Learning Network Design Objectives Using A Program Synthesis Approach

Yanjun Wang
Purdue University
West Lafayette, IN, USA
wang3204@purdue.edu

Chuan Jiang
Purdue University
West Lafayette, IN, USA
jiang486@purdue.edu

Xiaokang Qiu
Purdue University
West Lafayette, IN, USA
xkqiu@purdue.edu

Sanjay G. Rao
Purdue University
West Lafayette, IN, USA
sanjay@ecn.purdue.edu

ABSTRACT

While the networking community has extensively tackled network design problems using optimization or other techniques (e.g., in areas such as traffic-engineering, and resource allocation), much of this work focuses on efficiently generating designs assuming well-defined objectives. In this paper, we argue that in practice, the objectives of a network design task may not be easy to specify for an architect. We argue for, and present a structured approach where the objectives of a network design task are learnt through iterative interactions with the architect. Our approach is inspired by a programming-by-examples approach that has seen success in the programming languages community. However, conventional program synthesis techniques do not apply because in our context a user can only provide a relative comparison between multiple choices on which one is more desirable, rather than provide an exact output for a given input. We propose a novel comparative synthesis approach to tackle these challenges. We sketch the approach, present promising preliminary results, and discuss future research questions.

ACM Reference Format:
Yanjun Wang, Chuan Jiang, Xiaokang Qiu, and Sanjay G. Rao. 2019. Learning Network Design Objectives Using A Program Synthesis Approach. In *The 18th ACM Workshop on Hot Topics in Networks (HotNets '19), November 13–15, 2019, Princeton, NJ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3365609.3365861

HotNets '19, November 13–15, 2019, Princeton, NJ, USA

1 INTRODUCTION

Many network design problems such as planning, traffic engineering and resource allocation require making decisions in a manner guided by policies expressed by the architect. For instance, when managing wide area networks in cloud provider settings, a network architect must decide how to allocate bandwidth to different applications, and route traffic, in a manner that reflects application priorities, while balancing fairness, throughput and latency considerations [10–12].

The primary focus of much research in the networking community has been on realizing a design (e.g., determine how bandwidth must be allocated) given well specified objectives (e.g., given a max-min fair criteria, or given well designed utility functions). The typical practice is to formulate optimization problems with well-defined objectives, with the focus being on strategies for efficiently solving the problem. However, in practice, it may be challenging for an architect to precisely state the objectives in the first place. As one example, while many network design problems simultaneously trade-off multiple objectives (e.g., latency, bandwidth, throughput, fairness), it may not be easy for an architect to precisely express the trade-offs between these different criteria. While one approach is to provide objective functions with knobs that an architect could control, the functions could be overly restrictive, and it may not be intuitive to an architect how the knobs should be set to match her intent. As another example, many network allocation problems require architects to specify utility functions (e.g., [12]). However, such functions are fairly abstract and may not be intuitive for a designer to specify in practice.

In this paper, we take the position that when synthesizing network designs, the challenge is not just how to solve a well-defined formal problem specification, rather how to obtain a formal specification that reflects architect goals in the first place. To this end, we argue for an approach

where the objectives behind the synthesis are *learnt* through iterative interactions with the architect.

Our approach is inspired by, and draws from the program synthesis community. Program synthesis is the process of automatically generating programs that meet the user's intent, and has seen many successful applications recently in many domains, e.g., synchronization for high-performance parallel code [22], high-quality feedback for early programmers [16], and SQL queries [2, 23] among others. Since writing precise specification for a programming task is challenging especially for average programmers, there has been much interest in the programming-by-examples (PBE) techniques, populated by success stories such as Sketch [17, 18] and FlashFill [8, 9]. Here, the synthesizer is given a set of input-output examples and tasked to find a program whose behavior matches the given examples through iterative interactions with the user.

Unfortunately, PBE techniques are not directly applicable to our context of learning objectives when interacting with the network designer. First, PBE techniques assume that the user knows the exact output for a given input. Unfortunately, in the network synthesis context, the architects cannot provide exact outputs for a given configuration. For example, when evaluating a design which achieves a certain throughput for a set of flows, the architects cannot provide a computed total utility value across all the flows, because precisely formulating utility functions is itself the challenge that we seek to tackle. However, an architect can provide a *relative* comparison between two proposed designs (e.g., given two candidate flow allocations, an architect can rank which one is more acceptable).

