Spilling the TE: Lessons from AI-driven evolution of Traffic Engineering

Rahul Bothra
Google

Alexander Krentsel
Google

Brighten Godfrey
UIUC

Sylvia Ratnasamy
Google

Abstract

Traffic Engineering (TE) is crucial for network operators to enforce routing policy and maintain operational efficiency. TE solutions have been deployed in global cloud systems for 20+ years, during which networks have grown 100x in scale, with complex requirements, further accelerated by increasing AI demands. This rapid growth is outpacing the slow human-driven evolution of TE systems – once a decade for most global WANs, with continuous duct-taping to meet new constraints. This limitation makes TE ripe for rapid development using Evolutionary Discovery Techniques (EDT). In this paper, we report lessons learned as we start to evolve a production-grade TE system using an EDT; we posit that these apply broadly across system optimizations. We use production traces from a global WAN to design frontier-breaching algorithms. We discuss open questions, and what the future of EDT-TE integration would look like.

1 Introduction

Cloud operators spend billions of dollars to provision wide-area networks (WANs) connecting datacenters across the globe. They then employ traffic engineering (TE) systems [4, 5, 6, 7] that monitor traffic demand and optimize the routing of traffic through the network to satisfy performance goals, the most basic of which is to maximize total throughput. It is also important for a TE system to have low runtime, to react quickly to traffic bursts and hardware failures.

Global WANs have grown by orders of magnitude in both topological size and traffic volume [8], even more so recently to meet AI demands. This growth has made achieving optimal efficiency at fast runtimes increasingly intractable. TE systems inevitably begin to buckle, performing poorly on the core requirements they were originally designed to uphold. As systems scale rapidly, new TE systems and algorithms are needed to keep up, potentially making different tradeoffs between optimality and runtime. Furthermore, new constraints and requirements continue to change; beyond simple throughput maximization, operators may seek latency objectives for certain flows, policy constraints on paths used, fairness in throughput (which itself has multiple definitions, including proportional, alpha-fairness, and max-min fairness), and more. Even if a more fundamental re-design is called for, humans often escape these issues by *duct-taping* components to stay within acceptable bounds [9, 10].

The broader point is that the scale, complexity, and rate of change in requirements of WANs is starting to outpace human-driven evolution. This makes Traffic Engineering ripe for a paradigm shift toward automated, AI-driven systems design. We argue that Evolutionary Discovery Techniques (EDT), like AlphaEvolve and OpenEvolve [2, 3], offer a promising path to rapidly evolve global-scale TE architectures to meet modern demands. In this paper, we report our early experiences and the fundamental mindset shifts required to achieve a EDT-TE integeration. We believe several of these

AI Agents for Discovery in the Wild (AID-Wild), Workshop at ACM CAIS 2026.

experiences – such as the need to capture previously-implicit objectives, and representations that accelerate the search – may be of interest beyond TE.

2 Prototype System Design

2.1 TE Solver overview

A production TE system is built with multiple interacting components such as telemetry collection, path generation, weight computation (TE solver), traffic quantization, and hardware programming. For our experiments, we isolate and model the TE solver, as it has the biggest impact on the desired TE objectives.

Inputs and Outputs TE solver requires three inputs - (i) link capacities of the network, (ii) traffic demands requested by users, and (iii) set of paths over which each demand can be routed. Using these inputs, the TE algorithm produces the bandwidth rates to allocate over those paths.

Metrics We evaluate TE solver on three objectives (although the entire set of objectives is much larger): (i) Runtime, (ii) total bandwidth served, and (iii) fairness across user demands[1].

2.2 The EDT Framework

We use Darwin[1], an EDT framework similar to OpenEvolve [2], SkyDiscover [3], and AlphaEvolve [4], with a frontier LLM model. To optimize evolution, we always expose the definition / code of the score vector, alongwith any helper / surrounding functions.

EDT Inputs We use production traces from global-scale WAN topology ($\sim$100 nodes, $\sim$1000 edges). We collect 100 snapshots from a live production stream captured over three months in 2025. For each snapshot, we generate paths (required inputs) using the k-shortest algorithm.

EDT Outputs and Evaluation Darwin generates entire populations of candidate TE algorithms, outputting the code alongside a score vector for each. In our experiments, we terminate an Darwin instance either after it generates 10,000 distinct TE systems, or when the top-20 generated programs successfully converge to the known optimal value of the average of score vector.

3 Lessons Learned: Scaffolding AI for TE Design

3.1 Metrics for evaluating EDT Trajectories

We evaluate an EDT on two distinct axes: the maximum (or mean of top-20) mean score by any TE program, and its *sample efficiency* - i.e. the number of generated programs required to converge near that optimal reward. We base this with the thought that a good technique of evolving TE systems should be able to find optimal TE algorithms, and also find them quickly.

