Quality-aware Analysis and Optimisation of Virtual Network Functions

Daniel-Jesus Munoz
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
danimg@lcc.uma.es

Mónica Pinto
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
pinto@lcc.uma.es

Lidia Fuentes
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
lff@lcc.uma.es

ABSTRACT

The softwarisation and virtualisation of network functionality is the last milestone in the networking industry. Software-Defined Networks (SDN) and Network Function Virtualization (NFV) offer the possibility of using software to manage computer and mobile networks and build novel *Virtual Network Functions* (VNFs) deployed in heterogeneous devices. To reason about the variability of network functions and especially about the quality of a software product defined as a set of VNFs instantiated as part of a service (i.e., Service Function Chaining), a variability model along with a quality model is required.

However, this domain imposes certain challenges to quality-aware reasoning of service function chains, such as numerical features or configuration-level *Quality Attributes* (QAs) (e.g., energy consumption). Incorporating numerical reasoning with quality data into SPL analyses is challenging and tool support is rare. In this work, we present 3 groups of operations: model report, aggregate functions to dynamically convert QAs at the feature-level into the configuration-level, and quality-aware optimisation. Our objective is to test the most complete reasoning tools to exploit the extended variability with quality attributes needed for VNFs.

CCS CONCEPTS

• **Software and its engineering → Abstraction, modeling and modularity**; **Software product lines**; **Software performance**; *Requirements analysis*; • **Theory of computation → Automated reasoning**; • **Computing methodologies → Representation of mathematical objects**.

KEYWORDS

virtual network function, quality attribute, variability, numerical feature. reasoning, optimization

ACM Reference Format:
Daniel-Jesus Munoz, Mónica Pinto, and Lidia Fuentes. 2022. Quality-aware Analysis and Optimisation of Virtual Network Functions. In *Proceedings of 26th ACM International Systems and Software Product Lines Conference (SPLC'22).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The softwarisation and virtualisation of network functionality is a new trend in Industry 4.0, especially for emergent mobile technologies such as Beyond 5G networks (B5G). The objective is to turn networks into general-purpose platforms providing smart connectivity to a plethora of devices. Software-Defined Networking (SDN) [32] and Network Function Virtualization (NFV) [26], are widely accepted paradigms to address the new structure of network architectures. SDN aims to introduce network programming capability and NFV is an innovative, but complementary paradigm that promotes the virtualisation technology to disengage network functions from dedicated hardware appliances and transform them into software components, so-called virtual network functions (VNFs). Then, user applications demanding a network service turn out to be a request for running a set of VNFs at the application plane on servers. These application services (i.e., VNFs) can be tailored for certain applications family (e.g., virtual reality, video delivery or distributed games), domains like IoT, or allocated to a class of customers, or certain mobile network operators.

Next-generation networks such as 6G promise to provide a large set of agile services, custom-made and providing user-defined Quality of Service (QoS), such as latency or energy consumption. Indeed, there is a growing interest in energy-efficient orchestration of VNFs, being this the main goal of the DAEMON project that supports this work [1]. Therefore, to reason about the variability of network functions and especially about the quality of a software product defined as a set of VNFs instantiated as part of a service (i.e., Service Function Chaining), a variability model along with a quality model is required. Unfortunately, the heterogeneity in the network complicates the relationship between variability and quality of VNFs configurations [42]. In addition, this domain imposes certain challenges to quality-aware reasoning of service function chains, such as numerical features or configuration-level *Quality Attributes* (QAs) such as energy consumption. Incorporating numerical reasoning with quality data into SPL analyses is challenging and tool support is rare.

While the majority of works in SDN area focuses mainly on applying Artificial Intelligence approaches, such as deep learning, reinforcement learning or control theory to proactively adapt VNFs chains to network workload and current resources, little work focus on customizing a set of VNFs considering different alternatives providing variable QoS [11] [21]. In this work, we

[1]European H2020 funded project DAEMON: https://h2020daemon.eu/

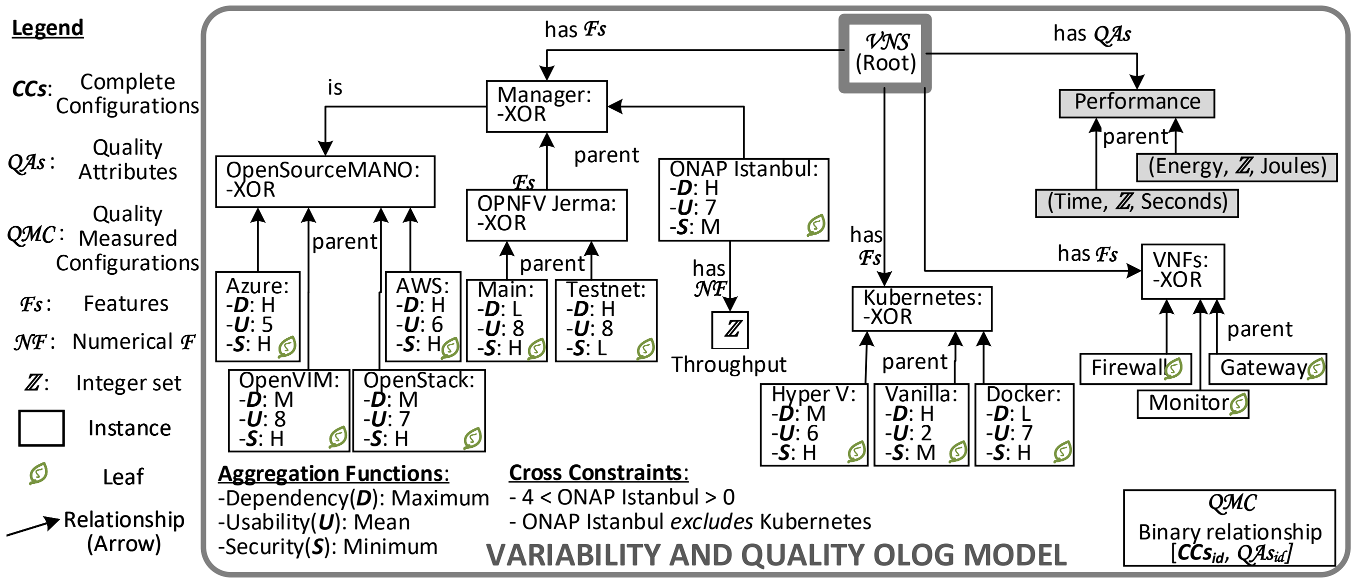

Figure 1: An extract of the variability and quality olog of the software product line *Virtual Network System*($\mathcal{VNS}$)

apply SPL technologies to the quality-aware reasoning, customization and optimization of VNFs chains for specific user services. Typical quality attributes considered in the DAEMON project are latency and energy efficiency, so we argue we need to incorporate *feature-level* and *configuration-level* QAs, the two main approaches for quality reasoning of variability models according to [39, 47]. *Feature-level QAs*, are modelled as attributes directly linked to single features, such as response time or any cost associated with each feature. Second, *configuration-level QAs*, which cannot be quantified at the feature-level and must be measured and associated at the configuration-level. Two examples of this are performance or energy consumption [39, 47]. However, existing works mainly focus on feature-level QAs using attributes [6, 33, 46, 52] so they do not apply to energy-efficient SDNs, and only a few of them deal with both [46, 57]. Further, these works do not fully support real-world SPLs and advanced reasoning, and neither are easy to extend. For example, FAMA framework [6] supports extended FMs with attributes as quality information. However, it cannot support generating the optimal configuration within a range of values for a specific quality (e.g., minimise a VNF's energy consumption above 1 Watt) [51]. With the objective of modelling and reasoning flexibility, we proposed a *Category Theory* (CT) [3] framework for SPLs. There, we unified FMs and *Quality Models* (QMs) [35] in a single model and tool. It has the potential to support complex relationships and the quality-reasoning of numerical FMs. We consider quality-reasoning to any Automatic Analysis of Feature Models operations with QAs on their resolution. For instance, request VNFs consuming 1 Watt and with a cost under 10$.

