Computing Monetary Risk Measures in Linear Time

Anonymous authors
Paper under double-blind review

Abstract

Monetary risk measures have gained popularity for expressing decision-makers' risk aversion. Value-at-Risk (VaR) and Conditional-Value-at-Risk (CVaR), in particular, are used commonly for this purpose. This paper proposes new efficient algorithms to compute these risk measures for a discrete random variable in expected linear time with respect to the size of its domain. First, we propose a QuickVaR algorithm that computes the VaR of a discrete random variable. Then, we leverage QuickVaR to propose QuickDivergence, an algorithm for computing a class of φ-divergence risk measures, including the popular CVaR risk measure. The QuickVaR algorithm adapts the well-known Quickselect algorithm, while QuickDivergence builds on polymatroid optimization algorithms. Numerical results show that our new algorithms offer an order-of-magnitude speedup for large domains, and a library implementation of the algorithms is available at [redacted for anonymity].

1 Introduction

Monetary measures of risk generalize the expectation operator to be able to represent the risk aversion of decision-makers in domains that range from robotics (Akella et al., 2024; Benrabah et al., 2024; Majumdar and Pavone, 2020) to finance (Follmer and Schied, 2016) to infrastructure maintenance (Inzunza et al., 2016) to disaster response (Barahona et al., 2013). Value-at-Risk (VaR) and Conditional-Value-at-Risk (CVaR) are the most popular risk measures and have made inroads in reinforcement learning, machine learning, and decision making (Chávez and Li, 2026; Ding and Feinberg, 2023; Godbout and Durand, 2025; Muni et al., 2026; Petrik et al., 2024; Sadana et al., 2026; Xia et al., 2023). Machine learning and reinforcement learning algorithms typically must compute the risk measures quickly for large discrete random variables. For example, in risk-averse reinforcement learning, one must evaluate the risk measure in every iteration of the algorithm, which can represent a significant computational bottleneck (Hau et al., 2023; 2025; Ho et al., 2018; 2021; 2022).

In this paper, we propose linear-time algorithms for computing risk measures of discrete random variables. First, we develop *QuickVaR*, an algorithm that computes VaR in linear time and generalizes the well-known quickselect algorithm. Second, we develop *QuickDivergence*, an algorithm that computes, in linear time, φ-divergence risk measures (Ahmadi-Javid, 2012) that satisfy a specific piecewise linearity property. This algorithm reduces φ-divergence risk measures to linear minimization problems over polymatroids (Follmer and Schied, 2016). We illustrate this algorithm for the well-known CVaR risk measure and a related TVaR risk measure. Both of these risk measures are in the φ-divergence family and satisfy the required linearity property.

Our algorithms' time complexity improves on the state of the art by a logarithmic factor. Virtually all implementations of VaR and CVaR algorithms require sorting the input arrays (Follmer and Schied, 2016). Comparison-based sorting requires $\Omega(n \log n)$ operations where n is the size of the probability space (Korte and Vygen, 2012). Our numerical results show that the logarithmic acceleration can lead to an order-of-magnitude speedup for realistic large problems without incurring any penalty for small problems. In addition, our linear-time algorithms have simple implementations that add minimal complexity over standard algorithms.

We now summarize some of the most relevant fast algorithms for solving problems similar to computing risk measures. Some examples of such structured optimization problems are the projection onto the

simplex (Adam and Mácha, 2019; Wang and Carreira-Perpiñán, 2013), projection onto the k-capped simplex (Ang et al., 2021; Lim and Wright, 2016), linear optimization on the intersection of L_p or φ-divergence balls and the probability simplex in robust optimization (Ho et al., 2022), optimization for φ-divergence risk measures (Ahmadi-Javid, 2012), and on polymatroids (Korte and Vygen, 2012). The preponderance of the literature proposes algorithms that require sorting with few exceptions (Condat, 2016; Lim and Wright, 2016). Finally, QuickVaR is closely related to the computation of the weighted quantile but returns the same value only when the quantile is unique (Cormen et al., 2009, chapter 9).

The remainder of the paper is organized as follows. Section 2 describes the necessary background including risk measures, φ-divergence, and polymatroid optimization. Then, Section 3 proposes *QuickVaR* to compute VaR in linear time. Section 4 builds on QuickVaR to develop *QuickDivergence* that computes certain φ-divergence risk measures in linear time. We evaluate the algorithms numerically on synthetic and realistic problems in Section 5.

2 Preliminaries

Notation: We restrict our attention to random variables defined on finite probability spaces $(\Omega, 2^\Omega, \boldsymbol{p})$ where $\Omega := \{1, \dots, n\}$, $\boldsymbol{p} \in \Delta_n$, and Δ_n is the n-dimensional probability simplex. We use a tilde to indicate a random variable, such as $\tilde{x} \colon \Omega \to \mathbb{R}$. Vectors are denoted with a lower-case bold font, such as $\boldsymbol{x} \in \mathbb{R}^n$ and indexed as $\boldsymbol{x}_i$ for $i = 1, \dots, n$. We use the function notation $\tilde{x}$ and vector notation $\boldsymbol{x}$ interchangeably, such that $x_\omega = x(\omega), \forall\, \omega \in \Omega$. Sets are denoted with calligraphic letters, and $\bar{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$ denotes the extended real line, and $\mathbb{R}_{\geq 0}$ are non-negative reals. The symbol $\mathbb{N}$ denotes all 0-based natural numbers and $a{:}b := a, a+1, \dots, b$ for $a \leq b \in \mathbb{N}$ and the empty set otherwise. We use $\boldsymbol{x}_{a:b}$ to denote a subset of elements of $\boldsymbol{x}$ and comparisons such as $\boldsymbol{x}_{a:b} < y$ are interpreted element-wise. Finally, the symbol $\mathbf{1} \in \mathbb{R}^n$ denotes a vector of all ones and $\mathbf{1}_{\mathcal{O}} \in \mathbb{R}^n$ for $\mathcal{O} \subseteq \Omega$ is an indicator vector of the set $\mathcal{O}$.

Monetary risk measures $\rho \colon \Delta_n \times \mathbb{R}^n \to \bar{\mathbb{R}}$ are functions of a probability measure and a random variable that are monotone and translation invariant (Follmer and Schied, 2016). Risk measures generalize the expectation operator to account for uncertainty in a random variable's distribution. We apply risk measures to random variables representing rewards and suppose that higher risk-adjusted values are preferable. For the sake of streamlining the notation, we omit the probability measure when it is obvious from the context and only write $\rho[\tilde{x}]$ to compute the risk value.

Value-at-Risk (VaR) is a popular risk measure and is defined for $\alpha \in [0, 1)$ and $\tilde{x} \in \mathbb{R}^n$ as (Follmer and Schied, 2016; Shapiro et al., 2014)

$$\mathrm{VaR}_\alpha\left[\tilde{x}\right] \;:=\; \max\, \left\{\tau \in \bar{\mathbb{R}} \mid \mathbb{P}\left[\tilde{x} < \tau\right] \leq \alpha\right\} = \max\, \left\{\tau \in \bar{\mathbb{R}} \mid \mathbb{P}\left[\tilde{x} \geq \tau\right] \geq 1 - \alpha\right\}, \tag{1}$$

where the maximum exists because $\tau \mapsto \mathbb{P}\left[\tilde{x} < \tau\right]$ is lower semi-continuous. One can readily see that $\mathrm{VaR}_0[\tilde{x}] = \operatorname{ess\,inf}[\tilde{x}]$, and typically the risk measure is extended as $\mathrm{VaR}_1[\tilde{x}] = \infty$.

A *φ-divergence risk measure* $\mathrm{DR}_{\varphi,\beta} \colon \mathbb{R}^n \to \bar{\mathbb{R}}$ is defined for $\varphi \colon \bar{\mathbb{R}} \to \bar{\mathbb{R}}$ and $\beta \in \mathbb{R}_{\geq 0}$ as (Ahmadi-Javid, 2012)

$$\mathrm{DR}_{\varphi,\beta}[\tilde{x}] := \inf\left\{\boldsymbol{q}^\mathsf{T}\boldsymbol{x} \mid \boldsymbol{q} \in \Delta_n, H_\varphi(\boldsymbol{q}\|\boldsymbol{p}) \leq \beta, \boldsymbol{q} \ll \boldsymbol{p}\right\} \tag{2}$$

$$= \sup_{z \in \mathbb{R}} \left(z - \inf_{t > 0} t \cdot \mathbb{E}\left[\varphi^\star\left(\frac{z - \tilde{x}}{t} + \beta\right)\right]\right), \tag{3}$$

where $\varphi^\star(x)$ is the convex conjugate of $\varphi(y)$, φ is a closed, convex function that satisfies $\varphi(1) = 0$, and H_φ is a φ-divergence measure (Bayraksan and Love, 2015):

$$H_\varphi(\boldsymbol{q}\|\boldsymbol{p}) := \sum_{\omega \in \Omega} p_\omega \cdot \varphi\left(\frac{q_\omega}{p_\omega}\right),$$

defined for $\boldsymbol{q} \ll \boldsymbol{p}$. Here, $\boldsymbol{q} \ll \boldsymbol{p}$ denotes that $\boldsymbol{q}$ is absolutely continuous with respect to $\boldsymbol{p}$, meaning that $p_\omega = 0 \implies q_\omega = 0$ for each $\omega \in 1{:}n$. We also adopt the common convention that $1/0 = \infty$ and $0 \cdot \infty = 0$ (Rockafellar and Wets, 1996). It is important to note that φ-divergence risk measures are

Table 1: Parameters for relevant φ-divergence risk measures for $\alpha \in (0,1)$.

Risk measure	$\varphi(y)$	$\varphi^\star(x)$	β
$\mathbb{E}$	$\begin{cases} 0 & \text{if } y = 1 \\ \infty & \text{otherwise} \end{cases}$	x	0
CVaR_α	$\begin{cases} 0 & \text{if } y \in [0, \frac{1}{\alpha}] \\ \infty & \text{otherwise} \end{cases}$	$\frac{1}{\alpha} \cdot \max\{0, x\}$	0
EVaR_α	$\begin{cases} y \log y & \text{if } y > 0 \\ 0 & \text{if } y = 0 \\ \infty & \text{if } y < 0 \end{cases}$	$\exp(x-1)$	$\log \frac{1}{\alpha}$
TVaR_α	$\lvert y - 1 \rvert$	$\begin{cases} x & \text{if } \lvert x \rvert \le 1 \\ \infty & \text{otherwise} \end{cases}$	$\min\left\{\sqrt{2 \log \frac{1}{\alpha}}, 2\right\}$

coherent, which means they are concave and positively homogeneous. Any coherent risk measure $\rho\colon \mathbb{R}^n \to \bar{\mathbb{R}}$ satisfies that $\rho[\tilde{x}] = \inf_{\boldsymbol{q} \in \mathcal{Q}} \boldsymbol{q}^\mathsf{T} \boldsymbol{x}$ for some closed convex non-empty set $\mathcal{Q} \subseteq \Delta_n$ (Follmer and Schied, 2016, proposition 2.84).

