

2018 INFORMS International Conference
Taipei, Taiwan, June 17–20, 2018
© 2018 INFORMS| ISBN 978-0-9906153-1-6
https://doi.org/10.1287/infp.2018.XXXX
pp. 000–000

New Exceedance Counter with Application to Network Optimization

Jeremy D. Jordan
Department of Mathematics & Statistics, Air Force Institute of Technology, jjordan810@gmail.com

Stan Uryasev
Industrial and Systems Engineering Department, University of Florida, uryasev@ufl.edu

Abstract We consider a new characteristic for counting the number of large outcomes in a data set that are considered to be large w.r.t. some fixed threshold x. A popular characteristic used for this purpose is the Cardinality of Upper Tail (CUT), which counts the number of outcomes with magnitude larger than the threshold. We consider a similar characteristic called the Cardinality of Upper Average (CUA), defined as the number of largest data points which have average value equal to the threshold. This characteristic not only counts the number of outcomes that are larger than the threshold, but also the largest outcomes that are less than the threshold such that the average of these outcomes is equal to x. For example, consider data points representing individual queue lengths for a set of servers such that a server is considered to be overloaded if its queue length exceeds x. CUA provides us with, not only an assessment of the number of overloaded servers, but an assessment of how severely overloaded these servers are. CUA also has superior mathematical properties, being a continuous function w.r.t. the threshold parameter, piecewise linear in its reciprocal, and directly optimizable via convex and linear programming. This means that it is possible to efficiently minimize the number of outcomes in the tail, with the tail including the largest outcomes that average to the threshold. This is in contrast to CUT, which does not assess the severity of tail outcomes, is discontinuous as a function of threshold, and is such that direct optimization yields numerically difficult non-convex problems. We show that CUA can be used to formulate meaningful optimization problems containing counters of the largest components of a vector without introduction of binary variables, leading to large improvement in computation speeds. In particular, we apply the CUA concept to create new formulations of network optimization problems involving node failures, where we aim to minimize the number of most burdened nodes which have loads averaging to x.

Keywords optimization, networks; quantile; Value-at-Risk; buffered Probability of Exceedance

1. Introduction

When analyzing a set of n data points, one often needs to count the number of outcomes that are large compared to some threshold $x \in \mathbb{R}$. For example, if the data points represent monetary losses, it is only natural that one would want to count the number of outcomes that are large compared to a threshold representing an 'acceptable' level of loss. A popular characteristic which acts as a counter of such large outcomes is the x-Cardinality of Upper Tail (CUT_x), which counts the number of data points with magnitude exceeding the threshold x.

We consider a similar characteristic called the x-Cardinality of Upper Average (CUA_x) that not only counts the number of data points with magnitude larger than the threshold, but also the largest outcomes with magnitude less than the threshold such that the average of these outcomes is equal to the threshold.

Suppose that we manage a network with 10,000 servers. We consider that "there is a concern" if a server has a queue containing at least 10 jobs (clients). In other words, we say that a server is overloaded if the length of the queue is greater than or equal to 10 jobs. We can ask the question: How many servers are overloaded? The answer is $\text{CUT}_{10} = 3$. However, it is completely unclear how significantly these servers are overloaded and how much additional resource would be required to serve the customers in these queues. An alternative characteristic is $\text{CUA}_{10} = 207.3$. This means that the average length of a queue over the 207.3 busiest servers is 10 jobs. The total number of unserved jobs in these servers is then $207.3 \times 10 = 2,073$. Therefore, we understand immediately how much additional resource is needed to process jobs in the longest queues. This important information is present in CUA_x, and is not present in CUT_x. CUA_x is also a continuous (nice mathematical) characteristic as a function of the threshold, while CUT_x is a discontinuous characteristic.

You may ask, what is a better characteristic, CUA_x or CUT_x? The simple answer is that these are different, complimentary characteristics. Therefore, while in some cases you may be interested in CUT_x and in other cases CUA_x, it is probably a good idea to calculate both characteristics.