In this work, we highlight three groups of operations needed for VNFs' quality-aware reasoning. The first one is a model report, which covers *type and number features, type and number of constraints, size of the configuration and measured spaces, and QAs metadata*. The second group are the aggregate functions, which define how to convert QAs in the feature-level into the configuration-level in the form of *addition, product, mean, and approximation arithmetic equations*. The last one is optimisation like *maximum, minimum, (weighted) multi-objective, and range objective*. In section 3, we analyse the support that current tools provide to those groups of operations. Additionally, we define and implement in our CT framework the necessary reasoning algorithms in Section 3. For the evaluation Section 5, we selected the 4 most complete tools: CQL IDE [35], ClaferMoo [39], AAFM Python Framework [18], and SATIBEA [23]. Alongside our new CT reasoning algorithms, we validated those tools for 5 different real-world SPLs with different QAs and a variety of reasoning operations of the three groups. Finally, we compare their capabilities and performance when generating results. Our main contribution is to provide a set of tools to perform the necessary quality-aware operations for VNFs orchestration.

2 QUALITY-AWARE REASONING OF SDN

VNFs orchestration to compose SFCs has complex requirements for low latency, energy and security among others [32]. In this section, we analyse with an SPL perspective the quality-aware reasoning operations that could guide such orchestration processes. We summarised the operations in 3 groups: model analysis, aggregation of attributes values, and optimal search. For illustration purposes, we will provide several examples based on a reduced SPL of our *Virtual Network Orchestration System* represented by the $\mathcal{VNS}$ model of Figure 1. To include a single model with variability alongside all types of QAs, we draw an olog based on our CT framework. In other words, ologs are the categorical counterpart of completely extended FMs [48].

$\mathcal{VNS}$ is a real-world problem that we are solving in the context of the DAEMON project and contains different virtual network managers, containers for virtualisation software (e.g., *Kubernetes*) and 3 common VNFs, following the proposed standard reference architecture for the Management And Network Orchestration (MANO) of VNFs [26]. Figure 1 shows a simplified version that comprises 18 boolean and 1 numerical features, 1 propositional and 1 arithmetic constraints, and 3 feature-level and 2 configuration-level QAs: hardware *Dependency*, *Usability*, *Security*, *Energy* and *Time* respectively. The performance and energy metrics are obtained after several runs with specialised tools like *Watts Up? Pro* and multimetres, while the other 3 are consensus from industrial partners.

We can find many reasoning operations in the SPL literature [4, 25]. We made a selection based on DAEMON requirements; they can be summarised on an analysis of the virtualised network properties and an optimal orchestration of VNFs:

- *Type and number of the features*: A reasoner counts every feature in the model per type and domain. Classic features have a boolean domain [27] indicating their existence in a configuration. Numerical features expand that domain to integer, real, etc. [36], and identify many states of a feature (e.g., *Throughput*). In Figure 1 the features are boxes identified by $\mathcal{F}s$ and $\mathcal{NF}$ respectively.
- *Type and number of model constraints*: A reasoner counts every model constraint. Classic constraints are propositional logic comprising connectives (i.e., and, or), negations, implications and exclusions [27]. Our SPLs also need arithmetic constraints like inequalities with additions, subtractions, multiplications, divisions and modules. Figure 1 contains one of each summarised at the bottom. Non-linear constraints are theoretically possible as a combination of them (e.g., power) [36].
- *Number of configurations*: A reasoner counts every valid complete configuration of a model; in other words, it counts the configuration space that the model represents. While the fastest reasoners perform it by pure model counting (e.g., sharpSAT and boolean decision diagrams), the most common alternative is to construct and enumerate them (e.g., Clafer) [36]. The configuration space represented by Figure 1 is 63 configurations.

To this set of operations, DAEMON requires specific ones for QAs. These are novel, as the integration of variability and quality models are mainly unexplored as already discussed in Section 1:

- *Number of quality measured configurations*: A reasoner counts how many configuration measurements are for all QAs. As Figure 1 is completely measured for its 5 QAs, its measured configuration space is 63 * 5 = 315 measurements.
- *Number, names, values and domains of feature-level and configuration-level QAs*: A reasoner counts every QA present in the model and details their type, range and domain. QAs can be grouped into 8 different types, where the most common ones are performance, usability and security [9]. Configuration-level QAs in a model provide the range of measurements in the configuration space. However, feature-level QAs provide a range of values for their respective feature space, and additionally, aggregate functions. Finally, any QA must define its domain (i.e., metric). As in Figure 1, where $\mathcal{VNS}$ comprises 5 QAs, hardware *Dependency* (D), *Usability* (U), and *Security* (S) at the feature-level, and *Time* and *Energy* at the configuration-level.

2.1 Aggregate Operations

To compute a configuration-level value of a QA based on feature-level values, we need one aggregate function per QA. Hence, aggregations are functions for approximating the quality of configurations [55]. Feature-level attributes have clear advantages like smaller space (feature versus configuration space) and can be used in prediction functions, but they come at the cost of accuracy, manageability and maintainability (e.g., energy consumption [37]). This means that, while in theory any QA can be represented by attributes and functions, in the real world that niche is shared between the two spaces. Consequently, feature attributes with aggregate functions have their place in SPLs.

The most trivial type of aggregation is the addition (e.g., calculating the final additive cost in $ of individual components [4]), but we could define any sort of arithmetic function, including non-linear functions like Gauss approximation or predictive performance model functions [50]. For instance, the configuration value of $\mathcal{VNS}$ dependency is the *Maximum* value of the individual values in each feature. On the other hand, *Minimum* calculates $\mathcal{VNS}$ security value. Finally, features' usability is calculated as the aggregate *Mean*. Not every QA is a numerical metric [14]. If we take a closer look to Figure 1, we can see that dependency and security range is a non-numerical scale (i.e. **L**ow, **M**edium and **H**igh), on the contrary, usability is numeric (i.e., [0,10]), and hence the aggregations must consider this in their definitions and implementations. To close the discussion about $\mathcal{VNS}$ QAs, we clarify that time in seconds and energy in joules are configuration-level QAs, and therefore they do not need aggregation functions, as these values were obtained directly by experimentation. Indeed, time and energy are two examples of complex QAs, which the most accurate values are obtained by experimentation because they are difficult to calculate with aggregation functions.

2.2 Optimal Search Operations

In SPLs, optimisation problems are the ones of finding, from the quality measured space, the best configurations with certain quality values [39]. To define and guide the search, we must define objective functions. The most common objectives in the literature are maximise and minimise functions. If an objective considers more than one QA, we are dealing with multi-objective optimisation. In this type of complex optimisation is common that no QA can be better off without making another one worse off. Hence, we are in the field of calculating a set of *similar* high-quality configurations - the Pareto frontier.

To prevent confusion, aggregation takes feature-level QAs as input and transforms them into configuration-level QAs by calculating each corresponding value with an aggregates function. While objective function acts more like a configuration space filter where its inputs and outputs are configuration-level QAs.

However, that is very interesting reasoning-wise, as we can apply function composition, allowing us to include feature-level QAs into optimisation problems by pre-aggregating QAs values. For instance, in Figure 1 dependency aggregation is the maximum, which provides the dependency of a $\mathcal{VNS}$ configuration. But then, we can apply the same function as an objective (i.e., maximise) to obtain the configurations with the highest dependency. That would not have been possible without aggregating before optimising.

Considering that composition, we could perform multi-objective optimisation considering both, feature-level and configuration-level QAs, in the same function. For example, only after aggregating, we can define the following objective search for $\mathcal{VNS}$: *Maximise Usability and Security while Minimise Dependency, Time and Energy.*

Additionally, we can define and use new quality domains based on the ones that already exist in the model. We elaborate on this with an example in $\mathcal{VNS}$. As we know that the energy rate of a system is its energy consumption divided by its runtime (e.g., $Watts = \frac{Joules}{Seconds}$), we could redefine the previous multi-objective as *Maximise Usability and Security while Minimise Dependency and Energy Rate*, with *Energy Rate* $= \frac{Energy}{Time}$.

3 RELATED WORK

This section summarises our search for proper reasoning solutions for virtualised networks, including QoS, SPL tools with support for quality reasoning, and optimisation algorithms.