Motivated by these observations, in this paper, we propose comparative synthesis, a novel synthesis framework for learning and realizing network design objectives. From the perspective of program synthesis, our key contribution is a novel way of interaction with the user based on a comparison (rather than exact output values). From the perspective of networking, our key contribution is to advocate the learning of formal specifications for network verification/synthesis problems through a comparative synthesis approach.

We present a preliminary evaluation to demonstrate the feasibility and potential of a comparative synthesis approach. Our work is preliminary, but indicates the promise, and opens several questions for future research.

2 MOTIVATING EXAMPLE

In this section, we present a concrete example to illustrate that (i) when designing networks, specifying objectives is often not straight-forward; and (ii) existing specification approaches may not be intuitive to the network designer.

Consider traffic engineering in the wide-area settings with SDN networks [10–12]. Typically such settings have multiple classes of traffic (e.g., higher priority latency sensitive traffic, and lower priority elastic traffic). An architect must decide how to allocate bandwidth to different applications and route traffic in a manner that considers application priorities, and fairness considerations, while maximizing throughput and minimizing latency. While existing methods [10–12] focus on designing for a given set of requirements, precisely specifying requirements may itself be challenging as we illustrate below:

Expressing latency, throughput and fairness tradeoffs. Given a set of flows, and tunnels, with flow i having demand d_i, and tunnel j having a weight w_j, consider the problem of determining the bandwidth b_i that must be allocated to flow i, and the portion of this bandwidth b_{ij} that must be sent on tunnel j. SWAN [10] tackles this problem through seperate formulations that either maximize throughput, or ensure a max-min fair allocation. Further, when considering throughput, SWAN also considers latency as an additional factor. Specifically, it optimizes an objective:

$$\sum_i b_i - \epsilon \sum_{i,j} w_j b_{i,j} \tag{2.1}$$

The first term corresponds to the total throughput, and the second term imposes a penalty for traffic on higher latency paths. The knob ϵ is a parameter that controls the penalty assigned to longer paths. While in theory an architect can tune ϵ to trade-off throughput, and latency, in practice it is not clear that (i) it is easy for an architect to find a desirable ϵ; and (ii) the above approach is sufficiently expressive to capture how the designer would really like to trade these parameters off. For instance, an architect may care about the latency of individual flows (not just the average latency across flows weighted by traffic), may be particular that latency and throughput meet certain absolute requirements, and trade the two goals differently depending on how far they are to these requirements.

More generally, a designer may also wish to jointly consider fairness and throughput. An approach to do so is presented in [3] which maximizes q_f such that every flow receives at least a fraction q_f of its max-min fair allocation, while the total allocation across flows is at least $q_t T_{opt}$, where T_{opt} is the optimal throughput that could be achieved across flows if fairness is not considered. However, it is unclear how an architect chooses q_f, or q_t, and if the above approach is reflective of how the architect would trade-off the two criteria. Finally, it may be necessary to simultaneously combine the above criteria with latency requirements.

Expressing fairness and priority requirements. When multiple traffic classes are involved, SWAN [10] strictly prioritizes traffic belonging to a higher class, and uses a max-min

fair allocation for traffic within the same class. An architect may wish more flexibility in specifying such requirements. For instance, rather than strict priority, a weighted max-min fair allocation may be more reflective of designer intent. Beyond a max-min fair allocation, other allocation schemes such as proportionally fair allocations, or alpha-fair allocations may better match the designer's goals [20]. While it is feasible to design optimization formulations for a given notion of fairness, it is unclear how an architect decides which of these (fairly abstract) choices is the most appropriate in her context. Another approach used by Google [12] is to use functions where the bandwidth requirement is specified as a concave function of fair share. However, specifying a bandwidth function (which requires knobs such the slope as a function of fair share to be specified) is a black-art.

