June 21, 2018

QUASI-OPTIMAL SOFTWARE RECOMMENDATION ANALYTICS SYSTEM (OPERA) FOR NETWORK DEVICES

Gyana Dash
Antonio Nucci
Jasmin Guliani
Cedric Van Labeke
Sam Grimee

See next page for additional authors

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Dash, Gyana; Nucci, Antonio; Guliani, Jasmin; Van Labeke, Cedric; Grimee, Sam; and Dexter, Chris, "QUASI-OPTIMAL SOFTWARE RECOMMENDATION ANALYTICS SYSTEM (OPERA) FOR NETWORK DEVICES", Technical Disclosure Commons, (June 21, 2018)
https://www.tdcommons.org/dpubs_series/1274

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.
QUASI-OPTIMAL SOFTWARE RECOMMENDATION ANALYTICS SYSTEM (OPERA) FOR NETWORK DEVICES

AUTHORS:

Gyana Dash
Antonio Nucci
Jasmin Guliani
Cedric Van Labeke
Sam Grimee
Chris Dexter

ABSTRACT

The Quasi-OPtimal Software Recommendation Analytics system (OPERA) is configured to scan a customer network on a 24/7 basis. OPERA is also configured to proactively notify the customer if a software upgrade is necessary, as well as notify the customer which device or group of devices should be upgraded to the exact needed software versions. The software recommendations are tailored on a per-customer basis and have improved accuracy by using reinforcement learning, constrained based optimization, and an Integer Linear Programming model. OPERA may also present customers with a composite risk score for the current network and recommend upgrades which will minimize the risk.

DETAILED DESCRIPTION

1. Introduction and Problem Statement

Being able to provide an accurate software update to a network device is very beneficial to customers. It allows them the opportunity to enhance the functionality and operational features of the device (and the overall network), and can also provide service and security vulnerabilities fixes for discovered issues hence ensuring the safe operation their network and overlaid business applications. With the ever-changing advances in technology, updating the device software to gain additional features and capabilities helps prevent a network device from soon becoming obsolete and maintain its competitive and operational value for a much longer time. As new network devices are released on the
market with improved features, software updates for existing devices may potentially offer the same improved features at zero or minimal cost to the customer. Therefore, it may not always be necessary to purchase a new device just to stay up-to-date with current technology if the customer is provided with a reliable process which can constantly monitor the network, promptly discover potential software issues and reliably recommend the customer with what steps to take, i.e., an end to end software upgrade plan.

Today’s approach to the network software upgrading process requires the customer to initiate the process by formally opening a service request. The technical service engineers then go through several steps in order to gather detailed knowledge on what network devices and configurations are currently deployed in the customer, correlate such information with advisory bulletins related to known software bugs, security vulnerabilities, product recommendations and manually draft a recommendation based on the outcome of their analysis. This translates into: (i) the customer is the one to initiate the process which is both nerve wrecking (much information shall be entered by the customer while the correct operation of the customer network may be in jeopardy); and (ii) a lengthy process that the technical service personnel has to go through to successfully close the service request (5 hours to 8 days depending on the specific customer deployment). Furthermore, because of the manual process used by the technical personnel, often (iii) the final recommendation is prone to human errors (missing a fix on a critical security vulnerability, or incompatibility of software versions been recommended, etc.), which may require unnecessary rework, add up cost and impact the overall customer experience.

The techniques presented herein attack the problems associated with the automation of an entire network software update process by introducing a novel methodology called Quasi-OPtimal SoftwarE Recommendation Analytics system (OPERA). The benefits of the OPERA are multiple. First, OPERA scans the customer network on a 24/7 basis and proactively notify the customer if a software upgrade is necessary and what device or group of devices shall be upgraded to exactly what software versions. Proactively reaching the customers (before the actual opening of a service request) translates into a great customer experience and a higher level of business intimacy between the vendor and his customers. Second, OPERA automates the entire process from knowledge gathering, to data set correlation and analysis to the final generation of the network software recommendation.
This translates into both a drastic reduction in the time required to generate the final software recommendation (from today 5 hours - 8 days to less than 2 minutes) and an accurate, error-free recommendation which can be safely deployed into the customer network with minimal risk of future rework. More specifically, OPERA addresses the below key business challenges:

