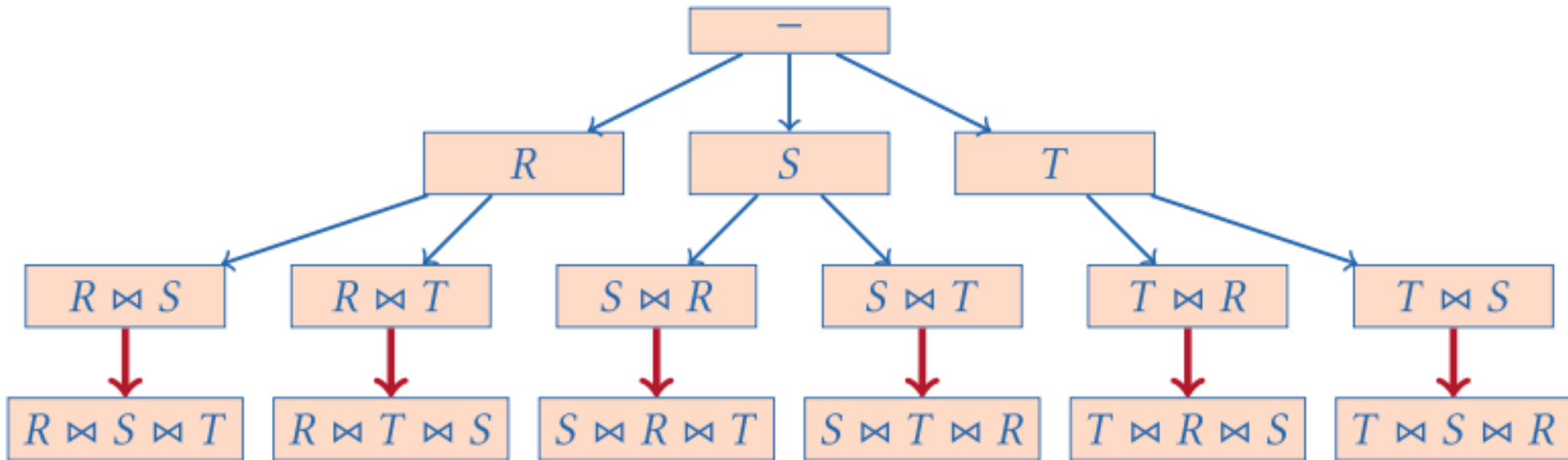


Fig. 2. We model join order choices for a query as an episodic Markov Decision Process. Actions (arrows) correspond to table choices and states (rectangles) to join order prefixes. A selected join order is executed on one data batch (Skinner-G and Skinner-H) or for one time slice (Skinner-C). Reward is calculated based on execution progress and associated with the final state (transition marked with thick red line).

Skinner-G

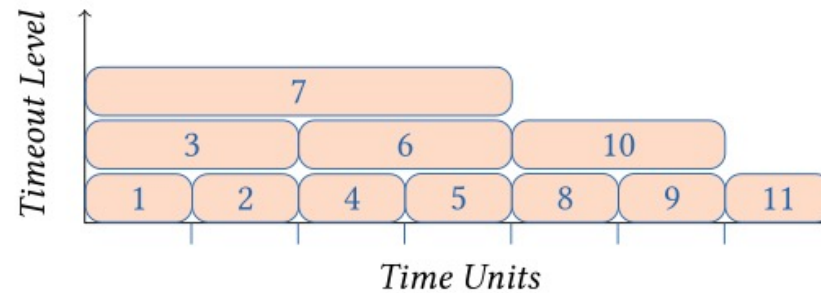


Fig. 3. Illustration of time budget allocation scheme: We do not know the optimal time per batch and iterate over different timeouts, allocating higher budgets less frequently.