3 A COMPARATIVE SYNTHESIS FRAMEWORK

To tackle the challenges discussed in §2, we present a general and structured approach to obtain a formal specification that reflects architect goals in the first place. Rather than precise specifications (e.g., exact utility functions), or limit designers to narrow ways of expressing objectives (e.g., constraining them to choosing an ϵ parameter to control the trade-off between bandwidth and latency), our approach supports richer and more expressive methods for intent to be expressed, and learns the objectives behind the synthesis task through iterative interactions with the architect.

Our approach is motivated by the programming-by-examples (PBE) techniques that have been well explored in the program synthesis community. In PBE, the synthesizer is given a set of input-output examples and tasked to find a program whose behavior matches the given examples. A successful use case is the FlashFill feature of Microsoft Excel [8, 9]. Rather than require a lay user to enter complex formulae or programs, these techniques automatically synthesize programs from examples provided by the user. An example-based specification approach is inherently ambiguous and potentially can be satisfied by many spurious programs. Hence, the success of a PBE system relies on two components: i) how to rank many candidate programs and pick the likely desired one; and ii) when the chosen program does not correctly produce outputs for all inputs, how to interact with the user and re-synthesize a better candidate.

Motivated by the above, we represent an objective function as a program. A program representation provides expressive power to capture a rich set of objectives that an architect may have in mind. We model the task of learning an objective function as one of synthesizing an appropriate program. Unfortunately, existing PBE techniques are not directly applicable in our context. Consider a context where the goal is to synthesize an objective function based on both the throughput and latency of a network design. Current PBE techniques require that the architect specifies the exact objective for a few example combinations of throughput and latency, and provides feedback on whether the output of a synthesized objective function is correct. Unfortunately, in our context, if the architect can provide these kinds of feedback, she would have written the precise objective that we want to synthesize.

An architect can however indicate which combination of throughput and latency is more preferable given two (or more) such combinations. To this end, we argue for a novel synthesis framework that we call *Comparative Synthesis* illustrated in Figure 1. We summarize the key ideas below:

- The synthesizer takes as input a set of metrics that summarize the performance of a network design, and are of interest to the designer. For example, in Eq (2.1). the relevant metrics are (i) the total throughput across all the flows; and (ii) the average latency of network flows weighted by the traffic on that flow. Henceforth, we refer to these metrics as throughput and latency respectively. More generally, the metrics could include the throughput and latency of individual flows (e.g., to capture requirements applicable at the flow level).
- The synthesizer may also take as input one or more sketches or templates of typical objective functions (we provide a concrete example later). Such sketches may be provided by a domain expert to incorporate the structure of objective functions typical for the context, and to constrain the synthesis process.
- The goal of the synthesizer is to produce an objective function that matches the architect's intent. At each step, the synthesizer requests the architect to provide a preference ranking for different sets of metric combinations (e.g., different throughput/latency combinations). Henceforth, we refer to each distinct metric combination as a scenario. Based on these inputs, the synthesizer generates multiple candidate objective functions that satisfies the user ranking. The synthesizer interacts with the architect, providing new scenarios to be ranked, and this process eventually converges to a desired objective function.

We present more concrete details in the next section, with a case study.

4 SYNTHESIS PROCESS

In this section, we briefly describe our synthesis process focusing on the SWAN example described in Section 2. We first describe a sketch of the objective and what the synthesis problem is, then formulate how the synthesis algorithm works under the user's guidance. Finally, we report our preliminary experimental results and our observations.

