MetaEase: Heuristic Analysis from Source Code via Symbolic-Guided Optimization

Pantea Karimi[1]

Siva Kesava Reddy Kakarla[2], Ryan Beckett[2], Santiago Segarra[3],

Pooria Namyar[2], Mohammad Alizadeh[1], Behnaz Arzani[2]

1 MIT CSAIL 2 Microsoft Research 3 RICE

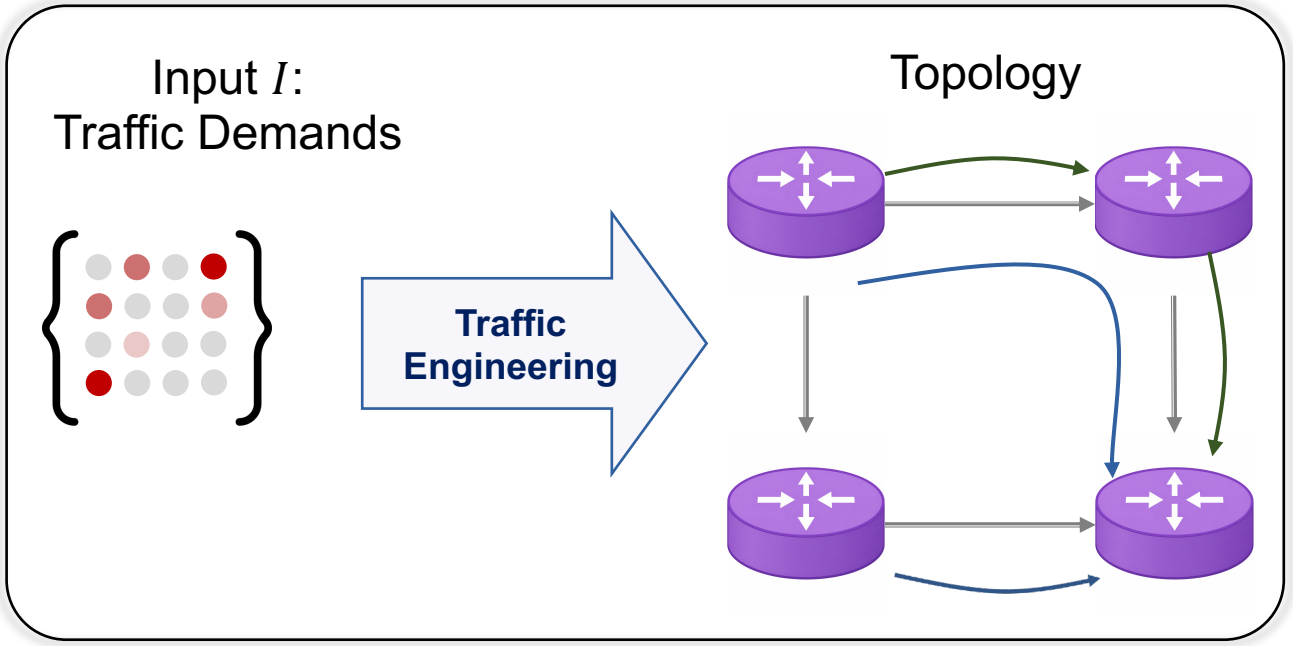
Heuristics Are Everywhere, Yet Can Break Unexpectedly

Risk of underperformance and inefficiency

Approximate Solution
(Heuristic)

Example of heuristics:
Demand Pinning in Traffic Engineering
First-Fit-Decreasing in VM Placement
...g

Traffic Engineering Example



Performance objective:
Maximize total routed traffic

$Opt(I)$ → Optimal's Performance on input I

Heuristic

Demand Pinning (DP)

$Heuristic(I)$ → DP's Performance on input I

If demand $\leq P$: *pin to shortest path*
else: *route optimally*

Performance Gap

$$Gap(I) = Opt(I) - Heuristic(I)$$

Demand Pinning *may* have 30% performance gap for some topologies!

Heuristic Analysis: Find Worst-Case Performance Gap

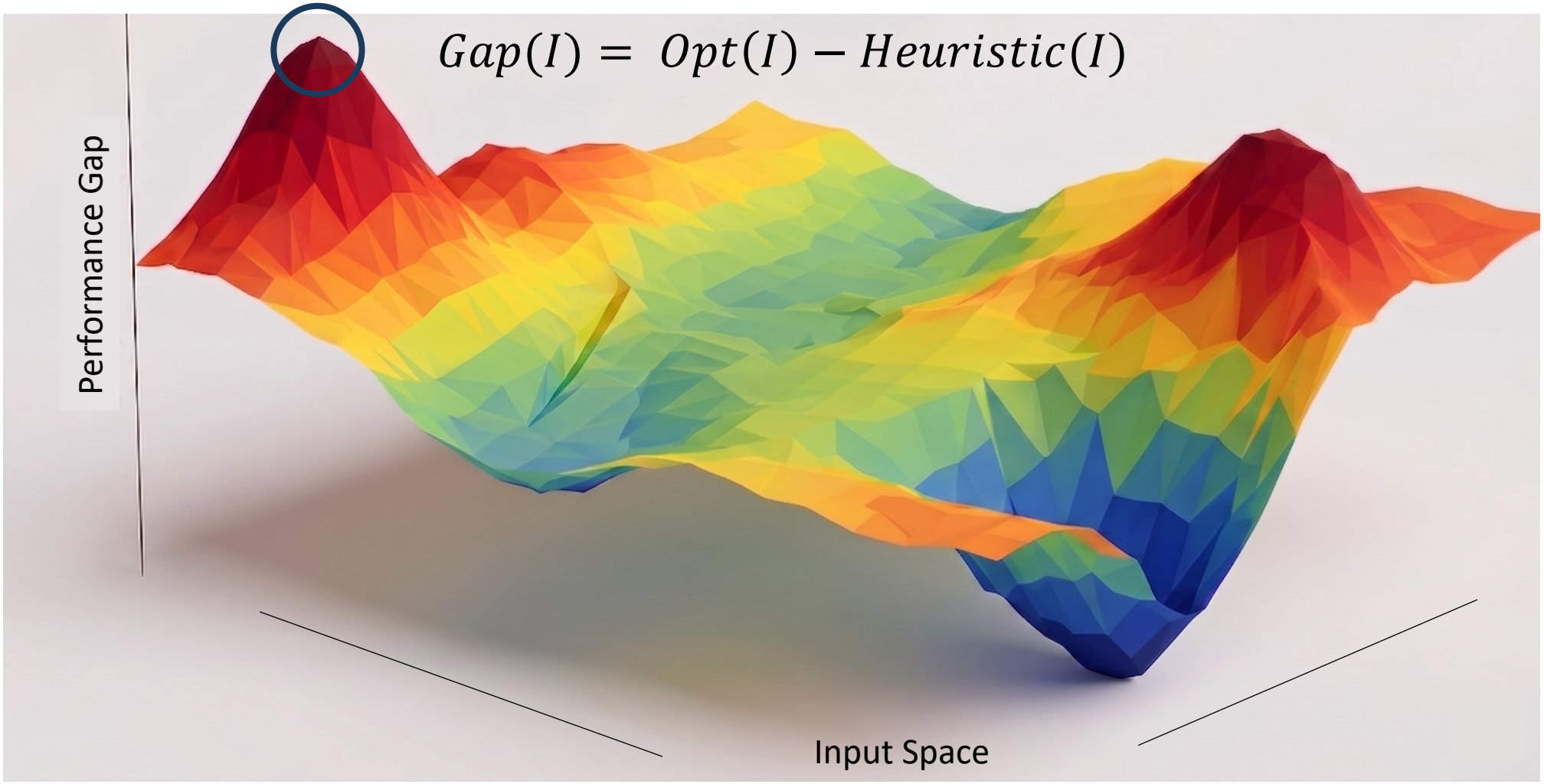


Image generated by Gemini

State-of-the-Art Solutions for Heuristic Analysis

Example: MetaOpt [NSDI ’24], FPerf[NSDI ‘23], Xplain [Hotnets’24]

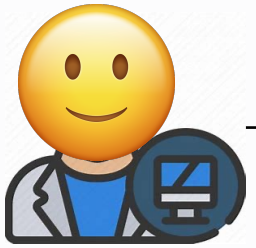
Encode any new heuristic into a **mathematical format** (e.g., optimization, SAT)

Hard to Use or Domain Specific

2 lines of logic → bi-level optimization + binary variables + big-M, … **~1-2 days, domain expert.**

MetaEase: The First Heuristic Analyzer from Code

Just bring ***heuristic code***
No modeling or encoding



```
for demand in demands:
    if demand <= P:
        # Pin to shortest path
        allocate_shortest_path(demand)
    else:
        # Route remaining demands optimally
        optimization_set.add(demand)

solve_optimization(optimization_set)
```