Table 1 summarizes several common φ-divergence risk measures; see Section 4.2 for the relevant definitions for $\alpha \in [0,1]$. We focus on the *Conditional-Value-at-Risk* (CVaR), which is a φ-divergence risk measure and defined equivalently for $\tilde{x} \in \mathbb{R}^n$ and $\alpha \in (0,1]$ as (Follmer and Schied, 2016)

$$\mathrm{CVaR}_\alpha[\tilde{x}] \;:=\; \max_{z \in \mathbb{R}} \left(z - \frac{1}{\alpha} \cdot \mathbb{E}\left[\, z - \tilde{x} \,\right]_+ \right) = \min\left\{ \boldsymbol{x}^\mathsf{T} \boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\, \boldsymbol{q} \le \alpha^{-1} \boldsymbol{p} \right\}, \tag{4}$$

and extended continuously to $\mathrm{CVaR}_0[\tilde{x}] = \operatorname{ess\,inf}[\tilde{x}]$.

We also study another φ-divergence risk measure, which we call *Total-Value-at-Risk* (TVaR) and which is defined for $\alpha \in (0,1]$ as

$$\mathrm{TVaR}_\alpha[\tilde{x}] \;:=\; \min\left\{ \boldsymbol{x}^\mathsf{T} \boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\, \|\boldsymbol{p} - \boldsymbol{q}\|_1 \le \min\left\{ \sqrt{2 \log \frac{1}{\alpha}}, 2 \right\}, \boldsymbol{q} \ll \boldsymbol{p} \right\}, \tag{5}$$

and extended continuously to $\mathrm{TVaR}_0[\tilde{x}] = \operatorname{ess\,inf}[\tilde{x}]$.

We also refer to the *Entropic-Value-at-Risk* (EVaR), which is the φ-divergence risk measure obtained from $\varphi(y) := y \log y$ for $y > 0$ and $\beta = \log \frac{1}{\alpha}$ for $\alpha > 0$ in Table 1. Because the corresponding φ-divergence is the Kullback–Leibler divergence $H_\varphi(\boldsymbol{q} \| \boldsymbol{p}) = \mathrm{KL}(\boldsymbol{q} \| \boldsymbol{p}) = \sum_{\omega \in \Omega} q_\omega \log \frac{q_\omega}{p_\omega}$, the EVaR is equivalently defined for $\alpha \in (0,1]$ and $\tilde{x} \in \mathbb{R}^n$ as (Ahmadi-Javid, 2012)

$$\mathrm{EVaR}_\alpha[\tilde{x}] \;:=\; \min\left\{ \boldsymbol{x}^\mathsf{T} \boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\, \mathrm{KL}(\boldsymbol{q} \| \boldsymbol{p}) \le \log \frac{1}{\alpha}, \boldsymbol{q} \ll \boldsymbol{p} \right\}, \tag{6}$$

and extended continuously to $\mathrm{EVaR}_0[\tilde{x}] = \operatorname{ess\,inf}[\tilde{x}]$.

An important property of TVaR is that it bounds EVaR (Ahmadi-Javid, 2012; Hau et al., 2023; Su et al., 2025) from below, as we show next and illustrate in Figure 1.

Proposition 2.1. *For each $\tilde{x} \in \mathbb{R}^n$, the risk measures above extended to $\alpha \in [0,1]$ satisfy that*

$$\mathrm{VaR}_\alpha[\tilde{x}] \;\ge\; \mathrm{CVaR}_\alpha[\tilde{x}] \;\ge\; \mathrm{EVaR}_\alpha[\tilde{x}] \;\ge\; \mathrm{TVaR}_\alpha[\tilde{x}].$$

Proof. The inequality for VaR and CVaR is well known, and the inequality for CVaR and EVaR follows from (Ahmadi-Javid, 2012, proposition 3.2). Recall that Pinsker's inequality (Boucheron et al., 2013, theorem 4.19) shows for $\boldsymbol{p}, \boldsymbol{q} \in \Delta_n$ that $\|\boldsymbol{p} - \boldsymbol{q}\|_1 \le \sqrt{2\,\mathrm{KL}(\boldsymbol{q} \| \boldsymbol{p})}$. Then we prove that TVaR lower-bounds EVaR

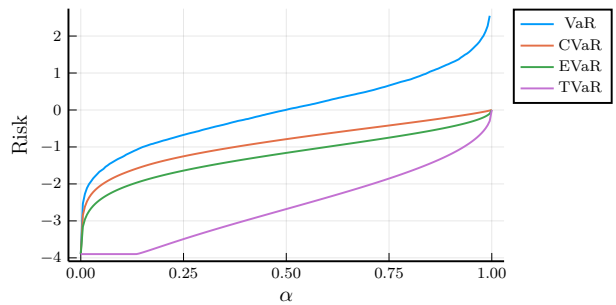

Figure 1: Comparison of risk measure values for a normal distribution with mean 0 and standard deviation 1 as a function of the risk level α.

as

$$\begin{aligned}
\mathrm{EVaR}_\alpha[\tilde{x}] &= \inf\left\{\boldsymbol{q}^\top \boldsymbol{x} \mid \boldsymbol{q} \in \Delta_n, \mathrm{KL}(\boldsymbol{q}\|\boldsymbol{p}) \le \log\frac{1}{\alpha}, \boldsymbol{q} \ll \boldsymbol{p}\right\} \\
&\ge \inf\left\{\boldsymbol{q}^\top \boldsymbol{x} \mid \boldsymbol{q} \in \Delta_n, \|\boldsymbol{p}-\boldsymbol{q}\|_1 \le \sqrt{2\log\frac{1}{\alpha}}, \boldsymbol{q} \ll \boldsymbol{p}\right\} \\
&= \inf\left\{\boldsymbol{q}^\top \boldsymbol{x} \mid \boldsymbol{q} \in \Delta_n, \|\boldsymbol{p}-\boldsymbol{q}\|_1 \le \min\left\{\sqrt{2\log\frac{1}{\alpha}}, 2\right\}, \boldsymbol{q} \ll \boldsymbol{p}\right\} = \mathrm{TVaR}_\alpha[\tilde{x}].
\end{aligned}$$

We use the fact that $\|\boldsymbol{p}-\boldsymbol{q}\|_1 \le 2$. □

3 Computing VaR in Linear Time

We now describe the QuickVaR algorithm, show that it computes VaR, and show that it runs in linear expected time.

The standard VaR algorithm for a discrete random variable $\boldsymbol{x} \in \mathbb{R}^n$ with distribution $\boldsymbol{p} \in \Delta_n$ and risk level α proceeds as follows (see listing 6.25 (Privault, 2026)). It sorts the entries of $\boldsymbol{x}$ to get a permutation $\sigma\colon 1{:}n \to 1{:}n$ such that $i \mapsto x_{\sigma(i)}$ is monotone. Then, it finds an index $i \in 1{:}n$ such that

$$\sum_{j=1}^{i-1} p_{\sigma(j)} \;<\; \alpha \;\le\; \sum_{j=1}^{i} p_{\sigma(j)}.$$

The value $x_{\sigma(i)}$ is the optimal solution in (1) by Lemma 3.2 below. The computational complexity of this algorithm is $\Omega(n \log n)$ because its runtime is dominated by the need to sort the entries of $\boldsymbol{x}$. For the sake of completeness, Algorithm A.1 in the appendix summarizes the standard greedy approach to computing VaR.

QuickVaR, described in Algorithm 3.1, computes VaR in $O(n)$ time by avoiding the need to sort the elements of the input random variable $\boldsymbol{x}$. In particular, QuickVaR partially sorts the indices of $\boldsymbol{x}$, analogous to the quickselect algorithm (Horowitz et al., 1998).

Algorithm 3.1 is recursive and applies to a random variable $\boldsymbol{x}$ with a distribution $\boldsymbol{p} \in \Delta_n$ as follows. In addition to the random variable, the input is f (front) and b (back), which delimit the indices of $\boldsymbol{x}$ it manipulates. The algorithm first picks a pivot element v uniformly at random from $f{:}b$. The random pivot selection is essential to achieve an average linear time complexity. Given a pivot, we use partition to partially sort the entries of $\boldsymbol{x}$ into three groups: 1. less than v: $\boldsymbol{x}_{f:(l-1)} < v$, 2. equal to v: $\boldsymbol{x}_{l:g} = v$, and 3. greater than v: $\boldsymbol{x}_{(g+1):b} > v$. Then, the algorithm checks for the conditions of Lemma 3.2 to determine which of the three groups contains the VaR value. Finally, if the VaR value is in group 1 or 3, the algorithm calls itself recursively with adjusted f, b, and α.

The partition procedure in Algorithm 3.1 uses the Dutch national flag algorithm to partition the input array into the three groups. The algorithm runs in linear time, and the properties of its output are given in Lemma A.3 below. For the sake of completeness, we provide the reader with the algorithm in Algorithm A.2 in Appendix A.

We present Algorithm 3.1 recursively to simplify its analysis. However, a loop-based, non-recursive implementation, such as Algorithm A.3 in Appendix A, typically performs better in imperative languages.

Algorithm 3.1: `QuickVaR`: Recursive VaR algorithm

Input : $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{p} \in \Delta_n$, $\alpha \in [0,1)$, $f, b \in 1{:}n$ such that $f \leq b$ and $\sum_{j=f}^{b} p_j > \alpha$
Output: $\mathrm{VaR}_\alpha[\tilde{x}]$

$v \leftarrow x_{\mathrm{rand}(f:b)}$; `// Pivot`
$l, g, \boldsymbol{x}', \boldsymbol{p}' \leftarrow \mathrm{partition}(\boldsymbol{x}, \boldsymbol{p}, v, f, b)$; `// Algorithm A.2`
$t \leftarrow \sum_{j=f}^{l-1} p'_j$; `//` $t = \mathbb{P}[\tilde{x} < v] - \sum_{i=1}^{f-1} p_i$
$e \leftarrow \sum_{j=l}^{g} p'_j$; `//` $e = \mathbb{P}[\tilde{x} = v]$
if $\alpha < t$ **then** `//` $\alpha + \sum_{i=1}^{f-1} p_i < \mathbb{P}[\tilde{x} < v]$
  return $\mathrm{QuickVaR}(\boldsymbol{x}', \boldsymbol{p}', \alpha, f, l-1)$
else if $t + e \leq \alpha$ **then** `//` $\mathbb{P}[\tilde{x} \leq v] \leq \alpha + \sum_{i=1}^{f-1} p_i$
  return $\mathrm{QuickVaR}(\boldsymbol{x}', \boldsymbol{p}', \alpha - t - e, g+1, b)$
else return v ;

The following theorem states the correctness of QuickVaR.