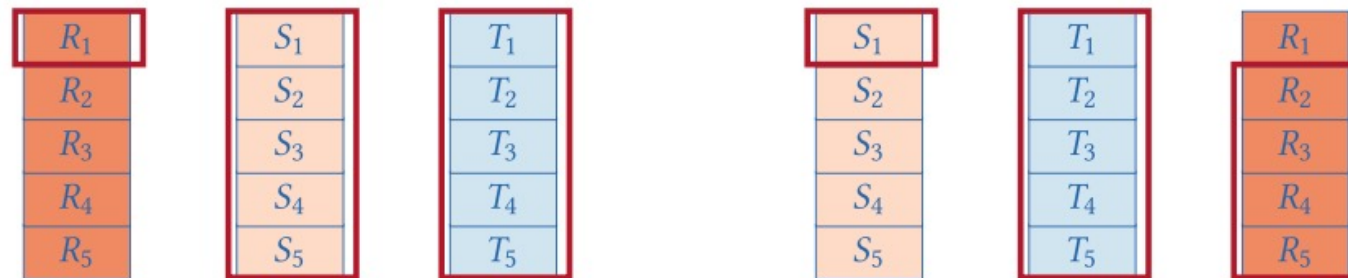
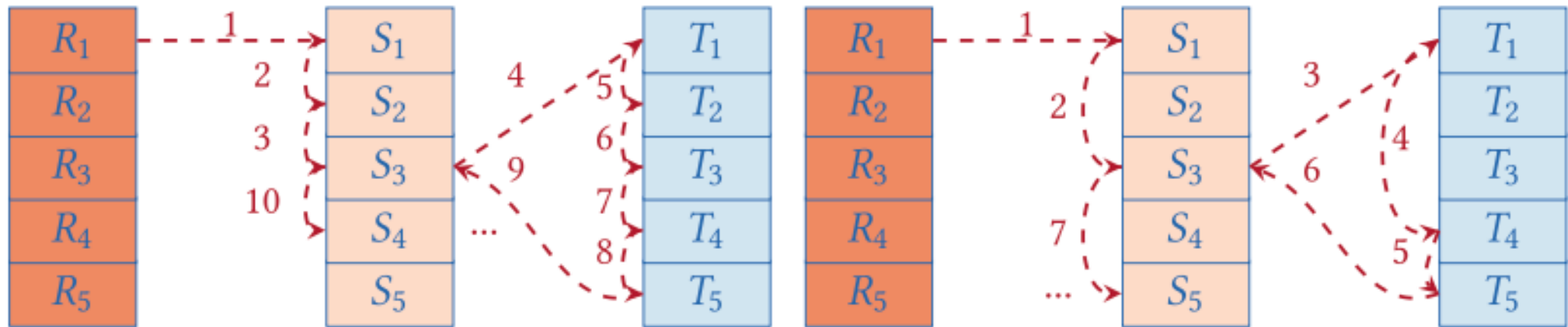


Fig. 4. Illustrating first (a) and second (b) episode of Skinner-G for an example query with three tables (R , S , and T). The red rectangles, surrounding the tables, highlight tuples joined in the corresponding episode.

Skinner-C Multi-way Join



(a) Depth-first multi-way join strategy: we increase the join order index once the first tuple satisfying all applicable predicates is found, we decrease it once all tuples in the current table were considered.

(b) Depth-first multi-way join strategy exploiting indices on join columns: using indices, the algorithm skips tuples that do not satisfy binary equality join predicates.

Fig. 6. Comparison of simple (left) and refined (right) variant of depth-first multi-way join algorithm.

Input-based Reward Function

- Measure how much of the “cartesian product” space is explored

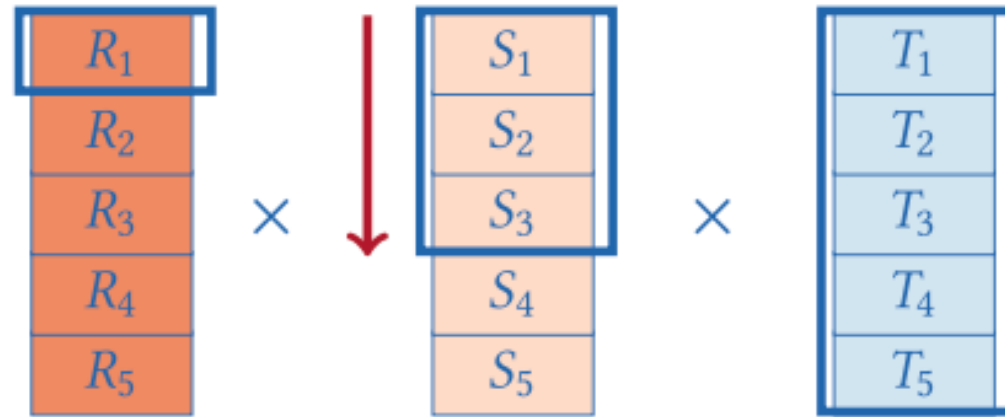


Fig. 8. The input reward function is based on the percentage of the space of Cartesian product tuples that has been considered per episode. When advancing by three tuples in the second table (for a fixed tuple in the first), we process a fraction of $(1/5) \cdot (3/5)$ of the Cartesian product space. All join orders must ultimately search 100% of the Cartesian product space, giving faster join orders a higher average reward per episode.