1. **Regular customer network scanning to detect network configuration changes.** Proactively understanding the specific configuration and active software features each network device is using is of pivotal importance to accurately identify the set of known software bugs and security vulnerabilities that need to be resolved. OPERA constantly scans each customer network according to a pre-defined time schedule (usually every 24 hours not to hit a device too often) to assess whether any network device was subject to a configuration change since the latest check. Then it precisely pinpoints the exact changes in the device configuration. This capability ensures that any change in network device configuration operated by any customer is automatically detected and promptly made available to the OPERA system.

2. **Auto-generation of the customer composite risk score.** OPERA retrieves all the information needed from each network device subject to a configuration change and derives the device exposure to open software bugs (and associated level of severity) and known security vulnerabilities (and associated level of criticality). It then computes the device composite risk score. If the risk score exceeds a maximum threshold of risk tolerance for the given customer, OPERA generates the customer software recommendation and then triggers a customer notification (with associated detailed recommendation). In addition to computing the risk score for each individual network device, OPERA provides the customer with an aggregated view of risk KPIs by aggregating the risk score across network devices belonging to the same product family (logical aggregation), or deployed in the same geographical site or across the entire customer network (geographical aggregation).

3. **Auto-generation of device (network) software recommendation.** OPERA models the network software recommendation problem as a RISK/REWARD problem which aims at maximizing the business benefits of upgrading a device (group of devices or the
overall network) to a new software version(s) (REWARD) while considering the inherent cost incurred by the customer to deploy the proposed software upgrade (RISK). An inherent cost could be represented by the need for the customer to upgrade the memory bank on one or more devices to host the newly recommended software version(s), or the cost for the customer to operate a network with a large number of distinct software versions running in the network. In deriving the final software recommendation, the OPERA system considers a variety of operational constraints like configuration compatibility between current and recommended software upgrades on a per-device (features and memory compatibility) and across network devices (logical adjacency compatibility).

4. Tailoring software recommendations on a per-customer basis. The OPERA system automatically monitors the acceptance rates of the provided software recommendations per each customer. It does it at the device or group of device level or by product family or group by product families. By tracking what the customer accepts or declines, OPERA automatically generates a customer profile which captures the customer bias towards accepting or declining specific software upgrades. It then uses these historical knowledges to tailor future recommendations per any given customer. Profiles are regularly updated and refreshed over time by weighting more the most recent actions and weight less older actions from any given customer such to keep the profile always current with time.

Although the OPERA system is presented herein in its totality, the following description focuses specifically on the two below technical aspects:

a) Integer Linear Programming (ILP) model that when solved with standard Operational Research (OR) solvers provides optimal software recommendations for network devices. For very large networks, solving the ILP to optimality may not be feasible but meta-heuristics like Taboo Search, Genetic Algorithms, Fuzzy Logic and alike can be used to solve the model for quasi-optimal solutions. The model does consider several operational and business constraints which need to be satisfied when deriving the final recommendations;
b) Reinforcement Learning model that allows the OPERA system to learn directly from historical customers’ actions their likelihood to accept or decline specific software recommendations and use those customer profiles to further fine-tune future recommendations. This enables OPERA to customer-tune the output of the ILP model (1) and weight-in the customer likelihood score of an acceptance or rejection.

Presented in Section 2, below, is an information flow diagram and a description of functional components of the overall OPERA system. Section 3, below, describes in more detail the primary analytics modules mentioned in a) and b) above, while Section 4 provides a few examples of software recommendation successfully provided by the overall OPERA when using real customer data.