Theorem 3.1. *Algorithm 3.1 executed with $f = 1$ and $b = n$ and $\alpha \in [0,1)$ returns $\mathrm{VaR}_\alpha[\tilde{x}]$.*

Before proving Theorem 3.1, we need to show several important lemmas. The following lemma, which holds only for discrete random variables, shows that the VaR value is the probability atom that satisfies specific probability properties.

Lemma 3.2. *Assuming $\alpha \in [0,1)$, there exists $\omega \in \Omega$ with $v := \mathrm{VaR}_\alpha[\tilde{x}] = x_\omega$ and*

$$\mathbb{P}[\tilde{x} < v] \leq \alpha < \mathbb{P}[\tilde{x} \leq v]. \tag{7}$$

Proof. First, we show that any v that satisfies (7) solves (1) optimally. The implication $\mathbb{P}[\tilde{x} < v] \leq \alpha \implies v \in \{t \in \mathbb{R} \mid \mathbb{P}[\tilde{x} < t] \leq \alpha\}$ follows immediately. Second, we prove $\alpha < \mathbb{P}[\tilde{x} \leq v] \implies \forall t \in \{t \in \mathbb{R} \mid \mathbb{P}[\tilde{x} < t] \leq \alpha\}, v \geq t$ by contradiction. Suppose that $\exists t \in \{t \in \mathbb{R} \mid \mathbb{P}[\tilde{x} < t] \leq \alpha\}, v < t$. Then, by the monotonicity of the probability operator, we get a contradiction as

$$\alpha < \mathbb{P}[\tilde{x} \leq v] \leq \mathbb{P}[\tilde{x} < t] \leq \alpha.$$

Then, the existence of ω follows from Lemma A.1 because (7) implies that

$$\mathbb{P}[\tilde{x} = \mathrm{VaR}_\alpha[\tilde{x}]] = \mathbb{P}[\tilde{x} \leq \mathrm{VaR}_\alpha[\tilde{x}]] - \mathbb{P}[\tilde{x} < \mathrm{VaR}_\alpha[\tilde{x}]] > \alpha - \alpha = 0.$$

□

The following lemma shows the recursive property of QuickVaR and is the main building block of Theorem 3.1.

Lemma 3.3. *Suppose that the input conditions of Algorithm 3.1 hold and $\boldsymbol{x} \in \mathbb{R}^n$ satisfies that*

$$\boldsymbol{x}_{1:(f-1)} < x_f, \qquad x_{f-1} < \boldsymbol{x}_{f:b} < x_{b+1}, \qquad x_b < \boldsymbol{x}_{(b+1):n}, \tag{8}$$

where $x_0 = -\infty$ and $x_{n+1} = +\infty$. Then Algorithm 3.1 returns $x_\omega, \omega \in f{:}b$ such that

$$\mathbb{P}[\tilde{x} < x_\omega] \quad \leq \quad \alpha + \sum_{i=1}^{f-1} p_i \quad < \quad \mathbb{P}[\tilde{x} \leq x_\omega]. \tag{9}$$

Proof. From (8) and Lemma A.3, t and e on lines 2 and 3 satisfy that

$$t = \mathbb{P}[\tilde{x} < v] - \sum_{i=1}^{f-1} p_i, \qquad e = \mathbb{P}[\tilde{x} = v]. \tag{10}$$

We proceed by induction on $k = b - f$. *Base case*: $k = 0$. Since $b = f$, we have that $v = x_b$ and $l = g = b$, $t = 0$, $e = p_b$, and $\alpha \geq t$. The input condition implies that $t + e = p_b > \alpha$. Because $t \leq \alpha < p_b$, the algorithm returns v on line 8 that satisfies

$$\mathbb{P}[\tilde{x} < v] \leq \alpha + \sum_{i=1}^{f-1} p_i < \mathbb{P}[\tilde{x} \leq v].$$

Above, the first inequality follows from $v = x_f$ and $0 \leq \alpha$. The second inequality follows from the algorithm's input condition.

Inductive step. Suppose that $k > 0$ and that the result holds for each $k' < k$. Then:

(i) If $\alpha < t$, the algorithm executes the first if statement on line 4. Because $0 \leq \alpha < t$, the recursive call satisfies the required input conditions: $f \leq l - 1$ and $\sum_{j=f}^{l-1} p_j = t > \alpha$. The recursive call also satisfies (8) by Lemma A.3, and we can conclude that (9) follows from the inductive hypothesis because $l - 1 - f \leq b - 1 - f \leq k - 1 < k$.

(ii) If $t + e \leq \alpha$, the algorithm executes the second if statement on line 6. We show that the input conditions for the recursive call are met. Because $t + e = \sum_{j=f}^{g} p_j \leq \alpha < \sum_{j=f}^{b} p_j$ (the second inequality is the input condition), it must be that $g < b$ and therefore $g + 1 \leq b$. Also, by the algorithm's input condition and algebraic manipulation, we get that

$$\sum_{j=g+1}^{b} p_j = \sum_{j=f}^{b} p_j - t - e > \alpha - t - e,$$

and the input conditions hold. Because (8) follows from Lemma A.3 and $b - g - 1 < b - f \leq k$, the inductive hypothesis implies that there exists x_ω such that

$$\mathbb{P}[\tilde{x} < x_\omega] \quad \leq \quad (\alpha - t - e) + \mathbb{P}[\tilde{x} \leq v] \quad < \quad \mathbb{P}[\tilde{x} \leq x_\omega].$$

Then, (9) follows by algebraic manipulation after substituting t, e from (10).

(iii) If $t \leq \alpha < t + e$, the algorithm executes the else statement on line 8, and the inequality (9) holds by substituting t, e from (10). □

We are now ready to prove the correctness of QuickVaR.

Proof of Theorem 3.1. With $f = 1$ and $b = n$, the condition in (8) holds vacuously, and the input condition $\sum_{j=1}^{n} p_j = 1 > \alpha$ holds since $\alpha \in [0, 1)$. Therefore, Lemma 3.3 shows that QuickVaR returns x_ω that satisfies

$$\mathbb{P}[\tilde{x} < x_\omega] \quad \leq \quad \alpha \quad < \quad \mathbb{P}[\tilde{x} \leq x_\omega],$$

which satisfies (7) and, therefore, it equals the VaR value by Lemma 3.2. □

With the correctness of Algorithm 3.1 established in Lemma 3.3, we now show its computational complexity.

Theorem 3.4. *Algorithm 3.1 runs in expected $O(n)$ time.*

Proof. The proof is analogous to the standard quickselect algorithm, as in (Horowitz et al., 1998, theorem 3.3), except that we need to keep track of the risk level α. Fix some $n \in \mathbb{N}$, $\boldsymbol{x}$, and $\boldsymbol{p}$, and we study the algorithm's properties. Let $T\colon 1{:}n \to [0, \infty]$ be the *expected* runtime of Algorithm 3.1 such that $T(k)$ is the

expected runtime when executed with $k := b - f + 1$. To simplify the notation, we assume without loss of generality in the remainder of the proof that $f = 1$ and $b = k$. Then, let

$$l := |\{i \in 1{:}k \mid x_i < \mathrm{VaR}_\alpha[\tilde{x}]\}|\,.$$

Note that by the assumption $\sum_{j=f}^{b} p_j > \alpha$, we have that $k \geq 1$ and $\mathrm{VaR}_\alpha[\tilde{x}] \leq \max_{i \in f:b} x_i$. Therefore, $l \in 0{:}(k-1)$. The algorithm terminates in constant time when $k = 1$, and we upper-bound the runtime for the worst-case choice of l for $k > 1$ as

$$\begin{aligned} T(k) &\leq d \cdot k + \max_{l \in 0:(k-1)} \frac{1}{k} \sum_{i=0}^{k-1} \left(T(i) \cdot 1_{l<i} + T(k-i-1) \cdot 1_{l>i}\right) \\ &= d \cdot k + \max_{l \in 0:(k-1)} \frac{1}{k} \left(\sum_{i=l+1}^{k-1} T(i) + \sum_{i=0}^{l-1} T(k-i-1) \right) \\ &= d \cdot k + \max_{l \in 0:(k-1)} \frac{1}{k} \left(\sum_{i=l+1}^{k-1} T(i) + \sum_{i=k-l}^{k-1} T(i) \right), \end{aligned} \tag{11}$$

where $d \cdot k$ is the local runtime of the algorithm for some $d > 0$ and $T(i)$ and $T(k-i-1)$ are the average runtimes of the recursive calls. The average over i corresponds to the uniformly random selection of the pivot with $i = |\{j \in 1{:}k \mid x_j < v\}|$ in the first line of the inequality. The last two equalities follow by algebraic manipulation. Then, an argument identical to theorem 3.3 (Horowitz et al., 1998), since (11) is the same as equation (3.8) in Horowitz et al. (1998), shows that $T(k) \leq 4 \cdot d \cdot k$ for all $k \geq 1$. □

Remark 3.5. An attentive reader may notice that QuickVaR resembles the algorithm for computing the weighted median (Korte and Vygen, 2012, section 17). However, the weighted median algorithm does not, in general, compute VaR. Define the *quantile* $\mathfrak{q}_\alpha, \alpha \in (0,1)$ for $\tilde{x}$ as the closed set

$$\mathfrak{q}_\alpha[\tilde{x}] := \{z \in \mathbb{R} \mid \mathbb{P}[\tilde{x} \leq z] \geq \alpha, \mathbb{P}[\tilde{x} \geq z] \geq 1 - \alpha\}\,,$$

with bounds $\mathfrak{q}_\alpha^+[\tilde{x}] := \max \mathfrak{q}_\alpha[\tilde{x}]$ and $\mathfrak{q}_\alpha^-[\tilde{x}] := \min \mathfrak{q}_\alpha[\tilde{x}]$. One can readily show that $\mathrm{VaR}_\alpha[\tilde{x}] = \mathfrak{q}_\alpha^+[\tilde{x}]$. Although the precise definition of a weighted median varies, it is typically a value in $\mathfrak{q}_{\frac{1}{2}}[\tilde{x}]$. Because $\mathfrak{q}_\alpha^+[\tilde{x}] \neq \mathfrak{q}_\alpha^-[\tilde{x}]$, in general, VaR also does not equal the median. A careful analysis shows that the weighted median algorithm in Korte and Vygen (2012, section 17) computes $\mathfrak{q}_{\frac{1}{2}}^-[\tilde{x}]$, which does not equal $\mathrm{VaR}_{\frac{1}{2}}[\tilde{x}]$.

4 Computing φ-divergence Risk Measures in Linear Time

In this section, we describe and analyze QuickDivergence, a new linear-time framework for computing a class of φ-divergence risk measures, including CVaR and TVaR. QuickDivergence treats risk evaluation as a linear optimization over a polymatroid. Standard polymatroid optimization algorithms must sort their input, which requires $\Omega(n \log n)$ runtime. We identify a structural condition of polymatroid risk measures that allows us to avoid sorting and instead use the linear-time QuickVaR algorithm described in Section 3.

