Cardinality of Upper Average and Its Application to Network Optimization

Matthew Norton, Aleksandr Mafusalov, Stan Uryasev

March 2015

RESEARCH REPORT 2015-1
Risk Management and Financial Engineering Lab
Department of Industrial and Systems Engineering
303 Weil Hall, University of Florida, Gainesville, FL 32611.
E-mail: *mdnorton@ufl.edu, mafusalov@ufl.edu, uryasev@ufl.edu*
Current Draft: December 2015, Version 2
Older Drafts: July 2015, Version 1

Abstract

When given a set of data, one is often interested in calculating the average value of a portion of largest data points in the tail of the data distribution, i.e. an Upper Average (UA) of the data set. This paper introduces a related quantity called the Cardinality of Upper Average (CUA), defined as the number of largest data points which have average value equal to a specified threshold. We provide multiple formulae for calculating CUA, showing it to be piecewise linear in its reciprocal w.r.t. the threshold parameter. We show CUA to be an upper bound for the Cardinality of Upper Tail (CUT), the number of largest data points exceeding the same threshold. Additionally, we find that CUA fits smoothly into optimization frameworks, reducing to convex and linear programming (LP). We show that CUA can be used to formulate optimization problems containing counters of the largest components of a vector without introduction of binary variables. Furthermore, CUA can be applied to several classes of optimization problems, leading to dramatic improvement of computation speeds. In particular, we apply the CUA concept to create new formulations of network optimization problems involving node failures.

1 Introduction

When given a set of n data points, there are multiple quantities one may be interested in calculating to characterize the data distribution. For example, one may want to calculate a k-Upper Average (UA_k) as the average of the k largest components of the data set. With UA_k, one specifies the number of largest components to average over, i.e. with the k parameter, and one then finds the UA_k to be equal to some number $x \in \mathbb{R}$. It may, though, be useful to instead specify a threshold $x \in \mathbb{R}$ and calculate the number of largest data points, k, which average to x. In this paper, we introduce this quantity as the x-Cardinality of Upper Average (CUA_x). With CUA_x, one specifies a threshold x and CUA_x at that particular threshold equals the number of largest data points, k, with UA_k equal to x.

After introducing CUA_x, we provide multiple ways to calculate this quantity. We first show that it is a unique solution to a simple minimization problem. We then show that it is a piecewise linear function in its reciprocal w.r.t. the threshold parameter, allowing for easy calculation of CUA_x over all possible threshold choices.

We also discuss CUA_x in relation to another characterization of the tail of a data distribution. Given a threshold $x \in \mathbb{R}$, it can be useful to calculate the x-Cardinality of Upper Tail (CUT_x), the number of data points with magnitude exceeding the threshold x. This quantity, though, can pose challenges in an optimization setting, as this may involve introduction of binary, or indicator variables. We show that given a threshold x, CUA_x is a natural upper bound for CUT_x that fits much more smoothly into optimization frameworks, leading to convex and linear programming.

Additionally, we show that 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 [6]. This special case helps to elucidate the inverse relationship between CUA_x and a deterministic variant of CVaR.

Lastly, we apply this concept to an optimization setting, showing that CUA's efficient calculation formula lends itself to efficient optimization. Specifically, we consider network optimization problems, showing that traditional Mixed Integer Programming (MIP) formulations can be transformed into Linear Programming (LP) formulations by use of CUA_x. In network optimization, it is often necessary to utilize binary variables to indicate the 'state' of a particular node or edge in the network graph structure. For example, a node in a network may be deemed as 'overloaded' if its capacity exceeds a particular capacity threshold. This overloaded state may be associated with a large increase in flow, or may simply be a state that must be avoided. Thus, when formulating a network optimization problem, in order to indicate an overloaded node state within the optimization framework, it is natural to introduce binary variables to indicate said state as 'on' or 'off.' This modeling choice, though, comes at a computation cost, with binary variables transforming optimization problems into MIP's. We aim to solve these types of problems, where the binary state of a node is critical to the optimization formulation, without introduction of binary variables by use of CUA_x. As we will show, CUA_x allows us to rid certain network optimization problems of binary variables, leading to convex and linear programming reformulations of network optimization MIP's.

The CUA_x concept is actually not a wholly new idea. It is a special case of so called Buffered Probability of Exceedance (bPOE). A detailed discussion of this concept, studied in [3, 5, 8, 9, 10], is beyond the scope of this paper. We maintain a deterministic setting,

while bPOE is studied in a probabilistic, stochastic optimization setting. We do, though, include some background connecting CUA_x and bPOE in Appendix B for the interested reader.

This paper is organized as follows. Section 2 discusses the task of analyzing the tail of a data distribution in a deterministic setting, which serves to set the stage for defining CUA_x. Section 3.1 defines CUA_x. We show that CUA_x is efficient to calculate and give an example illustrating CUA_x, particularly as an upper bound of CUT_x. Additionally, we provide an efficient method of calculating CUA_x which utilizes its piecewise linearity. Section 3.2 provides relations between CUA_x and CUT_x, showing that it is possible to simultaneously calculate CUA_x and CUT_x. Section 3.3 defines CAUA and shows that it is the inverse of the CVaR norm. Section 4 shows the power of the CUA_x concept when applied to an optimization setting. Specifically, we discuss classes of network optimization problems that are traditionally formulated as MIP's and show how application of CUA_x leads to convex or linear programming reformulations of these problems. We show the substantial computational advantages of this approach with a shipment network application.

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 of [11]. As already mentioned in the introduction, we can consider UA_k as well, which provides information about the magnitude of data points in the tail and upper bounds on CUT_x (a quantity which may still be of interest). 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 [7] 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 [7], 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 k and $\lceil k \rceil$ denotes the smallest integer greater than 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.

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

In this paper, we define CUA_x, the inverse of (2). As we will show, CUA_x can be efficiently calculated, provides valuable information about the largest components of our data vector $\mathbf{y}$, and is similar to CUT_x. Additionally, CUA_x can be easier to work with in an optimization setting. Specifically, we show that optimization problems involving CUT_x, which necessarily involve binary variables, can be replaced by optimization of CUA_x, achieving a similar objective without need for binary variables.