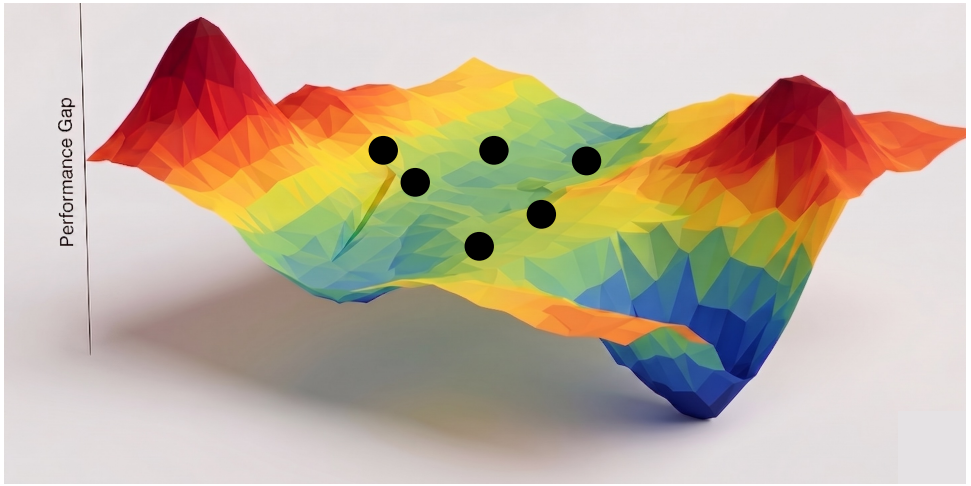
Heuristic Code

Opt Formulation

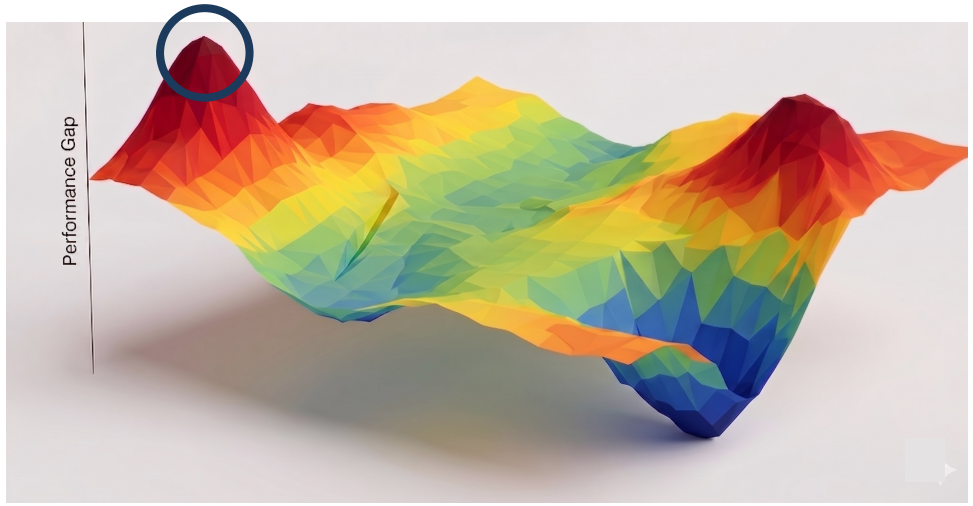
Fix, usually a well-known optimization

Key Challenges

$Gap(I)$ sampling is expensive



Avoid bad local Maxima

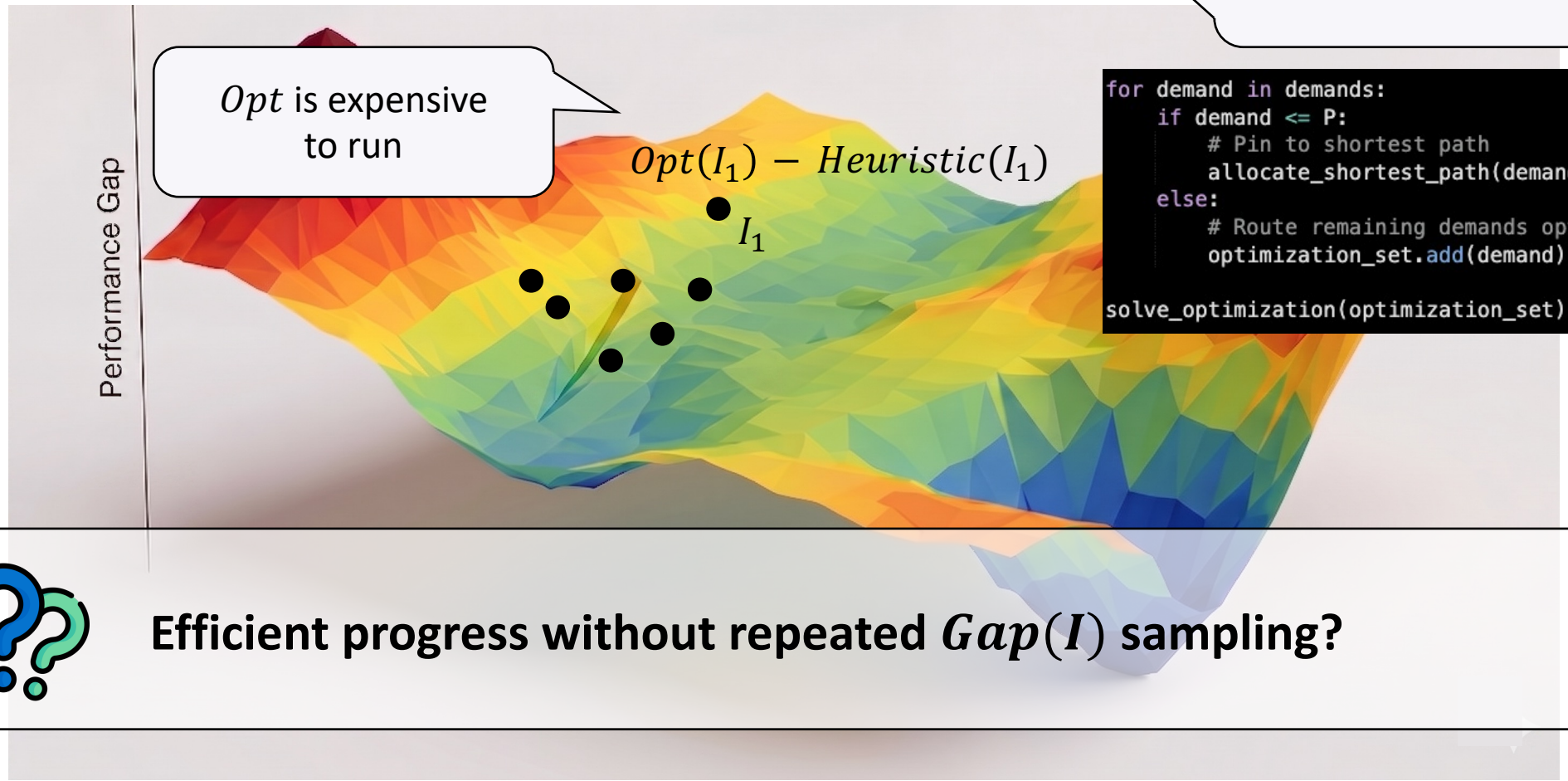


Challenge 1: Sampling Gap Function Is Expensive

$$Gap(I) = Opt(I) - Heuristic(I)$$

No encoding or model is available

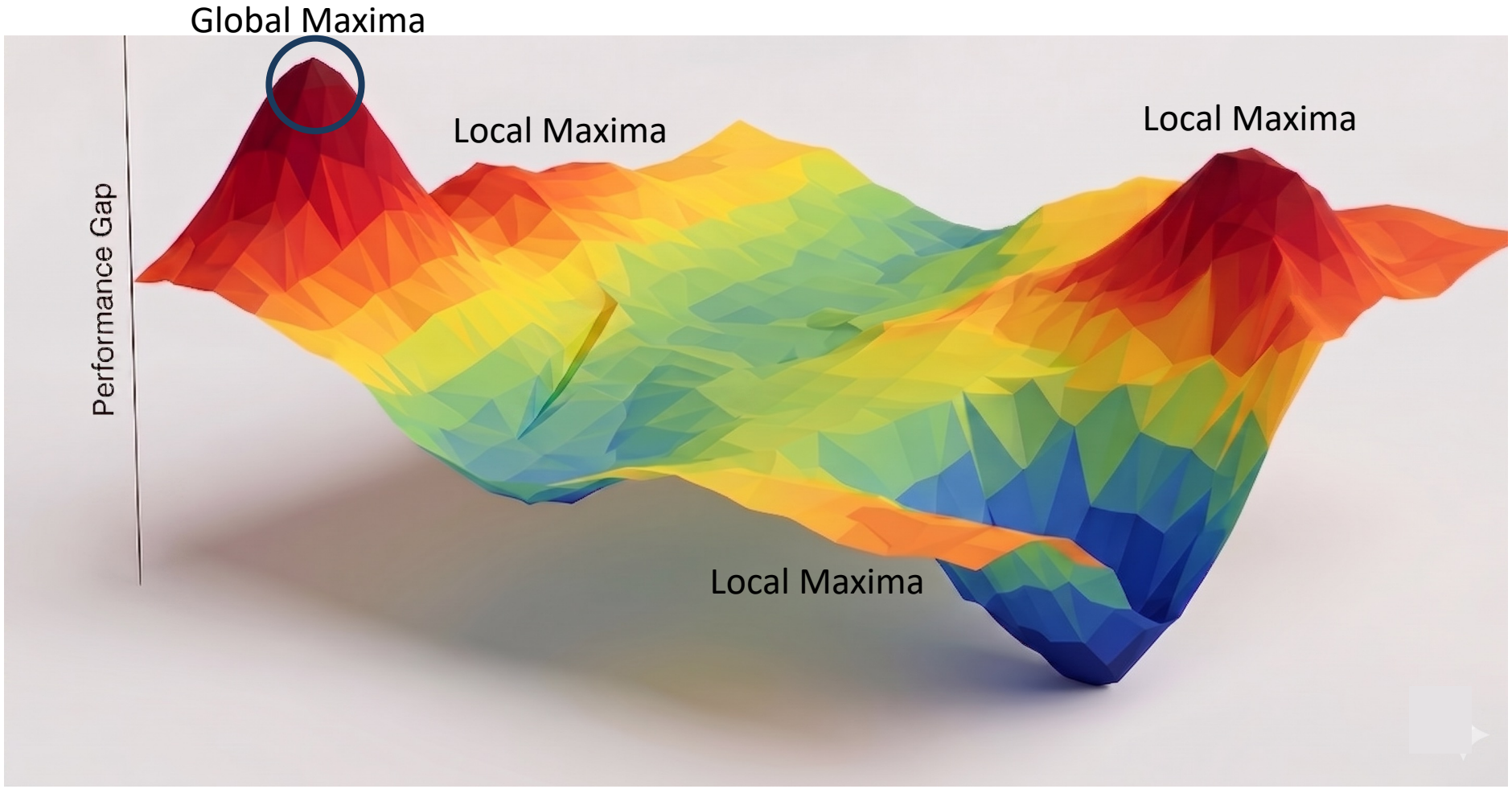
Opt is expensive to run




```
for demand in demands:
    if demand <= P:
        # Pin to shortest path
        allocate_shortest_path(demand)
    else:
        # Route remaining demands optimally
        optimization_set.add(demand)

solve_optimization(optimization_set)
```

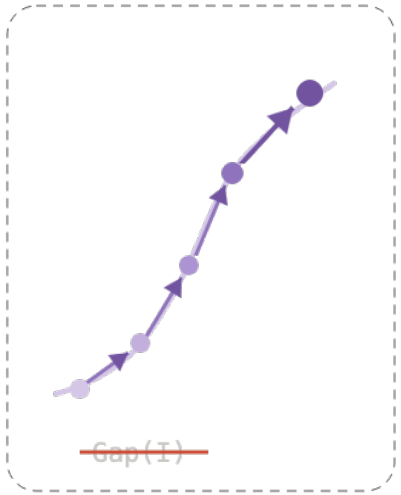
Efficient progress without repeated $Gap(I)$ sampling?

Challenge 2: Gap Function Can Have Local Maxima



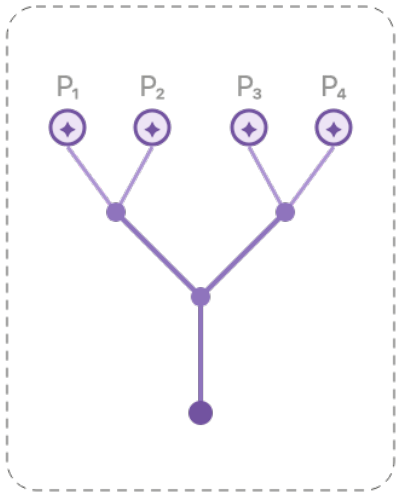
MetaEase Mechanisms

$Gap(I)$ sampling is expensive



Gradient ascent

Avoid bad local Maxima



Smart seeds

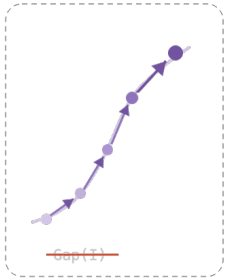
Gradient Ascent to Avoid Repeated Gap Sampling

$$Gap(I) = Opt(I) - Heuristic(I)$$

$$\nabla Gap(I) = \nabla Opt(I) - \nabla Heuristic(I)$$

MetaEase: Compute efficiently separately, subtract, use as gradient direction

Estimate ∇Opt without **repeatedly solving** Opt, using duality theory.
(we can take a step toward the optimal direction, but finding the exact Opt value takes many more steps.)



Gradient ascent

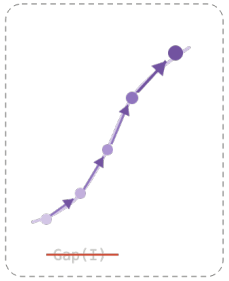
How MetaEase Computes ∇ $Heuristic(I)$

$Heuristic(I)$

```
for demand in demands:
    if demand <= P:
        # Pin to shortest path
        allocate_shortest_path(demand)
    else:
        # Route remaining demands optimally
        optimization_set.add(demand)

solve_optimization(optimization_set)
```

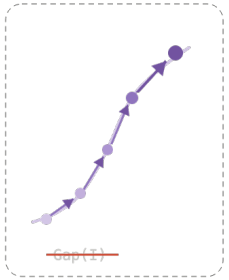
This is arbitrary code, how do we efficiently get the gradient?