The remainder of the section is organized as follows. Section 4.1 describes a general linear-time algorithm suitable for φ-divergence risk measures that satisfy an EWS property. Then Section 4.2 shows that both CVaR and TVaR satisfy the EWS property and can be computed in linear time. Note that we use a fixed distribution $\boldsymbol{p} \in \Delta_n$ and omit it in the notation when it is obvious from the context.

4.1 QuickDivergence

We first recall the definition of a submodular function (Korte and Vygen, 2012, section 2.1) and a polymatroid (Korte and Vygen, 2012, definition 14.9).

Definition 4.1. A function $f\colon 2^{1:n} \to \mathbb{R}_{\geq 0}$ is *submodular* if

$$f(\mathcal{A}) + f(\mathcal{B}) \;\geq\; f(\mathcal{A} \cup \mathcal{B}) + f(\mathcal{A} \cap \mathcal{B}), \quad \forall \mathcal{A}, \mathcal{B} \subseteq 1{:}n.$$

For a submodular *capacity* function $f\colon 2^{1:n} \to \mathbb{R}_{\geq 0}$, a *polymatroid* $\mathcal{Q}_f$ is a polytope defined as

$$\mathcal{Q}_f \;:=\; \left\{\boldsymbol{q} \in \mathbb{R}^n_{\geq 0} \mid \mathbf{1}^\mathsf{T}\boldsymbol{q}_\mathcal{U} \leq f(\mathcal{U}),\, \forall \mathcal{U} \subseteq 1{:}n\right\},$$

and a *normalized polymatroid* is $\widehat{\mathcal{Q}}_f := \mathcal{Q}_f \cap \Delta_n$.

Polymatroids are useful in optimization because any coherent, comonotonic monetary risk measure can be formulated as linear optimization over a polymatroid (Follmer and Schied, 2016, corollary 4.89). A risk measure ρ is *coherent* if it is concave and positively homogeneous. A risk measure ρ is *comonotonic* if for any comonotone $\tilde{x}, \tilde{y}$: $\rho[\tilde{x} + \tilde{y}] = \rho[\tilde{x}] + \rho[\tilde{y}]$, where the random variables $\tilde{x}, \tilde{y} \in \mathbb{R}^n$ are *comonotone* if $\tilde{x}(i) \geq \tilde{x}(j) \Leftrightarrow \tilde{y}(i) \geq \tilde{y}(j)$ for each $i, j \in 1{:}n$.

Proposition 4.2. *A monetary risk measure $\rho\colon \Delta_n \times \mathbb{R}^n \to \mathbb{R}$ is a coherent comonotonic risk measure if and only if*

$$\rho[\tilde{x}] = \min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_c} \boldsymbol{q}^\mathsf{T}\boldsymbol{x}, \qquad \textit{where} \qquad c(\mathcal{A}) := -\rho[-\tilde{1}_\mathcal{A}], \quad \forall \mathcal{A} \subseteq 1{:}n, \tag{12}$$

where $\tilde{1}$ is an indicator random variable. Moreover, if the risk measure is law-invariant, then there exists $g\colon \mathbb{R} \to \mathbb{R}$ such that $c(\mathcal{A}) = g(\mathbf{1}^\mathsf{T}\boldsymbol{p}_\mathcal{A})$ for each $\mathcal{A} \subseteq 1{:}n$.

Proposition 4.2 follows from corollary 4.88 and theorem 4.94 in Follmer and Schied (2016).

The standard solution to the polymatroid optimization problem in (12) is to use a greedy algorithm, such as Algorithm 4.1. This algorithm adapts the standard polymatroid optimization, such as Korte and Vygen (2012, section 14.9), to the normalized polymatroid $\widehat{\mathcal{Q}}_f$. See Proposition B.1 for the proof of correctness. Note that the greedy approach in Algorithm 4.1 requires sorting and, therefore, runs in $\Omega(n \log n)$ time.

Algorithm 4.1: Linear minimization over polymatroids.

Input : submodular monotone function $f\colon 2^{1:n} \to \mathbb{R}_{\geq 0}$, and an objective $\boldsymbol{x} \in \mathbb{R}^n$
Output: $\hat{\boldsymbol{q}} \in \arg\min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} \boldsymbol{x}^\mathsf{T}\boldsymbol{q}$

```
Use sort to get σ: 1:n → 1:n such that i > j ⟹ x_σ(i) ≥ x_σ(j), ∀i, j ∈ 1:n;
for k ∈ 1:n do
    q̂_σ(k) ← f(σ(1:k)) − f(σ(1:(k − 1)))
return q̂
```

We now define a special class of submodular functions, called EWS, that makes sorting unnecessary when optimizing a linear function over a polymatroid.

Definition 4.3. A function $f\colon 2^{1:n} \to \mathbb{R}_{\geq 0}$ is EWS$(\boldsymbol{\mu}, m, c)$, or *Element-Wise-Separable*, for $c \in \mathbb{R}_{\geq 0}$, $m \in \mathbb{R}_{>0}$, such that $c \leq 1$, $c + m \geq 1$, and $\boldsymbol{\mu} \in \Delta_n$ if $f(\mathcal{A}) := g(\mathbf{1}^\mathsf{T}\boldsymbol{\mu}_\mathcal{A})$ for all $\mathcal{A} \subseteq 1{:}n$ and

$$g(x) := \begin{cases} \min\{c + m \cdot x, 1\} & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

EWS functions satisfy the properties necessary to define a polymatroid as we show next.

Proposition 4.4. *An EWS function $f\colon 2^\Omega \to \mathbb{R}_{\geq 0}$ is submodular, monotone, and satisfies that $f(\emptyset) = 0$, and $f(\Omega) = 1$.*

Proof. To show that f is submodular, it is sufficient, from Lemma B.2, to show that g is concave on $[0, 1]$. The function g is concave on $(0, 1]$ because it is the minimum of two linear functions (Boyd and Vandenberghe, 2004). The concavity of g on $[0, 1]$ follows from the definition. Since $g(0) = 0$, for each $\lambda \in (0, 1]$ and $b \in (0, 1]$ we get that

$$(1 - \lambda)g(0) + \lambda \cdot g(b) = \lambda \cdot g(b) = \min\{\lambda \cdot c + m \cdot \lambda \cdot b, \lambda\} \leq \min\{c + m \cdot \lambda \cdot b, 1\} = g\left((1 - \lambda) \cdot 0 + \lambda \cdot b\right).$$

Because the inequality also holds for $\lambda = 0$, g is concave on $[0, 1]$ by the definition. The monotonicity, non-negativity and the boundary value conditions for f follow by algebraic manipulation from the assumptions on c and m. □

Algorithm 4.2: `QuickDivergence`: Linear minimization over EWS polymatroids.

Input : $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{p} \in \Delta_n$, $\mathrm{EWS}(\boldsymbol{p}, m, c)$ function $f\colon 2^{1:n} \to \mathbb{R}_{\geq 0}$
Output: $\boldsymbol{q}^\star \in \arg\min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} \boldsymbol{x}^\mathsf{T} \boldsymbol{q}$

Choose $k_{\min} \in \arg\min_{i \in 1:n} x_i$ s. t. $p_i > 0$; break ties to choose the smallest i `// Analogous to` $\operatorname{ess\,inf}$

$v \leftarrow \begin{cases} \mathrm{QuickVaR}(\boldsymbol{x}, \boldsymbol{p}, (1-c)/m), & \text{if } (1-c)/m < 1, \\ \infty, & \text{otherwise} \end{cases}$;

$\mathcal{K}_{<v} \leftarrow \{i \in 1{:}n \mid x_i < v\}$; $\mathcal{K}_{=v} \leftarrow \{i \in 1{:}n \mid x_i = v\}$;

$l \leftarrow |\mathcal{K}_{<v} \cup \{k_{\min}\}|$;

$\theta_l \leftarrow 1 - \min\left\{c + m \cdot \mathbf{1}^\mathsf{T} \boldsymbol{p}_{\mathcal{K}_{<v} \cup \{k_{\min}\}}, 1\right\}$;

for $i \in 1{:}n$ **do**

  if $i = k_{\min}$ **then** $q_i^\star \leftarrow \min\{m \cdot p_i + c, 1\}$;

  else if $i \in \mathcal{K}_{<v}$ **then** $q_i^\star \leftarrow m \cdot p_i$;

  else if $i \in \mathcal{K}_{=v}$ **then** $q_i^\star \leftarrow \min\{\theta_l, m \cdot p_i\}$; $\theta_{l+1} \leftarrow \theta_l - q_i^\star$; $l \leftarrow l + 1$;

  else $q_i^\star \leftarrow 0$;

return $\boldsymbol{q}^\star$

Algorithm 4.2 describes *QuickDivergence*, an algorithm for polymatroid optimization. QuickDivergence differs from the standard linear polymatroid algorithm in Algorithm 4.1 in that it does not require sorting the input vector. Instead, the algorithm uses QuickVaR to partially sort the input vector.

The following theorem states the correctness and runtime of Algorithm 4.2.

Theorem 4.5. *Algorithm 4.2 solves* (12) *in expected* $O(n)$ *runtime.*

Proof. To prove the correctness of Algorithm 4.2, we show that it returns $\boldsymbol{q}^\star$ that equals $\hat{\boldsymbol{q}}$ returned by the standard Algorithm 4.1. Suppose that Algorithm 4.1 chooses a permutation $\sigma\colon 1{:}n \to 1{:}n$ such that $i \leq j \Rightarrow x_{\sigma(i)} \leq x_{\sigma(j)}$ and $x_{\sigma(i)} = x_{\sigma(j)} \wedge i < j \Rightarrow \sigma(i) < \sigma(j)$ for each $i, j \in 1{:}n$. In other words, the permutation σ needs to be generated by a stable sort algorithm. We emphasize that the stability assumption is only needed for our proof technique; the actual implementation does not require it.

We now prove that $q_i^\star = \hat{q}_i$ for each $i \in 1{:}n$ by separately analyzing the following four cases.

(i) Suppose that $i = k_{\min}$. Because $p_i > 0$ from the construction of $k_{\min}$ then

$$q_i^\star = \min\{m \cdot p_i + c, 1\} = g(p_i) = g(p_i) - g(0) = f(\{i\} \cup \mathcal{J}) - f(\mathcal{J}) = \hat{q}_i,$$

where $\mathcal{J} := \{j \in 1{:}n \mid \sigma^{-1}(j) < \sigma^{-1}(i)\}$ and $\mathbf{1}_{\mathcal{J}}^\mathsf{T} \boldsymbol{p} = 0$ by the construction of $k_{\min}$.