2. OPERA: Functional Components

This section introduces OPERA’s information flow used to derive the ultimate recommendation to address the business challenges (1–4) listed in the introduction session. OPERA comprises of six functional components which interact as shown in Figure 1, below. These six functional components include: OPERA Data Modeling, OPERA Feature Identification, OPERA Bug Extraction, OPERA Possible Future State Analysis, OPERA Constrained Optimization and OPERA Reinforcement Learning.
Figure 1. OPERA Information Flow Diagram. Six functional components: Data Modeling and Feature Identification, Bug Extraction, Possible Future States, Constrained Optimization, and Reinforcement Learning.

The OPERA Data Modeling, connects to each device in the customer network and collects specific information by running a set of selected commands on each device. OPERA uses a bank of vendor-specific plugins to execute the correct CLI commands for any given device. For example, for certain devices, OPERA runs the following CLI commands: show version [1], show commands [1], show config [1], show module[1]. Then OPERA parses the output and organize the data into a device blob which includes: Device Name (such as, iahftotkt-sw01.delta.com), OS Type (such as, IOS, IOS-XR), OS Version (such as, 15.0(2)SE5), Image Name (such as, c3750e-universalk9-m), Product Family (such as, Manufactuer1 SwitchABC), PID (such as, WS-C3750X-24), Serial Number (such as, name-sa-20140605), Configuration running on the device, etc. Similar commands are executed for other types of devices to gather the same device information as per above.

The output of the OPERA Data Modeling is a structured device data object (DDO) which is sent to the OPERA Feature Identification Module which parses out the configuration file and identify the set of features which have been activated (and used) by the network device under investigation (like openssl=ON, tcpleak=ON, poodle=ON, etc.). This new information is appended to the device DDO to create the enriched device data object which is referred to herein as an Enriched Device Data Object or EDDO.

Next, the OPERA Bug Feature Extraction module, processes each EDDO record to identify the set of known open bugs (and associated risk level) and security vulnerabilities (and associated risk level) related to the current state of the device. It does it by connecting to vendor-specific data repositories and correlate all the information using the EDDO meta-fields. In case of certain devices, OPERA connects to internal backend systems, such as the bug database. The output of the OPERA Bug Feature Extraction Module is then consumed by the OPERA Possible Future States. In addition to that, OPERA retrieves Business Unit recommended OS Version. The Business Unit Recommended Version is then merged into the 2 lists of OS Versions obtained from previous steps to check the security vulnerability if the device is upgraded to these OS Versions. After this step, OPERA checks for known identified bugs for the candidate OS Versions to avoid any
mishap after the OS upgrade. In addition to these steps, OPERA checks as well if the candidate OS Version is compatible with the current device image, if it supports all the features currently activated on the device and if the OS Version is not due to an end-of-life event. Finally, the short-listed OS Version become candidates for future states, and OPERA retrieves the required minimum memory to properly install the OS Version on the device.

Next, the OPERA Constrained Optimization module consumes such information to provide the software recommendation for any given device, or group of devices or all devices comprising the customer observed network. The Opera Constrained Optimization module takes into account the below operational and business constraints in deriving the recommendation: (i) the reward of upgrading the software of any given device in the network (i.e., decrease the exposure to software bugs or security vulnerabilities), (ii) the cost of upgrading the software that the customer might be subject to, (iii) the network compatibility of software upgrades affecting network devices which are logically connected (i.e., topology compatibility software upgrade) and (iv) the cost of network operation that the customer may be subject to operate the network with different software configurations deployed. Notice, that not all network devices may be recommended to be upgraded to a new OS software version during this process; only devices for which a sizable business and operation can be encountered are proposed by the OPERA system.

The software recommendations are then presented to either a domain expert which is called to review the set of recommendations and select the ones which fits best the risk profile of the customer under analysis. Notice, that the user can modify both the reward and cost functions provided as an input to the model to account for different needs; each selection of reward/cost functions will drive the model towards a different recommendation and hence a pool of candidate software recommendation to select from.