Insight: Gaussian Process (GP) Surrogate can approximate it locally as a differentiable function.



Gradient ascent

MetaEase Uses GP Surrogate for $\nabla\, Heuristic(I)$

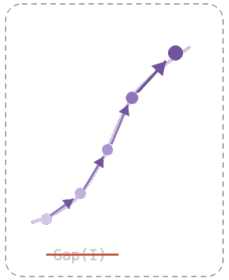


Gradient ascent

Fast and Sample Efficient

Graph-based Deterministic Policy Gradient for Repetitive Combinatorial Optimization Problems. *Zhongyuan Zhao et al*

MetaEase’s Gradient Ascent Steps



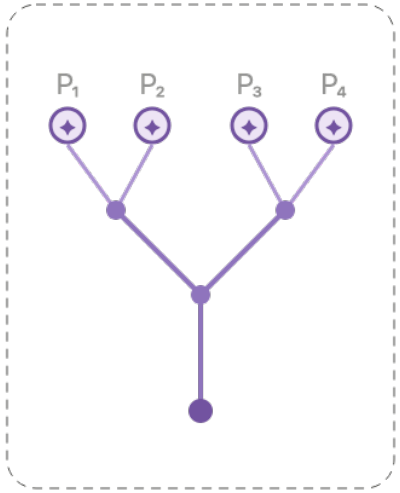
Gradient ascent

MetaEase Mechanisms

No repeated $Gap(I)$ sampling

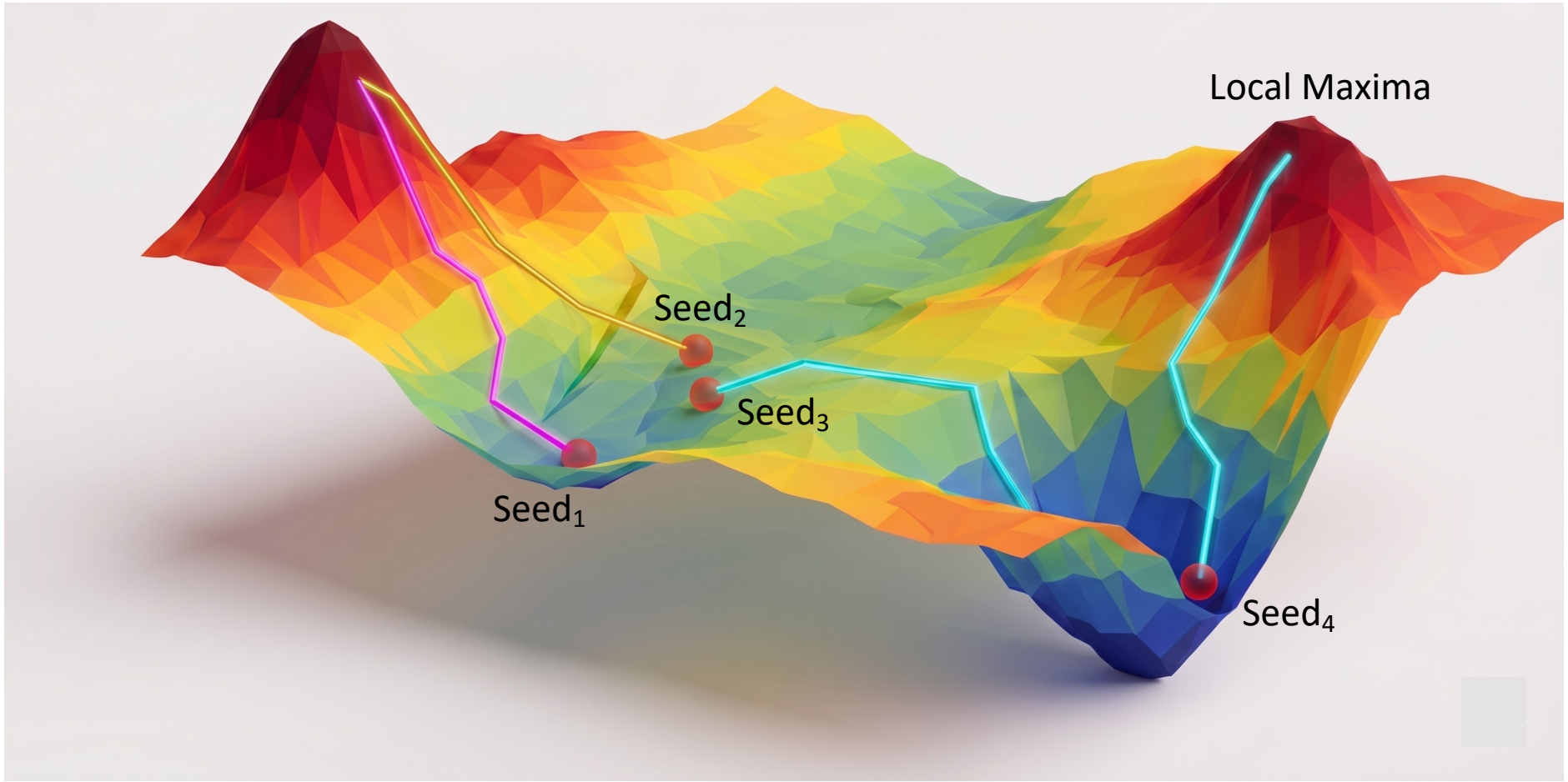
Gradient ascent

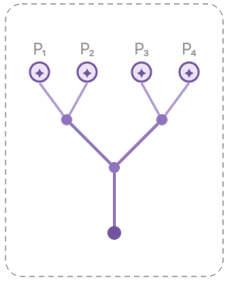
Avoid bad local Maxima



Smart seeds

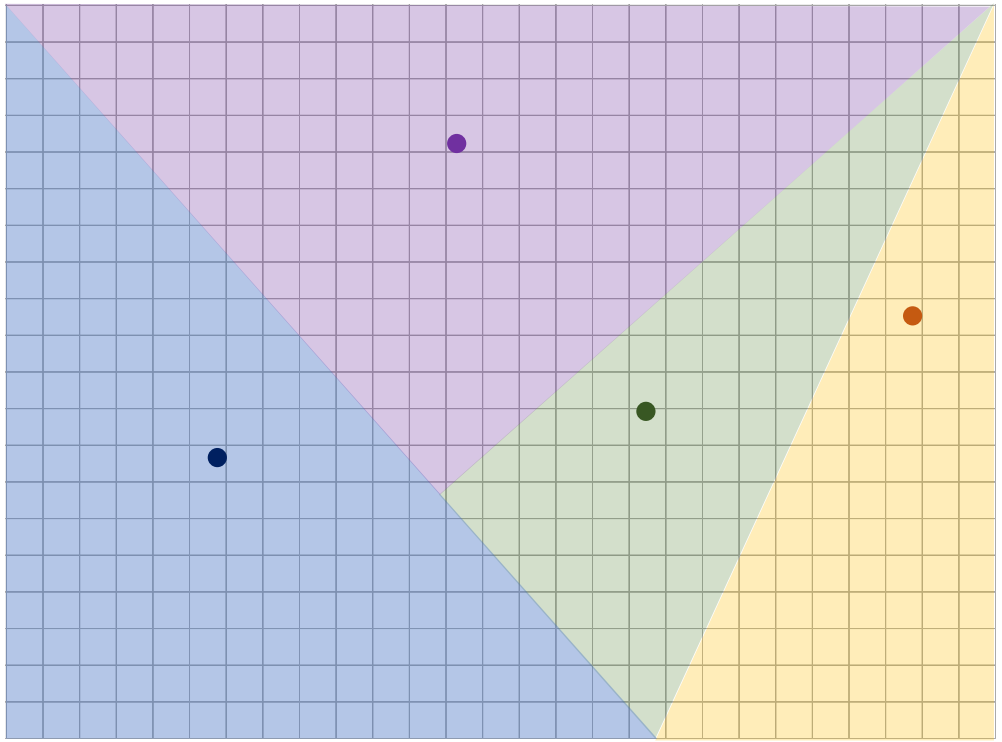
Selecting Smart Seeds Matters

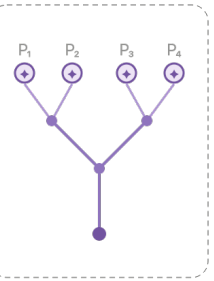




Smart seeds

Seed Different Heuristic Behavior to Avoid Local Maxima

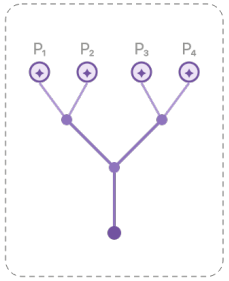




Smart seeds

In formal verification, these regions are equivalence classes.

Code Paths are Good Equivalence Classes for Heuristic

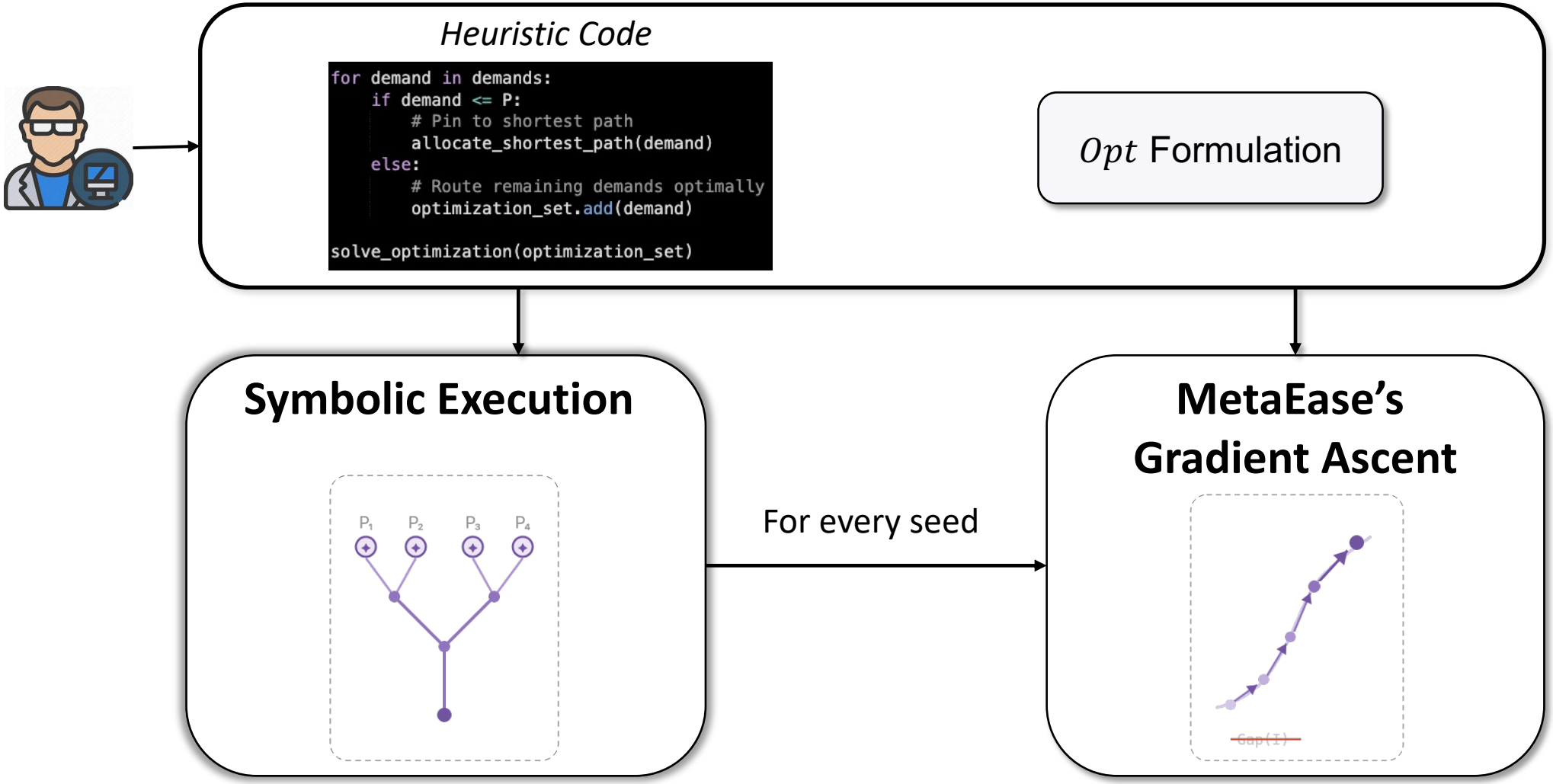


Smart seeds

MetaEase uses Symbolic Execution (KLEE) to find Seeds.