In the most basic sense, CUA_x is similar to CUT_x, with both acting as counters of large outcomes relative to a threshold. Beyond this, they have important differences. First, as illustrated in the example, CUA_x considers how far the outcome magnitudes are from the threshold. The CUT_x characteristic does not and only considers if an outcome is larger than x, not considering how far beyond the threshold these magnitudes lie. Second, CUA_x has superior mathematical properties. It is continuous w.r.t. the threshold parameter and a partial derivative can be taken w.r.t. threshold yielding information regarding the sensitivity of CUA_x to threshold changes. It is also piecewise linear in its reciprocal. As we will show, this allows for efficient calculation of CUA_x for all thresholds $x \in \mathbb{R}$.

In the context of optimization, the properties of CUA_x yield substantial benefits, with direct optimization of CUA_x reducing to convex, sometimes linear, programming. This means that we are able to efficiently minimize the number of tail outcomes, while taking into account the severity of these outcomes. Furthermore, the tail that is minimized is determined by the threshold, with the tail including the largest outcomes that average to the specified threshold. This is all in contrast to CUT_x, which does not account for the severity of the tail outcomes, is discontinuous w.r.t. the threshold parameter, and is such that direct optimization yields numerically challenging non-convex problems.

Before formally defining CUA_x, we will briefly introduce a related characteristic called the k-Upper Average (UA_k). This characteristic is, intuitively speaking, a generalization of an average. It calculates the average of the largest k outcomes, where k can take on non-integer values. Like CUA_x, UA_k is continuous w.r.t. the k parameter and shares similar mathematical properties. We use UA_k to motivate the definition of CUA_x, which is its inverse.

While being different from CUT_x in many ways, CUA_x is uniquely related. We show that CUA_x is, in a certain sense, the minimal quasi-convex upper bound of CUT_x. In an optimization context, this could be useful, suggesting that CUA_x minimization may yield efficient heuristic approaches to reducing CUT_x. This, though, is not our focus and we emphasize that CUA_x and its optimization are meaningful in their own right.

We apply CUA_x to a specific optimization setting, providing convex and Linear Programming (LP) CUA_x minimization formulations for solving network optimization problems. We specifically consider the network flow problem where product must flow from supply nodes to demand nodes through some intermediate transshipment nodes and we assume

that these intermediate nodes can become "overloaded" if too much flow is pushed through them. Therefore, it is desirable to minimize the number of overloaded nodes. We formulate this as a CUA_x minimization problem, which accounts for the magnitude of loads put upon nodes in the overloaded state, as well as the nodes with loads less than, but most near to the overloaded state. This information may be meaningful, as it may be undesirable to have nodes that, while not yet overloaded, are close to the overloaded state. Furthermore, it may be undesirable to have nodes that are dramatically overloaded. Thus, we minimize the number of most overloaded nodes that have average loads equal to x, where nodes with loads larger than x are considered to be overloaded.

We compare this CUA_x formulation with a similar formulation that minimizes CUT_x, directly minimizing the number of nodes in the overloaded state. With this problem involving binary variables to indicate the state of the node, Mixed Integer Programming (MIP) must be used. Not only is this much more difficult to solve than the LP CUA_x minimization, but we show that it may provide less appealing policies. By minimizing CUT_x, information about the magnitude of the loads is ignored and it only considers whether a node is overloaded or not. This information, as already mentioned, can be meaningful. Thus, the CUA_x formulation may be more appropriate and suggest more appealing policies.

Additionally, in the context of vector norms, one minus a special case of CUA_x, called Cardinality of Absolute Upper Average (CAUA), is exactly the inverse of the Conditional Value-at-Risk (CVaR) norm [5]. This special case helps to elucidate the inverse relationship between CUA_x and a deterministic variant of CVaR. The CUA_x concept is actually a deterministic variant of so called Buffered Probability of Exceedance (bPOE). A detailed discussion of this concept is studied in [3]. We maintain a deterministic setting, while bPOE is studied in a probabilistic, stochastic optimization setting.