3.1 QoS in Software-Defined Networking

QoS is the capability of a network to provide the required services for selected network traffic. Quality assurance of network services is based on measuring QAs, where the key factors are: path length, throughput, latency/performance, security, hardware dependency, capacity/usability, and energy [34]. To make things worst, those QAs are negatively influencing one another. For example, a high latency (e.g., due to incorrect ordering of VNFs) leads to failure of packet handling policies, thus increasing the vulnerabilities, which degrades security as it creates incidents. Similarly to SPL quality-aware reasoning, there are works in the literature that define QoS-aware approaches to solve these issues. Likewise, hybrid QoS approaches are multi-objective optimisations. Most of the approaches rely on machine learning, like [43] where the authors guide the orchestration of VNFs via Deep Reinforcement Learning. Another example of deep learning from DAEMON partners is vrAIn [1]. An alternative is heuristic algorithms for near-optimal orchestration. An example is [19], where heuristic formulas are based on linear programming. Another alternative is statistic approaches like [8] where a dynamic statistic multiplexing governs the network orchestrator. Finally, we find network intelligence as the new trend [2], where an example from DAEMON partners is Nuberu [20] based on Bayesian algorithms.

3.2 Tools Supporting Quality-Aware Reasoning

Existing SPL tools [25] provide at least the basic features and constraints defined in FODA (Feature-oriented Domain Analysis) [27]. Additionally, each tool supports a different set of extensions, such as numerical features, attributes, and complex constraints [25], and a different set of reasoning operations [4, 5]. We are interested in tools that allow performing some quality reasoning. Regarding the techniques to generate optimal configurations, we included a subset of them, since they are mostly based on the same algorithms (e.g. based on IBEA [22, 23, 44]) and offer similar operations.

Quality Modelling. Typically, an SPL engineer would like to obtain the configurations with a QA below a threshold (e.g., SFC configurations that consume less than 3 Joules) or generate the best-qualified configuration (e.g. trade-off between energy consumption and performance). We have already discussed the differences between feature-level and configuration-level QAs. These QAs are classified in [46] as feature-wise and variant-wise respectively. Feature-level QAs are the most common in the literature, and are supported by ClaferMoo [39], FAMA [6], FeatureIDE [33], pure::variants [2], SPL Conqueror [46] and STEAM [52]. In QAMTool [57] authors use an alternative representation and extend the FM by incorporating QA-specific features in a sub-tree. Another alternative is to have some external storage to relate features and quality measurements as usually done in genetic algorithms (SATIBEA [23], MILPIBEA [44], MO-DAGAME [40]). An exception is the GIA algorithm [38], defined to be applied to an attributed FM that also uses the Z3 solver. Only a few approaches, such as QAMTool [57] and HADAS [37], support QAs at the configuration-level. SPL Conqueror supports them only partially by calculating an approximated value for the feature attributes based on the set of measured configurations during the generation of the product configuration. Our CT framework [35] supports both the feature-level and configuration-level QAs.

Formalising and Solving Variability Models with Qualities. SPL tools (labelled with T: in Table 1) that only support feature-level QAs (ClaferMoo, FAMA, pure::variant) commonly use a declarative paradigm (e.g. CSP, BDD, SAT) to represent the FM and reason about its quality. In other cases, an external quality model is defined (e.g., a goal model), and the QAs measurements are usually linked to the configurations through a database. The FM is still represented using a declarative paradigm, but an additional structure is used to store and reason about configuration-level QAs. This is the case with the SPL Conqueror, HADAS and QAMTool tools. SPL Conqueror creates a performance model by using sampling and aggregation techniques and uses this model to approximate a near-optimal configuration. The HADAS tool uses Clafer plus a relational database, and the QAMTool uses the NFR framework [56] to externally represent QAs in a goal model. For algorithms generating optimum configurations (labelled with A: in Table 1), a genetic algorithm is usually complemented with a representation of the FM as genes and with a measurements database with the feature-level QA measurements. In some cases, a declarative solver is also used, as in the SATIBEA algorithm, which is defined as a combination of an SAT solver and the IBEA genetic algorithm and the GIA algorithm that uses a Z3 solver. In [35], authors discuss the benefits and drawbacks of approaches to defining an external quality model with two important conclusions: (1) most existing solutions are not directly compatible with automated quality-reasoning, and (2) SPL reasoning lacks a *"unified"* model that appropriately supports quality metrics. STEAM uses abduction and deduction reasoning. Our CT framework defines a unified model with native support for quality reasoning, although the algorithms must be provided at run-time, as they are not pre-established in CT tools.

[2]https://www.pure-systems.com/pv-update/additions/doc/latest/pv-user-manual.pdf

Table 1: Support for reasoning about quality in variability modelling.

	T:ClaferMoo	T:FAMA	T:FeatureIDE	T:pure::variants	T:SPL Conqueror	T:QAMTool	T:HADAS	T:STEAM	A:SATIBEA	A:MILPIBEA	A:GIA	A:MO-DAGAME	CT framework
Quality Modelling													
Feature-level QAs (Quality information is linked to individual features)	■	■	■	■	■	■	□	■	■	■	■	■	■
Configuration-level QAs (Quality information is linked to valid configurations)	□	□	□	□	▨	■	■	□	□	□	□	□	■
Formalising and Solving Variability Models with Qualities													
Declarative Paradigm (CSP solver, BDD solver, SAT solver, ...)	■	■	■	■	□	□	□	□	■	□	■	□	□
Declarative Paradigm + Additional Assets (SAT solver + database, ...)	□	□	□	□	■	■	■	□	□	□	□	□	□
Search-Based Software Engineering + FM representation	■	□	□	□	■	□	□	□	■	■	■	■	□
Alternative Formalisation (Abduction/Deduction reasoning, Category Theory)	□	□	□	□	□	□	□	■	□	□	□	□	■
Automatic Quality Reasoning													
Model Analyses Operations (satisfiability, count features, count configurations, ...)	■	■	■	■	■	■	■	■	░	░	░	░	Q
Aggregation Function Operations													
* Addition	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
* Product	Q	□	□	Q	Q	□	Q	□	□	□	Q	□	Q
* Mean	□	□	□	Q	□	□	□	□	□	□	□	□	Q
* Approximation arithmetic equation	□	□	□	□	Q	□	□	□	□	□	□	□	Q
Optimal Search Operations													
* Maximum	Q	Q	□	□	Q	□	□	Q	Q	Q	Q	Q	Q
* Minimum	Q	Q	□	□	□	□	□	Q	Q	Q	Q	Q	Q
* Multiobjective	Q	□	□	□	□	□	□	□	Q	Q	Q	Q	Q
* Range optimisation	Q	□	□	□	Q	□	□	□	□	□	□	□	Q

■ It supports the characteristic.
▨ It partially supports the characteristic.
□ It doesn't support the characteristic.
░ Out of the scope of the approach.
Q Quality-aware version of the operation.
T: The approach is a SPL tool.
A: The approach is an algorithm that takes a FM as input.

Automatic Quality Reasoning. All SPL tools (labelled with T: in Table 1) offer some level of model analysis operations. ClaferMoo, FAMA, FeatureIDE, pure::variants and STEAM provide implementations of all or a subset of the operations defined in [5] (e.g. satisfiability, type and number of features, type and number of model constraints, number of configurations). Others (e.g. SPL Conqueror, QAMTool, HADAS) use a third-party variability modelling language that provides such support. Algorithms (labelled with A: in Table 1) focus on optimisation, and these model-analysis operations are out of their scope. Regarding quality-aware operations, current approaches do not natively support the complete set of quality-aware operations. Native support would mean that the variability model implements quality-enriched operations as primitives. Regarding the aggregation function and the optimal search operations, the support is variable, as shown in Table 1. The operations supported by ClaferMoo are almost as complete as in our approach. It supports both addition and product aggregation functions, and it supports all the optimisation operations under consideration in this paper. pure::variants also supports addition, product and mean aggregation functions, although approximation arithmetic equations are not supported, and thus, reasoning about the combination of several quality attributes is not possible. Neither optimal search operations are supported. SPL Conqueror supports addition, product and some equations and allows optimal search operations with maximum and ranges. FeatureIDE, QAMTool and HADAS do not provide any support for optimisation. Regarding the genetic algorithms, they approximate optimal configurations using sampling strategies and considering feature-level QAs. They all support the addition aggregation function and the maximum, minimum and multi-objective search operations. They do not support range optimisation. Again, our CT framework has the potential to support all the quality-aware operations discussed in Table 1, but the reasoning algorithms must be implemented in CT and provided at run-time.

There are other works related to the quality of SPLs [13] [15] [54] [31, 41, 53] [10, 12, 16, 17, 30] that do not focus on the same quality-reasoning operations as our approach, so they are out of the scope of this paper.