What About the Cliffs?

MetaEase: Putting it All Together



Comparable Results to State-of-the-Art (MetaOpt)

MetaEase: Demand Pinning as Code

```
for demand in demands:
    if demand <= P:
        # Pin to shortest path
        allocate_shortest_path(demand)
    else:
        # Route remaining demands optimally
        optimization_set.add(demand)

solve_optimization(optimization_set)
```

MetaOpt: Demand Pinning as Optimization

Outer Problem (Leader):

$$\max_{\mathbf{I}\in\mathbb{R}^n} \quad \mathrm{OPT}(\mathbf{I}) - \mathrm{DP}(\mathbf{I})$$
$$\text{s.t.} \quad 0 \le I_k \le D_{\max} \quad \forall\, k$$

Inner Problem 1 — OPT (Max-Flow LP):

$$\mathrm{OPT}(\mathbf{I}) = \max_{f'} \quad \sum_k \sum_{p\in\mathcal{P}_k} f'_{k,p}$$
$$\text{s.t.} \quad \sum_k \sum_{\substack{p\in\mathcal{P}_k \\ e\in p}} f'_{k,p} \le \mathrm{Cap}_e \quad \forall\, e$$
$$\sum_{p\in\mathcal{P}_k} f'_{k,p} \le I_k \quad \forall\, k$$
$$f'_{k,p} \ge 0$$

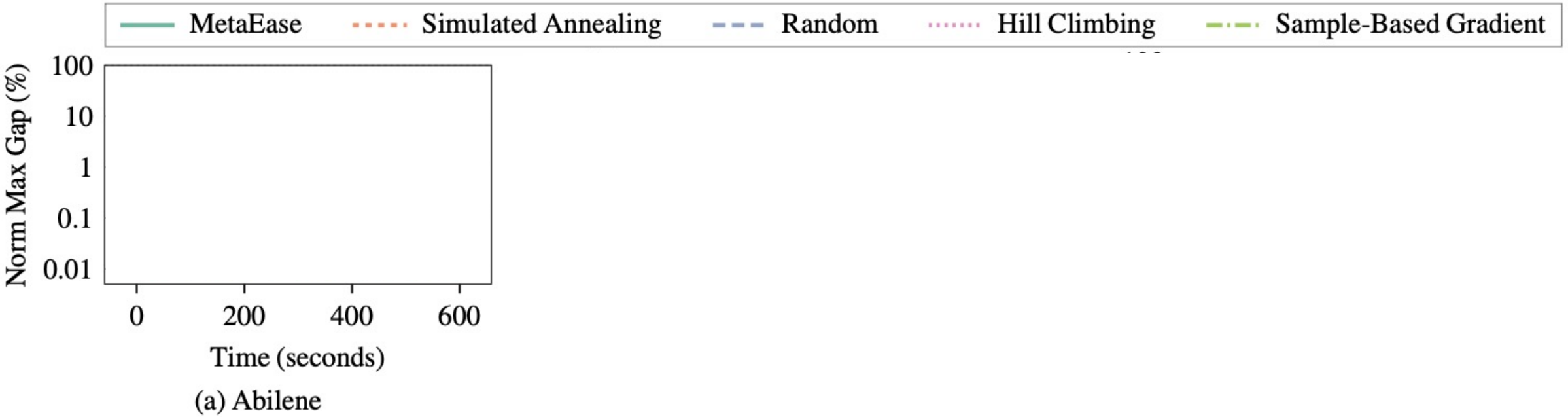
Inner Problem 2 — DP (Conditional + Max-Flow):

$$b_k \in \{0,1\} \qquad \text{(binary indicator)}$$
$$I_k \le P + M\,(1-b_k) \quad \forall\, k \qquad \text{(big-}M\text{)}$$
$$I_k \ge P + \varepsilon - M\,b_k \quad \forall\, k \qquad \text{(big-}M\text{)}$$
$$f_{k,\hat{p}_k} = I_k \cdot b_k \qquad \text{(pin if small)}$$
$$f_{k,p} \le M\,(1-b_k) \quad \forall\, p \ne \hat{p}_k$$
$$\sum_k \sum_{\substack{p\in\mathcal{P}_k \\ e\in p}} f_{k,p} \le \mathrm{Cap}_e \quad \forall\, e$$
$$\sum_{p\in\mathcal{P}_k} f_{k,p} \le I_k \quad \forall\, k$$

Then rewrite to single-level via KKT:

+ dual variables λ_e, μ_k for each constraint
+ stationarity: $\frac{\partial \mathcal{L}}{\partial f_{k,p}} = 0 \quad \forall\, k,p$
+ complementary slackness: $\lambda_e \cdot g_e = 0$
+ nonlinear terms requiring SOS / big-M

MetaEase Outperforms Black-Box Baselines



(a) Abilene

Unique Capability: Analyzing any Performance Heuristic

DNN-Based Heuristic:
DOTE [NSDI'23] in Traffic Engineering

MetaEase > Specialized DNN Analyzer HotNets'24

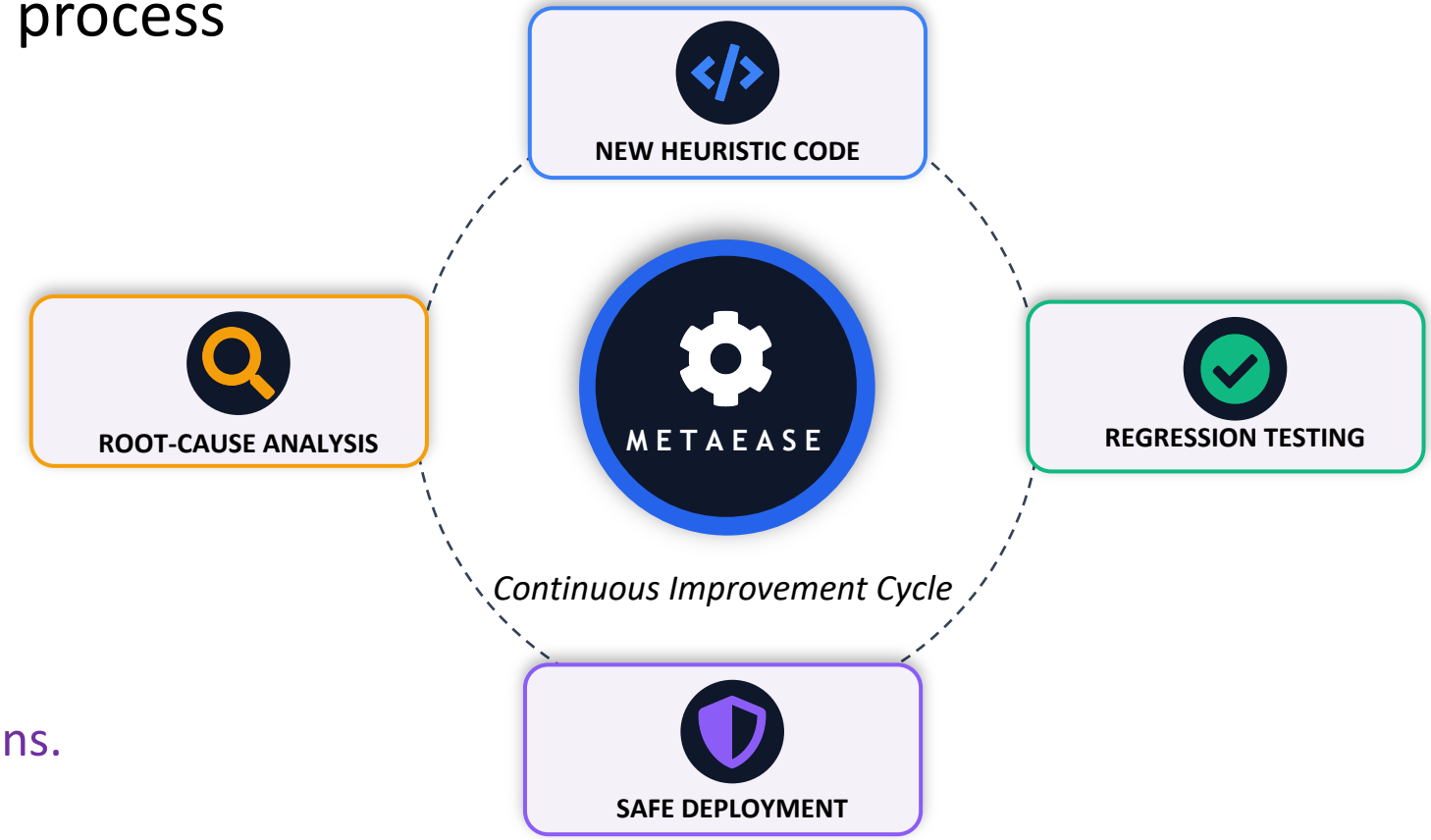
Randomized Heuristic:
Arrow [SIGCOMM'21] in Optical-IP TE

MetaEase > Black-Box

MetaEase can analyze heuristics that other analyzers cannot analyze or require difficult encoding.