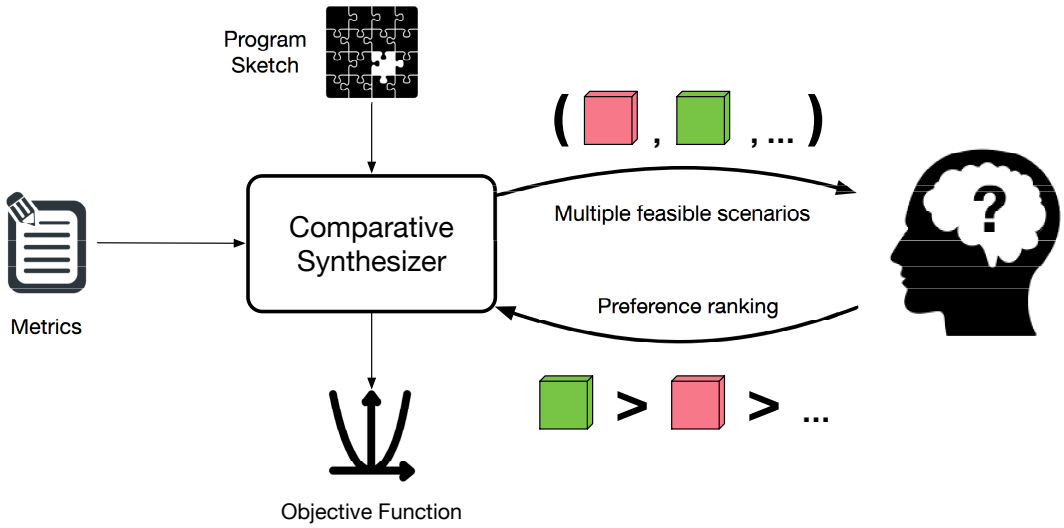

Figure 1: Overview of the comparative synthesis framework.

4.1 Sketching the Objective Function

We adopt the popular sketch-based synthesis approach [17, 18], in which the programmer specifies a synthesis problem as a *sketch* or *template*, which is a program that contains some unknowns to be solved for and some assertions to constrain the choice of unknowns.

To learn the network design objectives, we expect a sketch of the objective function reconciling throughput and latency has been provided by domain experts. In the case of SWAN, our objective function synthesis starts from a sketch as shown in Figure 2a. The objective function takes two arguments `throughput` and `latency` which are the two metrics that the user targets to optimize. The body of the function is a partial program containing four unknown holes (as highlighted in the figure): *tp_thrsh*, *l_thrsh*, *slope*1 and *slope*2. Then the synthesis task is to fill all four holes automatically to form a desirable objective function. Figure 2b shows a solution of the synthesis problem with values of filled holes colored red.

The sketch captures the following rules: 1) we prefer *satisfying scenarios*, i.e., those in which both the throughput is at least *tp_thrsh* and latency is at most *l_thrsh*, by giving an extra $factor$ bonus points to them (in Figure 2a we just use a large number 1000); 2) we weigh throughput and latency differently for satisfying and unsatisfying scenarios, and the weights can be represented by different slopes *slope*1 and *slope*2. We make several comments. First, the sketch allows for more flexibility than a single ϵ parameter, as it captures the notion that satisfying scenarios (which are intuitive to interpret) are prioritized, and allows for different slopes in different regions. Second, the sketching approach leaves the task of determining the various knobs such as *slope*1 and *slope*2 to the synthesis engine, and the parameters are chosen to match the architect's notion of which scenarios are more desirable. Third, the above sketch is just an example - it can be generalized to support multiple regions, and different ways of combining the objectives. The sketch can also encode requirements on throughput and latency of individual flows, or other summary metrics besides sum of throughput and weighted average latency of flows, and also even the exact functions in the summarization could be left unspecified.

4.2 Preference-Guided Synthesis

Recall that our goal is to fill the unknowns in the user-provided sketch, in the case of SWAN, *tp_thrsh*, *l_thrsh*, *slope*1 and *slope*2, such that the complete objective function matches the user's intent. As mentioned in Section 3, our system understands the user's intent by making queries about concrete scenarios. The system records the user's preference as a directed acyclic graph (DAG) G, in which each vertex represents a concrete scenario (in the case of SWAN, concrete throughput and latency values), and each directed edge represents a preference (the head scenario is more preferred than the tail scenario). For example, if there is an edge in G connecting a vertex (`throughput` $= 2$, `latency` $= 100$) to another vertex (`throughput` $= 5$, `latency` $= 10$), then the synthesizer must ensure the synthesized objective function satisfies

$$\texttt{objective\_func}(2, 100) > \texttt{objective\_func}(5, 10)$$

At the beginning, the synthesizer generates a set of randomly generated scenarios and asks the user to indicate her

```
objective_func(throughput, latency)
  if
    throughput >= tp_thrsh && latency <= l_thrsh
  then
    throughput - slope1*throughput*latency + 1000
  else
    throughput - slope2*throughput*latency
```

(a) Sketched objective function (the highlighted part is unknown holes to be filled).

```
objective_func(throughput, latency)
  if
    throughput >= 1 && latency <= 50
  then
    throughput - 1*throughput*latency + 1000
  else
    return throughput - 5*throughput*latency
```

(b) Target objective function (red indicates the synthesized values).

