**Overview of Approach**

Internally, the **ANYTIME LOOP** controls the interaction with the user and the resolution refinement. It uses the **PLAN GENERATOR** as a sub-function to refine the plan Pareto frontier approximation by generating new plans. The plan generator is incremental and indexes plans as candidates that might become useful in the future.

**Explanation.** After entering a query, the **USER** sees a continuously refining approximation of the query plan Pareto frontier. The user can set cost bounds to focus optimization on interesting segments of the Pareto frontier. Finally, the user selects the preferred cost tradeoff from the Pareto frontier.

**User**

- **Select Cost Tradeoff**
- **Visualize Pareto Frontier**

**Plan Generator**

- **Retrieve Candidates**
- **Prune Candidates**
- **Generate New Plans**

**Interactive Optimizer**

- **Approximate Pareto Frontier**
- **Refine Approximation**
- **User Input?**
- **Bound Change**
- **Tradeoff Selection**
- **Return Plan**

**Example Session**

We execute query plans in the Cloud and compare plans based on their execution time and their monetary execution fees.

After entering a query, the user obtains quickly a coarse-grained approximation of optimal cost tradeoffs within the default cost bounds:

- **Plan cost**
- **Cost bounds**

Without user interaction, the approximation is continuously refined:

- **Plan cost**
- **Cost bounds**

The user may adapt the bounds at any time and the visualization is quickly updated:

- **Plan cost**
- **Old bounds**
- **New bounds**

Optimization ends once the user selects his preferred cost tradeoff.

**Anytime Property**

Calculating a fine-grained approximation of the Pareto frontier takes a lot of time. To allow for a responsive user interface, our optimizer calculates a series of approximations with increasing resolution. We call this the anytime property.

**Incrementality**

The optimizer is invoked many times for the same query, for different resolution levels and cost bounds. We must avoid regenerating the same query plans over and over again. Our optimizer is incremental: it generates plans only once and discards them only if they cannot be relevant for future optimizer invocations.

**Goals**

Users do not specify their preferences beforehand but rather pick their preferred cost tradeoff from a visualization of available tradeoffs. Query optimization feels similar to using a hotel booking Web site: Users dynamically adapt their constraints based on a continuously refining visualization of optimal cost tradeoffs.

**Context**

In multi-objective query optimization, we search the query plan that represents the best tradeoff between different cost metrics such as execution time, monetary fees, energy consumption, or result precision. The best tradeoff is defined by user preferences. Prior approaches let users formalize their preferences before optimization starts. This is however tedious and error-prone.