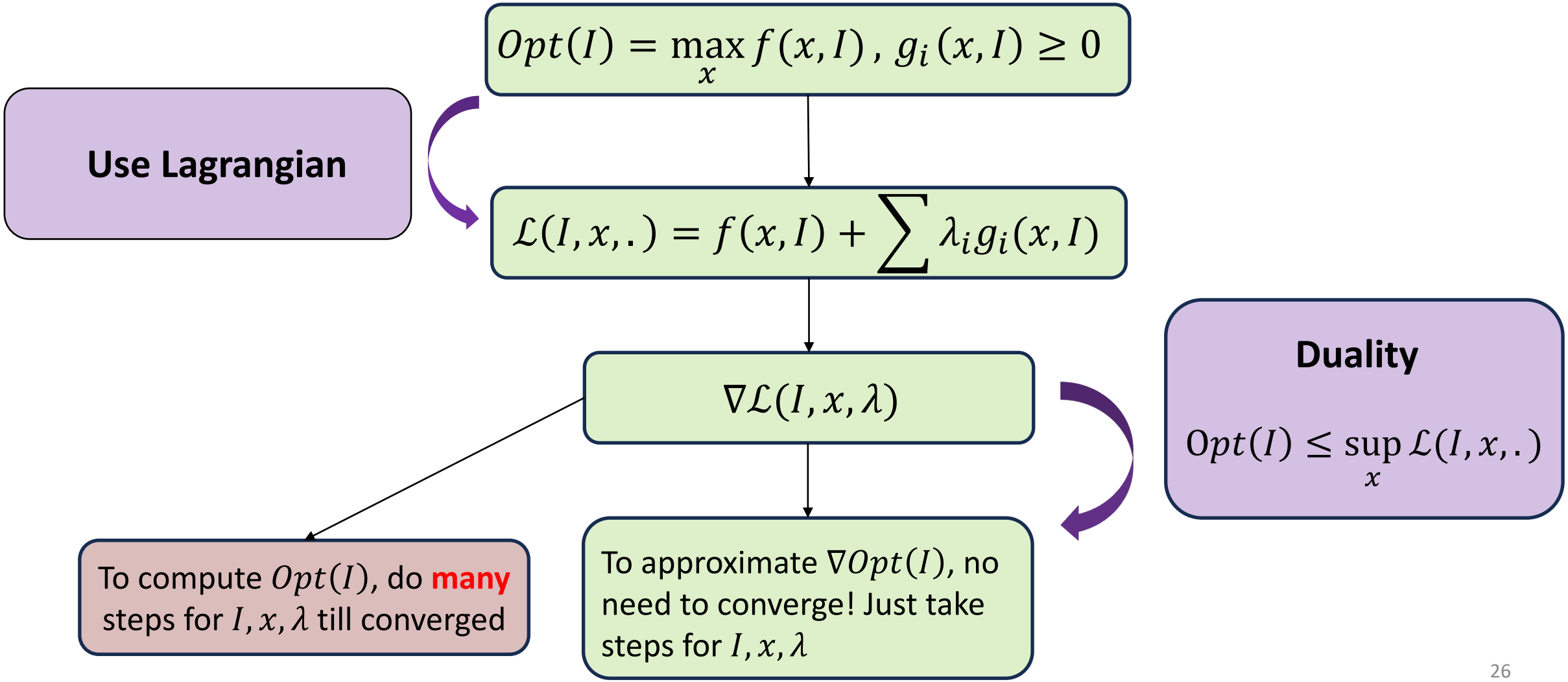
Conclusion

- First general-domain heuristic analyzer from code
- It matches or beats the SOTA across 5 domains and 8 heuristics
- MetaEase part of a design process



Thanks! I would love to answer your questions.
Contact: pkarimib@mit.edu

How MetaEase Computes $\nabla\, Opt(I)$



How MetaEase Gets the Optimal

give me the traffic engineering optimal formulation in python

5:21 PM 2 / 2

Architected network optimization solution using linear programming method…

Here's a classic traffic engineering (TE) optimal formulation — a multi-commodity flow problem minimizing maximum link utilization, implemented with `scipy.optimize.linprog`.

Marshaled traffic engineering optimization framework for Python implement…

The solver runs on a 4-node diamond topology and finds the optimal MLU of **87.5%** — the LP splits traffic across both paths (0→1→3 and 0→2→3, plus the cross-link 1→2) to balance utilization evenly across the bottleneck links.

The formulation covers the standard TE building blocks:

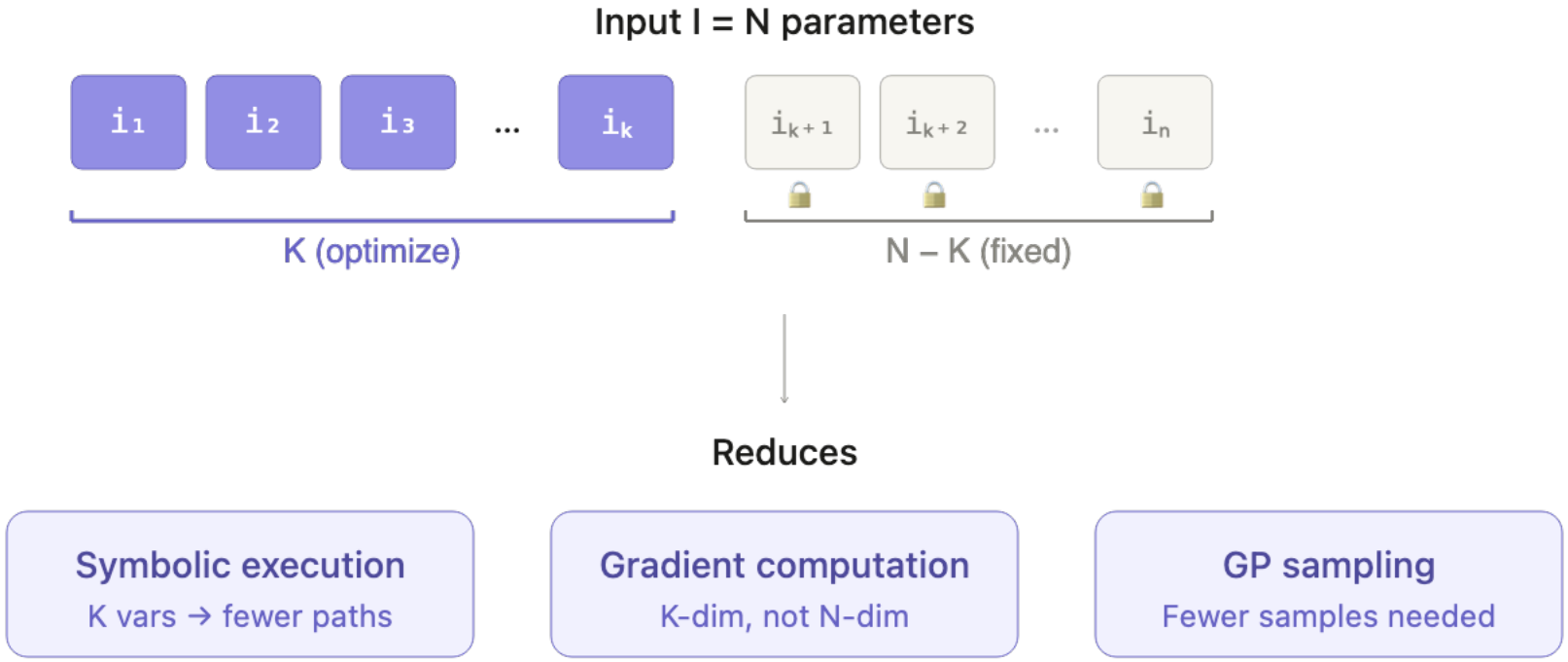
- **Objective:** minimize the maximum link utilization (u_max) across all links
- **Flow conservation:** at every node, incoming flow minus outgoing flow equals the demand (source/sink/transit)
- **Capacity coupling:** total flow on each link ⩽ capacity × u_max, which forces u_max up to the worst-case link
- **Multi-commodity flow splitting:** each demand can split across

Write a message…

```
"""
Traffic Engineering Optimal Formulation
=======================================
Multi-commodity flow LP that minimizes maximum link utilization (MLU).

Formulation:
  minimize   u_max
  subject to:
    (1) Flow conservation at each node for each demand
    (2) Link capacity: sum of flows on link (i,j) <= capacity(i,j) * u_max
    (3) Demand satisfaction: total flow for demand k = demand_k
    (4) Non-negativity: all flows >= 0, u_max >= 0

This is the standard Fortz-Thorup style TE objective used widely in WAN
and ISP backbone optimization.
"""

import numpy as np
from scipy.optimize import linprog
from dataclasses import dataclass


# ── Data Structures ──────────────────────────────────────────────

@dataclass
class Link:
    src: int
    dst: int
    capacity: float


```

Optimizations for Scalability

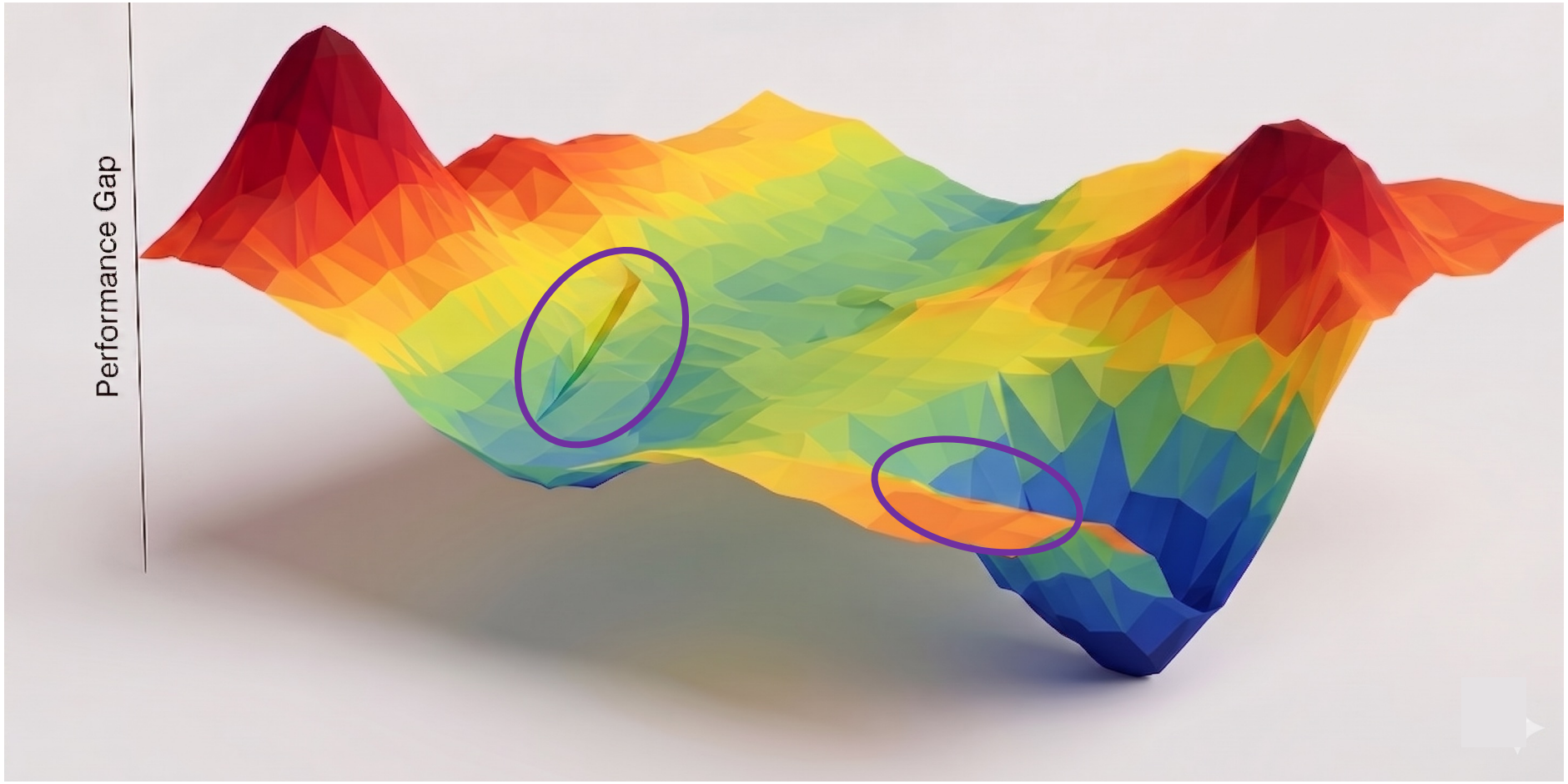
- Projected gradient ascent: Update K parameters at a time



Additional Optimizations for Scalability

- Adding domain customization “Hints”
  - Adding hints to prune and find harder samples
- Prune and prioritize equivalence classes