Last, the OPERA Reinforcement Learning module organizes customers in logical groups where each group containing customers with similar risk profiles; for example, one grouping strategy could be based on the verticals and business domains customer belong to, like ISP, CSP, Small, Medium and Large Enterprise, etc. (vertical) and associated business domains like Financials, E-Commerce, etc. Then, it records the accepted software recommendations (per device, group of devices) for each logical group and leverage this knowledge to automatically recalibrate the reward/cost functions to incorporate customer
preferences towards specific software upgrades. As a result, the OPERA Reinforcement Learning module can provide future recommendations for customers. Over time, the module is capable to provide future customers belonging to any logical group with software recommendations which do have a very high likelihood to be accepted at the first attempt.

Section 3, below, primarily focuses on the technical novelties of the Constrained Optimization Module (module 5 in Figure 1) and the Reinforcement Learning Module (module 6 in Figure 1).

3. OPERA: Constrained Optimization Module

3.1 Input Preparation: Reward Matrix \( W \) and Cost Matrix \( C \)

As discussed in Section 2, the output of the OPERA Bug Feature Extraction Module is a list of candidate OS Version for any given device with associated meta-information like software vulnerabilities and severity, software bugs and severity, memory requirement of its installation, etc. The OPERA Constrained Optimization Module uses this information as an input to calculate both the reward and cost factors for each candidate OS. More formally, the module prepares two matrices, the reward matrix \( W = [w_{ij}] \) and the cost matrix \( C = [c_{ij}] \), that are then used by the ILP model.

Next, introduced is an example of how the reward matrix and the cost matrix can be computed though other formulations can be provided to account for other factors like perceived risk or risk profile of any given customer or else. In this specific implementation, first defined is the risk \( r_i \) associated to a specific device with configuration \( u_i = \{P, OS, F\}_i \), where \( P_i \) is the product type of the device, \( OS_i \) the current Operating System software version and \( F_i \) the set of enabled features on the device. Now, it is possible to compute the device risk by considering the following risk factors: (i) \( \Theta_i = \{\theta_{ih}, \zeta_{ih}\} \) represent the set of known vulnerabilities (or PSIRTS) where \( \theta_{ih} \) be the specific vulnerability and \( \zeta_{ih} \) be its corresponding level of severity; \( \Gamma_i = \{\gamma_{ih}, \eta_{ih}\} \) represent the set of known open software bugs where \( \gamma_{ih} \) be the specific software bug and \( \eta_{ih} \) be its corresponding level of severity. With this definition, it is possible to compute the total risk \( r_i \) of a device configuration \( u_i \) as shown in Equation 1, below.

Equation 1:
\[ r_i = \sum_{h \in \Theta} \theta_{ih} \zeta_{ih} + \sum_{h \in \Gamma} \gamma_{ih} \eta_{ih} \]

That is Equation 1 represents the risk profile of any device with current configuration \( u_i \) when considering both the risk related to known and unfixed security vulnerabilities (with associated level of importance or severity) and known and unfixed software bugs (with associated level of criticality).

It is possible to define the relative weights \( \zeta_{ih} \) as 16 if the severity of the exposed security vulnerability is Critical, 8 if High, 4 if Medium and 2 if the severity is Low. Then, it is possible to compute the reward factor of upgrading a device from its current configuration state \( u_i \) to the new configuration state \( v_j \) as \( w_{ij} = r_i - r_j \).

Similarly, the cost \( c_{ij} \) of a software upgrade for a device with current configuration state \( u_i \) to the new software configuration state \( v_j \) can be defined as a combination of (i) the criticality of the software milestone \( \pi_j \) related to the new software configuration \( v_j \) and (ii) cost \( \tau_{ij} \) of upgrading the memory bank of the device if more memory is required on the device to host the new software configuration \( v_j \) compared to the current software configuration \( u_i \) as shown in Equation 2, below.

**Equation 2:**
\[ c_{ij} = \pi_j + \tau_{ij} \]

That is, Equation 2 represents the cost of a software upgrade from the current configuration state \( u_i \) to the new configuration state \( v_j \).