(ii) Suppose that $i \in \mathcal{K}_{<v} \wedge i \neq k_{\min}$. From Lemma 3.2, we get for each $\mathcal{O} \subseteq \mathcal{K}_{<v}$ that

$$\mathbf{1}^\mathsf{T} \boldsymbol{p}_{\mathcal{O}} \leq \mathbb{P}[\tilde{x} < v] \leq \frac{1-c}{m}, \tag{14}$$

and, hence, from (13): $f(\mathcal{O}) = g\left(\mathbf{1}^\mathsf{T} \boldsymbol{p}_{\mathcal{O}}\right) = c + m \cdot \mathbf{1}^\mathsf{T} \boldsymbol{p}_{\mathcal{O}}$, when $\mathbf{1}^\mathsf{T} \boldsymbol{p}_{\mathcal{O}} > 0$. Let $k := \sigma^{-1}(i)$ and note that $\sigma(1{:}k) \subseteq \mathcal{K}_{<v}$ because if $l \in 1{:}k$, then $x_{\sigma(l)} \leq x_{\sigma(k)} = x_i < v$ and $\sigma(l) \in \mathcal{K}_{<v}$. Then, since $i \neq k_{\min}$:

$$\hat{q}_i = \hat{q}_{\sigma(k)} = f(\sigma(1{:}k)) - f(\sigma(1{:}(k-1))) = m \cdot \mathbf{1}^\mathsf{T} \boldsymbol{p}_{\sigma(1:k) \setminus \sigma(1:(k-1))} = m \cdot p_{\sigma(k)} = m \cdot p_i = q_i^\star.$$

The third equality follows from $i \neq k_{\min}$ and $p_{k_{\min}} \neq 0$.

(iii) Suppose that $i \in \mathcal{K}_{=v} \wedge i \neq k_{\min}$. Here, we use the assumption that σ is a stable sort and prove the claim by induction on $l \in |\mathcal{K}_{<v} \cup \{k_{\min}\}|, \ldots, |\mathcal{K}_{<v} \cup \{k_{\min}\}| + |\mathcal{K}_{=v} \setminus \{k_{\min}\}|$ showing that

$$\theta_l = 1 - f(\sigma(1{:}l)), \tag{15}$$

$$\hat{q}_{\sigma(l+1)} = q_{\sigma(l+1)}^\star. \tag{16}$$

To prove the base case, suppose that $l = |\mathcal{K}_{<v} \cup \{k_{\min}\}|$. The equality in (15) holds by algebraic manipulation from (13) since $\mathcal{K}_{<v} \cup \{k_{\min}\} = \sigma(1{:}l)$. The equality in (16) follows from (13) and (14). First, assume that $\theta_l > 0$; then

$$\begin{aligned} \hat{q}_i = \hat{q}_{\sigma(l+1)} &= f(\sigma(1{:}(l+1))) - f(\sigma(1{:}l)) \\ &= \min\left\{1,\ c + m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:(l+1))}\right\} - c - m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:l)} \\ &= \min\left\{1 - c - m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:l)},\ m \cdot p_{\sigma(l+1)}\right\} \\ &= \min\{\theta_l, m \cdot p_i\} = q_i^\star. \end{aligned} \tag{17}$$

When $\theta_l = 0$, we get by algebraic manipulation that $\hat{q}_i = 0 = q_i^\star$.

To prove the inductive case, suppose that (15) and (16) hold for l, and we prove it for $l+1$. The equality in (15) can be derived from (16) and the construction in Algorithm 4.1 as

$$\theta_{l+1} = \theta_l - q^\star_{\sigma(l+1)} = \theta_l - \hat{q}_{\sigma(l+1)} = 1 - f(\sigma(1{:}l)) - (f(\sigma(1{:}(l+1))) - f(\sigma(1{:}l))) = 1 - f(\sigma(1{:}(l+1))).$$

The equality in (16) follows analogously to (17) if $\theta_{l+1} > 0$:

$$\begin{aligned} \hat{q}_i = \hat{q}_{\sigma(l+2)} &= f(\sigma(1{:}(l+2))) - f(\sigma(1{:}(l+1))) \\ &= \min\left\{1,\ c + m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:(l+2))}\right\} - (1 - \theta_{l+1}) \\ &= \min\left\{\theta_{l+1},\ c + m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:(l+2))} - f(\sigma(1{:}(l+1)))\right\} \\ &= \min\left\{\theta_{l+1},\ m \cdot p_{\sigma(l+2)}\right\} = q_i^\star. \end{aligned} \tag{18}$$

If $\theta_{l+1} = 0$, we get by algebraic manipulation that $\hat{q}_i = 0 = q_i^\star$.

(iv) Suppose that $i \in \mathcal{K}_{>v} := \{j \in 1{:}n \mid x_j > v\}$ and let $k = \sigma^{-1}(i)$. Then, using (7) we have that:

$$1 \geq f(\sigma(1{:}k)) \geq f(\sigma(1{:}(k-1))) \geq \min\{c + m \cdot \mathbf{1}^\top \boldsymbol{p}_{\sigma(1:(k-1))}, 1\} \geq 1,$$

since $\mathbf{1}^\top \boldsymbol{p}_{\sigma(1:(k-1))} > \frac{1-c}{m}$ from Lemma 3.2 and the construction of v in Algorithm 4.2. Then

$$\hat{q}_i = \hat{q}_{\sigma(k)} = f(\sigma(1{:}k)) - f(\sigma(1{:}(k-1))) = 0 = q_i^\star.$$

□

4.2 Solving CVaR and TVaR in Linear Time

We show that CVaR and TVaR are two examples of φ-divergence risk measures that can be formulated as linear optimization over a polymatroid with an EWS capacity function and, therefore, can be solved in linear time. To formulate the CVaR polymatroid optimization, we define the capacity function $f_\alpha^{\mathrm{CVaR}} \colon 2^\Omega \to \mathbb{R}$ for each $\mathcal{O} \subseteq \Omega$ and $\alpha \in [0,1]$ as

$$f_\alpha^{\mathrm{CVaR}}(\mathcal{O}) \ := \begin{cases} \min\left\{\frac{1}{\alpha} \cdot \mathbb{P}\left[\mathcal{O}\right], 1\right\} & \text{if } \alpha \in (0,1], \\ 0 & \text{if } \alpha = 0 \wedge \mathbb{P}\left[\mathcal{O}\right] = 0, \\ 1 & \text{if } \alpha = 0 \wedge \mathbb{P}\left[\mathcal{O}\right] > 0. \end{cases} \tag{19}$$

The capacity function f_α^{CVaR} for CVaR defined in (19) is well known, but is usually restricted to $\alpha \in (0,1)$ (Acerbi, 2002). The following proposition states its correctness.

Proposition 4.6. *For each $\alpha \in [0,1]$, $\tilde{x} \colon \Omega \to \mathbb{R}$, and $\boldsymbol{p} \in \Delta_n$:*

$$\mathrm{CVaR}_\alpha\left[\tilde{x}\right] \ = \min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_{f_\alpha^{\mathrm{CVaR}}}} \boldsymbol{x}^\top \boldsymbol{q}. \tag{20}$$

Moreover, f_α^{CVaR} is $\mathrm{EWS}(\boldsymbol{p}, 1/\alpha, 0)$ when $\alpha > 0$, and $\mathrm{EWS}(\boldsymbol{p}, 1, 1)$ when $\alpha = 0$.

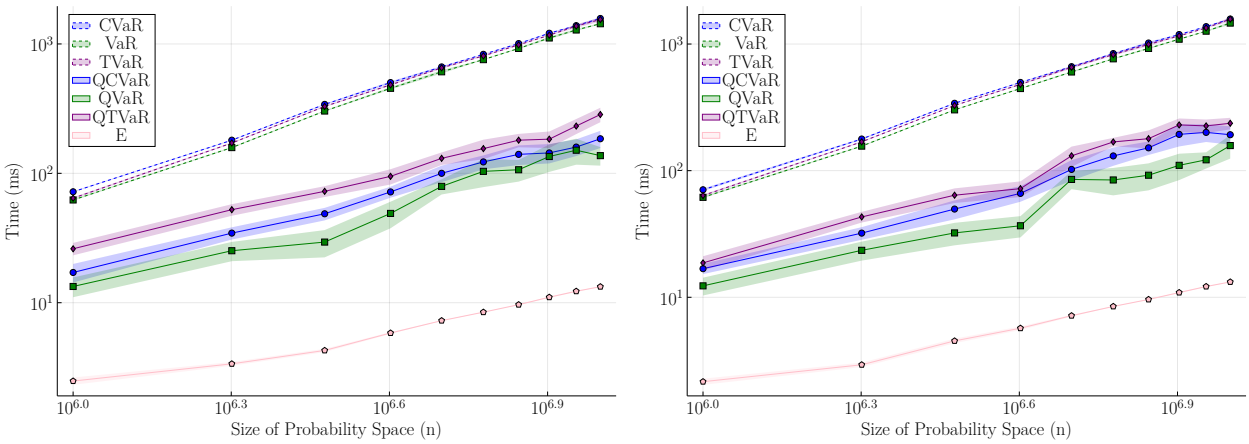

Figure 2: Algorithms' runtime for samples from a *uniform* random variable (left) and a *sparse* random variable (right). Shaded regions denote 95% confidence bands over 10 independent runs.

To compute TVaR, we define the capacity function $f_\alpha^{\mathrm{TVaR}} \colon 2^\Omega \to \mathbb{R}$ for each $\mathcal{O} \subseteq \Omega$ and $\alpha \in [0,1]$ as

$$f_\alpha^{\mathrm{TVaR}}(\mathcal{O}) \;:=\; \begin{cases} \min\left\{\mathbb{P}\left[\mathcal{O}\right] + \min\left\{\sqrt{\frac{1}{2}\log\frac{1}{\alpha}}, 1\right\}, 1\right\} & \text{if } \alpha \in (0,1] \wedge \mathbb{P}\left[\mathcal{O}\right] > 0, \\ 0 & \text{if } \mathbb{P}\left[\mathcal{O}\right] = 0, \\ 1 & \text{if } \alpha = 0 \wedge \mathbb{P}\left[\mathcal{O}\right] > 0. \end{cases} \tag{21}$$

TVaR was first proposed in the context of φ-divergence risk measures (Kruse et al., 2021) as well as in the context of robust stochastic programming (Shapiro, 2017). TVaR is also very closely related to the L_1 ball used in robust Markov decision processes (Ho et al., 2021; 2022). However, we are not aware of prior work that defines or uses the capacity function in (21). The following proposition shows that this function can be used to correctly formulate the TVaR evaluation using an EWS function.

Proposition 4.7. *For each* $\alpha \in [0,1]$, $\tilde{x} \colon \Omega \to \mathbb{R}$, *and* $\boldsymbol{p} \in \Delta_n$*:*

$$\mathrm{TVaR}_\alpha\left[\tilde{x}\right] = \min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_{f_\alpha^{\mathrm{TVaR}}}} \boldsymbol{x}^\mathsf{T}\boldsymbol{q}. \tag{22}$$

Moreover, f_α^{TVaR} *is* $\mathrm{EWS}(\boldsymbol{p}, 1, \min\left\{\sqrt{\frac{1}{2}\log\frac{1}{\alpha}}, 1\right\})$ *when* $\alpha > 0$*, and* $\mathrm{EWS}(\boldsymbol{p}, 1, 1)$ *when* $\alpha = 0$.