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 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 solution to a minimization problem. For interested readers, a detailed derivation is included in Appendix A. This detailed derivation also shows how CUA_x is also defined for thresholds $x \notin (\mathrm{UA}_n(\mathbf{y}), \mathrm{UA}_1(\mathbf{y}))$ where $\mathrm{UA}_n(\mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} y_i$ and $\mathrm{UA}_1(\mathbf{y}) = \max_i y_i$. Furthermore, Appendix B includes a brief discussion of the connection between CUA_x and bPOE, a new concept recently studied in [3, 5, 8, 9, 10]. CUA_x is a special case of

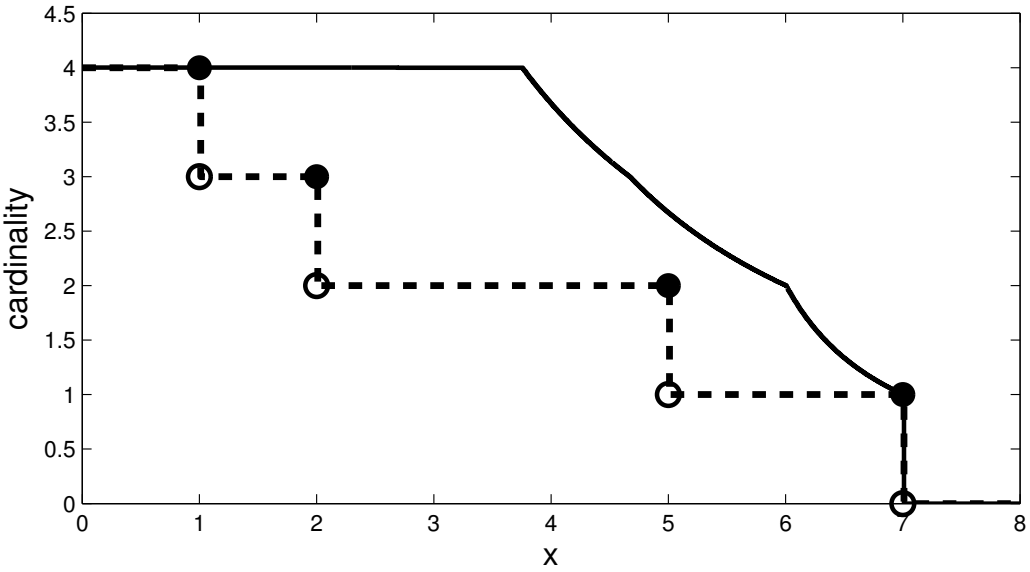

Figure 1: Cardinality of Upper Tail (CUT_x), $\eta_x(\mathbf{y})$, dashed line, and Cardinality of Upper Average (CUA_x), $\bar{\eta}_x(\mathbf{y})$, solid line, as functions of threshold x, where $\mathbf{y} = (1, 2, 5, 7)$.

Upper bPOE [5, 3]. A detailed discussion of this, though, is beyond the scope of this paper.

To begin discussing CUA_x, first notice that CUA_x is continuous w.r.t. the parameter x on the interval $x \in (-\infty, \text{UA}_1(\mathbf{y}))$; 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 $\text{UA}_k(\mathbf{y}) = x$, CUA_x is counting all components with magnitude greater than x and some components with magnitude less than x.

Consider now a simple example, illustrated in Fig. 1, with data vector $\mathbf{y} = (1, 2, 5, 7)$. For this vector, $\text{CUA}_\text{x} = 2$ for $x = 6$, because the average of the two largest components of the vector $\mathbf{y}$ equals $(7 + 5)/2 = 6$, and $\text{CUA}_\text{x} = 3$ for $x = 4\frac{2}{3} = (2 + 5 + 7)/3$. We can observe that CUA_x is an upper bound for CUT_x. Additionally, for this example, Fig. 1 shows that CUA_x is continuous w.r.t. x except at the maximum point where threshold $x = 7$, while CUT_x is discontinuous at $x = 1, 2, 5$, and 7.

3.1.1 CUA_x Calculation via Linear Interpolation

Suppose that one would like to calculate CUA_x for multiple threshold levels $x \in \mathbb{R}$. One could utilize formula (4) to achieve this task, but this proves quite inefficient if one would like to know CUA_x for all thresholds $x \in \mathbb{R}$. When one is simply interested in calculating CUA_x for all thresholds $x \in \mathbb{R}$ for a fixed vector $\mathbf{y}$, it is possible to utilize linear interpolation to calculate CUA_x at all thresholds $x \in \mathbb{R}$. To show this, we first introduce the following alternative calculation formula for CUA_x, which shows that CUA_x can be calculated via minimization over a finite set of points. It also provides additional insight into formula (4).

Proposition 1: *Consider a Euclidean vector* $\mathbf{y} = (y_1, \ldots, y_n)$ *and let* $y_0 = -\infty$. CUA_x

of $\mathbf{y}$ *at threshold* $x \in \mathbb{R}$ *equals*

$$\bar{\eta}_x(\mathbf{y}) = \min_{j \in \{0,1,\ldots,n\}} \frac{\sum_{i=1}^n [y_i - y_j]^+}{[x - y_j]^+} \ . \tag{5}$$

Proof: A change of variable $a = \frac{1}{x-\gamma}$ transforms minimization formula (4) to the following formula:

$$\bar{\eta}_x(\mathbf{y}) = \inf_{\gamma < x} \frac{\sum_{i=1}^n [y_i - \gamma]^+}{x - \gamma}. \tag{6}$$

Let us show that the objective is minimal for either $\gamma = y_j$ for $j = 1, \ldots, n$, or for $\gamma \to -\infty = y_0$. Suppose the contrary. Then the objective is differentiable at the optimal γ^*, since only $\gamma = y_i$ are the points where derivative over γ does not exist. Then the derivative of the objective w.r.t. γ must be 0 at γ^*:

$$(x-\gamma)^{-2} \left[(x-\gamma) \left(\sum_{i=1}^n [y_i - \gamma]^+ \right)'_\gamma - (x-\gamma)'_\gamma \sum_{i=1}^n [y_i - \gamma]^+ \right] \Bigg|_{\gamma=\gamma^*} = 0,$$

$$\sum_{y_i > \gamma^*} (-1)(x - \gamma^*) - (-1) \sum_{y_i > \gamma^*} (y_i - \gamma^*) = \sum_{y_i > \gamma^*} (y_i - x) = 0. \tag{7}$$

Suppose $\gamma^* \in (y_i, y_{i+1})$, or $\gamma^* \in (y_i, x)$, where $i = 0, \ldots, n$. From the derivative expression above it follows that the derivative should be equal to 0 on the corresponding interval, therefore, $\gamma = y_i$ is also an optimal solution, contradiction. $\square$

Utilizing this proposition, the following Corollary 1 shows that CUA_x can be calculated via simple linear interpolation. Proposition 1, by itself, though, provides interesting insights. First, we see that calculation can be performed by only considering the finite set of points $(y_0, y_1, \ldots, y_n)$. Second, it follows from Proposition 1 that if y_j is the argmin of (5), then $a^* = \frac{1}{x-y_j}$ is the argmin of (4). This fact can be seen more clearly in the proof of Proposition 1 (i.e. see the change of variable that takes place).

Moving to the discussion of linear interpolation, suppose we have the vector $\mathbf{y} \in \mathbb{R}^n$. Instead of using formula (4) to calculate CUA_x, one only needs to calculate $\mathrm{UA}_k(\mathbf{y})$ for $k \in \{1, \ldots, n-1\}$, which effectively calculates CUA_x for thresholds $x \in \{\mathrm{UA}_{n-1}(\mathbf{y}), \ldots, \mathrm{UA}_1(\mathbf{y})\}$, then utilize linear interpolation to calculate CUA_x for the intermediate threshold values.