4 QUALITY-AWARE ALGORITHMS FOR CATEGORY THEORY FRAMEWORKS

Our objective with this section is to provide a running alternative to the tools and algorithms of Table 1 covering as many operations as possible. Consequently, we use our CT framework for SPLs to define the algorithms necessary for the operations discussed in

Section 2. Although this part of our work can be applied to any quality-enriched SPL, we highlight that this work was developed in the context DAEMON project, and other projects that apply SPLs to networking, IoT and Edge-computing systems (see the acknowledgement section). The contributions of this work make possible practical use of SPLs for energy-aware orchestration of VNFs, which had been impossible or at least much harder with current SPL approaches with a limited capacity for quality-aware reasoning of configuration-level quality attributes like energy footprint.

4.1 Foundations of Category Theory

Category Theory (CT) is an algebraic theory of mathematical structures [3]. It allows to capture and relate similar structures while abstracting from the individual specifics of their dissimilarities. A category C represents spaces as a collection of *objects* with functional relationships via *arrows* (i.e., *morphisms*). The key concepts of CT are:

- **Object**: a structured class $X \in \mathrm{Ob}(C)$, graphically depicted as a node $\bullet^X$.
- **Arrow**: a structure-preserving function depicted $\overset{X}{\bullet} \xrightarrow{a} \overset{Y}{\bullet}$.
  - **Identity**: for every $X \in \mathrm{Ob}(C)$, we have the arrow $\overset{X}{\bullet} \xrightarrow{id} \overset{X}{\bullet}$.
  - **Composition**: if $\overset{X}{\bullet} \xrightarrow{a_1} \overset{Y}{\bullet}$ and $\overset{Y}{\bullet} \xrightarrow{a_2} \overset{Z}{\bullet}$, then also $\overset{X}{\bullet} \xrightarrow{a_2 \circ a_1} \overset{Z}{\bullet}$. Composition is associative, i.e., $a_1 \circ\ (a_2 \circ a_3) = (a_1 \circ\ a_2) \circ a_3$.
- **Category**: $\mathrm{Ob}(C) \cup \mathrm{Arr}(C)$ in a labelled directed graph.
- **Functor**: a process F between categories C and D depicted $\overset{C}{\bullet} \xrightarrow{F} \overset{D}{\bullet}$, which preserves identity and function composition.

Also, we shall introduce algebraic data integration CT concepts [7]:

- **Path**: a finite sequence of composed arrows: $\overset{X_0}{\bullet} \xrightarrow{a_1} \overset{X_1}{\bullet} \cdots \overset{X_{n-1}}{\bullet} \xrightarrow{a_n} \overset{X_n}{\bullet}$.
- **Generalised Element**: a morphism $\overset{U}{\bullet} \xrightarrow{elem} \overset{X}{\bullet}$, where U is a select "unit" object.
- **Instance**: a set-valued functor assigning values to elements.

4.2 Unifying Variability and Quality in a Categorical Model

In [35], we detail a CT framework that unifies numerical VMs with QAs as a category where features and QAs are objects, and data types, hierarchical relationships, and quality and feature constraints are arrows. We use this model to represent SPLs as categories. The transformation is graphically represented in Figure 2, being the basis for the algorithms for quality-aware operations that we detail in the next subsection.

Concretely, our framework comprises 3 data-type objects (i.e., Boolean, Integer, and String for characters sets) and 5 structured objects. Figure 2 helps an tiny example based on the SPL represented in Figure 1 model.

The most important one is the **Schema**, which defines the unified variability and quality model structure: elements, properties, hierarchical relationships and structural relationships. We can think about them as arrays of variables without set values. Naturally, *Features* represents any feature domain of FMs, *Feature Level Qualities* represents extended FMs attributes, and *Qualities* the items present in quality models (i.e., similar to features in an FM). The rest of the elements are sets of identifiers or a set of related identifiers (*Binary Relationship*). Those 3 last elements are necessary to relate configuration-level QAs to the respective set of features forming a specific *Configuration*.

The rest are **Instances** of certain **Domains** of the elements of the schema. In other words, they populate the schema. For simplicity, the schema would be blank FM, and the instances are the names of the features, cardinalities, etc.

4.3 Quality Operations in Category Theory

Considering that we can use the variability and quality modelling framework within CT and that we analysed the quality-aware reasoning operations, we need to find a flexible CT reasoner that supports the implementation. Consequently, we choose the CT state-of-the-art tool: the *Categorical Query Language* (CQL) IDE [3]. CQL is a functorial language used for functional programming based on lambda calculus. While for low-level details we kindly point the readers to [29], we present now an overview of the CQL main assets:

- Basic data types and functions are defined as global objects and arrows (e.g., $\mathbb{B}$ for boolean domain).
- A structured category is a *schema* of objects and different types of arrows (e.g., Figure 1).
- A functor is a *query* over an input schema to an output schema. For composed reasoning, the input schema of an intermediate functor must be the same as the output schema of the previous functor.
- A *literal instance* generates variables and assigns the values.
- The reasoning is an *eval instance* of a schema literal.

Having all the necessary background, we can now implement categorical reasoning in CQL IDE. We repeat the same sequence, hence starting with model analysis operations.

In Algorithm 1 we merged all the self-analysis operations in a single operation called *Model Report*. Its inputs are certain categorical objects of the model: the features (i.e., $\mathcal{F}$s), the complete configurations identifier (i.e., CCs), the QAs identifier (i.e., $\mathcal{QA}$s) and their relationships in the *Quality Measured Configurations* (QMC). Its outputs are calculated with a composition of 9 lambda functions, which sequentially are:

(1) Number of *bool*ean and *num*erical features given by instantiated elements in $\mathcal{F}$s.
(2) Number of first-order *logic* and *arithm*etic constraints given by instantiated elements in $\mathcal{F}$s without/with inequalities or numerical calculations respectively.
(3) The size of the *conf*iguration and the *measured* spaces given by the valid identifiers in CCs and their presence in QMC respectively. In other words, how many configurations have been measured for each QA.
(4) The meta-data of feature and configuration-level QAs, which are extracted from $\mathcal{F}$s and $\mathcal{QA}$s objects respectively.

While we defined model report as operations directly performed on the model (i.e., on the feature space), some reasoners perform them on the configuration space, once configurations are generated.

[3]CQL IDE main website: https://www.categoricaldata.net/

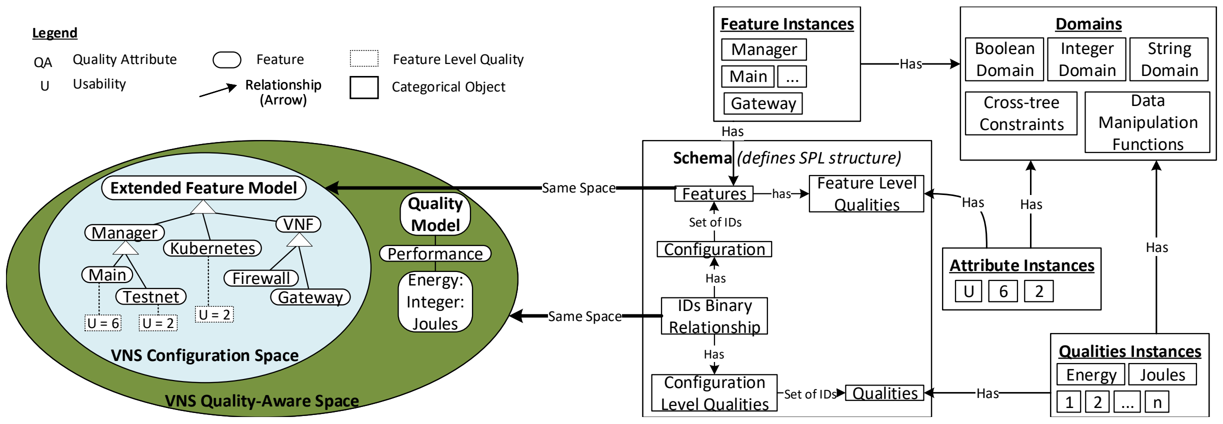

Figure 2: Unification of Extended Variability Models and Quality Models into a Category