Another way to quantify sample efficiency is by computing the mean score normalized by the program-number across the evolution, with a caveat that EDT frameworks inherently balance the *exploration* of radical, potentially flawed logic with the *exploitation* of known good structures, making early sample efficiency volatile.

3.2 Feasibility as an Objective – Penalize impossible solutions instead of eliminating them

A practical TE system is one that doesn't generate impossible results. For EDT to generate practical systems, it must learn what results are acceptable. Human designed algorithms already follow these rules, but EDT requires a complete description. For TE, this means enforcing the traditional flow constraints: flow rates should be non-negative, respect link capacities, and not exceed requested demand.

[1] Name anonymized

Initially, we eliminated any generated program that violated these constraints. However, this binary approach stifled discovery, as EDT discarded directions of algorithms that performed well, except with slight feasibility violations. We found that applying a continuous penalty gradient to the objective score—scaling with the magnitude of the violation—allowed AI to explore a wider set of algorithms. Note that many human-designed TE algorithms also violate feasibility before restoring it [1, 2]. Naturally, the penalty for such violations must be large (say quadratic) to avoid developing solutions with high performance along with high violations.

A related implication of this observation is that *the objective function should be smooth (continuous and differentiable) to make faster progress.*

3.3 Bounded Targets Accelerate Discovery Over Unbounded Optimization

We start evolving TE with the EDT score vector comprising of the raw throughput served across input snapshots – with no information of the upper bound. We observed that EDT made steady progress to achieve near-optimal (within $\approx 2.5\%$) throughput, but stagnated beyond that. Then, we normalize the score vector with the optimal throughput possible for each snapshot (calculated offline using an LP solver). As shown in Fig 1, running EDT with the normalized score formulation TE systems with equivalent performance nearly ~2.8x faster than the unbounded formulation.

A plausible explanation for why this is useful is because it gives EDT a mathematical ceiling to converge upon, rather than asking it to hunt indefinitely for an unbounded, arbitrary maximum.

An additional benefit of normalization – which by itself is often a requirement – is that it treats all input snapshots as equally important, restricting the reward for each in [0, 1]. This ensures that TE systems don't fine-tune for snapshots which can serve relatively large amounts of bandwidth.

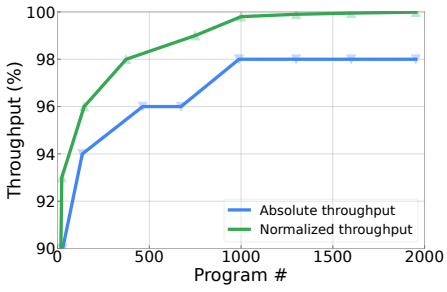

Figure 1: Throughput served across program # by configuring EDT bounded (normalized) and unbounded (absolute) score vectors.

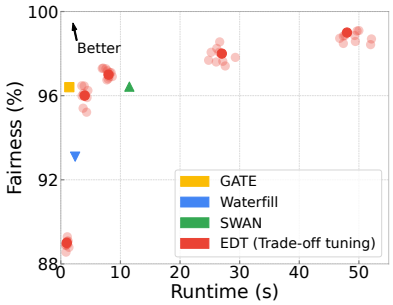

Figure 2: Fairness and runtime of top-20 TE solvers with tuning objectives in the EDT score vector (compared to SOTA).

3.4 Objectives and Trade-offs Must Be Well-Defined

On the surface, TE would seem to be a system that has very clearly stated objectives: at its core it is a multicommodity flow problem, and a performance-optimal TE solution can be derived with a linear program solver like Gurobi. But looking more closely, TE has other objectives that are not clearly stated.

At a high level, this occurs because humans have approached TE systems design from an *algorithms-first* perspective: an algorithm is chosen because it satisfies certain unstated objectives that the designer qualitatively looks for, without those objectives being concrete mathematical targets. This happens in several ways. First, some objectives are unstated within the algorithm: for TE, this happens with properties like path-churn and routing table size [6]. Second, the tradeoffs between these objectives are not clearly defined; for example, is it worth accepting 5% lower optimality in order to halve the routing table size or the runtime? Third, in some cases *proxies* are used instead of true goals. For example, human-designed TE systems heavily emphasize minimizing solver runtime. However, fast execution is not the ultimate objective; rather, it is a proxy for reacting quickly to link failures and traffic bursts to meet performance guarantees.

Due to the nature of human reasoning, we have gotten away without formally defining the desired objectives and tradeoffs over the high-dimensional space. EDT, however, forces a paradigm shift from an *algorithm-first* to an *objective-first* approach. To effectively use AI discovery, system operators must adhere to a *Unified Objective Theory* – i.e. express the quality of a TE system with one number. This requires (i) making *unstated* objectives explicit, (ii) quantifying the impact of a proxy objective on the true objective, and (iii) defining a trade-off between the quantified TE objectives to fold them into a single number. Fig 2 shows tuning the trade-off between fairness and runtime (score vector), and how some clusters come close to existing SOTA algorithms. Each cluster represents algorithms generated by one tuning choice.