It is then possible to define the criticality of the software milestone \( \pi_j \) associated to \( v_j \) as: 1 if the new software configuration has been disclaimed to be in End-Of-Life state, 0.5 if it is in End-Of-Software-Vulnerability state, 0.33 if in End-Of-Engineering state and 0 if in End-of-Sales state. For the cost term related to an upgrade of the memory bank \( \tau_{ij} \), it is possible to set it to 0 if no additional memory is required to upgrade the current configuration \( u_i \) to the new configuration state \( v_j \).

Both the reward matrix \( W \) and the cost matrix \( C \) are passed as input parameters to the OPERA Constrained Optimization which use them to derive the optimal software recommendation.
3.2 The ILP Model for Software Recommendations: Relaxation of the Maximum Weighted Semi-matching for Bipartite Graph with Linear Constraints

One of the classical combinatorial optimization problems is finding a maximum matching in a bipartite graph. Formally, a bipartite graph is a graph $G = (U \cup V, E)$ in which $E \subseteq U \cup V$. A matching in $G$ is a set of edges, $M \subseteq E$, such that each vertex in $U \cup V$ is an endpoint of at most one edge in $M$. In other words, each vertex in $U$ is matched with at most one vertex in $V$ and vice-versa. In this section, proposed is a relaxation of the maximum bipartite matching problem applied to the problem of software configuration upgrade. It is possible to define a semi-matching to be a set of edges, $M \subseteq E$, such that each vertex in $U$ is an endpoint of exactly one edge in $M$. Clearly a semi-matching does not exist if there are isolated $U$-vertices, so it is required that each $U$-vertex in $G$ have degree at least 1.

One objective is to find a semi-matching of maximum weight.

Next, introduced is some mathematical notation to later formalize the problem.

Let $X$ be a customer network be compromised of $D$ network devices. Each network device is univocally characterized by a combination of (i) product family the device belongs to, (ii) the OS software version (OS) installed in the device and (iii) a set of features $F$ which are enabled on the device. It is possible to organize the network devices $D$ into $I$ distinct groups where each group comprises devices with same properties (i) to (iii). As a consequence, each group of network devices $D_i$ is univocally identified by the group configuration state $u_i = \{P, OS, F\}_i$, where $P_i$ is the corresponding product type of the devices in the group, $OS_i$ the associated Operating System software version software version and $F_i$ the set of enabled features on those devices. The cardinality of all active configuration states in the customer network equals to the number of derived groups $I$. Notice that any network device belongs to one and only one cluster, i.e. $\cap D_i = \{0\}$, and that $D = \bigcup D_i$. Let $U = \{u_i\}$ represent the current configuration of the network. Similarly, let $V = \{v_j\}$ be the new network configuration where $v_j = \{P, OS, F\}_j$ represent an admissible configuration state of a device after a possible upgrade of its OS software image. Because not all network devices may be recommended to be upgraded to a new software version (case in which no new configuration state does exist which reduces its risk of not been upgraded), it may be that $U \subseteq V$, which means that there are some groups of devices...
$D_i$ which retain the exact same configuration $v_j = u_i$. Furthermore, because any network device can be upgraded to more than one possible OS software image, the cardinality of the admissible network configuration states $J \geq I$. Next, let $e_{ij} \in E$ represents an admissible software upgrade for network devices in cluster $D_i$ from their current configuration state $u_i$ to a new configuration state $v_j$. In this context, a configuration upgrade is defined from $u_i$ to $v_j$ to be admissible if and only if the operating system can be upgraded from $OS_i$ to $OS_j$ and the set of supported features on the new $OS_j$ is a superset of features enabled in the current $OS_i$, i.e., $F_i \subseteq F_j$. Notice that $e_{ij}$ can be expressed as a binary variable that equals to 1 if and only if an edge is established between $u_i$ and $v_j$ and 0 otherwise. Let $W = [w_{ij} \geq 0]$ be the reward matrix where each entry $w_{ij}$ represent the reward factor when upgrading devices in group $D_i$ from the current configuration state $u_i$ to the new configuration $v_j$. The reward factor can be pre-computed as $w_{ij} = r_i - r_j$ where $r_i$ represents the risk profile of devices with configuration $u_i$ and $r_j$ represents the risk profile of devices with configuration $v_j$. As noted above, the values are pre-computed and provided by OPERA Possible Future States Module.