5 Experiments

We evaluate the proposed algorithms empirically to confirm that their theoretical performance guarantees translate into faster practical runtimes. We compare each linear-time algorithm against its standard sorting-based counterpart. Throughout this section, we prefix the name of our linear-time implementation with a "Q" to distinguish it from the sorting-based baseline, giving QVaR, QCVaR, and QTVaR. We also report the runtime of computing the standard expectation $\mathbb{E}$ as a baseline.

Our implementation, together with the code and data needed to reproduce all results, is available at [redacted for anonymity], and the actual implementation of the risk measures can be found at [redacted for anonymity]. All experiments were run on an AMD Ryzen Threadripper 3970X with 32 cores and 64 threads at 4.55 GHz with 256 GB of RAM using Julia 1.12.5. To reduce noise in the timing measurements, we disable the Julia garbage collector during each run.

We begin with synthetic data drawn from a uniform distribution with n samples, varying n from 10^6 to 10^7. For each value of n we run every algorithm 10 times on independently generated samples and report the mean runtime. Figure 2 shows the results on a log-log scale for random variables with a uniform (left) and

Table 2: Algorithms' runtime (ms) in stock market scenario.

Method	Mean	St. D.
VaR	0.37	0.00
QVaR	0.13	0.02
CVaR	0.39	0.00
QCVaR	0.22	0.03
TVaR	0.40	0.01
QTVaR	0.25	0.05
$\mathbb{E}$	0.01	0.00

Table 3: Algorithms' mean runtime (ms) on small sparse domains.

n	$\mathbb{E}$	VaR	QVaR	CVaR	QCVaR	TVaR	QTVaR
1000	0.00	0.03	0.01	0.03	0.02	0.03	0.02
2000	0.00	0.06	0.02	0.06	0.03	0.06	0.04
3000	0.01	0.09	0.04	0.10	0.05	0.10	0.07
4000	0.00	0.13	0.04	0.14	0.06	0.13	0.08
5000	0.01	0.16	0.05	0.17	0.10	0.18	0.10
6000	0.00	0.20	0.08	0.21	0.10	0.21	0.11
7000	0.00	0.23	0.08	0.25	0.12	0.24	0.13
8000	0.00	0.27	0.09	0.28	0.13	0.28	0.15
9000	0.01	0.30	0.09	0.32	0.12	0.31	0.17
10000	0.01	0.33	0.11	0.35	0.16	0.35	0.19

sparsely-supported (right) distributions. We fix the risk level to $\alpha = 0.95$, though the results are similar for other levels except the degenerate cases $\alpha = 0$ and $\alpha = 1$. The shaded regions denote 95% confidence bands over the 10 runs; the bands are tight because the measured runtimes vary little between runs.

The results confirm that the quick algorithms are substantially faster than their sorting-based counterparts, with the gap widening as n grows. This behavior matches the theoretical analysis of Sections 3 and 4: the sorting-based methods require $O(n \log n)$ operations, whereas our algorithms run in $O(n)$ expected time, so the logarithmic factor separating them grows with the size of the probability space. The acceleration is most consequential for VaR and CVaR, which are the most widely used risk measures in the applications discussed in Section 1 and are frequently evaluated on large random variables.

To confirm the speedups persist beyond large synthetic inputs, we repeat the comparison on two more realistic settings. The first one is a Stock Market scenario, in which the random variable models the returns of a financial portfolio. Table 2 reports the mean runtime and standard deviation of each method. The second one is small-scale problems, where the asymptotic advantage is least pronounced but can be practically important when the risk measures must be computed repeatedly. Table 3 reports the mean runtime on sparse random variables over probability spaces of size n ranging from 1000 to 10 000, using the same protocol as above but with 50 trials per configuration in place of 10. Even at these sizes, the quick algorithms outperform their sorting-based counterparts for every method and every value of n, confirming that the linear-time algorithms deliver consistent improvements across the full range of input sizes without incurring any penalty on small inputs.

6 Conclusion

We propose and analyze new algorithms for computing several common risk measures, including VaR, CVaR, and TVaR, in $O(n)$ expected time, where n is the size of the probability space. This compares favorably with the $O(n \log n)$ time required by sorting-based methods. This speedup is significant because risk measures are frequently evaluated on large random variables in applications such as robotics, finance, and infrastructure maintenance. An important direction for future work is to develop fast algorithms for other risk measures, such as EVaR and Expectile.

References

C. Acerbi. Spectral measures of risk: A coherent representation of subjective risk aversion. *Journal of Banking & Finance*, 26(7):1505–1518, 2002.

L. Adam and V. Mácha. Projections onto the canonical simplex with additional linear inequalities, 2019.

A. Ahmadi-Javid. Entropic Value-at-Risk: A new coherent risk measure. *Journal of Optimization Theory and Applications*, 155(3):1105–1123, 2012.

P. Akella, A. Dixit, M. Ahmadi, L. Lindemann, M. P. Chapman, G. J. Pappas, A. D. Ames, and J. W. Burdick. Risk-aware robotics: Tail risk measures in planning, control, and verification. *arXiv preprint arXiv:2403.18972*, 2024.

A. Ang, J. Ma, N. Liu, and K. Huang. Fast projection onto the capped simplex with applications to sparse regression in bioinformatics. In *Conference on Neural Information Processing Systems*, volume 35, 2021.

F. Barahona, M. Ettl, M. Petrik, and P. M. Rimshnick. Agile logistics simulation and optimization for managing disaster responses. In *Winter Simulation Conference*, pages 3340–3351, 2013.

G. Bayraksan and D. K. Love. Data-driven stochastic programming using phi-divergences. In *The Operations Research Revolution*, INFORMS TutORials in Operations Research, chapter 1, pages 1–19. INFORMS, 2015.

M. Benrabah, C. Orou Mousse, E. Randriamiarintsoa, R. Chapuis, and R. Aufrère. A review on traversability risk assessments for autonomous ground vehicles: Methods and metrics. *Sensors*, 24(6):1909, 2024.

J. Bilmes. Submodular functions, optimization, and applications to machine learning — spring quarter, lecture 4, 2018.

S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.

S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

C. Chávez and Y. Li. Sample complexity for Markov decision processes and stochastic optimal control with static risk measures, 2026.

L. Condat. Fast projection onto the simplex and the L1 ball. *Mathematical Programming*, 158(1-2):575–585, 2016.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, N.J, 1976.

R. Ding and E. A. Feinberg. Sequential optimization of CVaR, 2023.

H. Follmer and A. Schied. *Stochastic Finance: Introduction in Discrete Time*. De Gruyter Graduate, fourth edition, 2016.

M. Godbout and A. Durand. On the fundamental limitations of dual static CVaR decompositions in Markov decision processes, 2025.

J. L. Hau, E. Delage, M. Ghavamzadeh, and M. Petrik. On dynamic programming decompositions of static risk measures in Markov decision processes. In *Neural Information Processing Systems (NeurIPS)*, 2023.

J. L. Hau, E. Delage, E. Derman, M. Ghavamzadeh, and M. Petrik. Q-learning for quantile MDPs: A decomposition, performance, and convergence analysis. In *Artificial Intelligence and Statistics (AISTATS)*, 2025.

C. P. Ho, M. Petrik, and W. Wiesemann. Fast Bellman updates for robust MDPs. In *International Conference on Machine Learning (ICML)*, pages 1979–1988, 2018.

C. P. Ho, M. Petrik, and W. Wiesemann. Partial policy iteration for L1-robust Markov decision processes. *Journal of Machine Learning Research*, 22:1–46, 2021.

C. P. Ho, M. Petrik, and W. Wiesemann. Robust phi-divergence MDPs. In *Neural Information Processing Systems (NeurIPS)*, 2022.

E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, New York, 1998.

A. Inzunza, R. Moreno, A. Bernales, and H. Rudnick. CVaR constrained planning of renewable generation with consideration of system inertial response, reserve services and demand participation. *Energy Economics*, 59:104–117, 2016.

B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, fifth edition, 2012.

T. Kruse, J. C. Schneider, and N. Schweizer. A toolkit for robust risk assessment using f-divergences. *Management Science*, 67(10):6529–6552, 2021.

C. H. Lim and S. J. Wright. Efficient Bregman projections onto the permutahedron and related polytopes. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1205–1213. PMLR, 2016.

A. Majumdar and M. Pavone. How should a robot assess risk? towards an axiomatic theory of risk in robotics. In *Robotics Research: The 18th International Symposium ISRR*, pages 75–84. Springer, 2020.

A. Muni, V. Taboga, E. Derman, P.-L. Bacon, and E. Delage. Reward Redistribution for CVaR MDPs using a Bellman Operator on L-infinity, 2026.

M. Petrik, G. Tennenholtz, and M. Ghavamzadeh. Bayesian regret minimization in offline bandits. In *Proceedings of the 41st International Conference on Machine Learning*, pages 40502–40522. PMLR, 2024.

N. Privault. Notes on financial risk and analytics with R and Python. Technical report, Nanyang Technological University, 2026.

R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis*. Springer, 1996.

U. Sadana, E. Delage, and A. Georghiou. Mitigating optimistic bias in entropic risk estimation and optimization, 2026.

A. Shapiro. Distributionally robust stochastic programming. *SIAM Journal on Optimization*, 27(4):2258–2275, 2017.

A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, 2014.

X. Su, J. Grand-Clement, and M. Petrik. Risk-averse total-reward MDPs with ERM and EVaR. In *AAAI Conference on Artificial Intelligence*, 2025.

W. Wang and M. Á. Carreira-Perpiñán. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application, 2013.

L. Xia, L. Zhang, and P. W. Glynn. Risk-sensitive Markov decision processes with long-run CVaR criterion. *Production and Operations Management*, 32(12):4049–4067, 2023.

A Supplemental Results for Section 3

In this section, we summarize the standard and supporting algorithms for solving VaR. We also include the selected technical proofs for the correctness of the proposed algorithm.

Lemma A.1. *For each* $t \in \mathbb{R}$*:*

$$\mathbb{P}\left[\tilde{x} = t\right] > 0 \implies \exists \omega \in \Omega, x(\omega) = t.$$

Proof. The result follows by induction on the size $|\Omega|$, dropping the constraint $\mathbf{1}^\mathsf{T}\boldsymbol{p} = 1$. □

Proposition A.2. *For each* $\boldsymbol{x} \in \mathbb{R}^n$*,* $\boldsymbol{p} \in \Delta_n$*, and* $\alpha \in [0, 1)$*, Algorithm A.1 returns* $\mathrm{VaR}_\alpha\left[\tilde{x}\right]$.