While the feature space is more complex, is also more scalable. The flexibility to functionally configure a reasoner is one of the main advantages of CT tools, and that alongside compositions is a declared need for advanced reasoning of SPLs [52].

```
Algorithm 1: Categorical Model Report
  Input: Populated $\mathcal{F}s$, $CCs$, $\mathcal{QMC}$, $\mathcal{QA}s$
  (bool, num, logic, arithm, confs, measured[]) = [0, . . ., 0];
  bool = Add($\lambda(x \in \mathcal{F}s \mid !x.num) : 1$);
  num = Add($\lambda(x \in \mathcal{F}s \mid x.num) : 1$);
  logic = Add($\lambda(x \in \mathcal{F}s.where \mid x.ops \notin [=,+,...,\%]) : 1$);
  arithm = Add($\lambda(x \in \mathcal{F}s.where \mid x.ops \in [=,+,...,\%]) : 1$);
  confs = Add($\lambda(CCs.id) : 1$);
  measured[QA]=Add($\lambda(CC.f.att \vee CC.id \in \mathcal{QMC}) : 1$);
  (FQAs, CCQAs) = [' ', ' '];
  FQAs = ConcatIfNew($\lambda(x \in \mathcal{F}s.att)$ :
    $[x.name, x.val, x.dom, x.AggregationFunction]$);
  CCQAs =
    ConcatIfNew($\lambda(x \in QAs) : [x.name, x.val, x.dom]$);
  Result: bool, num, logic, arithm, confs, measured, FQAS, CCQAs
```

Next in line is Algorithm 2 where we approximate the configuration-level value of each feature-level QA based on its specific aggregation function. The method consists of going over every configuration identifier in CCs, and retrieving their respective features, attributes and functions (originally located in $\mathcal{F}s$). That retrieved information is the input of a lambda function, which simply runs aggregate functions with their related attributes (i.e., $f(x)$).

Finally, we present Algorithm 3 where we search for configurations with desired QAs. To cover all types of QAs in this algorithm, we pre-composed Algorithm 2, whose results are temporarily stored in the extended $\mathcal{QMC}$ (i.e., eQMC) structured object. Provided that, the algorithm goes through every configuration identifier where the lambda function filters them based on the provided objective function. For clarity reasons we simplified the resulting data as only generating identifiers, but if we needed feature names and final QA values, we would need to provide and access $\mathcal{F}s$ and $\mathcal{QA}s$ objects within the algorithm.

```
Algorithm 2: Categorical Aggregation of Attributes
  Input: Populated $CCs$
  AggregatedQAs = [][];
  forall $cc \in CCs$ do
      Func = [];
      Func = Push($\lambda(x \in cc.feature.att) : x.function$);
      AggregatedQAs[cc.id] = Push(
      $\lambda(x \in cc.feature.att, f \in Func | f = x.function) : f(x)$);
  end
  Result: AggregatedQAs
```

```
Algorithm 3: Categorical Quality Optimisation
  Input: Populated $\mathcal{QMC}$, $CCs$, Objectives
  eQMC = [$\mathcal{QMC}$.cc, ($\mathcal{QMC}$.qa ∪ Aggregation($CCs$))];
  Result: $(\lambda x \in eQMC \mid Objectives(x.qas)$ :
          $[x.cc.features, x.qas])$
```

5 EMPIRICAL VALIDATION

In this section, we are going to test Section 4 algorithms, comparing their capabilities and reasoning times with the most complete solvers, in the context of VNFs orchestration and configuration. The concrete research questions are:

RQ 1: *Which is the level of empirical support that the state-of-the-art currently has to represent and provide reasoning to quality-measured VNFs?*

RQ 2: *Is our CT framework for SPLs a feasible alternative to analyse and optimise VNFs orchestration?*

RQ 3: *How do the alternative tools scale for the complete set of quality-aware reasoning operations present in SDNs systems?*

5.1 Methodology and Setup

Table 2 shows the 5 SPLs we have used for validation. They are all real-world SPLs with different properties (e.g., size) as a means to reinforce the results and conclusions. As far as we know, our approach is the only work that applies SPLs to SDN/NFV domain, so we had to use models from other domains as validation objects. Also,

Table 2: Variability details generated with reporting operations in CQL IDE of the SPLs with quality attributes analysed

SPL	Description	Features	Constraints	Configurations	Quality Attributes
$\mathcal{Pizza}$	Italian vendor machine [28]	• Boolean: 12 • Numerical: 1	*Empty*	• Total SAT: 42 • Measured: 84	• Feature level: (1) Cost ∈ (5,25) $: Function: Addition • Configuration level: (2) Time ∈ (1,10^3) *seconds*
$\mathcal{Truck}$	Truck factory [45]	• Boolean: 33	• Logic: 10	• Total SAT: 234 • Measured: 234	• Feature level: (1) Size ∈ [0,10) *metres*^{2}: Function: Product
$\mathcal{JHipster}$	Software generator [24]	• Boolean: 45	• Logic: 13	• Total SAT: 26,256 • Measured: 105,024	• Feature level: (1) Usability ∈ (0, 10): Function: Addition (2) Battery ∈ (0, 20): Function: Addition (3) Footprint ∈ (0, 10): Function: Addition • Configuration level: (4) Compileable ∈ [true, false]
(1) $\mathcal{VNS}$ and (2) *Full*$\mathcal{VNS}$	Virtual Network System	Figure 1 version: • Boolean: 18 • Numerical: 1 Full version: • Boolean: 40 • Numerical: 3	Figure 1 version: • Logic: 1 • Arithmetic: 1 Full version: • Logic: 63 • Arithmetic: 4	Figure 1 version: • Total SAT: 63 • Measured: 315 Full version: • Total SAT: 2,130,000 • Measured: 10,650,000	• Feature level: (1) Dependency ∈ [L, M, H]: Function: Maximum (2) Usability ∈ (1, 10): Function: Mean (3) Security ∈ [L, M, H]: Function: Minimum • Configuration level: (4) Time ∈ (1,10^3) *seconds* (5) Energy ∈ (1,10^3) *joules*

quality-measured virtualised networks use cases are not available in the literature, so 3 of the SPLs are from different domains, but, they are third-party use cases, well-known in the literature, and share certain quality-reasoning requirements like *Cost in $* and *Battery* minimisation.

We present them ordered by the size of their configuration space. The smallest is $\mathcal{Pizza}$, taken from [28] and including the QAs additive *Cost* in $ at the feature-level and *Time* in seconds at the configuration-level. With 6 times its configurations space, we have $\mathcal{Truck}$, which we extracted from [45] and extended with random values of a multiplicative *Size* in squared metres as a feature-level QA. A larger case with a measured space 449 times bigger is $\mathcal{JHipster}$, which already included the QAs: *Usability*, *Battery*, and memory *Footprint* as additive QAs at the feature-level, and textit-Compileable, a binary QA (i.e., yes or no) at the configuration-level. The largest model is the complete version of the one represented in Figure 1, the DAEMON's $\mathcal{VNS}$ SPL. For completeness, we also included the reduced version represented in Figure 1. We distinguish them as *Full*$\mathcal{VNS}$ and $\mathcal{VNS}$ respectively. *Full*$\mathcal{VNS}$ comprises 40 boolean and 3 numerical features, 64 logic and 4 arithmetic constraints, and space of 2+ million configurations. Its QAs are the same as in Figure 1; consequently, its QAs space is of 10+ million configurations measurements.

Considering our analysis summarised in Table 1, we selected the most complete open-source tools for each group of reasoning operations: (1) ClaferMoo [39] due to its support of feature-level QAs and certain flexibility to define functions; (2) AAFM Python framework for its speed at the cost of not supporting QAs, (3) SATIBEA [23] as the representative of IBEA based genetic algorithm for optimisation due to its documentation and user support, and finally (4) CQL IDE, the state-of-the-art CT tool. The different models and data-sets are available at:

https://github.com/danieljmg/SPLC22

We ran the presented SPLs and tools on a desktop computer comprising an Intel(R) Core i7-4790 CPU@3.60 GHz processor with 16 GB of memory RAM and an SSD running an up-to-date Windows 10 H22H1 X86_64 with the latest supported versions of the tools and shared libraries (e.g., Java JDK 18.0.2). Besides double-checking the internal statistics of each tool, we measured reasoning time with the Windows PowerShell *Measure-Command {. . .}*. Initial JAVA virtual machine overhead was purposely removed, and it is not affecting the time results.