Corollary 1: *The function* $\frac{1}{\bar{\eta}_x(\mathbf{y})}$ *is a piecewise-linear convex function of* x *with knots at* $x \in \{\mathrm{UA}_n(\mathbf{y}), \mathrm{UA}_{n-1}(\mathbf{y}), \ldots, \mathrm{UA}_1(\mathbf{y})\}$. *Specifically, for any threshold* $x = \lambda \mathrm{UA}_{i+1}(\mathbf{y}) + (1-\lambda)\mathrm{UA}_i(\mathbf{y})$, $i \in \{1, \ldots, n-1\}$, $\lambda \in (0,1)$, *we have that*

$$\frac{1}{\bar{\eta}_x(\mathbf{y})} = \frac{\lambda}{\bar{\eta}_{\mathrm{UA}_{i+1}(\mathbf{y})}(\mathbf{y})} + \frac{1-\lambda}{\bar{\eta}_{\mathrm{UA}_i(\mathbf{y})}(\mathbf{y})} \tag{8}$$

Proof: The derivative expression (7) for minimization problem (6) implies that at $x \in [\mathrm{UA}_{n-j-1}(\mathbf{y}), \mathrm{UA}_{n-j}(\mathbf{y})]$ and $\gamma = y_j$, the derivative $\sum_{y_i > \gamma - \epsilon}(y_i - x) \le 0$, the derivative $\sum_{y_i > \gamma + \epsilon}(y_i - x) \ge 0$ for all $\epsilon > 0$. Therefore, $\gamma = y_j$ is optimal for $x \in [\mathrm{UA}_{n-j-1}(\mathbf{y}), \mathrm{UA}_{n-j}(\mathbf{y})]$ and

$$\bar{\eta}_x(\mathbf{y}) = \frac{\sum_{j=1}^n [y_i - y^{(j+1)}]^+}{x - y^{(j+1)}} \ .$$

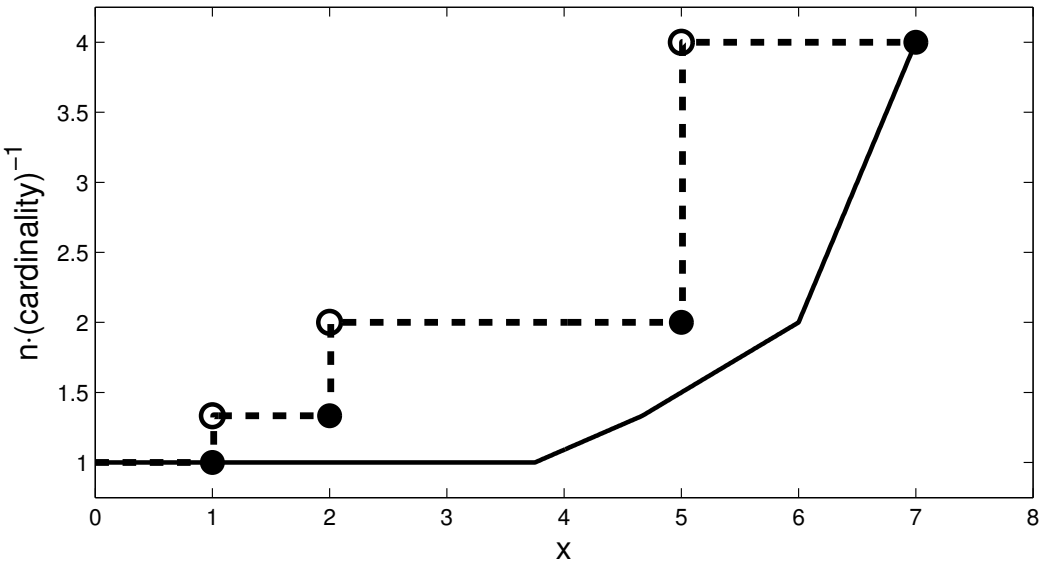

Figure 2: $\frac{1}{\eta_x(\mathbf{y})}$, dashed line, and $\frac{1}{\bar{\eta}_x(\mathbf{y})}$, solid line, as functions of threshold x, where $\mathbf{y} = (1, 2, 5, 7)$.

Hence,

$$\frac{1}{\bar{\eta}_x(\mathbf{y})} = \frac{\lambda}{\bar{\eta}_{\mathrm{UA}_{i+1}(\mathbf{y})}(\mathbf{y})} + \frac{1-\lambda}{\bar{\eta}_{\mathrm{UA}_i(\mathbf{y})}(\mathbf{y})}$$

for $x = \lambda \mathrm{UA}_{i+1}(\mathbf{y}) + (1-\lambda)\mathrm{UA}_i(\mathbf{y})$, $i \in \{1, \ldots, n-1\}$, $\lambda \in (0, 1)$ and values $\{\mathrm{UA}_n(\mathbf{y}), \ldots, \mathrm{UA}_1(\mathbf{y})\}$ are knots for the piecewise linear function $\frac{1}{\bar{\eta}_x(\mathbf{y})}$. $\square$

Thus, we only need to calculate CUA_x for n-thresholds, namely $x \in \{\mathrm{UA}_{n-1}(\mathbf{y}), \ldots, \mathrm{UA}_1(\mathbf{y})\}$ and we can 'fill in' the missing thresholds via linear interpolation. Using the example $\mathbf{y} = (1, 2, 5, 7)$ from Section 3.1, we illustrate the piecewise linearity in Fig. 2 which plots $\frac{1}{\bar{\eta}_x(\mathbf{y})}$ on the vertical axis and x on the horizontal axis.

3.2 Connecting CUA_x and CUT_x

3.2.1 CUA_x as Upper Bound on CUT_x

Fig. 1 shows for a specific example that CUA_x acts as an upper bound for CUT_x. Specifically, this can be posed as the following relation, which follows intuitively from the definitions of CUA_x, UA_k, and CUT_x.

$$\bar{\eta}_x(\mathbf{y}) = k \iff \mathrm{UA}_k(\mathbf{y}) = x \implies \eta_x(\mathbf{y}) \leq k \,. \tag{9}$$

This may be quite useful in practice. As a standalone counter of vector components, CUA_x may provide useful information. Additionally, though, if CUT_x is still of importance, CUA_x can be viewed as an efficiently calculable upper bound for CUT_x. As will be seen in later sections, this is especially applicable when minimization of CUT_x in an optimization setting leads to MIP formulations. Minimizing CUA_x in place of CUT_x acts as a minimizer of a CUT_x upper bound that can be solved with extreme efficiency via an LP.

3.2.2 Simultaneous Calculation of CUA_x and CUT_x

An important and interesting property of CUA_x calculation formula (4) is that it simultaneously calculates CUA_x and CUT_x. Specifically, we have the following property of formula (4).

Proposition 2: *Suppose that for a vector* $\mathbf{y} \in \mathbb{R}^n$ *and threshold* $x \in (\mathrm{UA}_n(\mathbf{y}), \mathrm{UA}_1(\mathbf{y}))$ *we have that*

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

meaning that a^* *is an optimal solution to (4) and* y_j *is optimal to (5) with* $a^* = \frac{1}{x - y_j}$. *Then, we know that*

$$\lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1 = \eta_{x - \frac{1}{a^*}}(\mathbf{y}) = \eta_{y_j}(\mathbf{y}) \ ,$$

meaning that CUT_x *at threshold* $x - \frac{1}{a^*} = y_j$ *equals one plus the largest integer less than or equal to* CUA_x *at threshold* x.