Figure 2: Example of sketch-based objective function synthesis.

preferences over them. The user's feedback forms the initial graph G and helps the synthesizer discard most of the obviously incorrect solutions.

Next, the synthesizer repeatedly makes more queries and expands G with the user's answers. In each interaction, the synthesizer finds multiple candidate solutions that are consistent with the current preference graph G, say f_a and f_b. Then to determine which candidate is closer to the user's expectation, the synthesizer further generate two scenarios, say (t_1, l_1) and (t_2, l_2) such that f_a and f_b give different preferences over these two scenarios. The synthesis task is automated by making a logical query to a satisfiability solver such as Z3 [4]:

$$
\begin{aligned}
&\exists f_a, f_b, t_1, l_1, t_2, l_2.\\
&\wedge \mathit{Viable}(f_a) \wedge \mathit{Viable}(f_b) \wedge\\
&\quad \bigwedge_{((\mathtt{t},\mathtt{l}),(\mathtt{t}',\mathtt{l}'))\in G} f_a(\mathtt{t},\mathtt{l}) > f_a(\mathtt{t}',\mathtt{l}') \wedge f_b(\mathtt{t},\mathtt{l}) > f_b(\mathtt{t}',\mathtt{l}')\\
&\wedge f_a(t_1, l_1) > f_a(t_2, l_2) \wedge f_b(t_2, l_2) > f_b(t_1, l_1)\\
&\wedge \mathit{ClosedInRange}
\end{aligned}
$$

The query asks the SMT solver to find candidates f_a, f_b and scenarios (t_1, l_1) and (t_2, l_2) such that a set of constraints are satisfied. First, both candidates should be viable, i.e., they can be implemented with some knobs the user can tune. Note that the viability check is ad-hoc for different network design problems and can be very sophisticated. However, in the SWAN example, arbitrary combination of thresholds and slopes can be implemented by an appropriate knob ϵ, hence we simply skip the viability check.

Secondly, both candidates should honor existing preferences in G, and have different preferences on the two scenarios. The last condition is *ClosedInRange*, which represents a set of constraints that enforce the metrics (throughput and latency) are closed in the range that we assume. For SWAN, we assume both `throughput` and `latency` should be greater than or equal to 0; the `throughput` is under 10Gbps; the `latency` should be under 200ms.

Once an appropriate set of scenarios is generated, the synthesizer passes it to the user. As the scenarios are concrete metrics, the user can easily understand and order them based on her preferences, which will be added to the preference graph G. The synthesis continues to the next iteration.

Metrics	Average	Median	SIQR
# Iterations	31.33	30	4.25
Synthesis Time per Iteration (s)	2.45	2.37	0.12
Total Synthesis Time (s)	76.13	71.67	11.16

Table 1: Summary of experimental results.

Note that the user does not have to give a full rank of all concrete scenarios. If some scenarios are indistinguishable or incomparable from the user's view, the synthesizer can still update the preference graph with the partial rank and make progress toward finding a solution.

The SMT solver may return unsatisfiable, which means the preference graph G is rich enough such that there is only one viable solution. In that case, the synthesis process and the unique solution will be returned.

4.3 Experimental Evaluation

We developed a prototype of the comparative synthesis framework and evaluated the effectiveness of our approach by synthesizing objective functions for the SWAN system. As a preliminary evaluation, we use an oracle to play the role of an ideal user in the synthesis process. For any set of scenarios given by the synthesizer, the oracle evaluates the scenarios using the ground truth (the target function in Figure 2b) and returns the preference ranking accordingly. Our framework leverages Z3 [4], a state-of-art SMT solver as our back-end constraint solver. We conducted our experiments on a laptop with a dual-core, 2.9GHz CPU and 8GB memory running macOS 10.12.