5.2 Self-analysis and optimisation operations results

In Table 3 we present the first set of results in the form of reasoning time in seconds; as SATIBEA does not currently support these operations, it is not present in this comparison. We grouped them into 4 operations which should return similar information to what

Table 3: Averaged performance in seconds of reporting operations in state-of-the-art reasoners for Table 2 models We tagged the performance with asterisks if the reasoner ignored QAs at the configuration-level

Reasoner:	ClaferMoo				AAFM Python Framework				CQL IDE			
SPLs:	$\mathcal{P}izza$	$\mathcal{VNS}\|Full$	$\mathcal{T}ruck$	$\mathcal{JH}ipster$	$\mathcal{P}izza$	$\mathcal{VNS}\|Full$	$\mathcal{T}ruck$	$\mathcal{JH}ipster$	$\mathcal{P}izza$	$\mathcal{VNS}\|Full$	$\mathcal{T}ruck$	$\mathcal{JH}ipster$
Features	0.2 s	0.2|2.7 s	0.2 s	0.3 s	0.02 s	0.02|0.39 s	0.03 s	0.05 s	0.04 s	0.05|0.55 s	0.06 s	0.07 s
Constraints	0.2 s	0.2|2.73 s	0.2 s	0.3 s	0.02 s	0.02|0.39 s	0.03 s	0.05 s	0.04 s	0.05|0.55 s	0.06 s	0.07 s
Configurations	*0.5 s	*1.05|213 s	*1.4 s	*2.7 s	*0.15 s	*0.2|60 s	*0.24 s	*5.03 s	0.17 s	0.19|156.4 s	0.22 s	1.98 s
QAs	*0.9 s	*1.7|4.3 s	*2.1 s	*2.9 s	Unsupported	Unsupported	Unsupported	Unsupported	0.06 s	0.11|3.4 s	0.14 s	0.19 s
Mean:	***0.45 s**	***0.79|55.6 s**	***0.98 s**	***1.55 s**	***0.06 s**	***0.09|20.2 s**	***0.1 s**	***1.71 s**	**0.08 s**	**0.1|40.3 s**	**0.12 s**	**0.58 s**

has been presented in Table 2. While AAFM Python framework reasoning is fast, it did not support QAs. Likewise, ClaferMoo did not support QAs at the configuration-level. When an operation is completely unsupported, we show in the table *Unsupported*. However, if the support is partial, they are tagged with an asterisk. In Table 4 we present the second set of results that involves the aggregate reasoning. Each column is a variation of the aggregate reasoning, and they are ordered from the simplest to the most complex. The first row is a direct aggregation without constraints of all the QAs of that SPL. The *Constrained* row implies reasoning by randomly excluding one feature. Similarly, the *Range* row limits the values of a (feature-level) QA. Note that we are not constraining based on the aggregated total value but the individual feature-level QA value. In these results, an asterisk means that the reasoner did not support the specific aggregate function. As detailed in Table 1 of the related work in Section 3, ClaferMoo only supports addition and product in $\mathbb{Z}$ domain, while SATIBEA just addition. Hence, they did not support any $\mathcal{VNS}$ aggregate function (i.e., maximum, mean and minimum). In those cases, we swap the domains to $\mathbb{Z}$ *Addition* to allow time comparisons. The last set of results is in Table 5, where we perform an optimisation operation with different types of objectives. We first compose the aggregations for the feature-level QAs. In the first row, we *Min/Maximise* individual QAs and average the runtime results. In the second row, we defined the following multi-objectives:

- $\mathcal{P}izza$: Minimise Time and total Cost.
- $\mathcal{T}ruck$: Minimise total $Size_1$ * $Size_2$.
- $\mathcal{JH}ipster$: Maximise compileable and total Usability $\wedge$ Minimise total Battery and Footprint.
- $\mathcal{VNS}$s: Maximise total Usability and Security $\wedge$ Minimise total Dependency, Time and Energy.

As we can see, in the case of $\mathcal{T}ruck$ we duplicated its single QA *Size* into $Size_1$ and $Size_2$ in order to be able to perform a multi-objective test. The weighted objectives for the third row were similar but included random weights for the different QAs (e.g., Minimise $\mathcal{VNS}$ 0.3**Time*, 0.7**Energy*). Finally, for the last row we defined the following objectives:

- $\mathcal{P}izza$: Minimise total Cost per second.
- $\mathcal{T}ruck$: Minimise $Size_1$ * $Size_2$.
- $\mathcal{JH}ipster$: Minimise total Battery + total Footprint.
- $\mathcal{VNS}$s: Minimise energy rate.

In this case, an asterisk indicates that the reasoner did not support modelling configuration-level QAs. In that case, we provided random values to allow some level of comparison. Nevertheless, ClaferMoo and SATIBEA do not currently support weighted and new domain objectives. SATIBEA sampling parameters are configured as suggested in the documentation: 20000 evolutions [23].

5.3 Discussion and Scalability Results

Which is the level of empirical support that the state-of-the-art currently has to represent and provide reasoning to quality-measured VNFs

In this subsection we answer the RQs by considering the results of Tables 1 to 5. The goal of RQ1 is to assess if we can successfully apply SPL tools to reason about quality-aware orchestration of VNFs, and in general to the SDN/NFV domain [21]. Only those SPLs tools that include numerical features, complex constraints and support reasoning and optimization of configuration-level QA apply to this domain. While in the current situation we find academic tools for very specific and basic reasoning, in practice the network industry will discard them as they are not enough by themselves. Additionally, those tools are not directly configurable or extendable, and there is where CQL IDE with our CT operations highlights. The **RQ1 answer is that the best current tools could be viable if they are extended like we are doing with CQL IDE, that is, they provide a unified solution** beyond boolean features, logic constraints, additive attributes, and max/minimise optimisation.

The **RQ2 answer is that our algorithms feasibly extend CQL IDE for all the quality-aware operations, and hence it provides support for managing QoS in VNFs orchestration**. Although, we should mention that CT knowledge is not very common in the industry, which is required to properly adjust such flexible tools.

Regarding run-times, if we are only analysing SDNs variability, AAFM Python Framework is the fastest with a worst performance of 60 seconds. Similarly, for simple aggregation and optimisation functions, SATIBEA is the fastest. This was expected, as it always works with the same number of samples, and increasing them is not translated to higher accuracy as is stated in the literature [23]. ClaferMoo tends to be the slowest, sometimes due to how their reasoning algorithms work, like counting by enumeration [36]. However, it covers more operations than the average. Nevertheless, we consider CQL IDE the proper alternative for the DAEMON project, as it is among the fastest ones while supporting all sorts of quality-aware reasoning. In summary, and **answering RQ3, all the solutions scale linearly**, but without being always the fastest, CQL IDE shines for its large application domain.

Table 4: Averaged performance in seconds of aggregation operations in state-of-the-art reasoners for Table 2 models
For unsupported advanced aggregations, we switched them for addition and tagged their performance with an asterisk

Reasoner:	ClaferMoo				SATIBEA				CQL IDE			
SPLs:	*Pizza*	*VNS|Full*	*Truck*	*JHipster*	*Pizza*	*VNS|Full*	*Truck*	*JHipster*	*Pizza*	*VNS|Full*	*Truck*	*JHipster*
Unconstrained	0.95 s	*1,8|236 s	2.15 s	2.99 s	1.67 s	*1.79|2.33 s	*1.85 s	1.78 s	0.27 s	0.39|211 s	0.42 s	2.68 s
Constrained	0.97 s	*1.89|237 s	2.17 s	3.01 s	Unsupported	Unsupported	Unsupported	Unsupported	0.27 s	0.4|212 s	0.42 s	2.69 s
Range	0.95 s	*1.8|232 s	2.15 s	2.94 s	Unsupported	Unsupported	Unsupported	Unsupported	0.27 s	0.41|212 s	0.43 s	2.69 s
Mean:	**0.957 s**	***1.83|235 s**	**2.16 s**	**2.98** s	**1.67 s**	***1.79|2.33 s**	***1.85 s**	**1.78** s	**0.27 s**	**0.4|212 s**	**0.423 s**	**2.69 s**

Table 5: Averaged performance in seconds of optimisation operations in state-of-the-art reasoners for Table 2 models
We tagged the performance with an asterisk if the reasoner ignores quality attributes at the configuration level