2. Cardinality of Upper Tail and Related Quantities

2.1. Cardinality of Upper Tail and Upper Average

Consider a Euclidean vector $\mathbf{y} = (y_1, \ldots, y_n)$ containing n data points. It is often important in applications to know the number of components of $\mathbf{y}$ that exceed a particular threshold $x \in \mathbb{R}$. We call this the Cardinality of the Upper Tail (CUT_x), denoted as

$$\text{CUT}_x(\mathbf{y}) = \eta_x(\mathbf{y}) = |\{y_i | y_i \geq x, i = 1, \ldots, n\}| \,.$$

This quantity, though, can be difficult to work with. For example, it is discontinuous w.r.t. the x parameter. Additionally, this quantity does not provide information about the magnitude of the components above the threshold x. As we will show in later sections, these are undesirable properties in an optimization setting. We can also consider other quantities that provide useful information about the tail of the data distribution that are easy to work with. For example, one could consider the ordered weighted averaging aggregation operators. We can also consider a type of *tail average* called the k-Upper Average (UA_k), which provides information about the magnitude of data points in the tail and upper bounds on CUT_x. If we let $(y^{(1)}, y^{(2)}, \ldots, y^{(n)})$ represent a permutation of $(y_1, \ldots, y_n)$ with components listed in nondecreasing order, $y^{(1)} \leq y^{(2)} \leq \ldots \leq y^{(n)}$, we can denote this quantity as

$$\text{UA}_k(\mathbf{y}) = \frac{1}{k} \sum_{i=n-k+1}^{n} y^{(i)} \,. \tag{1}$$

2.2. Generalized UA_k

Notice that (1) only applies to integer values of the k parameter. This function, $\text{UA}_k(\cdot)$, can also be defined in a more general manner for non-integer values of k. Popularized in the financial engineering literature under the name CVaR, paper [6] defines this quantity in a

broad, probabilistic setting. Here, we maintain a deterministic setting, but emphasize that this is simply a special case of the general CVaR definition and formula. Thus, following directly from [6], we have that for any $k \in [1, n]$, letting $[\cdot]^+ = \max\{0, \cdot\}$,

$$\mathrm{UA}_k(\mathbf{y}) = \min_{\gamma} \left\{ \gamma + \frac{1}{k} \sum_{i=1}^{n} [y_i - \gamma]^+ \right\} . \tag{2}$$

Though this formula may not seem intuitive, it can be clarified by noticing two facts. First, for any integer $k \in \{1, \ldots, n\}$, (2) is equal to (1). Second, for noninteger values of $k \in [1, n]$, this formula simply gives a weighted average of $\mathrm{UA}_{\lfloor k \rfloor}(\mathbf{y})$ and $\mathrm{UA}_{\lceil k \rceil}(\mathbf{y})$, where $\lfloor k \rfloor$ denotes the largest integer less than or equal to k and $\lceil k \rceil$ denotes the smallest integer greater than or equal to k. Thus, intuitively, formulation (2) is still averaging the largest k data points, but it is now a continuous function w.r.t. the k parameter.

In addition to providing useful information about the magnitude of data points in the tail, $\mathrm{UA}_k(\cdot)$ provides an upper bound for CUT_x. Specifically, we have the following relation, which follows intuitively from (2) and the definition of CUT_x.

$$\mathrm{UA}_k(\mathbf{y}) = x \implies \eta_x(\mathbf{y}) \leq k \, . \tag{3}$$

3. Cardinality of the Upper Average

3.1. Definition of CUA_x

For a specified threshold x, CUA_x calculates the value of $k \in [1, n]$ such that $\mathrm{UA}_k(\mathbf{y}) = x$. In other words, CUA_x is equal to the number of largest components of the vector $\mathbf{y}$ such that the average of those components is equal to x. We now give the strict mathematical definition of CUA_x, followed by examples illustrating key differences between CUT_x and CUA_x.