Proof: For this proof, let us denote the kth largest component of $\mathbf{y}$ as $y^{(k)}$ and for any nonnegative real number N let $\lceil N \rceil$ denote the smallest integer greater than or equal to N and let $\lfloor N \rfloor$ denote the largest integer less than or equal to N. From [7], we know that if $\mathrm{UA}_k(\mathbf{y}) = x$ and $\gamma^* = \operatorname{argmin}_{\gamma} \gamma + \frac{1}{k} \sum_{i=1}^{n} [y_i - \gamma]^+$, then $y^{(\lceil n-k \rceil)} = \gamma^*$. (Just a note for clarification of this point: Treating $\mathbf{y}$ as a uniformly distributed discrete random variable, and speaking in terms of quantiles and the notation defined in Appendix B, this is equivalent to saying that $q_{(\frac{n-k}{n})}(\mathbf{y}) = \gamma^*$, where $k = n(1-\alpha)$ and $\alpha \in (0,1)$ is a fixed probability level.) As shown in Case 1, Appendix A,

$$\begin{aligned} \bar{\eta}_x(\mathbf{y}) &= \{k | \mathrm{UA}_k(\mathbf{y}) = x\} \\ &= \min\{k | \min_{\gamma} \gamma + \frac{1}{k} \sum_{i=1}^{n} [y_i - \gamma]^+ \leq x\} \\ &= \min_{x - \gamma > 0} \frac{\sum_{i=1}^{n} [y_i - \gamma]^+}{x - \gamma} \ , \end{aligned} \tag{10}$$

where the optimal γ is the same throughout the last two equalities. We can then see that if

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

then $\mathrm{UA}_{\bar{\eta}_x(\mathbf{y})}(\mathbf{y}) = x$ and $\gamma^* = \operatorname{argmin}_{\gamma} \gamma + \frac{1}{k} \sum_{i=1}^{n} [y_i - \gamma]^+$ and $y^{(\lceil n - \bar{\eta}_x(\mathbf{y}) \rceil)} = \gamma^*$. Stating this in terms of the change of variable $a^* = \frac{1}{x - \gamma^*}$, we get $y^{(\lceil n - \bar{\eta}_x(\mathbf{y}) \rceil)} = x - \frac{1}{a^*}$. Finally, we note that for any $k \in \{1, \ldots, n-1\}$, $y^{(n-k)} = \hat{y}$ implies that $\eta_{\hat{y}}(\mathbf{y}) = n - (n-k) + 1 = k + 1$. Therefore, $y^{(\lceil n - \bar{\eta}_x(\mathbf{y}) \rceil)} = x - \frac{1}{a^*}$ implies that $\eta_{x - \frac{1}{a^*}}(\mathbf{y}) = n - \lceil n - \bar{\eta}_x(\mathbf{y}) \rceil + 1 = \lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1$. Furthermore, we know from Proposition 1 that if a^* is optimal solution to (4) and y_j is optimal to (5), then $a^* = \frac{1}{x - y_j}$. Thus, we finally have that $\eta_{x - \frac{1}{a^*}}(\mathbf{y}) = \eta_{y_j}(\mathbf{y}) = \lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1$. □

This property is quite useful, as it shows that simply calculating CUA_x provides information about CUT_x. Note that, although a^* may not be a unique optimal solution, the equality $\eta_{x - \frac{1}{a^*}}(\mathbf{y}) = \lfloor \bar{\eta}_x(\mathbf{y}) \rfloor + 1$ still holds.

3.3 CVaR Norm and CAUA: A Special Case

So far we have discussed CUA_x as a characterization of the largest components of a vector, or data set. In this context, it is natural to consider its relationship with vector norms. The norm of a vector, e.g. L-2 or L-1 norm, can be utilized in a similar vein to characterize the components of a vector. Here we show that a special case of CUA_x can be viewed as the inverse of the CVaR norm, a vector norm defined in [6] and furthermore generalized in [4]. It should be noted that a similar norm was introduced by [1], which is a special case of an Ordered Weighted Averaging (OWA) Operator [11].

The CVaR norm, as defined in [6], is defined in terms of a probability, or confidence level. To avoid discussion of probabilities, we write the CVaR norm equivalently in terms of $\text{UA}_k(\cdot)$ in the following way.

Definition 2: Consider the absolute value of a Euclidean vector $|\mathbf{y}| = (|y_1|, \ldots, |y_n|)$. The CVaR norm of $\mathbf{y}$ at confidence level $\alpha \in [0, 1]$, denoted by $\langle\langle \mathbf{y} \rangle\rangle_\alpha$, equals $\text{UA}_k(|\mathbf{y}|)$ with $k = n(1-\alpha)$. Specifically, letting $k = n(1-\alpha)$,

$$\langle\langle \mathbf{y} \rangle\rangle_\alpha = \text{UA}_k(|\mathbf{y}|) = \min_\gamma \left\{ \gamma + \frac{1}{k} \sum_{i=1}^{n} [|y_i| - \gamma]^+ \right\} . \tag{11}$$

Now, we define the following special case of CUA_x, which we call Cardinality of Absolute Upper Average (CAUA).

Definition 3: Consider the absolute value of a Euclidean vector $|\mathbf{y}| = (|y_1|, \ldots, |y_n|)$. CAUA of $\mathbf{y}$ at threshold $x \in \mathbb{R}$ equals CUA_x of $|\mathbf{y}|$ at threshold $x \in \mathbb{R}$. Specifically, CAUA of $\mathbf{y}$ at x equals

$$\bar{\eta}_x(|\mathbf{y}|) = \min_{a \geq 0} \sum_{i=1}^{n} [a(|y_i| - x) + 1]^+ . \tag{12}$$

Since CUA_x is the inverse of $\text{UA}_k(\cdot)$, it is now easy to see that CAUA is the inverse of the CVaR norm. In other words, CUA_x applied to the absolute value of a vector gives the inverse of the CVaR norm.

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. On the other hand, since CUA_x serves as an upper bound on CUT_x, we can instead consider minimization of CUA_x, which is posed as follows.

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

As we will show by example in Section 4.1, 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.

4.1 Applications to Network Optimization

We now show that network flow problems, specifically ones utilizing MIP formulations for CUT_x minimization, can be transformed into Linear Programming problems by replacing CUT_x with CUA_x. Section 4.1 shows this for a broad class of network flow problems, specifically the minimum network overflow problem. Section 4.2 presents proof that the CUT_x minimization problem is NP hard, showing that regardless of the strategy used to solve the MIP given by CUT_x minimization, CUA_x minimization will be more efficient and solvable in polynomial time. Section 4.3 and 4.4 provide two special cases of the minimum network overflow problem, where we carry out numerical experiments demonstrating the numerical efficiency of the CUA_x formulation. For this, we simply report the general model parameters and solution times, as the optimal flow through the network does not provide much additional insight. Section 4.5, gives an example of a small graph structure that provides insight into the benefits of the optimal CUA_x solution, as compared to the optimal MIP solution.