Reasoner:	ClaferMoo				SATIBEA				CQL IDE			
SPLs:	*Pizza*	*VNS|Full*	*Truck*	*JHipster*	*Pizza*	*VNS|Full*	*Truck*	*JHipster*	*Pizza*	*VNS|Full*	*Truck*	*JHipster*
Min/Maximise	*0.98 s	*1.87|344 s	2.23 s	*4.36 s	*1.67 s	*1.79|2 s	1.85 s	*1.78 s	0.38 s	0.52|274 s	0.68 s	3.48 s
Multiobjective	*1.02 s	*1.97|355 s	2.38 s	*4.5 s	*1.85 s	*2.01|2,1 s	2.03 s	*1.95 s	0.42 s	0.67|312 s	0.78 s	3.95 s
Weighted	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	0.42 s	0.68|317 s	0.82 s	4.02 s
New Domain	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	0.41 s	0.63|274 s	0.73 s	3.47 s
Mean:	***1 s**	***1.92|349 s**	**2.31 s**	***4.43 s**	***1.76 s**	***1.9|2,05 s**	**1.94 s**	***1.87 s**	**0.41 s**	**0.63|294 s**	**0.75 s**	**3.73 s**

5.4 Threats to Validity

Internal Validity. To control randomness, we repeated the experiments 97 times and averaged the results for a confidence level of 95% with a 10% margin of error [49]. Additionally, we are aware of the need to extend this validation with tool's accuracy and the reasons based on their specific reasoning techniques. Nevertheless, we considered the selected experiments and discussion sufficient for the aim of this work.

External Validity. By choosing the real-world SPLs of Table 2 we pretended to cover a variety of properties, QAs and functions commonly found in VNF cases. Nonetheless, we are aware that they do not cover every possible casuistic. While one could argue that large spaces are not enough, and colossal spaces should be tested, we should mention that larger spaces are very rare for VNF orchestrators. The problem in SDN systems is the complexity of the reasoning and not the size of it. Testing our algorithms with just one CT reasoner could be another threat. The problem is that CT tools besides CQL IDE are also rare due to the intrinsic abstraction and knowledge requirement.

6 CONCLUSION AND FUTURE WORK

The domain of SDN and NFV, Edge computing and IoT is challenging for quality-aware reasoning of configurations. AAFM provides reasoning tools and algorithms that we can apply to improve the quality of service, being energy efficiency the most critical in those domains. However, we found limitations when applied to the context of VNFs orchestrations in the DAEMON project. In short, there is a lack of understanding, methods, and tools designed explicitly for advanced quality-aware analysis and optimisation that consider interactions between feature and configuration-level QAs.

In this work, we start by uncovering the quality-based reasoning operations necessary in the DAEMON project and grouped them into: model analysis, aggregation functionality, and optimisation based on objectives. We follow by analysing the state-of-the-art of AAFM methods and tools that supports any share of those operations and summarised the outcomes in Table 1. As we found the need for a complete alternative, we defined and implemented in CQL IDE the quality-aware reasoning algorithms of those operations for our CT framework for SPLs. Next, we empirically tested the state-of-the-art alongside our proposal for 5 different real-world SPLs with several QAs and up to 20 different quality-aware reasoning operations.

For RQ1 we conclude that current tools could be viable if they are extended like we are doing with CQL IDE, that is, they provide a unified solution. For RQ2 we state that a CT tool like CQL IDE has the flexibility and potential to cover all the operations, but its feasibility depends on having CT knowledge in the team – as in our case with the DAEMON project. Finally, in RQ3 we highlight that the selection of the reasoning tool will depend on the set of operations that the SDN system needs; while all the tools scale linearly, some of them are faster than others for specific operations (e.g., SATIBEA for basic near-optimal search). As a final statement, if the objective is that VNF orchestrators automatically rely on AAFM, all the tools in the current literature need to extend their support beyond boolean features, logic constraints, additive attributes and Pareto optimisation. Our CQL IDE algorithms are a solution for that.

As an extension, we plan to analyse the trade-off between scalability and accuracy in optimisation operations. Additionally, we also plan to implement sampling and learning techniques in CQL IDE, as well as exploit other tools.

ACKNOWLEDGMENTS

This work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109, by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 and Rhea P18-FR-1081 and the PRE2019-087496 grant from the Ministerio de Ciencia e Innovación.

REFERENCES

[1] Jose A Ayala-Romero, Andres Garcia-Saavedra, Xavier Costa-Perez, and George Iosifidis. 2021. Demonstrating a Bayesian Online Learning for Energy-Aware Resource Orchestration in vRANs. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–2.

[2] Albert Banchs, Marco Fiore, Andres Garcia-Saavedra, and Marco Gramaglia. 2021. Network intelligence in 6G: Challenges and opportunities. In *Proceedings of the 16th ACM Workshop on Mobility in the Evolving Internet Architecture*. 7–12.

[3] Michael Barr and Charles Wells. 1990. *Category theory for computing science*. Prentice Hall, Hoboken, New Jersey, USA.

[4] David Benavides. 2019. Variability Modelling and Analysis During 30 Years. In *From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*. Springer Berlin Heidelberg, Berlin, Heidelberg, 365–373. https://doi.org/10.1007/978-3-030-30985-5_21

[5] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615 – 636. https://doi.org/10.1016/j.is.2010.01.001

[6] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*, Oscar Pastor and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 491–503.

[7] Kristopher S Brown, David I Spivak, and Ryan Wisnesky. 2019. Categorical data integration for computational science. *Computational Materials Science* 164 (2019), 127–132.

[8] Shubhajeet Chatterjee, Mohammad J Abdel-Rahman, and Allen B MacKenzie. 2021. On Optimal Orchestration of Virtualized Cellular Networks with Statistical Multiplexing. *IEEE Transactions on Wireless Communications* (2021).

[9] Michel R. V. Chaudron. 2012. Quality Assurance in Model-Based Software Development: Challenges and Opportunities. In *Software Quality. Process Automation in Software Development*, Stefan Biffl, Dietmar Winkler, and Johannes Bergsmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–9.

[10] João Choma Neto., Thelma E. Colanzi., and Aline M. M. Miotto Amaral. 2017. Application of Memetic Algorithms in the Search-based Product Line Architecture Design: An Exploratory Study. In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*. INSTICC, SciTePress, Setubal, Portugal, 178–189. https://doi.org/10.5220/0006363201780189

[11] Emanuel et al. Coutinho. 2018. Research Opportunities in Quality Assessment of Internet of Things, Software Defined Networks and Network Function Virtualization Environments. *International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2018)* (2018), 43.

[12] Xavier Devroey, Gilles Perrouin, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2016. Search-Based Similarity-Driven Behavioural SPL Testing. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-Intensive Systems* (Salvador, Brazil) *(VaMoS '16)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/2866614.2866627

[13] Sascha El-Sharkawy, Adam Krafczyk, and Klaus Schmid. 2019. MetricHaven: More than 23,000 Metrics for Measuring Quality Attributes of Software Product Lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 25–28. https://doi.org/10.1145/3307630.3342384

[14] Sascha El-Sharkawy, Adam Krafczyk, and Klaus Schmid. 2020. Fast Static Analyses of Software Product Lines: An Example with More than 42,000 Metrics. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems* (Magdeburg, Germany) *(VAMOS '20)*. Association for Computing Machinery, New York, NY, USA, Article 8, 9 pages. https://doi.org/10.1145/3377024.3377031

[15] Sascha El-Sharkawy, Nozomi Yamagishi-Eichler, and Klaus Schmid. 2019. Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology* 106 (2019), 1–30. https://doi.org/10.1016/j.infsof.2018.08.015

[16] Thiago do Nascimento Ferreira, Silvia Regina Vergilio, and Marouane Kessentini. 2020. Applying Many-Objective Algorithms to the Variability Test of Software Product Lines. In *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing* (Natal, Brazil) *(SAST 20)*. Association for Computing Machinery, New York, NY, USA, 11–20. https://doi.org/10.1145/3425174.3425211

[17] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2018. Achieving Feature Location in Families of Models Through the Use of Search-Based Software Engineering. *IEEE Transactions on Evolutionary Computation* 22, 3 (2018), 363–377. https://doi.org/10.1109/TEVC.2017.2751100

[18] José A. Galindo and David Benavides. 2020. A Python Framework for the Automated Analysis of Feature Models: A First Step to Integrate Community Efforts. In *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B (SPLC '20)*. ACM, New York, NY, USA, 52–55. https://doi.org/10.1145/3382026.3425773

[19] Meihui Gao, Bernardetta Addis, Mathieu Bouet, and Stefano Secci. 2018. Optimal orchestration of virtual network functions. *Computer Networks* 142 (2018), 108–127.