Sharing Progress

- For two permutations that share a prefix, the prefix computation doesn't need to be repeated

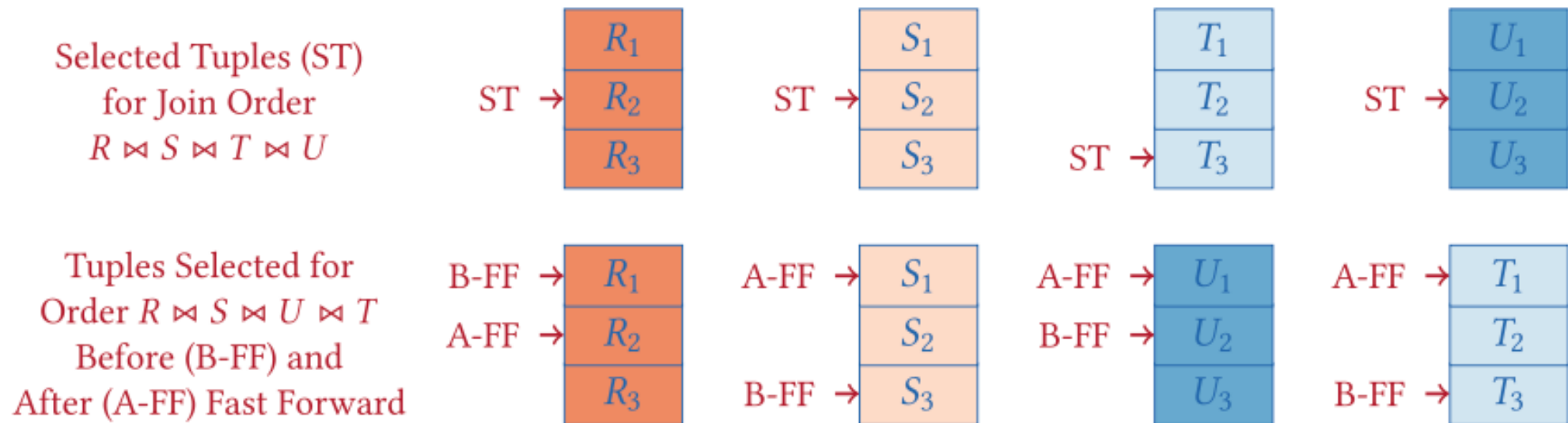


Fig. 9. We fast-forward the execution of join order $R \bowtie S \bowtie U \bowtie T$ (lower part) by integrating evaluation progress made for join order $R \bowtie S \bowtie T \bowtie U$ (upper part).

Regret Guarantees

- Skinner-G: at most worse by a factor of $O(m \log(n))$

THEOREM 7.10. *Expected execution time regret of Skinner-G is upper-bounded by $(1 - 1/(\log(n) \cdot m \cdot 4)) \cdot n + O(\log(n))$.*

- Skinner-C: at most worse by a factor of $O(m)$ (in limit)

THEOREM 7.13. *Expected execution time regret of Skinner-C is upper-bounded by $(1 - 1/m) \cdot n + O(\log(n))$.*

THEOREM 7.14. *The ratio of expected to optimal execution time for Skinner-C is upper-bounded and that bound converges to m as n grows.*