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 the transshipment vertices can only handle a particular amount of flow. In other words, it is possible for the transshipment vertices to become overloaded, which is undesirable. 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 $(i,j) \notin E$.
- $x_{ij} \geq 0$ = flow through the edge (i,j).
- $l_i = \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(k,i)\in E} c_{ki}x_{ki}$ = 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.
- $\lambda \geq 0$ = "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 λ.
- 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 $\eta_\lambda(\mathbf{l})$.

$$\min_{\xi,\mathbf{l},x} \quad \sum_{i\in V'} \xi_i \tag{13}$$

$$s.t. \quad \xi_i \geq (l_i - \lambda)/M,\ l_i = \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(k,i)\in E} c_{ki}x_{ki}\ \forall i \in V', \tag{14}$$

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(k,i)\in E} x_{ki} = b_i\ \forall i \in V, \tag{15}$$

$$\xi_i \in \{0,1\}\ \forall i \in V',\ x_{ij} \geq 0\ \forall (i,j) \in E. \tag{16}$$

Though this MIP will solve the desired problem, that of minimizing $\eta_\lambda(\mathbf{l})$, the introduction of binary variables greatly increases the computational burden. Consider, though, minimization of CUA_x in place of CUT_x. This gives us the following formulation:

$$\min_{a,\mathbf{l},x} \quad \sum_{i=1}^{n} [a(l_i - \lambda) + 1]^+ \tag{17}$$

$$s.t. \quad l_i = \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(k,i)\in E} c_{ki}x_{ki}\ \forall i \in V', \tag{18}$$

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(k,i)\in E} x_{ki} = b_i\ \forall i \in V, \tag{19}$$

$$a \geq 0 \tag{20}$$

$$x_{ij} \geq 0\ \forall (i,j) \in E. \tag{21}$$

The CUA_x optimization problem, though, can be further reduced to LP form with a simple change of variable. Specifically, optimization problem (17)–(21) with the change of variables $y_{ij} = ax_{ij}$ can be reduced to the following LP.

$$\min_{a,z,y} \quad \sum_{i\in V'} z_i \tag{22}$$

$$s.t. \quad z_i \geq \sum_{(i,j)\in E} c_{ij}y_{ij} + \sum_{(k,i)\in E} c_{ki}y_{ki} - a\lambda + 1\ \forall i \in V', \tag{23}$$

$$\sum_{(i,j)\in E} y_{ij} - \sum_{(k,i)\in E} y_{ki} = ab_i\ \forall i \in V, \tag{24}$$

$$a \geq 0,\ y_{ij} \geq 0\ \forall (i,j) \in E,\ z_i \geq 0\ \forall i \in V'. \tag{25}$$

This is a powerful result, which implies that we can achieve the same goal (i.e. minimizing the number of transshipment vertices with loads exceeding a specified threshold λ) using linear programming as opposed to mixed-integer programming. In Section 4.3 we present a special case of the minimum network flow problem, and show numerically the efficiency of the LP formulation versus the MIP formulation. In the next section, though, we prove that the CUT_x minimization problem for general graphs is NP hard.

4.2 CUT_x minimization is NP hard

It should be noted that the MIP formulation we consider for the CUT_x minimization problem is a Big-M type formulation. Though there may be more efficient methods for formulating and approximating such a MIP, e.g. by considering a Disjunctive Programming approach, we emphasize that the CUT_x minimization problem is NP hard. In this section, we prove that for arbitrary graphs, the CUT_x minimization problem is NP hard. Therefore, regardless of the strategy used to solve (exactly or approximately) the CUT_x minimization problem, CUA_x minimization will reduce to convex and linear programming which have polynomial-time solvers.

In Proposition 3 we show that the problem of minimizing CUT_x for an arbitrary graph is an NP-hard problem with polynomial-time reduction from an NP-complete set covering problem to CUT_x minimization problem. The NP-hardness implies that the considered problem is at least as hard as any P, NP, or NP-complete problem, that is, time consumed by solving this problem will probably grow exponentially with the size of the problem.

Proposition 3: CUT_x minimization is an NP-hard problem.

Proof: The set covering problem is NP-complete. Solving a set-covering problem with one run of the CUT_x minimization problem of the same size will prove that CUT_x minimization problem is NP-hard. Suppose there is a set S consisting of sets S_i: $S = \{S_1, \ldots, S_n\}$, where $S_i \subseteq U \equiv \{1, \ldots, m\} = \cup_{i=1}^{n} S_i$. It is required to find minimal covering set $S^* = \{S_1^*, \ldots, S_k^*\}$ with $S_k^* \in S$ and minimal k. Consider a three-layer graph. On the first layer of the graph there are m demand vertices with demand 1, each vertex corresponds to an object from the union set U. On the second layer of the graph there are n transshipment nodes with demand/supply value 0, which correspond to sets S_i. The first-layer vertex j and the second-layer vertex i are connected iff $j \in S_i$. The transportation costs along all edges are equal to 1. The third layer of the graph consists of a single auxiliary supply vertex with the supply m. CUT_x is measured for transshipment vertices, and the critical threshold is 0. That is, if transshipment vertex is involved in transportation, it is violated. Finally, for the described graph it can be seen that picking a covering subset with minimal number of sets is equivalent to minimizing the number of overloaded transshipment vertices.
□

4.3 Customer Transportation Costs Optimization

This example is a special case of the Minimum Network Overflow Problem. Take $V' = V_D$, where c_{ij} $\forall (i,j) \in E$ is associated with usage cost of the channel (i,j). The goal is to assign routing in a such way that the end users do not pay too much. Let there be a critical "acceptable" total usage cost λ for each end user. It is assumed that for each end user it is undesirable to exceed the cost value λ. One way to solve this problem is to minimize a number of overpaying end users, which can be solved via mixed-integer optimization problem (13)–(16). Alternatively, one can optimize CUT_x for end user costs with respect to level λ, i.e., minimize the number of end users with largest costs such that these tail costs are averaging in λ. This would result in optimization problem (17)–(21),

which is equivalent to the LP formulation (22)–(25).

4.4 Minimum Overloaded Servers

To demonstrate the performance of the CUA_x formulation versus the CUT_x formulation, we form another special case of the generic overload flow problem described in Section 4.1. Specifically, assume that $V = V_S \cup V_T \cup V_D$ — vertices are divided into three categories: supply vertices $b_i > 0 \; \forall i \in V_S$; transshipment vertices $b_i = 0 \; \forall i \in V_T$; demand vertices $b_i < 0 \; \forall i \in V_D$. Then, take $V' = V_T$, $c_{ij} > 0 \; \forall (i,j) \in E$. One possible interpretation comes from an information flow networks area. Supply vertices are associated with data servers, demand vertices are associated with end users, transshipment nodes are associated with routing servers. The goal is to assign routing in a such way that the routing servers are not overloaded with information flow. Let λ be a critical flow value, i.e., it is undesirable to exceed the load λ.

One way to solve this problem is to minimize a number of overloaded servers, which can be solved via mixed-integer optimization problem (13)–(16). Alternatively, one can minimize CUA_x with respect to level λ, i.e., minimize the number of routing servers with largest loads such that these tail loads are averaged to λ. This would result in optimization problem (17)–(21), which is equivalent to the LP formulation (22)–(25).