[20] Gines Garcia-Aviles, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Pablo Serrano, and Albert Banchs. 2021. Nuberu: Reliable RAN virtualization in shared platforms. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 749–761.

[21] M. Gramaglia, L. Fuentes, M. Pinto, Daniel-Jesus Munoz, and et al. 2021. DAEMON Deliverable 3.1: Initial design of real-time control and VNF intelligence mechanisms. This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No.101017109..

[22] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2019. SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* 18, 2 (2019), 1447–1466.

[23] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. Association for Computing Machinery, New York, NY, USA, 517–528. https://doi.org/10.1109/ICSE.2015.69

[24] Jose-Miguel Horcas, José A. Galindo, Ruben Heradio, David Fernandez-Amoros, and David Benavides. 2021. *Monte Carlo Tree Search for Feature Model Analyses: A General Framework for Decision-Making*. Association for Computing Machinery, New York, NY, USA, 190–201. https://doi.org/10.1145/3461001.3471146

[25] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. Software Product Line Engineering: A Practical Experience. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. ACM, New York, New York, USA, 164–176. https://doi.org/10.1145/3336294.3336304

[26] ETSI ISG. 2014. Network Functions Virtualisation standards. (2014).

[27] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[28] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is There a Mismatch between Real-World Feature Models and Product-Line Research?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 291–302. https://doi.org/10.1145/3106237.3106252

[29] Georg P Loczewski. 2018. *A++ and the Lambda Calculus: Principles of Functional Programming*. tredition, Berlin, Germany.

[30] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. 2015. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology* 61 (2015), 33–51. https://doi.org/10.1016/j.infsof.2015.01.008

[31] Lars Luthmann, Timo Gerecth, and Malte Lochau. 2019. Sampling strategies for product lines with unbounded parametric real-time constraints. *International Journal on Software Tools for Technology Transfer* 21 (2019), 613–633. https://doi.org/10.1007/s10009-019-00532-4

[32] T. Mahmoodi M. Condoluci. 2018. Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges. *Computer Networks* 146 (2018), 65–84.

[33] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Quality Assurance for Feature Models and Configurations*. Springer International Publishing, Cham, 81–94. https://doi.org/10.1007/978-3-319-61443-4_8

[34] Seyedakbar Mostafavi, Vesal Hakami, and Maryam Sanaei. 2021. Quality of service provisioning in network function virtualization: a survey. *Computing* 103, 5 (2021), 917–991.

[35] Daniel-Jesus Munoz, Dilian Gurov, Monica Pinto, and Lidia Fuentes. 2021. Category Theory Framework for Variability Models with Non-functional Requirements. In *Advanced Information Systems Engineering*, Marcello La Rosa, Shazia Sadiq, and Ernest Teniente (Eds.). Springer International Publishing, Cham, 397–413.

[36] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France). ACM, New York, New York, USA, 289–301. https://doi.org/10.1145/3336294.3336297

[37] Daniel-Jesus Munoz, Mónica Pinto, and Lidia Fuentes. 2018. Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant. *Computing* 100, 11 (2018), 1155–1173.

[38] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference - Volume 1* (Florence, Italy) *(SPLC '14)*. Association for Computing Machinery, New York, NY, USA, 92–101. https://doi.org/10.1145/2648511.2648521

[39] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. 2012. Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*

(Innsbruck, Austria). ACM, New York, New York, USA, Article 2, 6 pages. https://doi.org/10.1145/2420942.2420944

[40] Gustavo G. Pascual, Roberto E. Lopez-Herrejon, Mónica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *J. Syst. Softw.* 103 (2015), 392–411. https://doi.org/10.1016/j.jss.2014.12.041

[41] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product Sampling for Product Lines: The Scalability Challenge. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 78–83. https://doi.org/10.1145/3336294.3336322

[42] Tie Qiu, Jiancheng Chi, Xiaobo Zhou, Zhaolong Ning, Mohammed Atiquzzaman, and Dapeng Oliver Wu. 2020. Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2462–2488.

[43] Joan S Pujol Roig, David M Gutierrez-Estevez, and Deniz Gündüz. 2019. Management and orchestration of virtual network functions via deep reinforcement learning. *IEEE Journal on Selected Areas in Communications* 38, 2 (2019), 304–317.

[44] Takfarinas Saber, David Brevet, Goetz Botterweck, and Anthony Ventresque. 2020. MILPIBEA: Algorithm for Multi-objective Features Selection in (Evolving) Software Product Lines, In Evolutionary Computation in Combinatorial Optimization - 20th European Conference, EvoCOP 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15-17, 2020, Proceedings. *Evolutionary Computation in Combinatorial Optimization - 20th European Conference, EvoCOP 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15-17, 2020, Proceedings* 12102, 164–179. https://doi.org/10.1007/978-3-030-43680-3_11

[45] Anna Schmitt, Christian Bettinger, and Georg Rock. 2018. Glencoe–a tool for specification, visualization and formal analysis of product lines. In *Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0*. IOS Press, Amsterdam, The Netherlands, 665–673.

[46] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL Conqueror: Toward Optimization of Non-Functional Properties in Software Product Lines. *Software Quality Journal* 20, 3–4 (sep 2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9

[47] Norbert Siegmund, Stefan Sobernig, and Sven Apel. 2017. Attributed Variability Models: Outside the Comfort Zone. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. ACM, New York, New York, USA, 268–278. https://doi.org/10.1145/3106237.3106251

[48] David I Spivak and Robert E Kent. 2012. Ologs: a categorical framework for knowledge representation. *PloS one* 7, 1 (2012), e24274.

[49] C.R. Systems. 2022. Sample Size Calc. https://www.surveysystem.com/sscalc.htm.

[50] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, Léo Noel-Baron, and José Galindo. 2017. *Learning-based performance specialization of configurable systems*. Ph. D. Dissertation. IRISA, Inria Rennes; University of Rennes 1.

[51] Pablo Trinidad. 2012. *Automating the analysis of stateful feature models*. Ph. D. Dissertation. Universidad de Sevilla. https://idus.us.es/handle/11441/55765

[52] Pablo Trinidad, Antonio Ruiz-Cortés, and David Benavides. 2013. *Automated Analysis of Stateful Feature Models*. Springer Berlin Heidelberg, Berlin, Heidelberg, 375–380. https://doi.org/10.1007/978-3-642-36926-1_30

[53] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1* (Gothenburg, Sweden) *(SPLC '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3233027.3233035

[54] Tobias Wägemann, Ramin Tavakoli Kolagari, and Klaus Schmid. 2019. ADOOPLA - Combining Product-Line- and Product-Level Criteria in Multi-objective Optimization of Product Line Architectures. In *Software Architecture*, Tomas Bures, Laurence Duchien, and Paola Inverardi (Eds.). Springer International Publishing, Cham, 126–142.

[55] Lanxin Yang, He Zhang, Haifeng Shen, Xin Huang, Xin Zhou, Guoping Rong, and Dong Shao. 2021. Quality Assessment in Systematic Literature Reviews: A Software Engineering Perspective. *Information and Software Technology* 130 (2021), 106397. https://doi.org/10.1016/j.infsof.2020.106397

[56] Anton Yrjönen and Janne Merilinna. 2009. Extending the NFR framework with measurable non-functional requirements. In *Proceedings of the 2nd International Workshop on Non-functional System Properties in Domain Specific Modeling Languages*, Marko Boškoviæ, Dragan Gaševiæ, Claus Pahl , and Bernhard Schätz (Eds.). ACM, New York, New York, USA, 0–14. 2nd International Workshop on Non-functional System Properties in Domain Specific Modeling Languages, NFPinDSML2009, NFPinDSML2009 ; Conference date: 04-10-2009 Through 04-10-2009.

[57] Guoheng Zhang, Huilin Ye, and Yuqing Lin. 2014. Quality attribute modeling and quality aware product configuration in software product lines. *Softw. Qual. J.* 22, 3 (2014), 365–401. https://doi.org/10.1007/s11219-013-9197-z