Definition 1: Consider a Euclidean vector $\mathbf{y} = (y_1, \ldots, y_n)$. CUA_x of $\mathbf{y}$ at threshold $x \in \mathbb{R}$ equals

$$\mathrm{CUA}_x(\mathbf{y}) = \bar{\eta}_x(\mathbf{y}) = \min_{a \geq 0} \sum_{i=1}^{n} [a(y_i - x) + 1]^+ . \tag{4}$$

Initially, it may be surprising that this particular formula calculates the number of biggest components of the vector $\mathbf{y}$ such that the average of those components is equal to x. In short, this formula is derived from equation (2), partially explaining its form as the unique minimal value to the minimization problem.

To begin discussing CUA_x, first notice that CUA_x is continuous w.r.t. the parameter x on the interval $x \in (-\infty, \mathrm{UA}_1(\mathbf{y}))$; this will be shown later on in Corollary 1. As already mentioned in Section 2, CUT_x is discontinuous w.r.t. this threshold parameter. Second, notice that by calculating the $k \in [1, n]$ such that $\mathrm{UA}_k(\mathbf{y}) = x$, CUA_x is counting all components with magnitude greater than x and some components with magnitude less than x. These magnitudes can contain important and meaningful information that is ignored by CUT_x. For example, consider a disaster relief agency that is analyzing historical hurricane damage data where damages are in dollars [2]. Assume that the agency is attempting to determine if allocating $\$B$ dollars to the relief fund for the next hurricane is sufficient. First, of the historical damage amounts that exceed $\$B$, it is important to know by how much these damages exceeded $\$B$, especially if the exceedance is large. Secondly, it is important to know the magnitude of the largest damage amounts that are less than $\$B$. Are these damages very close to $\$B$? Or are they much smaller than $\$B$? Answers to these questions are clearly important for budgetary considerations.

With CUA_x being the inverse of UA_k, both quantities relay similar information. We mention, though, that CUA_x may be more intuitive to use when data have meaningful units,

particularly because the threshold parameter x is posed in the associated units. For example, if the data represent monetary losses (in the unit of dollars) from hurricane damage, it can be more intuitive to work with a threshold x, which will be some dollar amount, rather than with the k-parameter. Consider, as before, the disaster relief agency that is attempting to determine if allocating $\$B$ dollars to the relief fund for the next hurricane is sufficient. It would be much more intuitive to look at CUA_B than to look at UA_k for many different values of k. Additionally, this also applies when considering some optimization tasks. Assume an investor is trying to form a portfolio by analyzing historical stock behavior (where each possible portfolio generates some data set of historical losses it would have incurred in the market). If an investor does not want to incur losses larger than $\$B$ dollars, it is more intuitive to try and devise a portfolio that minimizes CUA_B rather than to find a portfolio that minimizes UA_k for some appropriate value of k which would need to be determined.

We also highlight the simple, but useful fact that CUA_x tells you immediately the total weight that is in the tail. Consider again the example from the introduction where we have a network of 10,000 servers and we find that $\text{CUA}_{10} = 207.3$. This immediately tells us that 10×207.3 additional units of resource are needed to process jobs in the 207.3 longest queues. On the other hand, $\text{CUT}_{10} = 3$ is much less informative, telling us only that the 3 longest queues need *at least* 30 additional units of resource, even though this number could be much larger. We can put this more precisely, though. Assume that a^* an optimal point for CUA_x calculation (4) at threshold x. We prove in Section 3.1.1 that,

$$x\bar{\eta}_x(\mathbf{y}) \leq \sum_i \{y_i | y_i \geq x - \frac{1}{a^*}\} \leq x(\lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1)\,,$$

where $\lfloor k \rfloor$ denotes the largest integer less than or equal to k. Therefore, we know that the total amount of resource needed to process all unserved jobs waiting in queues with length longer than or equal to $x - \frac{1}{a^*}$ is somewhere between $x\bar{\eta}_x(\mathbf{y})$ and $x(\lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1)$. Additionally, note that the continuity of CUA_x can help determine where these resources are needed. For example, consider the case where all queues have load less than 10 except for three servers, which are extremely overloaded. In this case, one can take a partial derivative of CUA_x w.r.t. the threshold to analyze how dramatically CUA_x is changing as the threshold changes continuously.