We initialized the preference graph G by asking the user to rank 5 randomly generated scenarios. Then in each following interaction, we asked the user to rank one additional pair of scenarios. Since G was randomly initialized, our experiments were not deterministic. The solving time and the

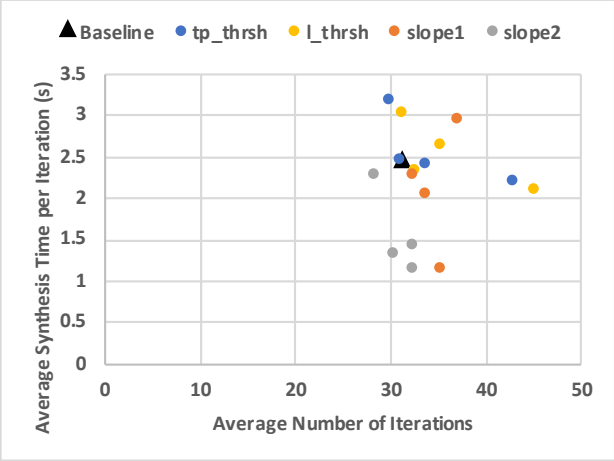

Figure 3: Experimental results with tuned threshold or slope.

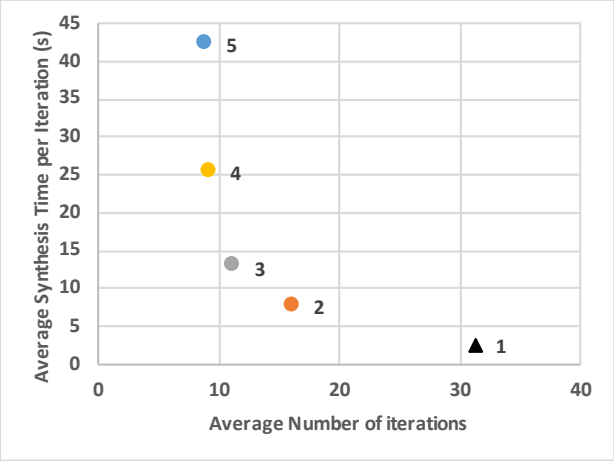

Figure 4: Experimental results with more pairs of scenarios ranked per iteration.

number of interactions (we use iterations and interactions interchangeably) between the user and the synthesizer varied across runs. We ran each experiment **nine** times and collected the iteration number, synthesis time for each iteration and the total synthesis time. Since the user is simulated by an oracle, we omit the time spent by the oracle. Table 1 gives the average, the median and the semi-interquartile range (SIQR) for these numbers. We found the total synthesis time is reasonable and stable regardless of different initial input pairs.

We further investigate the robustness of our approach by tuning various components of the synthesis framework, including different target functions and different numbers of scenarios.

Effect of target function. It is important to ensure the framework can synthesize various target functions, since these represent different user's intents. We tuned the thresholds *tp_thrsh* and *l_thrsh*, and slopes *slope*1 and *slope*2 in the target function separately, each with 5 different values. *l_thrsh* varies in range between 20 and 80, and others vary in the range between 1 and 5. Figure 3 reports the average number of iterations and average synthesis time per iteration for each variant target function. While the number of interactions and the synthesis time vary, we successfully synthesized all different correct objective functions.