Challenge 3: Gap Function Can Be Non-Smooth



MetaEase Mechanisms

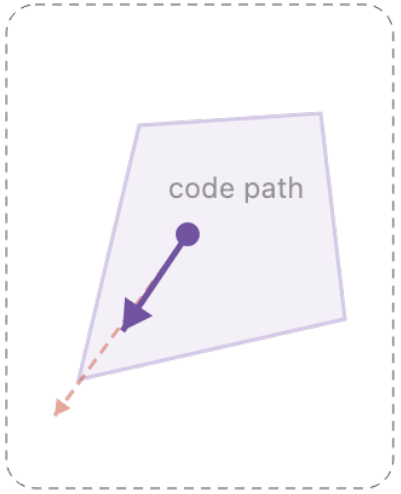
No repeated $Gap(I)$ sampling

Avoid bad local Maxima

Handle non-differentiability

Gradient ascent

Smart seeds



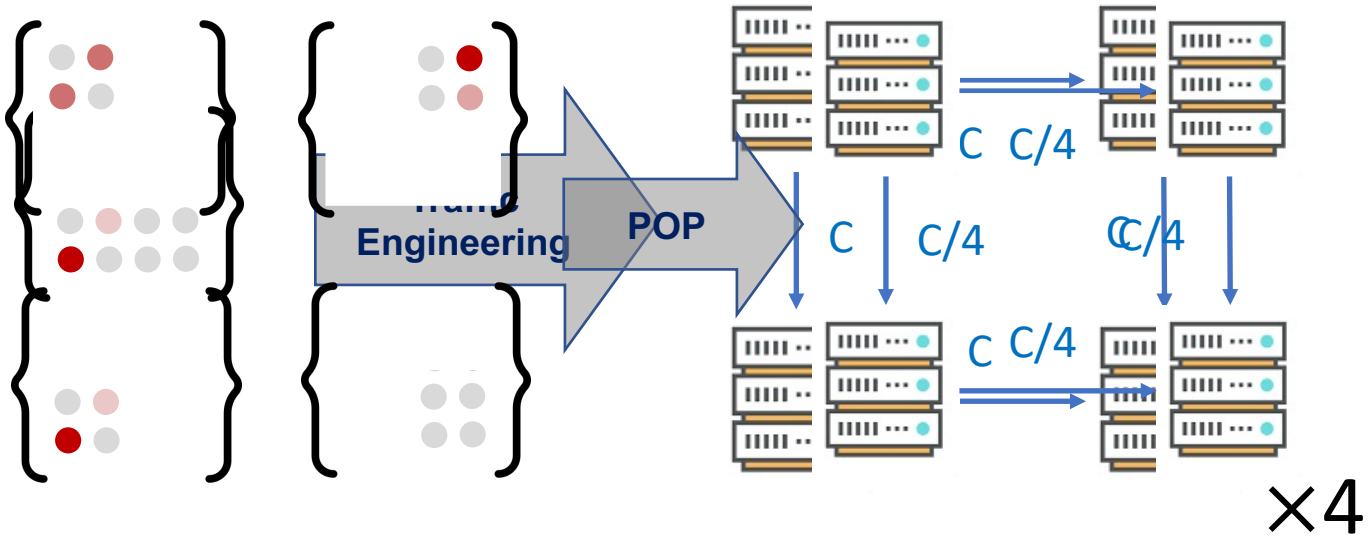
Path projection

MetaEase Projects the Gradients Back to Code Path

MetaEase vs MetaOpt Gap Ratio

$$Gap\ ratio = \frac{MetaEase\ Max\ Gap}{MetaOpt\ Max\ Gap}$$

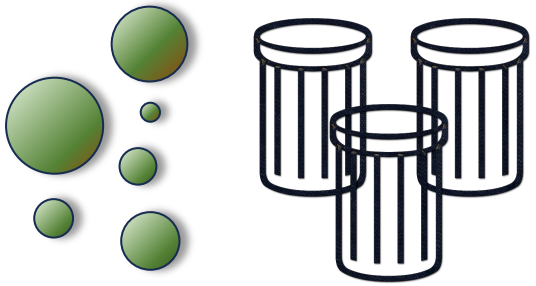
Heuristic \ Topo	Abilene	B4	Cogentco	Swan	Uninett2010
POP	**98%**	89%	88%	**91%**	65%



MetaEase vs MetaOpt - Vector Bin Packing

$$\frac{MetaEase\ Max\ Gap}{Metaopt\ Max\ Gap}$$

Heuristic \ #Balls, Dim	10, 1D	10, 2D	15-1D	15-2D	20-1D	20-2D
First-Fit Decreasing	**100%**	**66%**	**100%**	**75%**	**100%**	**250%**



Hints In MetaEase

```
for (int k = 0; k < K; ++k) {
  // Restrict
     each demand to a bounded interval.
  klee_assume(0 <= demand[k] && demand[k] <= D_MAX);
}

bool any_edge_overloaded = false;

for (int e = 0; e < num_edges; ++e) {
  // Compute the total
     load induced on edge e by all demands.
  int edge_load
     = sum_demands_on_edge(demand, edges[e]);

  // Track whether
     this input stresses at least one edge.
  // We want: exists
     e such that edge_load(e) > capacity(e).
  any_edge_overloaded = any_edge_overloaded
     || (edge_load > edges[e].Capacity);
}

// Hardness predicate:
// If no edge is overloaded
    , the instance is typically "easy",
// and is unlikely to expose a gap
     between the heuristic and the benchmark.
klee_assume(any_edge_overloaded);
```

Path-Aware Gradient Step

Algorithm 1: Path-aware gradient step in MetaEase

Input: Block size Δ, samples N, step size η

$h \leftarrow \text{path}(I)$ // current code path

$\mathbf{X} \leftarrow \{u \in \text{SAMPLEBOX}(I, \Delta) \mid \text{path}(u) = h\}$

$\mathbf{Y} \leftarrow [\text{Heuristic}(x) \mid x \in \mathbf{X}]$

$\text{Heuristic}_{GP} \leftarrow \text{FITGP}(\mathbf{X}, \mathbf{Y})$

Compute $g_b \leftarrow \nabla \text{Benchmark}(I)$

Compute $g_h \leftarrow \nabla \text{Heuristic}_{GP}(I)$

$g \leftarrow g_b - g_h$

$\tilde{I} \leftarrow I + \eta \cdot g$

if $\text{path}(\tilde{I}) = h$ **then**

  $I \leftarrow \tilde{I}$ // stay in same path

else

  Pick $z \in \mathbf{X}$ most aligned with g

  $I \leftarrow z$ // project back into path

Encoding LLM-Generated Heuristic In MetaEase

Code 1: New Sort-based Traffic Engineering Heuristic

```
double SortBasedHeuristic(Graph *G) {
   // collect and sort demands
   Demands D = positive_demands(G);
   sort_desc(D);

   // split top 20% as critical
   int k = max(1, 0.2 * size(D));
   Demands Critical = top_k(D, k);
   Demands Remaining = rest(D, Critical);

   // solve optimal on critical demands first
   Routing
     criticalRoutes = SolveOptimal(G, Critical);
   // Adjust the capacities
   Graph G = AdjustCapacities(G, criticalRoutes);
   // solve optimal on non critical demands
   Routing
     nonCriticalRoutes = SolveOptimal(G, Remaining);

   return G->total_met_demands;
}
```

Encoding LLM-Generated Heuristic In MetaOpt

Two-Stage Heuristic in MetaOpt — Selection & Stage 1

Input: $\mathcal{D}, \mathcal{E}, \{\mathcal{P}_k\}, \{\text{Cap}_e\}, \{d_k\}$
Vars: $f_{k,p} \geq 0, g_{k,p} \geq 0;\ z_k = \sum_{p \in \mathcal{P}_k} f_{k,p},\ t_k = \sum_{p \in \mathcal{P}_k} g_{k,p}$
Selection (Top-20%)

1: $\tau \leftarrow \lfloor 0.2|\mathcal{D}| \rfloor$

2: $r_k \leftarrow \texttt{Rank}\left(d_k, [d_i]_{i \in \mathcal{D}}\right)$

3: $c_k \leftarrow \texttt{IsLeq}(r_k, \tau) \quad (c_k \in \{0,1\})$

4: $\sum_k c_k = \tau$

5: $d_k^{\text{crit}} \leftarrow \texttt{Multiplication}(c_k, d_k)$

6: $d_k^{\text{non}} \leftarrow \texttt{Multiplication}(1 - c_k, d_k)$

Stage 1 (Critical on full capacity)
Constraints: $\forall e:\ \sum_k \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} f_{k,p} \leq \text{Cap}_e$

7: $z_k \leq d_k^{\text{crit}}$

8: $\texttt{IfThen}\left(1 - c_k, [(z_k, 0)]\right)$

Path aggregation

9: $z_k = \sum_{p \in \mathcal{P}_k} f_{k,p}$

Two-Stage Heuristic in MetaOpt — Residual & Stage 2

Residual capacity (from Stage 1 flows)

10: $\text{rawRes}_e \leftarrow \text{Cap}_e - \sum_k \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} f_{k,p}$

11: $\text{ResCap}_e \leftarrow \texttt{MAX}([\text{rawRes}_e], 0)$

Stage 2 (Non-critical on residual capacity)
Constraints: $\forall e:\ \sum_k \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} g_{k,p} \leq \text{ResCap}_e$

12: $t_k \leq d_k^{\text{non}}$

13: $\texttt{IfThen}\left(c_k, [(t_k, 0)]\right)$

Path aggregation

14: $t_k = \sum_{p \in \mathcal{P}_k} g_{k,p}$

Objective:

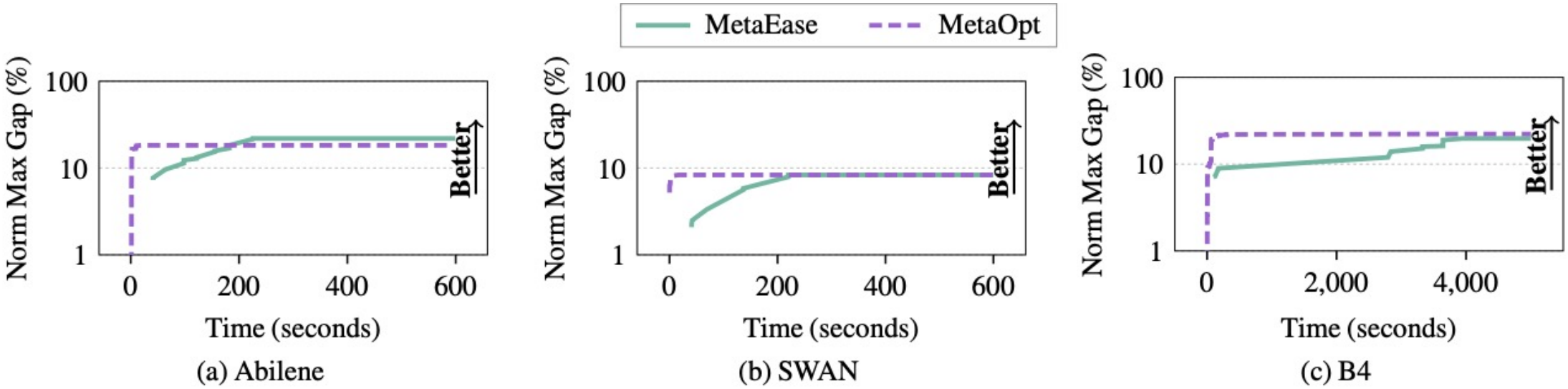
15: $\texttt{IfThenElse}\left(1, [\sum_k z_k + \sum_k t_k, \text{MaxFlow}()], [,]\right)$

(equivalently, maximize $\sum_k z_k + \sum_k t_k$*)*

Optional Big-M gating: $z_k \leq M c_k,\ t_k \leq M(1 - c_k)$; using `Multiplication` avoids tuning M.

Figure 19: Encoding of Traffic Engineering heuristic in MetaOpt. This heuristic selects the top-20% of demands as critical (1) route critical set optimally, (2) then route remaining non-critical demands on residual capacities. This required a lot of attention and LLMs can't do it without human supervision to check for correctness.