4.4.1 Numerical Example

Using the network structure from Section 4.3, we solved differently sized problems using Gurobi Solver on a PC via a Python interface. The graph structure and model setup are specified in the following list. Additionally, Python code and full problem description can be found online.[1] Note that threshold was chosen so that the formulations would not yield trivial solutions where the optimal objective equals 0 or T. Additionally, with edge cots being randomly generated, run times are reported for graph instances yielding non-trivial solutions :

- Number of Demand nodes = 10,000
- Number of Transshipment nodes= T
- Number of Supply Nodes = 1
- Demand per demand node = 10
- Supply for supply node = 100,000 = (10)x(10,000)
- Each demand node is connected to all transshipment nodes with edge cost c_{ij} uniformly distributed in the interval $(0, 1)$.
- The supply node is connected to all transshipment nodes with edge cost c_{ij} uniformly distributed in the interval $(0, 1)$.
- No edge connecting supply node and demand nodes. No edges connecting demand nodes.

[1] http://www.ise.ufl.edu/uryasev/research/testproblems/logistics/network-optimization-by-minimization-of-cardinality-of-upper-averages-cua/

T	**MIP Run Time (seconds)**	**LP Run Time (seconds)**
100	2244.94	31.53
75	492.42	14.52
50	191.34	13.23
25	109.33	5.95
10	21.26	6.47

Table 1: Number of transshipment nodes, T, and approximate time to solve with Gurobi in Python interface.

- Threshold for CUA_x minimization = λ =.5 x 100,000 / T
- Threshold for CUT_x minimization = λ = .2 x 100,000 / T
- M-value for big M = 100,000

With this setup, we compared the performance of the MIP formulation (13)–(16) and the LP formulation (22)–(25) for varying values of T. Run time comparisons can be seen in Table 1. We see that the LP formulation has a clear advantage as the number of transshipment nodes, i.e. the number of binary variables in the MIP formulation, increases. Thus, we see that the CUA_x minimization problem is significantly faster. For large problems, CUA_x solving time will be dramatically lower than MIP solving time.

4.5 Small Graph: Optimal CUA_x vs. CUT_x Flow

An example below illustrates CUA_x vs CUT_x minimization for loads on transshipment vertices. Graph structure is shown in Figure 3. For each of the demand vertices there is a "regular" edge to a corresponding transshipment vertex. There are also two "costly" vertices. There is a single supply vertex connected to all transshipment vertices, and transportation between supply and transshipment vertices is free of cost.

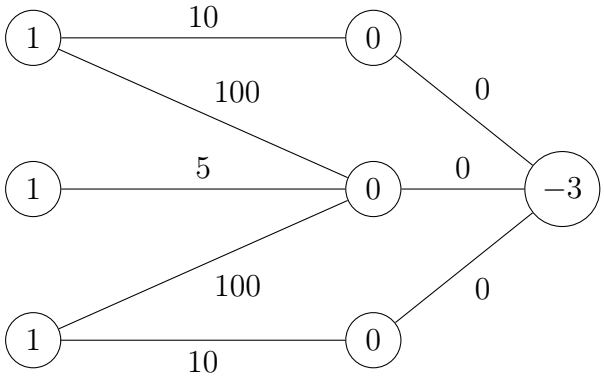

Figure 3: Graph structure. Numbers inside circles represent demand/supply. Numbers aside of line segments represent edge costs per unit transported.

Suppose that the threshold for overloading is $\lambda = 9$. It is easy to see that an optimal solution in CUA_x minimization problem will have nonzero flows only on "regular"

edges, and therefore the vector of loads on transshipment vertices is $\mathbf{l} = [10, 5, 10]$, which corresponds to CUA_x value $\bar{\eta}_\lambda(\mathbf{l}) = \bar{\eta}_9([10, 5, 10]) = 2.5$. The optimal network flow is illustrated in Figure 4.

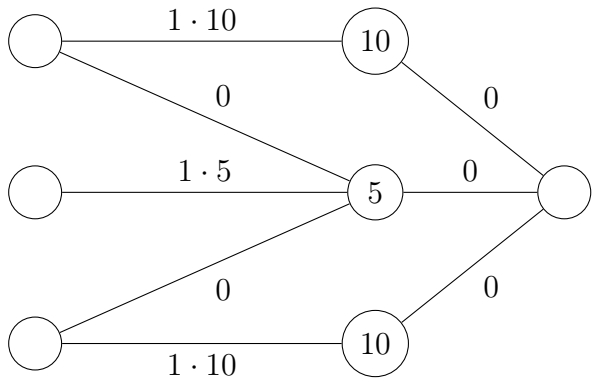

Figure 4: Optimal network flow for CUA_x minimization. Numbers inside circles represent loads. Numbers aside of line segments represent transportation costs.

The solution obtained in CUA_x minimization will not be optimal for CUT_x minimization, since $\eta_9([10, 5, 10]) = 2$, and there are feasible solutions with $\eta_9(\cdot) = 1$. However, all these solutions are forced to overload the second transshipment node. Two extreme optimal solutions with smallest and lardest total loads are illustrated in Figure 5. It can be seen that second transhipment node state varies from very overloaded (i.e. load equal to 25) to extremely overloaded (i.e. load equal to 205).

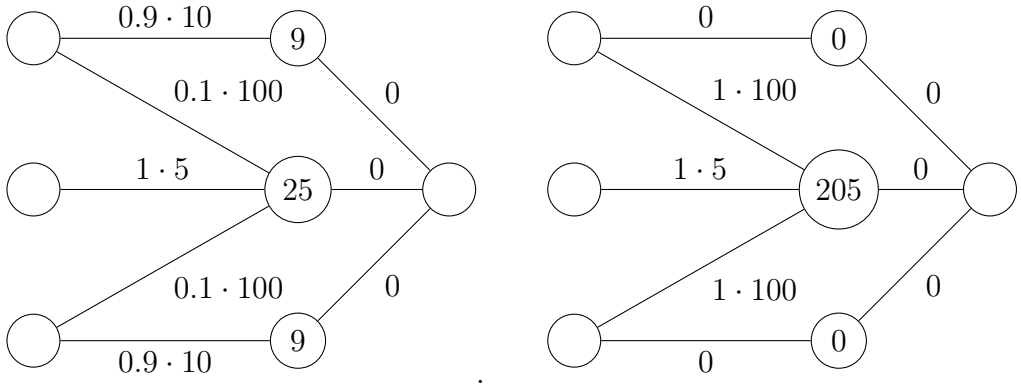

Figure 5: Optimal network flow for CUT_x minimization: (a) solution with smallest total cost among optimal solutions; (b) solution with largest total cost among optimal solutions. Numbers inside circles represent loads. Numbers aside of line segments represent transportation costs.

Note that if costs of "costly" edges are increased, then there is no change in optimal flows for both CUA_x solution and CUT_x solutions, however, total loads in CUT_x optimal solutions will grow indefinitely. This example shows that CUT_x minimization approach, despite being very intuitive, can lead to quite unappealing and impractical results.

5 Conclusion

In this paper, we have introduced a new concept called CUA_x, a continuous function which acts as a counter of the largest components of a vector. We have shown that CUA_x has merits as a standalone measure of vector components, while also providing upper bound information about CUT_x. Additionally, we have shown that CUA_x is the inverse of UA_k, a function that measures the average magnitude of the largest k components of a vector. Applying this to a special case, we have shown that one minus CAUA is the inverse of an existing vector norm, namely the CVaR norm.