Effect of number of ranked scenarios. The number of iterations interacting with oracle was around 30 in the original setting, which is a bit excessive if a human user were participating. Therefore, we consider generating multiple pairs of scenarios per iteration for the user to compare. As illustrated in Figure 4, ranking 2 scenarios per iteration helped the synthesizer find a solution in similar amounts of time but

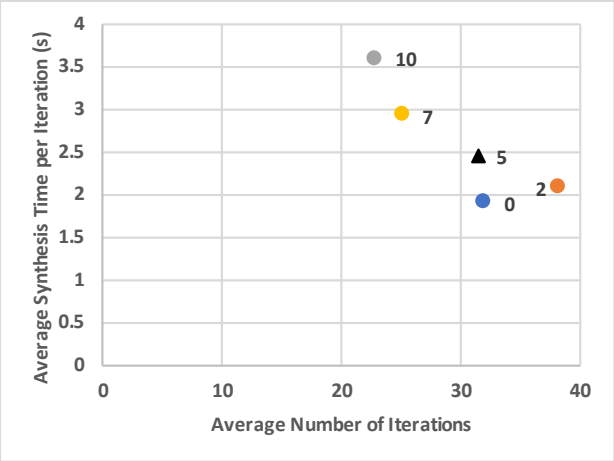

Figure 5: Experimental results with different number of initial scenarios.

with fewer interactions,. However, ranking 3 or more scenarios per iteration only reduced the interaction moderately but increased the total synthesis time significantly.

Effect of number of initial scenarios. We also investigated if the randomly generated scenarios help. We tried various number of initial random scenarios ranging from 0 to 10. As shown in Figure 5, initial randomly generated scenarios slowed down synthesizing target function but helped decreasing the number of interactions.

5 RELATED WORK

The research community has started exploring the use of program synthesis techniques to networking [6, 7, 14, 15, 19,

21, 25]. These systems all focus on synthesizing forwarding tables, or router configuration files. In contrast, our framework takes the first step toward learning network design objectives using synthesis techniques. Recently Birkner *et al.* [1] has also recognized the challenge of writing network specifications. However, they do not account for generating quantitative design objectives that match the architect's intent.

The PBE techniques have been applied to synthesize network configurations. For example, NetEgg [25] can generate stateful configurations, e.g., those look up and maintain internal states, from user-provided scenarios. While we are inspired by PBE techniques, a direct application is not viable in our context, and we propose a comparative synthesis approach.

6 DISCUSSION AND CHALLENGES

In this paper, we have taken the position that when designing networks, it is not sufficient to address difficulties in solving a well-defined formal problem specification. Rather, it is also important to tackle challenges in obtaining a formal specification in the first place. To this end, we have proposed a novel **comparative synthesis** approach that can allow learning of design objectives through iterative interactions with network architects.

Our work is preliminary, and many challenges need to be addressed to realize our approach at scale in practice. Our work also has potential applications in many other contexts beyond those discussed in this paper. We discuss below.

6.1 Challenges

Considering tractability of realizing an objective. In this paper, we have focused on synthesizing objective functions that capture architect intent. However, ultimately, it is also necessary to synthesize designs that meet these goals. It may be intractably hard to synthesize designs for arbitrary objective functions. One approach is for the domain architect to create templates that guide the architect towards objective functions that are tractable, and pick functions that closely approximate architect intent. Another approach is for the solver to use a more tractable objective function and generate multiple (rather than a single) solution that are good for that objective function, and subsequently pick one of these based on the exact architect objective.

Robustness to user inputs. As a PBE approach, comparative synthesis also depends on the quality of inputs from the architect. Since an architect can potentially provide inconsistent or vague relative preference information, the synthesis approach must be robust to detect and remove noise in user inputs. A potential approach is to use machine learning techniques to this end.

Comparing scenarios through simulators. A key question for comparative synthesis is how architects can rank a given set of scenarios. In some domains, an architect can intuitively reason about the scenarios, and indicate which is more preferable. In other contexts, it may be helpful to create simulations that mimic the scenarios, to allow architects to decide which scenario is more preferable. Such an approach is particularly relevant to the video streaming application that we discuss below.

6.2 Other applications

Beyond network design, comparative synthesis may apply to many other networking contexts. We present some below.