Now, it is possible to formalize the problem as an Integer Linear Programming (ILP) problem which can be solved with standard Optimization tools like CPLEX [1], as shown below in Equation 3.

**Equation 3:**

$$\max \sum_{i \in I, j \in J} e_{ij} w_{ij} |D_i|$$

That is, Equation 3 illustrates the objective Function: Maximization of Total Reward (or Minimization of Total Composite Risk) subject to Equation 4, below.

**Equation 4:**

$$\sum_{j \in J} e_{ij} = 1 \ \forall i \in I$$

That is, Equation 4 (Constraint 1) illustrates a constraint ensuring that each cluster $D_i$ is mapped to ONE and only ONE new configuration state $v_j$. Equation 5, below, illustrates a second constraint.

**Equation 5:**

$$e_{ij}(r_i - r_j) \geq 0 \ \forall (i,j) \in E$$
That is, Equation 5 (Constraint 2) is a constrain ensuring that only upgrades producing positive rewards will be considered during the assignment. Notice that a negative value of reward $w_{ij} = r_i - r_j$ will force the $e_{ij} = 0$; conversely, a positive value of $w_{ij} = r_i - r_j$ allows the variables $e_{ij}$ to assume either 0 or 1 values.

### 3.2.1 Generalization to include the cost of a software upgrade

In the formulation provided above, not considered was the potential cost that a customer may incur in adopting a new configuration state $v_j$ if its deployment would require a memory upgrade on some of the its network devices or even worse if the new OS software recommendation may have a limited software supportability. To address the above challenges, this section extends the formulation as following. Let $C = [c_{ij} \geq 0]$ be the cost matrix where $c_{ij}$ represents the cost incurred by the customer when upgrading devices in group $D_i$ from the current configuration state $u_i$ to the new configuration $v_j$. As noted above, the cost matrix $C$ is pre-computed and provided as an input by the OPERA Future States Module. It is possible to add the constraint shown below in Equation 6 to ensure that only edges $e_{ij}$ with $c_{ij} = 0$ can be selected during the assignment.

**Equation 6:**

$$e_{ij}(1 + c_{ij}) \leq 1 \quad \forall \ (i,j) \in E$$

That is, Equation 6 (Constraint 3) is a constraint added if there is a need to guarantee that only edges $e_{ij}$ with $c_{ij} = 0$ will be selected during the assignment. Notice that a cost $c_{ij} > 0$ will force the $e_{ij} = 0$; conversely, a $c_{ij} = 0$ allows the variables $e_{ij}$ to assume either 0 or 1 values.

It is also possible to relax this hard constraint by allowing the assignment to select edges with a non-null cost by removing Equation 6 from the list of constraints and add a new term which accounts for the cost to select an assignment in the objective function (Equation 3) weighted by a number $B_1 \geq 0$ (see Equation 7). If $B_1$ is chosen to be 0, then the formulation collapses to the Reward-Only formulation as presented above; if $B_1$ is chosen to be a much larger number than the average total reward, then the assignment will try to minimize the total cost first before taking into consideration the total reward; conversely, if $B_1$ is chosen to be much smaller than the average total reward, then the
assignment will give priority to an assignment which will maximize the total reward before considering the associate cost of the assignment. In Equation 7, below, shown is the modified objective function which includes both the reward of an upgrade and the expected cost incurred by the customer to upgrade to the new network configuration state.

**Equation 7:**

\[
\max \sum_{i \in I, j \in J} e_{ij} w_{ij} |D_i| - B_1 \sum_{i \in I, j \in J} e_{ij} c_{ij} |D_i| = \max \sum_{i \in I, j \in J} e_{ij} |D_i| [w_{ij} - B_1 c_{ij}]
\]

That is, Equation 7 (Objective Function) represents the revisited Objective Function which maximizes the balance between Total Reward and Total Cost of a software upgrade network-wise.