4 Future of EDT-driven Traffic Engineering

4.1 Iteratively Shaping the Objective Contour

As noted in §3.4, human can easily design bounds of what is *unacceptable* for a single objective. However, an EDT not only requires what to avoid, but also a unified number combining individual objectives. Providing this in a high-dimensional space is exceptionally difficult. Historically, human intuition has been limited to quantifying tradeoffs between simple pairs of objectives, such as balancing throughput against latency or fairness. We posit that extending this across multiple objectives is exceedingly difficult.

We argue that discovering the right reward function requires an iterative, human-in-the-loop approach. Operators must experiment with different weights, penalty curves, and tradeoff scalarizations, to sculpt a complete and smooth "objective contour." Something that could accelerate this contour development is to use the objectives tradeoffs achieved by human-designed systems as a warm-start to explore around. We leave these challenges to be addressed in future work.

4.2 Scaffolding the Co-Optimization of Siloed Components

Large-scale systems (including TE) are often broken into discrete components that interact together to satisfy constraints and objectives. This modularization simplifies human-reasoning and algorithm design. For TE, this modularization appears with path computation, weight computation, traffic quantization [5], network partitioning [8, 9]. This modularization often leads to a loss of optimal quality, since siloed components cannot exchange information to achieve the globally optimal solution.

EDT presents a very unique opportunity here. First, EDT can enable fusing two (or more) modules into a larger module with better performance. Second, EDT can help in quantifying the cohesiveness of different siloed algorithms. For example: which path selection algorithm works best with a chosen weight selection algorithm. It remains an open question as to how many components can EDT fuse at once – would it be able to design a unifying TE algorithm that subsumes all components (with good performance)? Would it breakdown the problem into similar components by itself?

4.3 Handling Out-of-Distribution Events

EDT frameworks do what we ask – which is to maximize the average of the provided score vector, which includes the entire distribution of training snapshots. While this produces a highly optimized "jack-of-all-trades" solver for typical network conditions, it has the potential to struggle with out-of-distribution (OOD) events, such as link cuts or unplanned traffic spikes. Because these events are statistically rare, their signal can be easily drowned in the averaging of the score vector. In other words, EDT will readily sacrifice emergency robustness for marginal efficiency gains in steady-state scenarios. Conversely, human engineers intuitively understand that one size does not fit all. We design multi-modal networks with specialized, hardcoded fallback mechanisms [11], which explicitly take over during sudden localized failures before the global TE controller can react.

A key open question is what is the right mechanism to guide EDT to discover and adopt this multi-modal architectural design – i.e. rather than forcing the AI to evolve a single universal algorithm, it would be useful if we could generate an *algorithm portfolio* of complementary algorithms that together optimal performance at all times.

References

[1] Bothra, R., Krentsel, A., Mandal, S., Godfrey, P.B., Ratnasamy, S., Shakir, R. & Srikant, R. (2026) GATE: GPU-Accelerated Traffic Engineering for the WAN. *arXiv preprint* arXiv:2605.01748.

[2] Rudes, R., Hou, J. & Mehta, A. (2025) OpenEvolve: Codebase-scale evolutionary algorithms for code optimization, powered by LLMs. *GitHub repository*, https://github.com/ryanrudes/openevolve.

[3] UC Berkeley Sky Computing Lab (2026) Early Discoveries of Algorithmist I: Promise of Provable Algorithm Synthesis at Scale. *arXiv preprint* arXiv:2603.22363.

[4] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M. & Wattenhofer, R. (2013) Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference*.

[5] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Holzle, U., Stuart, S. & Vahdat, A. (2013) B4: Experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference*.

[6] Abuzaid, F., Huang, Y., Phothilimthana, P. M., Marcus, J. & Zheng, J. (2021) Contracting wide-area network topologies to solve flow problems quickly. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*.

[7] Facebook Engineering (2021) EBB: Reliable and Evolvable Express Backbone Network in Meta. *Engineering at Meta*.

[8] Hong, C.-Y., Mandal, S., Al-Fares, M., Zhu, M., Alimi, R., Chandrashekar, K., Bhagat, C., Jain, S., Koley, B., Muthukrishnan, S., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J. & Vahdat, A. (2018) B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the ACM SIGCOMM 2018 Conference*.

[9] Krishnaswamy, U., Devraj, A., Singh, R., Zhang, Y. & others (2022) Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*.

[10] Bogle, J., Bhatia, N., Ghobadi, M., Moshref, M., Rexford, J. & Gill, P. (2019) TEAVAR: Striking the right utilization-availability balance in WAN traffic engineering. In *Proceedings of the ACM SIGCOMM 2019 Conference*.

[11] Pan, P., Swallow, G. & Atlas, A. (2005) Fast Reroute Extensions to RSVP-TE for LSP Tunnels. *IETF RFC 4090*.