Algorithm design for video streaming. Consider the design of Adaptive Bit Rate (ABR) algorithms in the context of HTTP-based video streaming [13, 24]. It is well understood that many video delivery metrics impact user experience, such as the average video bit rate, start up latency, rebuffering ratio, and variations in bit rate [5]. However, it is not clear how best to simultaneously reconcile these metrics. State-of-the-art approaches in the field [13, 24] formulate objective functions that combine these metrics in fairly ad-hoc ways (e.g., simple linear combinations), and use optimization or machine learning techniques to optimize these composite measures. It is however unclear if these composite measures meaningfully reflect user experience. A comparative synthesis approach can be used by video publishers to learn richer objective functions that better relate to user experience. The ranking of preferences could be combined with simulations described above to create scenarios corresponding to different video delivery metrics, with the rating from users guiding the design of objective functions.

Configuring home networks. Another application domain is the configuration of home networks to manage policy regarding how bandwidth must be utilized across competing applications (e.g., different home users, IoT devices). It would be particularly challenging for home users to configure weights and utility functions for different kinds of traffic, and a comparative synthesis approach can potentially be beneficial in such settings.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under Grant No. CCF-1837023.

REFERENCES

[1] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. 2020. Config2Spec: Mining Network Specifications from Network Configurations. In *Proceedings of 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*.

[2] Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. 2013. Optimizing Database-backed Applications with Query Synthesis. In

Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13). ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/2491956.2462180

[3] Emilie Danna, Subhasree Mandal, and Arjun Singh. 2012. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *2012 Proceedings IEEE INFOCOM*. IEEE, 846–854.

[4] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS'08 (LNCS)*, Vol. 4963. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[5] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 362–373.

[6] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2017. Network-Wide Configuration Synthesis. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 261–281.

[7] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2018. NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 579–594. https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[8] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. ACM, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

[9] Sumit Gulwani, William R. Harris, and Rishabh Singh. 2012. Spreadsheet Data Manipulation Using Examples. *Commun. ACM* 55, 8 (Aug. 2012), 97–105. https://doi.org/10.1145/2240236.2240260

[10] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-driven WAN. In *SIGCOMM '13*. 15–26.

[11] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-deployed Software Defined Wan. In *SIGCOMM '13*. 3–14.

[12] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. 2015. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *SIGCOMM '15*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/2785956.2787478

[13] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[14] Leonid Ryzhyk, Nikolaj Bjørner, Marco Canini, Jean-Baptiste Jeannin, Cole Schlesinger, Douglas B. Terry, and George Varghese. 2017. Correct by Construction Networks Using Stepwise Refinement. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 683–698. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/ryzhyk

[15] Shambwaditya Saha, Santhosh Prabhu, and P. Madhusudan. 2015. NetGen: Synthesizing Data-plane Configurations for Network Policies. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*. ACM, New York, NY, USA, Article 17, 6 pages. https://doi.org/10.1145/2774993.2775006

[16] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated Feedback Generation for Introductory Programming Assignments. In *PLDI*. 15–26.

[17] Armando Solar-Lezama, Gilad Arnold, Liviu Tancau, Rastislav Bodik, Vijay Saraswat, and Sanjit Seshia. 2007. Sketching stencils. In *PLDI'07*. ACM, 167–178.

[18] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. 2006. Combinatorial sketching for finite programs. In *ASPLOS'06*. ACM, 404–415.

[19] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. 2014. Merlin: A Language for Provisioning Network Resources. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*. ACM, New York, NY, USA, 213–226. https://doi.org/10.1145/2674005.2674989

[20] Rayadurgam Srikant. 2012. *The mathematics of Internet congestion control*. Springer Science & Business Media.

[21] Kausik Subramanian, Loris D'Antoni, and Aditya Akella. 2017. Genesis: Synthesizing Forwarding Tables in Multi-tenant Networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, New York, NY, USA, 572–585. https://doi.org/10.1145/3009837.3009845

[22] Martin Vechev, Eran Yahav, and Greta Yorsh. 2010. Abstraction-Guided Synthesis of Synchronization. In *POPL*. ACM, New York, NY, USA.

[23] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. (2017). arXiv:cs.CL/1711.04436

[24] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.

[25] Yifei Yuan, Dong Lin, Rajeev Alur, and Boon Thau Loo. 2015. Scenario-based Programming for SDN Policies. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '15)*. ACM, New York, NY, USA, Article 34, 13 pages. https://doi.org/10.1145/2716281.2836119