### 3.2.2 Generalization to include the cost of software versions heterogeneity in the new network configuration state

This section expands the formulation to take into consideration the complexity and associated perceived risk that a customer may have in adopting a network-wise software upgrade which requires many new software versions to be deployed, operated and maintained. To ensure that the model will minimize the number of distinct new configurations states upon the achievement of maximum reward of a network-wise software upgrade with zero or minimal incurred cost, the following variables and constraints are introduced into the model. Let \( \alpha_j \) be a binary variable which is equal to 1 if and only if the new configuration state \( v_j \) is used by one or more groups of devices \( D_i \) and \( v_j \) is a new configuration state which is not included in the set of current configuration states, i.e., \( v_j \cap U = \{0\} \). Now, it is possible to introduce the following two constraints, shown in Equations 8A and 8B.

**Equation 8A**

\[
\alpha_j \leq \sum_{i \in I} e_{ij} \forall j \in J : v_j \cap U = \{0\}
\]

That is, Equation 8A (Constraint 4) is a constraint that guarantees that \( \alpha_j = 0 \) if no group of devices \( D_i \) selected the new configuration \( v_j \); conversely, \( \alpha_j \) can assume either 0 or 1 values.

**Equation 8B**

\[
B_2 \alpha_j \geq \sum_{i \in I} e_{ij} \forall j \in J : v_j \cap U = \{0\}
\]
That is, Equation 8B (Constraint 5) is a constraint where $B_2 \gg 0$ is a big positive number. This constraint forces the variables $\alpha_j$ to assume value equal to 1 if there is at least one group of devices which has selected the new configuration state $v_j$.

It is now possible to modify the objective function by including a penalty factor in selecting new configuration states $v_j$ which do not contribute to the increase the total reward for the customer. Next, the new penalty term is reported in the context of the objective function provided in Equation 3 (Maximization of Total Reward). It is noted that the same term can be added to the more general objective function provided in Equation 7 (Maximization of Total Reward/ Cost balance). Let $B_3 \geq 0$ be a positive number provided as an input to the formulation. If $B_3$ is chosen to be 0, then the formulation collapses to the Reward-Only formulation as presented in Equation 1; if $B_3$ is chosen to be a much larger number than the average total reward, then the assignment will try to minimize the total number of distinct software configurations first before taking into consideration the total reward; conversely, if $B_3$ is chosen to be much smaller than the average total reward, then the model will give priority to an assignment which will maximize the total reward before trying to minimize the total number of distinct software recommendations required.

**Equation 9**

$$\max \sum_{i \in I, j \in J} e_{ij} w_{ij} |D_i| - B_3 \sum_{j \in J} \alpha_j$$

That is, **Equation 9** (Objective Function) represents Maximization of Total Reward (or Minimization of Total Composite Risk) with penalty factor accounting the number of distinct software configurations required.

### 3.3 Modeling the Topology Compatibility of a software upgrade

In the formulation provided in the previous sections, all network devices were grouped together ($D_i$) if sharing the exact same current configurations $\alpha_i$ and seek for the optimal set of new configuration states at the group level, i.e., $\{v_j\}$ with $e_{ij} = 1$. By doing so, the result may be a set of recommendations in which some devices may have incompatible new configuration states when considering the how they are laid out in the network, i.e., from a network topology perspective.
To expand the formulation to cover the topology compatibility as well, it is possible to let \( d_i \) to represent the atomic network device, with current configuration \( u_i \) which can be migrated to a new configuration \( v_j \). Let \( N_i \) be the set of network devices which are neighbor of device \( d_i \), i.e., \( N_i = \{ d_{im} \} \). Let \( \Gamma = \{ y_{hk} \} \) be an indicator function which is equal to 1 if the new configurations states \( v_h \) and \( v_k \) are compatible to each other, and 0 otherwise. Then it is possible to add a new constraint which forces all neighbor devices to only select new configuration states which are compatible to each other, i.e., as shown in Equation 10.