3.1.1. Simultaneous Calculation of CUA_x and CUT_x An important and interesting property of CUA_x calculation formula (4) is that calculation of CUA_x provides us information about CUT_x. Specifically, we have the following property of formula (4).

Proposition 1: *Suppose that for a vector $\mathbf{y} \in \mathbb{R}^n$ and threshold $x \in (\text{UA}_n(\mathbf{y}), \text{UA}_1(\mathbf{y}))$ we have that*

$$\bar{\eta}_x(\mathbf{y}) = \sum_{i=1}^{n} [a^*(y_i - x) + 1]^+ = k\ ,$$

where a^* is an optimal point for (4). Then, if k is noninteger, we have that $a^* = \frac{1}{x - y^{(n-\lceil k \rceil)}}$ and is the unique minimizer of (4). Furthermore, we have that

$$\eta_{x - \frac{1}{a^*}}(\mathbf{y}) = \lceil k \rceil\ .$$

Otherwise, if k is integer, the set of minimizers of (4) is given by the interval $\mathcal{I} = [\frac{1}{x - y^{(n-(k+1))}}, \frac{1}{x - y^{(n-k)}}]$ where $a^* \in \mathcal{I}$. Furthermore, we have that

$$\eta_{x - \frac{1}{a^*}}(\mathbf{y}) = \begin{cases} k+1\,, & \text{if } a^* = \frac{1}{x - y^{(n-(k+1))}}\,, \\ k\,, & \text{otherwise.} \end{cases} \tag{5}$$

Proposition 1 is quite useful, as it shows that simply calculating CUA_x provides information about CUT_x.

In the context of our earlier example involving 10,000 servers, we said that $x \times \text{CUA}_x$ told us how many additional resources were needed to cover the CUA_x busiest queues. Using Proposition 3, we can expand upon this statement with the following Corollary.

4. Optimization of CUA_x

When entered into an optimization setting, CUA_x provides substantial benefits, particularly when compared to the analogous use of CUT_x. Consider the following CUT_x minimization problem, where $S \subseteq \mathbb{R}^n$ is a convex set and $x \in \mathbb{R}$ is a specified threshold:

$$\min_{\mathbf{y} \in S} \eta_x(\mathbf{y})$$

In order to solve this problem, it is natural to reformulate it as an MIP. In other words, to indicate the state of a vector component as being larger than x or not, introduction of binary variables is a natural consideration.

We consider minimization of CUA_x, which is posed as follows.

$$\min_{\mathbf{y} \in S} \bar{\eta}_x(\mathbf{y}) = \min_{\mathbf{y} \in S} \{\min_{a \geq 0} \sum_{i=1}^{n} [a(y_i - x) + 1]^+\} = \min_{\mathbf{y} \in S, a \geq 0} \sum_{i=1}^{n} [a(y_i - x) + 1]^+ \ .$$

As we will show by example in Section 4.1.1, if the set S is convex, the CUA_x minimization problem can be reduced to convex programming. Furthermore, if the set S is a polyhedron, the CUA_x minimization problem can be reduced to linear programming. Additionally, CUA_x minimization itself is meaningful, as it considers the magnitude of the components around the threshold x. Therefore, while also minimizing an upper bound for CUT_x, the CUA_x minimization is interesting in its own right. To demonstrate this, we provide an example network flow problem where CUA_x optimization may be more appealing than the analogous CUT_x minimization.

4.1. Applications to Network Optimization

We now show that network flow problems can be formulated as CUA_x minimization problems and can further be transformed into Linear Programming problems.