Experiments

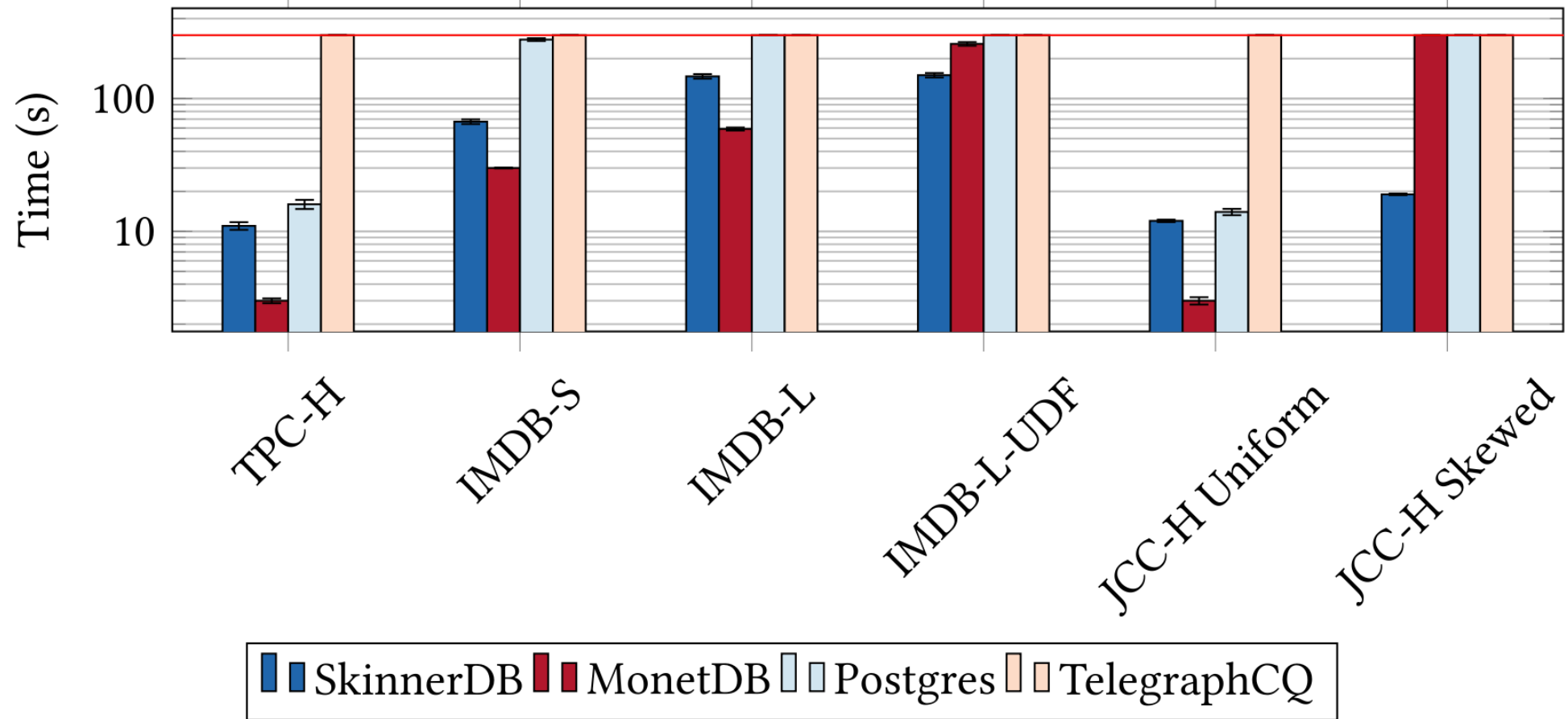


Fig. 12. Total runtime for different benchmarks and systems, the red line marks the timeout.

Experiments

Table 2. Comparison of Search Space for Join Ordering

Metric	IMDB-S		TPC-H	
	Average	Maximum	Average	Maximum
# UCT Nodes	211	4,779	18	64
# Plans	143	1,029	6	17