**Equation 10**

\[
e_{mh} + e_{nk} \leq 1 + y_{hk} \forall (m, n) \in N_i, \forall (h, k) \in \Gamma
\]

That is, Equation 10 (Constraint 6) is a constraint that guarantees that only compatible configurations \( v_h \) and \( v_k \) can be selected for any pair of neighboring devices during the assignment. Notice that a value \( y_{hk} = 0 \) (meaning the two new configuration states \( v_h \) and \( v_k \) are incompatible with each other) will force either \( e_{mh} \) or \( e_{nk} \) to be 0, i.e., neighboring devices cannot select this specific pair of new configurations; conversely, a value of \( y_{hk} = 1 \) allows both \( e_{mh} \) and \( e_{nk} \) to assume either 0 or 1 values, i.e., neighboring devices can select this pair of new configurations states.

4. **OPERA: Reinforcement Learning Module**

The OPERA Constrained Optimization Module as presented in Section 3, provides the customer with a best recommendation for each device or group of identical devices \( D_i \) (i.e., same configuration state \( u_i \)) given some input provided to the model by the domain experts like the reward matrix \( W=[w_{ij}] \) of an admissible software upgrade, i.e., weights on edges \( e_{ij} \) of the bipartite graph, and the cost matrix \( C=[c_{ij}] \) of such upgrade. Because the selection of these input configurations may not be precise in nature but mostly driven by user knowledge, domain experts will likely provide the customer \( X \) with a set of different software recommendations by running the OPERA Constrained Optimization Module under different input configurations; then have the customer to select the software upgrade which fits his policies and risk profile. This process allows the customer to tailor the software recommendation based on his own risk profile (or tolerance to specific risk levels) and hence customize the recommendation. For example, the OPERA system may
recommend a software migration which resolves all the known security vulnerabilities and the closes all open software bugs for a given group of devices, but the customer many not accept the recommendation if the release has not been out in production long enough (it will consider unstable and too risky).

The OPERA Reinforcement Learning module comes into the picture to learn the most popular software upgrade selections for any given group of devices provided by customers and reuse this information to automatically re-rank (i.e., refinement of reward and cost parameters of a software upgrade for a given device group) its recommendations based on customer adoption, with most popular / adopted software upgrade on the top and the least popular on the bottom.

More formally, let $S_i = \{s_{i1}, ..., s_{ik}\}$ be the set of the $K$ software upgrades per each group of devices $D_i$ with current configuration state $u_i$ provided by the OPERA package with $s_{i1}$ been the most trusted software upgrade by the domain expert and $s_{ik}$ been the least trusted software upgrade. As customers will be exposed to this software recommendation and make their own selection, the ANSR Reinforcement Learning Module will compute in real-time the popularity of each software upgrade and refresh the ranking of proposed upgrades to turn $s_{i1}$ to be the most popular among customers software upgrade and $s_{ik}$ to be the least popular.

5. Network and Group Level Software Recommendation

Provided in this section are two snapshots of the OPERA User Interface portal used to present the final software recommendation to customers. In Figure 2, below, shown is an example of a device recommendation deployed in a specific customer, omitted for confidentiality reason. OPERA identifies a network device deployed in a customer network that after a configuration change has drastically increased its risk level, i.e., risk value above a threshold set by the customer. The OPERA system reports the contributing factors to the risk value (end of sale, 7 security vulnerabilities, one software bug), and automatically provides the suggested software recommendation for the given device which is an upgrade from its current OS version (15.0(2)SE5) to the new software version (15.2(2)E5a which would zero out all the issues identified.
In Figure 3, below, shown is a different view provided by OPERA for all devices deployed in the same customer network. This includes different device vendors, and for each vendor a further breakdown by devices into product families, etc. OPERA does provide an overall risk score of the current network software configuration, i.e., 5009, and all security and software bug issues identified per each group. Furthermore, the customer can click on a specific group, e.g., a certain type of switches that includes 55 deployed devices contributing with a risk value of 2063 out of the overall network risk value of 5009, and check the software recommendation per each device in the group.