4.1.1. Minimum Network Overflow Problem Consider a network represented by a graph, where loads of product flow from supply vertices, through the graph edges and transshipment vertices, to the destination demand vertices. Furthermore, assume that if the flow through a transshipment vertex exceeds a threshold x, then it is considered to be overloaded, which is an undesirable state. For example, if flow exceeds a threshold, equipment malfunctions may begin to occur with high probability or additional unknown costs may be incurred. A common optimization problem arising from this scenario is that of minimizing the number of overloaded transshipment vertices. Specifically, consider the following notation:

- $G = (V, E)$ = directed graph with vertices V and edges E.
- $\mathbf{b} = (b_1, \ldots, b_{|V|}) \in \mathbb{R}^{|V|}$ = supply/demand vector, where b_i is a supply/demand at vertex i; $\sum_{i=1}^{|V|} b_i = 0$.
- $c_{ij} \geq 0 \ \forall (i,j) \in E$ = shipment costs associated with graph edges. One can set $c_{ij} = \infty$ for any $(i,j) \notin E$.
- $f_{ij} \geq 0$ = flow through the edge (i,j).

- $l_i = \sum_{(i,j)\in E} c_{ij} f_{ij} + \sum_{(k,i)\in E} c_{ki} f_{ki}$ = weighted load of the vertex $i \in V$.
- $\mathbf{l}$ = vector of loads l_i for $i \in V'$, where $V' \subseteq V$ is a subset of vertices.
- $x \geq 0$ = "acceptable" load level, it is undesirable to exceed that level.
- $\xi_i \in \{0,1\}\ \forall i \in V'$ = indicator of the load l_i exceeding the threshold x.
- M = sufficiently large constant.

If one would like to find the flow configuration that minimizes the number of overloaded transshipment vertices, it is natural to consider the following MIP problem which minimizes $\eta_x(\mathbf{l})$.

$$\min_{\xi,\mathbf{l},f} \quad \sum_{i\in V'} \xi_i \tag{6}$$

$$s.t. \quad \xi_i \geq (l_i - x)/M,\ l_i = \sum_{(i,j)\in E} c_{ij} f_{ij} + \sum_{(k,i)\in E} c_{ki} f_{ki}\ \forall i \in V', \tag{7}$$

$$\sum_{(i,j)\in E} f_{ij} - \sum_{(k,i)\in E} f_{ki} = b_i\ \forall i \in V, \tag{8}$$

$$\xi_i \in \{0,1\}\ \forall i \in V',\ f_{ij} \geq 0\ \forall (i,j) \in E. \tag{9}$$

Though this MIP will solve the problem of minimizing the number of overloaded nodes, it is ignoring potentially meaningful information about the magnitude of loads put upon nodes. Specifically, it does not account for the magnitude of loads put upon nodes in the overloaded state, as well as the nodes with loads less than, but most near to the overloaded state. For nodes that are already overloaded, it may be important to consider by how much they are overloaded. Maybe, for a certain system, it is far worse for a node to be overloaded by 1000 units than by 100 units. Additionally, it may be undesirable to have nodes that, while not yet overloaded, are very close to being in the overloaded state.

As a concrete example, consider a system where the flow through a transshipment node relies on equipment that can malfunction. If a malfunction occurs, there is a large cost C that must be payed to fix the node so that flow can finish moving through it. Assume that for a configuration of flow, a malfunction can only occur once, and that when the malfunction is fixed, flow moves through the network as specified. Let $I\{l_i > x\}$ be an indicator function, indicating if a node is overloaded. Now suppose that, for a given flow configuration and associated loads l_i, the probability of a malfunction occurring at node i is given by,

$$P(\text{malfunction at node } i) = .5 I\{l_i > x\} + .5(1 - \frac{1}{e^{[l_i - x]^+}}) .$$

Therefore, we have that a transshipment node will not malfunction until it is overloaded and once overloaded, the probability jumps to .5 and increases exponentially based on the magnitude of the load vector. Solving the CUT_x minimization problem to calculate flow configuration for this system is ignoring critical information, and may produce a very undesirable result. First, notice that the CUT_x minimization problem will ignore the magnitude with which a node is overloaded, thus ignoring the fact that *very* overloaded nodes will, with higher probability, incur the additional cost C. Second, notice that the CUT_x minimization will ignore information about nodes with loads less than, but very near to x. This can cause major stability issues if there is uncertainty regarding exact system specifications at test-time. For example, consider a CUT_x minimization solution which has many nodes with loads less than, but very near to x. If, at test-time, a small amount of extra, unplanned, flow is forced through each node, we will have a cascade of increasing malfunction probabilities as the nodes with flow close to x are pushed over the threshold with their failure probability increasing from zero to around .5.