MetaEase vs MetaOpt Progress Over Time



MetaEase vs Black-Box

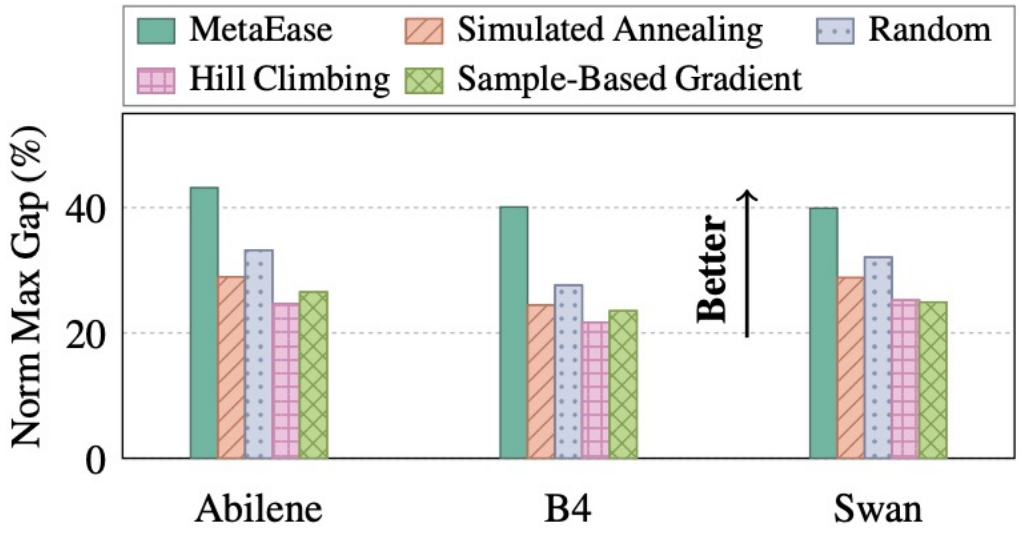


Figure 10: MetaEase finds a larger performance gap than black-box baselines for POP across production topologies under the same time budget.

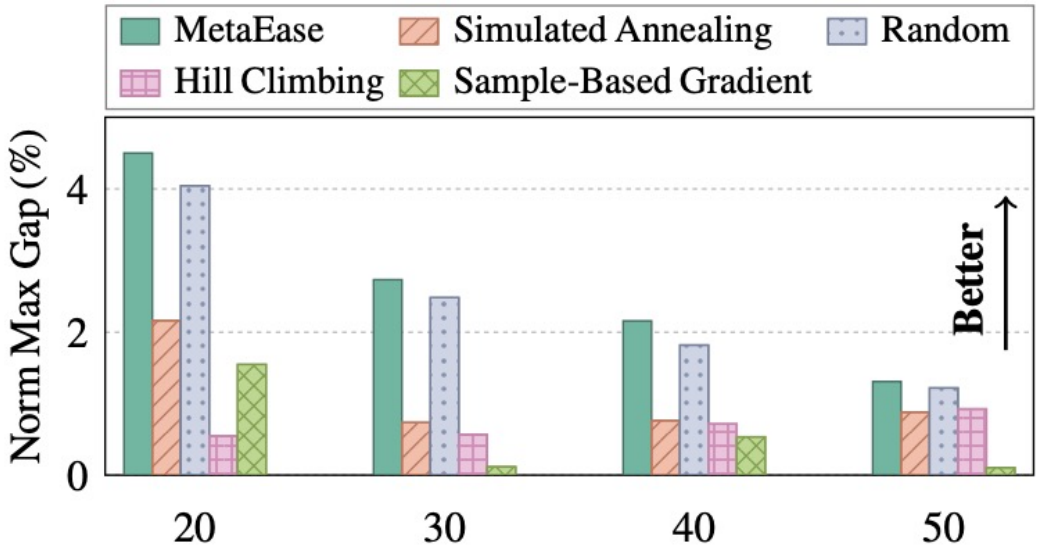


Figure 11: MetaEase outperforms all the baselines when it analyzes knapsack problems for 20, 30, 40, and 50 items. We normalized the gaps by number of items multiplied by the upper bound on each item's weight.

MetaEase Ablations

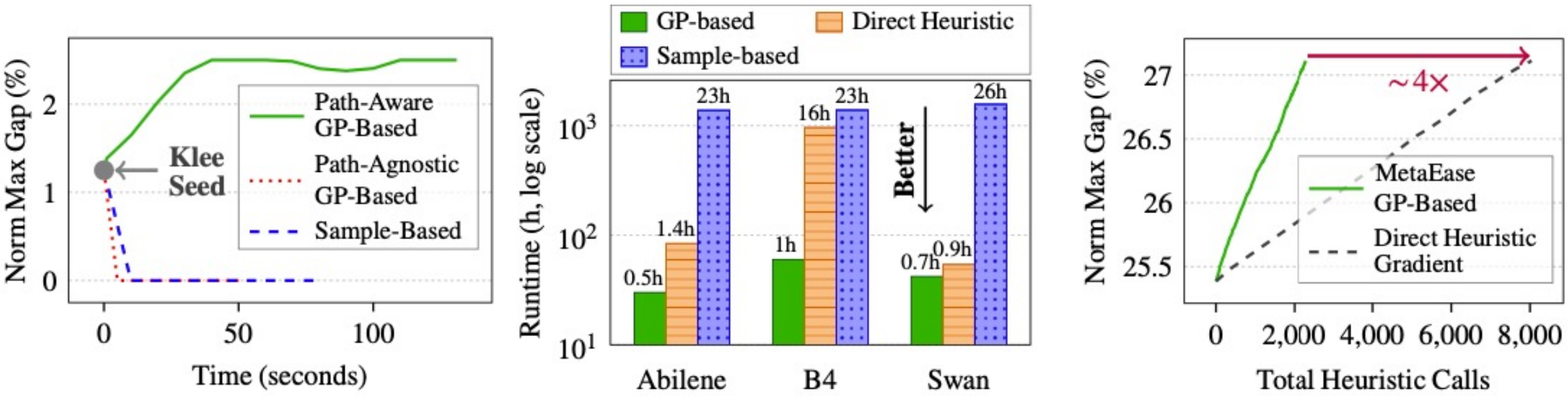


Figure 15: "Path-based" and "Surrogate-based" methods are effective: (left) MetaEase's gradient ascent improves upon the initial point and handles the discontinuities; (middle) surrogate-based is faster than sample-based variants, including vanilla sample-based and one where it only estimates heuristic's gradient; (right) MetaEase reduces the number of heuristic calls.

MetaEase: Putting It All Together

Simple interface for developers

Heuristic Code

```
for demand in demands:
    if demand <= P:
        # Pin to shortest path
        allocate_shortest_path(demand)
    else:
        # Route remaining demands optimally
        optimization_set.add(demand)

solve_optimization(optimization_set)
```

$Opt(I)$ *formulation*

Seed Generator (Symbolic Execution)

For every seed

Gradient Ascent

Smart seeds based on heuristic behavior

Efficient compute of gradients and non-differentiability handling

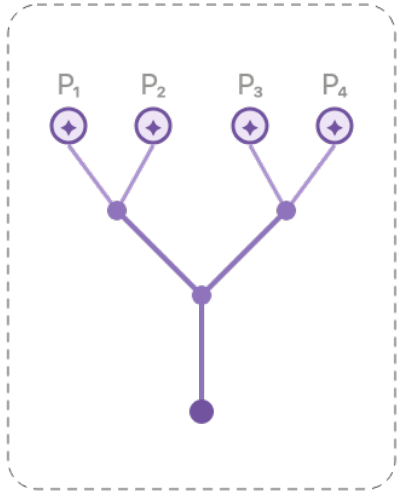
MetaEase Mechanisms

No repeated $Gap(I)$ sampling

Avoid bad local Maxima

Handle non-differentiability

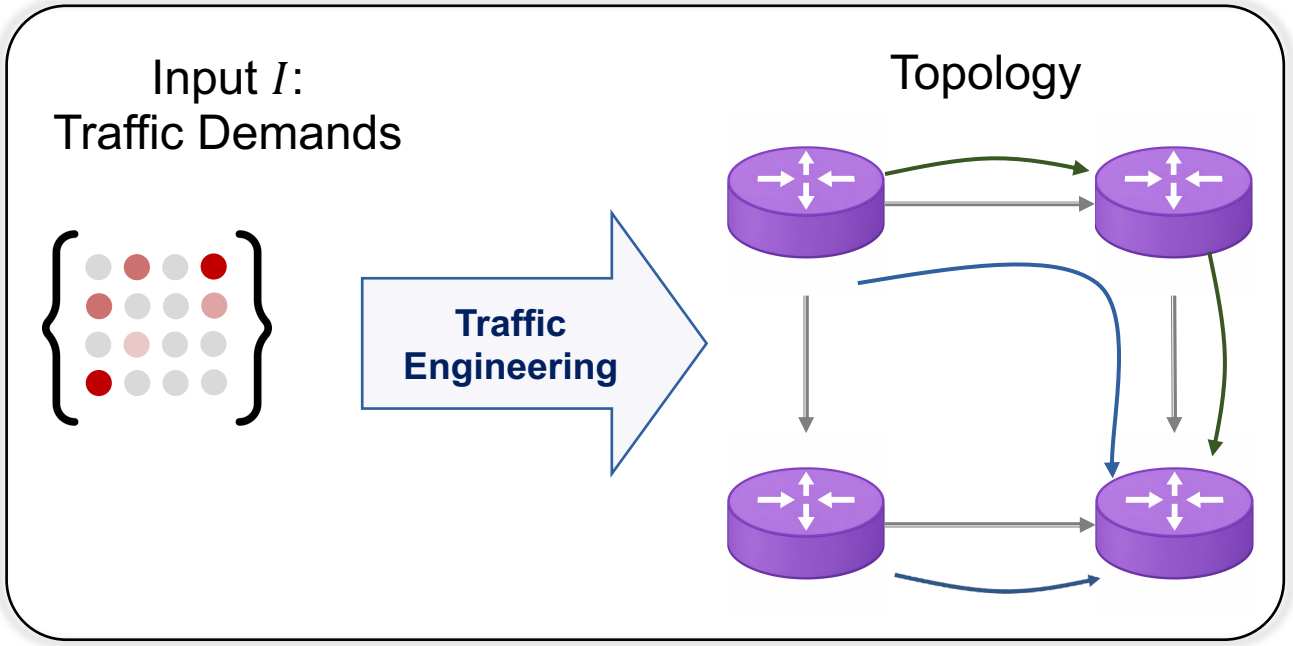
Gradient ascent



Smart seeds

Path projection

Traffic Engineering Example



$Opt(I)$ → Optimal Performance on input I

$Heuristic(I)$ → DP's Performance on input I

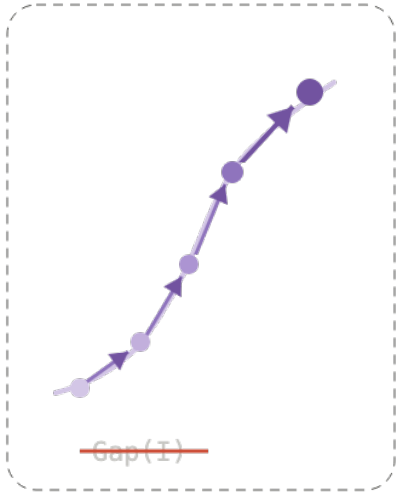
Performance Gap

$$Gap(I) = Opt(I) - Heuristic(I)$$

Demand Pinning *may* have 30% performance gap for some topologies!