We have also shown that CUA_x can be calculated very efficiently, and it is this calculation which lends itself to efficient optimization. Network flow optimization problems often involve minimization of the number of nodes with flow exceeding a critical threshold. This minimization problem often yields a mixed integer formulation involving binary variables, which causes it to suffer from computational complexity. We have introduced an alternative to the CUT_x minimization problem utilizing a concept called CUA_x. Specifically, we have shown that solving the CUA_x minimization problem via an LP formulation can be used in place of the CUT_x minimization problem utilizing an MIP formulation. The CUA_x minimization problem can be solved much more efficiently, as it does not involve any binary variables. In addition, the CUA_x minimization problem can achieve solutions to network flow problems that are more appealing than the CUT_x minimization alternative.

A Derivation of Formula for CUA_x

Here we show that for a Euclidean vector $\mathbf{y} \in \mathbb{R}^n$, the equation (4) given for CUA_x at threshold $x \in \mathbb{R}$ equals the value of k such that $\mathrm{UA}_k(\mathbf{y}) = x$. We must address four cases. **Case 1:** Assume $x \in (\mathrm{UA}_n(\mathbf{y}), \mathrm{UA}_1(\mathbf{y}))$. This assumption ensures that there exists a value of $k \in [1, n]$ such that $\mathrm{UA}_k(\mathbf{y}) = x$. We want to find the value of k such that $\mathrm{UA}_k(\mathbf{y}) = x$, so we write CUA_x as

$$\bar{\eta}_x(\mathbf{y}) = \{k | \mathrm{UA}_k(\mathbf{y}) = x\} \ . \tag{26}$$

Notice now that $\mathrm{UA}_k(\mathbf{y})$ is a strictly increasing function of k on $k \in [\frac{n-m}{n}, n]$, where $m = |\{y_i | y_i = \max_i y_i\}|$ (i.e. m equals the number of components of $\mathbf{y}$ that are equal to the largest component of $\mathbf{y}$). This follows from the known result (i.e. [7]) that CVaR is strictly increasing on an equivalent interval. Because of this, we see that equation (26) can be rewritten as the unique solution to

$$\min\{k | \mathrm{UA}_k(\mathbf{y}) \leq x\} \ . \tag{27}$$

Substituting equation (2) for $\mathrm{UA}_k(\mathbf{y})$, equation (27) becomes

$$\bar{\eta}_x(\mathbf{y}) = \min \left\{ k \ \middle| \ \min_{\gamma} \gamma + \frac{1}{k} \sum_{i=1}^{n} [y_i - \gamma]^+ \leq x \right\} \ . \tag{28}$$

This can then be simplified as

$$\begin{aligned} \bar{\eta}_x(\mathbf{y}) = \min_{k,\gamma} \quad & k \\ s.t. \quad & \gamma + \frac{1}{k}\sum_{i=1}^{n}[y_i - \gamma]^+ \leq x \ . \end{aligned} \tag{29}$$

Explicitly enforcing the constraint $x-\gamma > 0$, we can further simplify (29) without changing the optimal solution to become

$$\begin{aligned} \bar{\eta}_x(\mathbf{y}) = \min_{k,x-\gamma>0} \quad & k \\ s.t. \quad & \frac{\sum_{i=1}^{n}[y_i - \gamma]^+}{x-\gamma} \leq k \ . \end{aligned} \tag{30}$$

This, though, can be further simplified to become

$$\bar{\eta}_x(\mathbf{y}) = \min_{x-\gamma>0} \quad \frac{\sum_{i=1}^{n}[y_i - \gamma]^+}{x-\gamma} \ . \tag{31}$$

Finally, with the change of variable $a = \frac{1}{x-\gamma}$, (31) can be transformed into

$$\bar{\eta}_x(\mathbf{y}) = \min_{a\geq 0} \quad \sum_{i=1}^{n}[a(y_i - x) + 1]^+ \ . \tag{32}$$

Case 2: Assume $x = \max_j y_j$. With $x = \max_j y_j$, we show that CUA_x equals the number of components equal to $\max_j y_j$. For notational convenience let $\max_j y_j = y_{max}$. Also, assume there are m components of $\mathbf{y}$ that are equal to y_{max}, i.e. $|\{y_i | y_i = y_{max}\}| = m$. Also, let $\hat{y} = \max\{y_i | y_i < y_{max}\}$, i.e. $\hat{y}$ equals the largest component of $\mathbf{y}$ that is less than y_{max}. [2]

Since $y_j - y_{max} \leq 0$ for any $j \in \{1, ...n\}$, we have that for any $a^* \geq \frac{-1}{\hat{y}-y_{max}}$

$$\min_{a\geq 0} \sum_{i=1}^{n}[a(y_i - y_{max}) + 1]^+ \geq \min_{a\geq 0} \sum_{y_i = y_m ax}^{n} [a(y_i - y_{max}) + 1]^+$$

$$= \sum_{i=1}^{n}[a^*(y_i - y_{max}) + 1]^+ = |\{y_i | y_i = y_{max}\}| = m \ ..$$

To see this, notice that for any $y_i \leq \hat{y}$ and any $a^* \geq \frac{-1}{\hat{y}-y_{max}}$ we get

$$[a^*(y_i - y_{max}) + 1]^+ \leq [a^*(\hat{y} - y_{max}) + 1]^+ = [-1 + 1]^+ = 0 \ .$$

Furthermore, for any $y_j = y_{max} = x$, we have that

$$[a(y_i - y_{max}) + 1]^+ = [a(0) + 1]^+ = 1 \ .$$

.

Case 3: Assume $x > \max_i y_i$. We need to show that equation (4) equals the value of k such that $\text{UA}_k(\mathbf{y}) = x$. Since $x > \max_i y_i$ we show that CUA_x equals 0.

[2] If all components of $\mathbf{y}$ are equal, then apply Case 4. In this case, although $\hat{y}$ does not exist, $\text{UA}_n(\mathbf{y}) = \max_j y_j$ and Case 4 can be applied

Let us denote $\max_j y_j = y_{max}$. Since $x > y_{max}$, then for any $i \in \{1, ..., n\}$ we have $y_i - x < 0$. This implies that for any $a^* \geq \frac{-1}{y_{max}-x}$, $a^*(y_i - x) \leq -1$ for all i, therefore,

$$\sum_{i=1}^{n}[a^*(y_i - x) + 1]^+ = 0 = \min_{a\geq 0} \sum_{i=1}^{n}[a(y_i - x) + 1]^+ \ .$$

Case 4: Assume $x \leq \mathrm{UA}_n(\mathbf{y})$. We need to show that equation (4) equals the value of k such that $\mathrm{UA}_k(\mathbf{y}) = x$. Since $x \leq \mathrm{UA}_n(\mathbf{y})$, we show that CUA_x equals n.
Since $x \leq \mathrm{UA}_n(\mathbf{y})$, we have $0 \leq \mathrm{UA}_n(\mathbf{y}) - x$. This implies that for any $a \geq 0$

$$\sum_{i=1}^{n}[a(y_i - x) + 1]^+ \geq \sum_{i=1}^{n}[a(y_i - x) + 1] \geq n[a(\mathrm{UA}_n(\mathbf{y}) - x) + 1] \geq n \ .$$

This result implies that $\min_{a\geq 0} \sum_{i=1}^{n}[a(y_i - x) + 1]^+ = n$. (attained at $a = 0$)
□

B CUA_x: A special case of bPOE

B.1 bPOE and Tail Probabilities

When working with optimization of tail probabilities, one frequently works with constraints or objectives involving *probability of exceedance* (POE), $p_x(X) = P(X > x)$, or its associated quantile $q_\alpha(X) = \min\{x | P(X \leq x) \geq \alpha\}$, where $\alpha \in [0, 1]$ is a probability level. The quantile is a popular measure of tail probabilities in financial engineering, called within this field Value-at-Risk by its interpretation as a measure of tail risk. The quantile, though, when included in optimization problems via constraints or objectives, is quite difficult to treat with continuous (linear or non-linear) optimization techniques.