By minimizing CUA_x, we formulate a network flow problem that accounts for these factors (i.e. the magnitudes of loads). Specifically, we have the following optimization problem which aims to minimize the number of most loaded nodes that have loads averaging to x.

$$\min_{a,\mathbf{l},f} \quad \sum_{i=1}^{n}[a(l_i - x) + 1]^+ \tag{10}$$

$$s.t. \quad l_i = \sum_{(i,j)\in E} c_{ij} f_{ij} + \sum_{(k,i)\in E} c_{ki} f_{ki} \ \forall i \in V', \tag{11}$$

$$\sum_{(i,j)\in E} f_{ij} - \sum_{(k,i)\in E} f_{ki} = b_i \ \forall i \in V, \tag{12}$$

$$a \geq 0 \tag{13}$$

$$f_{ij} \geq 0 \ \forall (i,j) \in E. \tag{14}$$

The CUA_x optimization problem, though, can be further reduced to LP form with a simple change of variable. Specifically, optimization problem (10)–(14) with the change of variables $y_{ij} = a f_{ij}$ can be reduced to the following LP.

$$\min_{a,z,y} \quad \sum_{i\in V'} z_i \tag{15}$$

$$s.t. \quad z_i \geq \sum_{(i,j)\in E} c_{ij} y_{ij} + \sum_{(k,i)\in E} c_{ki} y_{ki} - ax + 1 \ \forall i \in V', \tag{16}$$

$$\sum_{(i,j)\in E} y_{ij} - \sum_{(k,i)\in E} y_{ki} = a b_i \ \forall i \in V, \tag{17}$$

$$a \geq 0, \ y_{ij} \geq 0 \ \forall (i,j) \in E, \ z_i \geq 0 \ \forall i \in V'. \tag{18}$$

This is a powerful result, which implies that we can achieve a similar goal (i.e. minimizing the number of transshipment vertices with loads exceeding, or slightly below a specified threshold) using linear programming as opposed to mixed-integer programming. In terms of the network optimization problem, it considers how much load is burdening the overloaded nodes (i.e. by how much they are overloaded). Additionally, it considers how much load is burdening the nodes with loads less than, but most near to the threshold (i.e. how close to being overloaded are the most burdened nodes). Thus, in addition to being more efficiently solvable, it can also be a more appealing formulation than CUT_x minimization, such as in the malfunction probability example.

References

[1] Bertsimas, D., Pachamanova, D., Sim, M. Robust linear optimization under general norms Operations Research Letters, 32(6), 510-516. (2004)

[2] Davis, J.R., Uryasev S. Analysis of Hurricane Damage using Buffered Probability of Exceedance. Research Report 2014-4, ISE Dept., University of Florida. (2014)

[3] Mafusalov A., Uryasev S. Buffered Probability of Exceedance: Mathematical Properties and Optimization Algorithms. Research Report 2014-1, ISE Dept., University of Florida. (2014)

[4] Mafusalov, A., Uryasev, S. Conditional value-at-risk (CVaR) norm: Stochastic Case. Research Report 2013-5, Department of Industrial Systems and Engineering, University of Florida, Gainesville, FL (2013)

[5] Pavlikov, K., Uryasev S. CVaR norm and applications in optimization. Optimization Letters 8.7 (2014): 1999-2020. (2014)

[6] Rockafellar R.T. and S. Uryasev Optimization of Conditional Value-At-Risk. The Journal of Risk, Vol. 2, No. 3, 2000, 21-41. (2000)