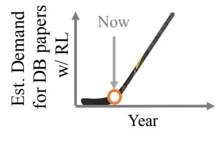
CIDR^2: *C*razier *I*nnovations in *D*atabases ⋈ *R*einforcement-learning *R*esearch

Eugene Wu
Columbia University, ewu@cs.columbia.edu

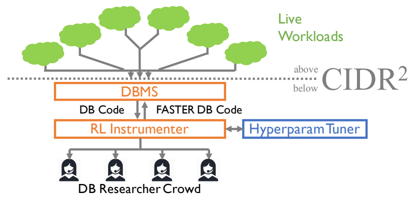
🔮 *One does not polish a round ball. The ball polishes itself.* 🔮

Since CIDR 2017, advances in reinforcement learning (RL) and neural networks have promised a world where robots excel at games (e.g., Go, Chess, Atari, Defense of the Ancients) to such an extent that humans are finally relieved from the burden of manually playing games. This has sparked a deep yearning for RL to similarly relieve humans from the burden of manually optimizing databases.

To this end, intrepid database researchers have proposed to replace or augment bits and pieces of the DBMS with RL: join order optimization, cardinality estimation, database driving, and even the humble index structure have been targeted, and shown improved performance, optimization overhead, and/or DBA costs. As the hockey stick curve in Figure 1.a forecasts, every corner of the DBMS must be illuminated by the warm glow of RL. However, the authors worry about a looming scalability challenge. Namely, that the database is a big piece of software[1] and growing at a tremendous pace. In contrast, the set of DB researchers only grows at a linear or possibly even sublinear rate. Thus, automated techniques are needed to scale *ML for Systems and Systems for ML* research.



(a) Est. demand for DB papers that use RL over time.



(b) `CIDR`^{2} System Architecture.

Figure 1

This abstract[2] outlines a world where the database is reinforced so hard that we never have to touch database code ever again. The primary idea is as follows: rather than point solutions that apply RL to specific components of the DBMS, `CIDR`^{2} reinforces everything. Similar to Mike Tyson's legendary defense, we will reinforce DBMSes so hard that DBMSes never have to worry about performance issues nor developers (in expectation). Amazingly, reinforcement learning algorithms such as UTC will guarantee that our regret over this entire exercise is bounded.

"*Wait*," you may wonder, "Isn't this Andy's self-driving database?" Please stay in your lane. It's more like everything is driving everywhere, all the time. "Bu..but,", you now ask, "*How*?"

We will use an advanced program analysis technique called "pattern matching". RL is really applied in two ways: to train a standalone prediction model and to learn a search policy. Thus, `CIDR`^{2} is an automated system that identifies any and all code fragments in the database that smell like a prediction decision or a search algorithm, and replaces such code with RL. Any binary branching logic (e.g., `if()`) will be replaced with RL to train a binary predictor; switching logic with RL to train a multi-class classifier; numeric functions with RL to train a regression model; you get the point. Similarly, any function that behaves like a state machine or search process will be replaced with RL to learn a search policy. Note that these functions can be as small as a single arithmetic expression buried deep in the bowels of the codebase, or as high-level as the ODBC driver handlers. Nested instrumentation is called a *Cascade*.

Figure 1.b presents the system architecture, which takes as input a DBMS codebase and a live workload from a centralized blockchain service (a cloud DB), and instruments the codebase. Using recent performance profiling techniques such as VProfiler, `CIDR`^{2} will identify all code paths that affect database performance and reinforce that code. The hyperparameter tuner uses a state-of-the-art technique called random search via DICE rolling to toggle RL at different locations and to learn parameters.

The lynchpin in RL is the reward function. `CIDR`^{2} leverages the power of *lineage* to trace and identify the program state that influences the replaced code fragments. For `if(a > b)`, we extract input state that contributed to `a` and `b` using *backwards lineage queries* and use their linear combination as the reward. Naturally, the linear weights are learned using RL. However, imperfect reward functions lead to robot uprisings, or Marios that gobble coins all day instead of saving the princess (did she really need saving?)

No sweat, we have *hybrid technique* to the rescue! If automated tuning is too slow or wrong, we employ a hierarchical crowd to both tune the reward functions and toggle instrumentation points. The crowd employs experts with varying expertise and cost: from DB admins (accurate, medium cost), researchers (possibly accurate, expensive, paid in citations), to stackoverflow participants (unpredictable accuracy, low cost). Hybrid is always better.

In case hybrid does not perform well, or the query result is "incorrect", `CIDR`^{2} uses adversarial generation (often called data cleaning) to modify the database contents so that the data consistent with the (now correct) results.

[1] Just the query specification was 1400+ pages over 10 years ago and its size has certainly withstood the test of time.
[2] References available after publication as a generative RNN model.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235