Algorithm A.1: Standard VaR algorithm

Input : $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{p} \in \Delta_n$, and $\alpha \in [0, 1)$
Output: $\mathrm{VaR}_\alpha[\tilde{x}]$

Sort $\boldsymbol{x}$ indices to get $\sigma\colon 1{:}n \to 1{:}n$ such that $i > j \implies x_{\sigma(i)} \geq x_{\sigma(j)}$, $\forall i, j \in 1{:}n$;
$q_0 \leftarrow 0$; // $q_i = \mathbb{P}\left[\tilde{x} \leq x_{\sigma(i)}\right]$ `when` $x_{\sigma(i)}$ `unique`
for $i \in 1{:}n$ **do**
  $q_i \leftarrow q_{i-1} + p_{\sigma(i)}$;
  if $q_i > \alpha$ **then return** $x_{\sigma(i)}$;

Proof. First, one can readily show by induction that for each $i \in 1{:}n$ (note that values of $\boldsymbol{x}$ need not be unique):

$$\mathbb{P}\left[\tilde{x} < x_{\sigma(i)}\right] \leq q_{i-1} \leq q_i \leq \mathbb{P}\left[\tilde{x} \leq x_{\sigma(i)}\right]. \tag{23}$$

Since $\alpha < 1 = q_n$, the algorithm is guaranteed to terminate with some $i^\star \in 1{:}n$ and return $x_{\sigma(i^\star)}$. Therefore, $q_{i^\star - 1} \leq \alpha < q_{i^\star}$ and applying (23) gives us that

$$\mathbb{P}\left[\tilde{x} < x_{\sigma(i^\star)}\right] \leq q_{i^\star - 1} \leq \alpha < q_{i^\star} \leq \mathbb{P}\left[\tilde{x} \leq x_{\sigma(i^\star)}\right],$$

and the result holds by Lemma 3.2. □

Algorithm A.2: `partition`: Dutch national flag algorithm (Dijkstra, 1976, chapter 14)

Input : $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{p} \in \Delta_n$, pivot $v \in \mathbb{R}$, and front and back $f, b \in 1{:}n$, $f \leq b$
Output: $l, g \in f{:}b$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{p} \in \Delta_n$ such that Lemma A.3 holds

$l_0 \leftarrow e_0 \leftarrow f$; $g_0 \leftarrow b$; $k \leftarrow 0$;
while $e_k \leq g_k$ **do**
  if $x_{e_k} < v$ **then**
    swap: $x_{e_k} \leftrightarrow x_{l_k}$ and $p_{e_k} \leftrightarrow p_{l_k}$;
    $l_{k+1} \leftarrow l_k + 1$; $e_{k+1} \leftarrow e_k + 1$;
  else if $x_{e_k} > v$ **then**
    swap: $x_{e_k} \leftrightarrow x_{g_k}$ and $p_{e_k} \leftrightarrow p_{g_k}$;
    $g_{k+1} \leftarrow g_k - 1$;
  else $e_{k+1} \leftarrow e_k + 1$;
  $k \leftarrow k + 1$
return $(l_k, g_k, \boldsymbol{x}, \boldsymbol{p})$;

The following lemma proves the properties of the partition operator defined in Algorithm A.2.

Lemma A.3. *If* $(l, g, \boldsymbol{x}', \boldsymbol{p}') = \mathrm{partition}(\boldsymbol{x}, \boldsymbol{p}, v, f, b)$ *for* $f \leq b$*, then:*

$$\boldsymbol{x}'_{f:(l-1)} < v, \qquad \boldsymbol{x}'_{l:g} = v, \qquad \boldsymbol{x}'_{(g+1):b} > v. \tag{24}$$

Moreover, $\boldsymbol{x}_{f:b} = \boldsymbol{x}'_{\sigma(f:b)}, \boldsymbol{p}_{f:b} = \boldsymbol{p}'_{\sigma(f:b)}$ *for some permutation* $\sigma\colon f{:}b \to f{:}b$*, and* $\boldsymbol{x}_{1:(f-1)} = \boldsymbol{x}'_{1:(f-1)}, \boldsymbol{p}_{1:(f-1)} = \boldsymbol{p}'_{1:(f-1)}$*, and* $\boldsymbol{x}_{(b+1):n} = \boldsymbol{x}'_{(b+1):n}, \boldsymbol{p}_{(b+1):n} = \boldsymbol{p}'_{(b+1):n}$*. The elements outside of the range* $f{:}b$ *are not modified. The algorithm runs in* $O(b - f + 1)$ *time.*

Proof of Lemma A.3. Let v be an input to Algorithm A.2 and consider the following invariants.

$$x_j < v, \qquad j \in f_k{:}(l_k - 1), \tag{25}$$
$$x_j = v, \qquad j \in l_k{:}(e_k - 1), \tag{26}$$
$$x_j > v, \qquad j \in (g_k + 1){:}b_k. \tag{27}$$

Initialization. For $k = 0$, conditions (25), (26), (27) are true because the pertinent intervals are empty.

Algorithm A.3: `QuickVaR`: Non-recursive adaptation of Algorithm 3.1

Input : vector $\boldsymbol{x}^0$, pmf $\boldsymbol{p}^0$ and $\alpha_0 \in [0, 1)$
Output: $\mathrm{VaR}_\alpha[\tilde{x}]$

1 $f_0 \leftarrow 1$; $b_0 \leftarrow n$; $k \leftarrow 1$;
2 **while** $b_{k-1} > f_{k-1}$ **do**
3 $\quad v_k \leftarrow x_{\mathrm{rand}(f_{k-1}:b_{k-1})}$; `// Pivot`
4 $\quad l_k, g_k, \boldsymbol{x}^k, \boldsymbol{p}^k \leftarrow \mathrm{partition}(\boldsymbol{x}^{k-1}, \boldsymbol{p}^{k-1}, v_k, f_{k-1}, b_{k-1})$; `// Alg. A.2, Lemma A.3`
5 $\quad t_k \leftarrow \sum_{j=f_{k-1}}^{l_k-1} p_j^k$; `//` $\mathbb{P}\left[\tilde{x}^k < v_k\right]$
6 $\quad e_k \leftarrow \sum_{j=l_k}^{g_k} p_j^k$; `//` $\mathbb{P}\left[\tilde{x}^k = v_k\right]$
7 $\quad$ **if** $\alpha_{k-1} < t_k$ **then** $f_k \leftarrow f_{k-1}$; $b_k \leftarrow l_k - 1$; $\alpha_k \leftarrow \alpha_{k-1}$;
8 $\quad$ **else if** $t_k + e_k \leq \alpha_{k-1}$ **then** $f_k \leftarrow g_k + 1$; $b_k \leftarrow b_{k-1}$; $\alpha_k \leftarrow \alpha_{k-1} - t_k - e_k$;
9 $\quad$ **else** $f_k \leftarrow g_k$; $b_k \leftarrow g_k$;
10 $\quad k \leftarrow k + 1$;
11 **return** $x_{b_{k-1}}^{k-1}$;

Maintenance. At each iteration of the loop that starts on line 2 of Algorithm A.2, there are 3 possible branches determined by the conditional logic: $x_{e_k} < v$, $x_{e_k} > v$, and $x_{e_k} = v$. Since $v, x_{e_k} \in \mathbb{R}$ one of these branches must be executed for each iteration k.

Each time the $x_{e_k} < v$ or $x_{e_k} > v$ branch is executed, conditions (25) and (27), respectively, remain true since in each case we swap the necessary elements to preserve each condition. Notice also that e_k is incremented alongside l_k to maintain $e_k - l_k = e_{k+1} - l_{k+1}$ as is necessary for condition (26).

When $x_{e_k} = v$, e_k is incremented; since $e_0 = l_0$, this creates a difference between e_{k+1} and l_{k+1}. The algorithm terminates when $e_t = g_t + 1$, so $g_t - l_t = e_t - l_t - 1 \geq 0$ and (26) remains true.

Termination. In the loop on line 2 there are 3 possible paths of execution determined by the conditional logic $x_{e_k} < v$, $x_{e_k} > v$, and $x_{e_k} = v$. Each of these paths of execution decreases $g_k - e_k$ by 1. This occurs on lines 5 and 9 by increasing e_k by 1 and keeping g_k the same, or on line 8 by decreasing g_k by 1 and keeping e_k the same.

Observe that $g_0 - e_0 = b - f$ is non-negative since $f \leq b$. The difference $g_k - e_k$ decreases by 1 at each iteration, and each iteration requires constant time to complete. Because the algorithm returns when $g_t - e_t = -1$, it finishes in $O(b - f)$ time.

On the last iteration $g_t = e_t$, therefore by the same reasoning as above $g_{t+1} - e_{t+1} = -1$ and condition (26) holds for $g_{t+1} = e_{t+1} - 1$. □

B Supplemental Results for Section 4

The following proposition states the correctness of the standard polymatroid optimization algorithm when applied to a normalized polymatroid.

Proposition B.1. *Suppose that $f\colon 2^{1:n} \to \mathbb{R}$ is submodular, monotone, $f(\emptyset) = 0$ and $f(1{:}n) = 1$, and let $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$. Then, Algorithm 4.1 computes $\hat{\boldsymbol{q}}$ such that*

$$\hat{\boldsymbol{q}} \in \operatorname*{arg\,min}_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} \boldsymbol{x}^\mathsf{T} \boldsymbol{q}. \tag{28}$$

Proof. First, we show that $\hat{\boldsymbol{q}} \in \widehat{\mathcal{Q}}_f$. Because the construction of $\hat{\boldsymbol{q}}$ in Algorithm 4.1 matches the construction in proposition 14.10 in Korte and Vygen (2012), since $f(\emptyset) = 0$ and $\hat{q}_{\sigma(1)} = f(\{\sigma(1)\}) - f(\emptyset) = f(\{\sigma(1)\})$, we have that $\hat{\boldsymbol{q}} \in \mathcal{Q}_f$. In addition,

$$\sum_{i=1}^n \hat{q}_i = \sum_{i=1}^n \hat{q}_{\sigma(i)} = f(1{:}n) - f(\emptyset) = 1,$$

which implies that $\hat{\boldsymbol{q}} \in \Delta_n \cap \mathcal{Q}_f = \widehat{\mathcal{Q}}_f$, $\widehat{\mathcal{Q}}_f \neq \emptyset$, and the minimum (28) exists.

To show that $\hat{\boldsymbol{q}}$ is optimal, we reduce the optimization in (28) to the standard linear polymatroid maximization problem with non-negative weights (Korte and Vygen, 2012, section 14.9) as follows:

$$\begin{aligned}
\min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} \boldsymbol{x}^\top \boldsymbol{q} &= \min_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} (\boldsymbol{x} - x_{\max} \cdot \mathbf{1})^\top \boldsymbol{q} + x_{\max} \\
&= - \max_{\boldsymbol{q} \in \widehat{\mathcal{Q}}_f} (x_{\max} \cdot \mathbf{1} - \boldsymbol{x})^\top \boldsymbol{q} + x_{\max} \\
&\geq - \max_{\boldsymbol{q} \in \mathcal{Q}_f} (x_{\max} \cdot \mathbf{1} - \boldsymbol{x})^\top \boldsymbol{q} + x_{\max} \\
&= -(x_{\max} \cdot \mathbf{1} - \boldsymbol{x})^\top \hat{\boldsymbol{q}} + x_{\max} \\
&= \boldsymbol{x}^\top \hat{\boldsymbol{q}}.
\end{aligned}$$

Here, $x_{\max} := \max_{i \in 1:n} x_i$ and the equalities follow because $\widehat{\mathcal{Q}}_f$ is a normalized polymatroid and $\mathbf{1}^\top \hat{\boldsymbol{q}} = 1$. The second-to-last equality follows from theorem 14.11 in Korte and Vygen (2012) and the construction in the greedy polymatroid algorithm. □

The following lemma is folklore in submodular optimization (Bilmes, 2018, theorem 4.5.1). We state and prove it for completeness.