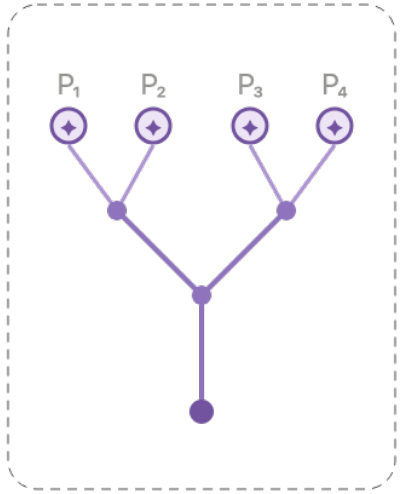
MetaEase Mechanisms

No repeated $Gap(I)$ sampling



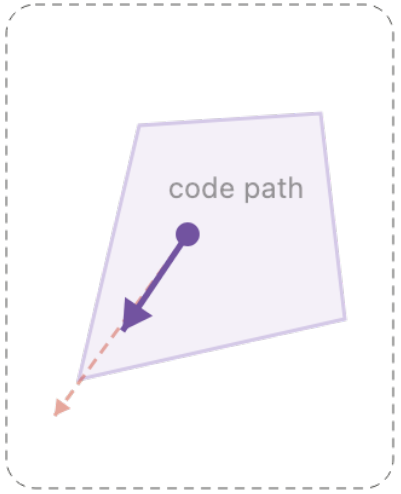
Gradient ascent

Avoid bad local Maxima



Smart seeds

Handle non-differentiability



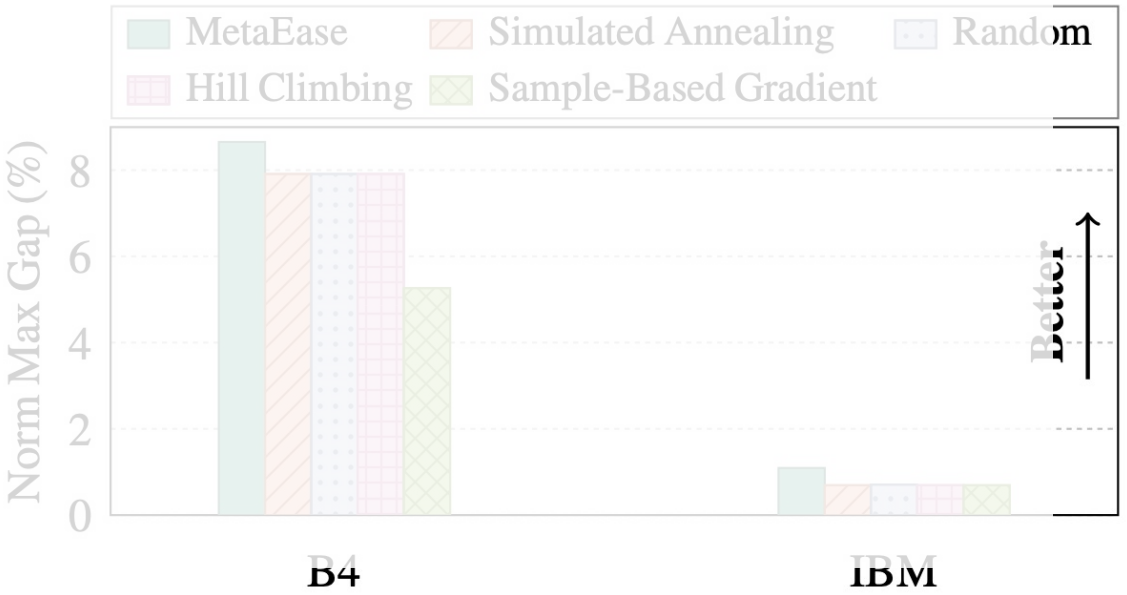
Path projection

Unique Capability: Analyzing Any Numeric Heuristic

DNN-Based Heuristic:
DOTE [NSDI'23] in Traffic Engineering

Method	Normalized Max Gap
MetaEase	73.13%
DNN Analyzer in [59]	71.84%
Random	63.02%
Simulated Annealing	61.39%
Hill Climbing	58.78%
Sample-based Gradient	58.78%

Randomized Heuristic:
Arrow [SIGCOMM'21] in Optical-IP TE



MetaEase can analyze heuristics that other analyzers cannot analyze or require difficult encoding.

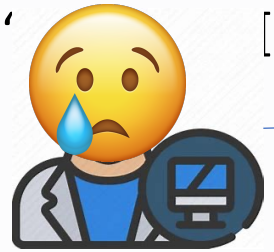
Existing Solutions

Model-based Tools

Encode any new heuristic into a mathematical format

Example: MetaOpt [NSDI '24], FPerf[NSDI ' [Hotnets'24]

✗ **Hard To Use**
Domain Specific



Encode

OuterVar: Y(size of ball
Input: C(capacity of bins)
for all ball i **and** bin j **do**

$$r_{ij} = C_j - Y_i - \sum_{\text{ball } u<i} x_{uj}^d$$

$$f_{ij} = \text{AllLeq}([-r_{ij}^d]_d, 0)$$

$$\gamma_{ij} = \text{AllEq}([x_{ik}^d]_{d,k<j}, 0)$$

$$\alpha_{ij} = \text{AND}(f_{ij}, \gamma_{ij})$$

$$\text{IfThenElse}(\alpha_{ij}, [(x_{ij}, Y_i)], [(x_{ij}, 0)])$$

end for

This is hard!

Black-Box Tools

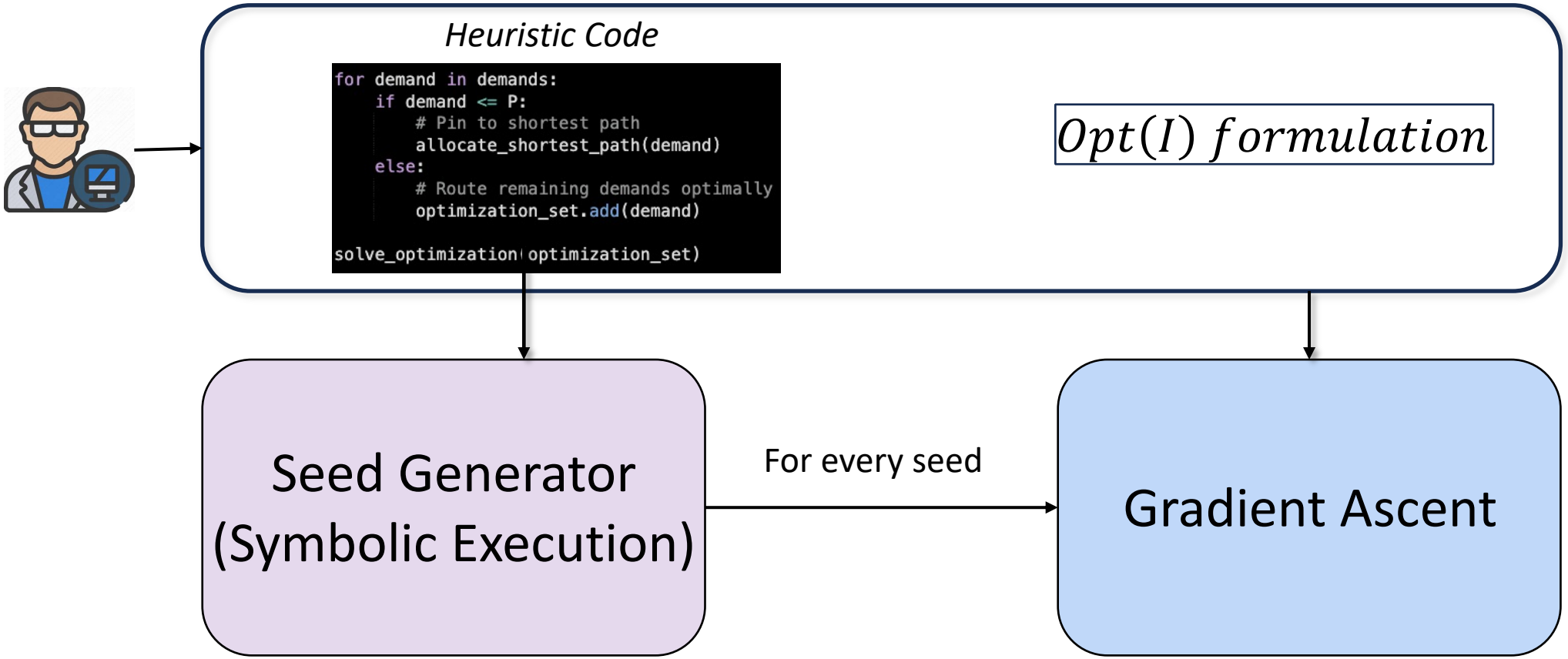
Example: Random Sampling, Hill Climbing, Simulated Annealing

Easy to use, progress by sampling many $Gap(I)$ values

✗ Inefficient

MetaEase Overview

Simple interface for developers

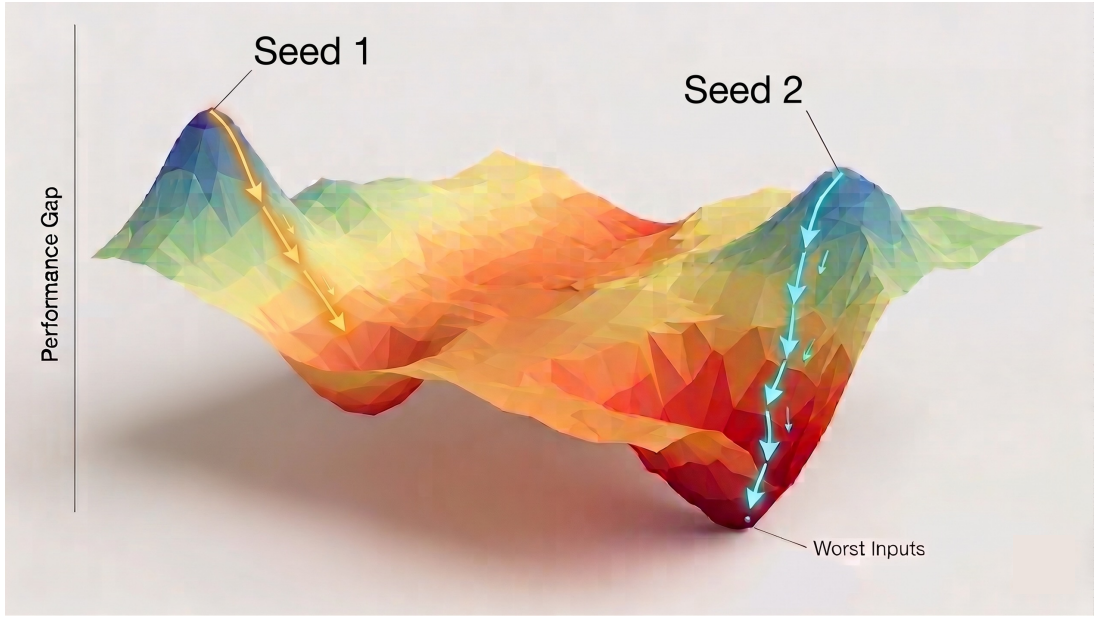


Smart seeds based on heuristic behavior

Efficient compute of gradients and non-differentiability handling

MetaEase Mechanism's For Gradient Ascent

$$\nabla Gap(I) = \nabla\, Opt(I) - \nabla Heuristic(I)$$



Compute $\nabla\, Opt$ efficiently ✓

Compute $\nabla Heuristic$ efficiently ✓

Smart gradient ascent seeds

Handling non-differentiability