A significant advancement was made in [7] in the development of an approach to combat the difficulties raised by the use of the quantile function in optimization. Rockafellar and Uryasev explored a replacement for the quantile, called CVaR within the financial literature, and called the superquantile in a general context. The superquantile is a measure of uncertainty similar to the quantile, but with superior mathematical properties. Formally, the superquantile (CVaR) for a continuously distributed X is expressed as

$$\bar{q}_\alpha(X) = E\left[X | X > q_\alpha(X)\right].$$

For general distributions, the superquantile is defined by the following formula,

$$\bar{q}_\alpha(X) = \min_\gamma \left\{\gamma + \frac{E[X - \gamma]^+}{1 - \alpha}\right\} ,$$

where $[\cdot]^+ = \max\{\cdot, 0\}$. Similar to $q_\alpha(X)$, the superquantile can be used to assess the tail of the distribution. The superquantile, though, is far easier to handle in optimization contexts. It also has the important property that it considers the magnitude of events within the tail. Therefore, in situations where a distribution may have a heavy tail, the

superquantile accounts for magnitudes of low-probability large-loss tail events while the quantile does not account for this information.

Working to extend this concept, bPOE was developed as the inverse of the superquantile in the same way that POE is the inverse of the quantile. Specifically, there exists two slightly different variants of bPOE, namely Lower and Upper bPOE. Paper [3] defines so-called Lower bPOE in the following way, where $\sup X$ denotes the essential supremum of the random variable X.

Definition (Lower bPOE): Let X be a real-valued random variable and $x \in \mathbb{R}$ a fixed threshold parameter. Lower bPOE of random variable X at threshold x equals

$$\bar{p}_x^L(X) = \begin{cases} 0, & \text{if } x \geq \sup X \ , \\ \{1 - \alpha | \bar{q}_\alpha(X) = x\}, & \text{if } E[X] < x < \sup X \ , \\ 1, & \text{otherwise.} \end{cases}$$

In words, for any threshold $x \in (E[X], \sup X)$, Lower bPOE can be interpreted as one minus the probability level at which the superquantile equals x.

Similarly, paper [5] defines so-called Upper bPOE as follows.

Definition (Upper bPOE): Upper bPOE of random variable X at threshold x equals

$$\bar{p}_x^U(X) = \begin{cases} \max\{1 - \alpha | \bar{q}_\alpha(X) \geq x\}, & \text{if } x \leq \sup X \ , \\ 0, & \text{otherwise.} \end{cases}$$

Upper and Lower, in fact, do not differ dramatically. This is shown by the following property, proved in [5].

Upper vs. Lower bPOE:

$$\bar{p}_x^U(X) = \begin{cases} \bar{p}_x^L(X), & \text{if } x \neq \sup X \ , \\ P(X = \sup X), & \text{if } x = \sup X. \end{cases}$$

It is important to notice that Upper and Lower bPOE are equivalent when $x \neq \sup X$. The difference between the two definitions arises when threshold $x = \sup X$. In this case, we have that $\bar{p}_x^L(X) = 0$ while $\bar{p}_x^U(X) = P(X = \sup X)$. Thus, for a threshold $x \in (E[X], \sup X)$, both Upper and Lower bPOE of X at x can be interpreted as one minus the probability level at which the superquantile equals x. Roughly speaking, Upper bPOE can be compared with $P(X \geq x)$ while Lower bPOE can be compared with $P(X > x)$. To read further about the differences between Upper and Lower bPOE, see [3].

B.2 CUA_x and Upper bPOE

In [5], an important calculation formula for bPOE was introduced. Specifically, [5] found that Upper bPOE has the following calculation formula.

Proposition: Given a real valued random variable X and a fixed threshold x, bPOE for random variable X at x equals

$$\bar{p}_x^U(X) = \inf_{\gamma<x} \frac{E[X-\gamma]^+}{x-\gamma} = \begin{cases} \lim\limits_{\gamma\to-\infty} \frac{E[X-\gamma]^+}{x-\gamma} = 1 \ , & \text{if } x \leq E[X] \ , \\ \min\limits_{\gamma<x} \frac{E[X-\gamma]^+}{x-\gamma} \ , & \text{if } E[X] < x < \sup X \ , \\ \lim\limits_{\gamma\to x^-} \frac{E[X-\gamma]^+}{x-\gamma} = P(X = \sup X) \ , & \text{if } x = \sup X \ , \\ \min\limits_{\gamma<x} \frac{E[X-\gamma]^+}{x-\gamma} = 0 \ , & \text{if } \sup X < x. \end{cases} \tag{33}$$

With this calculation formula, we can then show that CUA_x is simply a special case of Upper bPOE. First, let us represent our deterministic vector $(y_1, y_2, ..., y_n) = \mathbf{y} \in \mathbb{R}^n$ as a real valued discrete random variable Y taking on values $(y_1, y_2, ..., y_n)$ with equal probabilities, i.e. $P(Y = y_i) = \frac{1}{n}$. Second, let us consider the quantity $n\bar{p}_x(Y)$. Using calculation formula (33) with the change of variable $a = \frac{1}{x-\gamma}$, we see that

$$\begin{aligned} n\bar{p}_x^U(Y) =& \min_{a\geq 0} \ nE[a(Y-x)+1]^+ \\ =& \min_{a\geq 0} \ \sum_{i=1}^{n} [a(y_i - x) + 1]^+ \end{aligned} \tag{34}$$

Thus, we see that this is exactly the definition of CUA_x. In other words, we see that CUA_x is a deterministic variant of bPOE.

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] Norton M., Uryasev S. Maximization of AUC and Buffered AUC in Classification Research Report 2014-2, ISE Dept., University of Florida. (2014)

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

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

[8] Rockafellar, R.T. Safeguarding Strategies in Risky Optimization. Presentation at the International Workshop on Engineering Risk Control and Optimization, Gainesville, FL, February, 2009.

[9] Rockafellar R.T., Royset J.O. On Buffered Failure Probability in Design and Optimization of Structures. Reliability Engineering & System Safety, Vol. 95, 499-510. (2010)

[10] Uryasev, S. Buffered Probability of Exceedance and Buffered Service Level: Definitions and Properties. Research Report 2014-3?, ISE Dept., University of Florida. (2014)

[11] Yager, R. R. On ordered weighted averaging aggregation operators in multicriteria decision making. Systems, Man and Cybernetics, IEEE Transactions on, 18(1), 183-190. (1988)