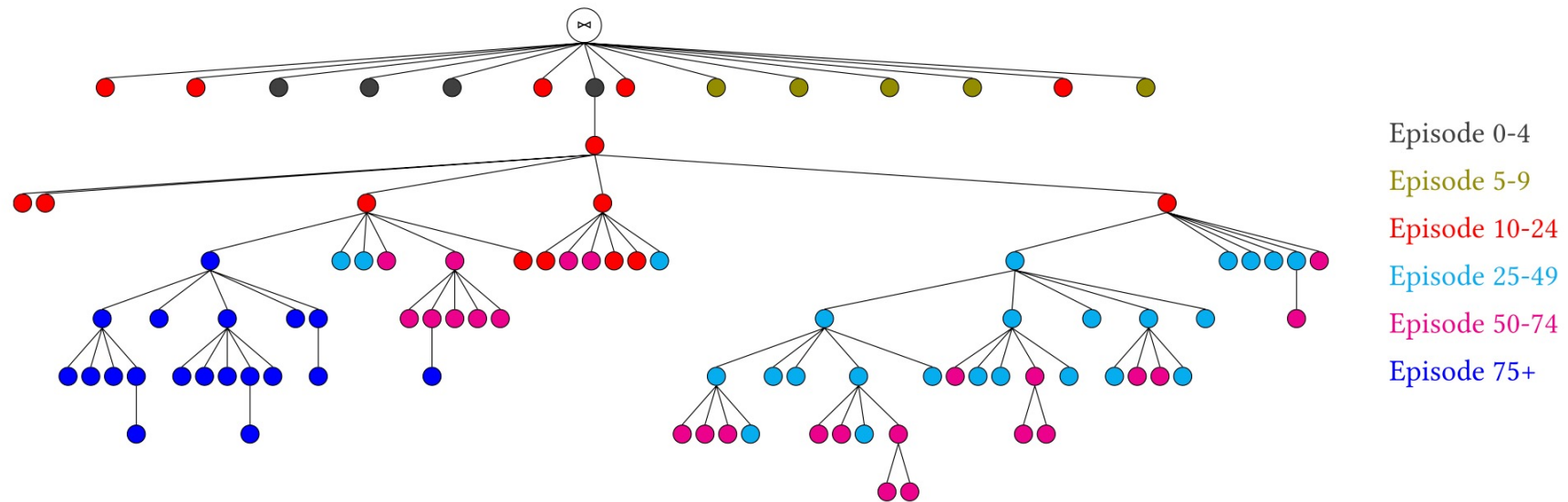


Fig. 22. Illustration of UCT growth over start of processing of IMDB-S query 33c: After random initial exploration, the tree is expanded towards join orders that seem most interesting. The node color indicates in which episode the corresponding node was added.

Experiments

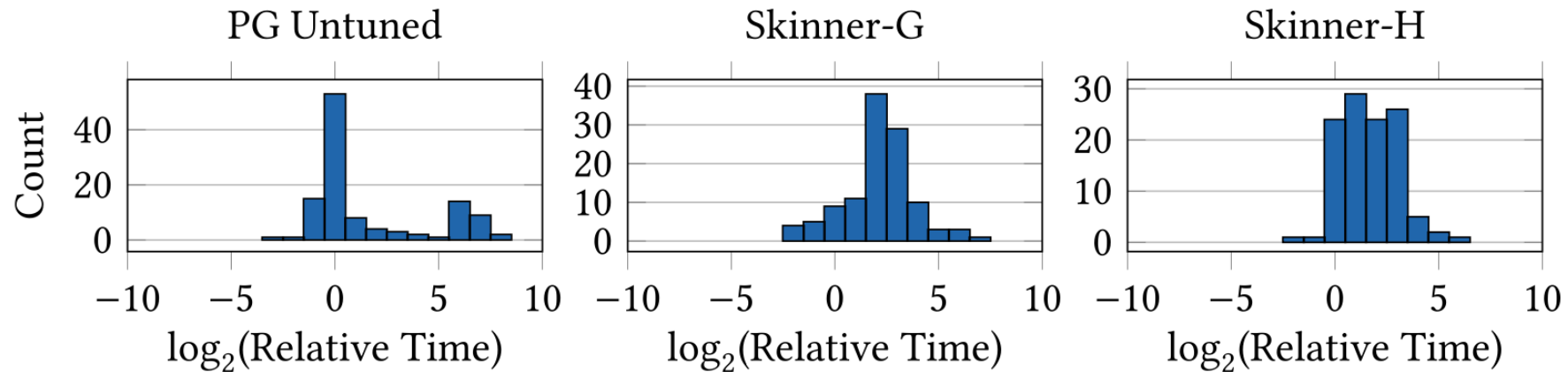


Fig. 29. We compare performance of different approaches on IMDB-S against performance of a manually tuned Postgres installation. Compared to an untuned installation, using Skinner-G or Skinner-H reduces the number of outlier queries where execution time is significantly above the fine-tuned version.

Table 3. Performance of SkinnerDB on Postgres with a Timeout of Five Minutes per Query

System	Time (s)	# Timeouts
Postgres - No Tuning	6,493	16
SkinnerG(PG)	2,049	1
SkinnerH(PG)	1,539	0
Postgres with Tuning	297	0

Some Discussion Points

- What's the main take-away from this paper?
- Major concerns with the paper?
- Possible improvements?