Lemma B.2. *Suppose that $\boldsymbol{\mu} \in \mathbb{R}^n_{\geq 0}$ and $g \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$ is concave and monotone. Then $f(\mathcal{A}) := g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}})$ is a submodular monotone function.*

Proof. The fact that f is monotone follows from the composition of two monotone functions. To prove that f is submodular, we use the following properties that hold for each $\mathcal{A}, \mathcal{B} \subseteq 1{:}n$:

$$\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}} \leq \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}} \leq \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}, \qquad \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}} \leq \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}} \leq \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}, \tag{29}$$

$$\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}} + \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}} = \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} + \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}. \tag{30}$$

The inequalities in (29) hold because $\boldsymbol{\mu} \geq \mathbf{0}$ and the equality in (30) holds because $\mathcal{A} \mapsto \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}}$ is modular. Suppose that $\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}} < \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}$; otherwise the submodularity condition in (33) below holds with equality. Then, writing $\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}}$ and $\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}}$ as a convex combination of $\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}$ and $\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}$ using (29) and the concavity of g we get that

$$g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}}) \geq \frac{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}}}{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}} g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}) + \frac{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}}{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}} g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}), \tag{31}$$

$$g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}}) \geq \frac{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}}}{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}} g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}) + \frac{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}}{\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}} - \mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}} g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}). \tag{32}$$

Then summing (31) and (32) and algebraic manipulation yields that

$$f(\mathcal{A}) + f(\mathcal{B}) = g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A}}) + g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{B}}) \geq g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cap \mathcal{B}}) + g(\mathbf{1}^\top \boldsymbol{\mu}_{\mathcal{A} \cup \mathcal{B}}) = f(\mathcal{A} \cap \mathcal{B}) + f(\mathcal{A} \cup \mathcal{B}), \tag{33}$$

which proves that f is submodular. □

Proof of Proposition 4.6. The function f_α^{CVaR} is EWS by algebraic manipulation. The equality (20) for $\alpha = 0$ follows by algebraic manipulation. To prove (20) for $\alpha > 0$, we use Proposition 4.2 and show for each $\mathcal{O} \subseteq \Omega$ that

$$f_\alpha^{\mathrm{CVaR}}(\mathcal{O}) = \min \left\{ \alpha^{-1} \mathbf{1}_{\mathcal{O}}^\top \boldsymbol{p},\ 1 \right\} = \max \left\{ \mathbf{1}_{\mathcal{O}}^\top \boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \boldsymbol{q} \leq \alpha^{-1} \boldsymbol{p} \right\} = -\operatorname{CVaR}_\alpha \left[-\tilde{1}_{\mathcal{O}} \right], \tag{34}$$

using the dual CVaR formulation in (4). We prove (34) as two inequalities.

(i) The inequality

$$\min \left\{ \alpha^{-1} \mathbf{1}_{\mathcal{O}}^\top \boldsymbol{p},\ 1 \right\} \geq \max \left\{ \mathbf{1}_{\mathcal{O}}^\top \boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \boldsymbol{q} \leq \alpha^{-1} \boldsymbol{p} \right\},$$

holds because $\boldsymbol{q}$ feasible in the right-hand side satisfies that $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \leq \alpha^{-1}\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p}$, and $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \leq \mathbf{1}^{\mathsf{T}}\boldsymbol{q} = 1$.

(ii) To prove that

$$\min\left\{\alpha^{-1}\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p},\ 1\right\} \leq \max\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \boldsymbol{q} \leq \alpha^{-1}\boldsymbol{p}\right\}, \tag{35}$$

we analyze the following two cases.

(a) When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} \geq \alpha$, let $\boldsymbol{q} := (\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p})^{-1} \cdot \boldsymbol{p} \odot \mathbf{1}_{\mathcal{O}}$, which is feasible in the right-hand side of (35) and $\odot$ is the Hadamard vector product.

(b) When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} < \alpha$, let $\boldsymbol{q} := \alpha^{-1} \cdot \boldsymbol{p} \odot \mathbf{1}_{\mathcal{O}} + d \cdot \boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}})$, which is feasible in the right-hand side of (35) and $d \in \mathbb{R}$ solves that $d \cdot \mathbf{1}^{\mathsf{T}}\boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}}) = 1 - \alpha^{-1}\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p}$. Such d always exists because $\mathbf{1}^{\mathsf{T}}\boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}}) = 1 - \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} > 1 - \alpha \geq 0$. It is easy to verify that $\boldsymbol{q} \geq \mathbf{0}$ and $\mathbf{1}^{\mathsf{T}}\boldsymbol{q} = 1$. The condition $\boldsymbol{q} \leq \alpha^{-1}\boldsymbol{p}$ holds because $d \leq \alpha^{-1}$ is satisfied by multiplying both sides of the inequality by $\mathbf{1}^{\mathsf{T}}\boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}})$:

$$d \cdot \mathbf{1}^{\mathsf{T}}\boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}}) \ \leq\ 1 - \frac{1}{\alpha}\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} \ \leq\ \frac{1}{\alpha} \cdot \mathbf{1}^{\mathsf{T}}\boldsymbol{p} \odot (\mathbf{1} - \mathbf{1}_{\mathcal{O}}).$$

□

Proof of Proposition 4.7. The function f_α^{TVaR} is EWS by algebraic manipulation. The equality (22) for α such that $\sqrt{\frac{1}{2}\log\frac{1}{\alpha}} \geq 1$, including $\alpha = 0$, follows by algebraic manipulation.

It remains to prove (22) when $\sqrt{\frac{1}{2}\log\frac{1}{\alpha}} < 1$. We use Proposition 4.2 and show for each $\mathcal{O} \subseteq \Omega$ that

$$\begin{aligned} f_\alpha^{\mathrm{TVaR}}(\mathcal{O}) &= \min\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}},\ 1\right\} \\ &= \max\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \|\boldsymbol{p} - \boldsymbol{q}\|_1 \leq \sqrt{2\log\frac{1}{\alpha}}\right\} = -\operatorname{TVaR}_\alpha\left[-\tilde{1}_{\mathcal{O}}\right], \end{aligned} \tag{36}$$

using the TVaR definition in (5). We need to handle two special cases first. When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} = 0$, then we have that $0 = f_\alpha^{\mathrm{TVaR}}(\mathcal{O}) = g(0) = -\operatorname{TVaR}_\alpha[-\tilde{1}_{\mathcal{O}}]$ since $\boldsymbol{q} \ll \boldsymbol{p}$ in the TVaR definition in (5). When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} = 1$, then we have that $1 = f_\alpha^{\mathrm{TVaR}}(\mathcal{O}) = g(1) = -\operatorname{TVaR}_\alpha[-\tilde{1}_{\mathcal{O}}]$ since $\boldsymbol{q} \ll \boldsymbol{p}$ and $\mathbf{1}^{\mathsf{T}}\boldsymbol{q} = 1$ in the TVaR definition in (5).

Finally, we prove (36) as two inequalities when $0 < \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} < 1$. (i) To prove that

$$\min\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}},\ 1\right\} \geq \max\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \|\boldsymbol{p} - \boldsymbol{q}\|_1 \leq \sqrt{2\log\frac{1}{\alpha}}\right\},$$

observe that $\boldsymbol{q}$ feasible in the right-hand side implies that $\|\boldsymbol{p}-\boldsymbol{q}\|_1 = \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}|\boldsymbol{p}-\boldsymbol{q}| + (\mathbf{1}-\mathbf{1}_{\mathcal{O}})^{\mathsf{T}}|\boldsymbol{p}-\boldsymbol{q}| \leq \sqrt{2\log\frac{1}{\alpha}}$ and $(\mathbf{1} - \mathbf{1}_{\mathcal{O}})^{\mathsf{T}}(\boldsymbol{p} - \boldsymbol{q}) = \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}(\boldsymbol{q} - \boldsymbol{p})$ since $\boldsymbol{p}, \boldsymbol{q} \in \Delta_n$. Then

$$\begin{aligned} \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} &= \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}(\boldsymbol{q} - \boldsymbol{p}) + \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} = \frac{1}{2}\left(\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}(\boldsymbol{q} - \boldsymbol{p}) + (\mathbf{1} - \mathbf{1}_{\mathcal{O}})^{\mathsf{T}}(\boldsymbol{p} - \boldsymbol{q})\right) + \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} \\ &\leq \frac{1}{2}\left(\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}|\boldsymbol{q} - \boldsymbol{p}| + (\mathbf{1} - \mathbf{1}_{\mathcal{O}})^{\mathsf{T}}|\boldsymbol{p} - \boldsymbol{q}|\right) + \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} = \sqrt{\frac{1}{2}\log\frac{1}{\alpha}} + \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p}, \end{aligned}$$

and $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \leq \mathbf{1}^{\mathsf{T}}\boldsymbol{q} = 1$.

(ii) To prove that

$$\min\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}},\ 1\right\} \leq \max\left\{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{q} \mid \boldsymbol{q} \in \Delta_n,\ \|\boldsymbol{p} - \boldsymbol{q}\|_1 \leq \sqrt{2\log\frac{1}{\alpha}}\right\},$$

we analyze the following two cases:

1. When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}} \geq 1$, let $\boldsymbol{q} \;:=\; (\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p})^{-1} \cdot \boldsymbol{p} \odot \mathbf{1}_{\mathcal{O}}$, where $\odot$ is the Hadamard vector product. This $\boldsymbol{q}$ is feasible in the right-hand side and satisfies the inequality.
2. When $\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}} < 1$, let

$$\boldsymbol{q} \;:=\; \frac{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} + \sqrt{\frac{1}{2}\log\frac{1}{\alpha}}}{\mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p}} \cdot \boldsymbol{p} \odot \mathbf{1}_{\mathcal{O}} + \frac{(\mathbf{1}-\mathbf{1}_{\mathcal{O}})^{\mathsf{T}}\boldsymbol{p} - \sqrt{\frac{1}{2}\log\frac{1}{\alpha}}}{(\mathbf{1}-\mathbf{1}_{\mathcal{O}})^{\mathsf{T}}\boldsymbol{p}} \cdot \boldsymbol{p} \odot (\mathbf{1}-\mathbf{1}_{\mathcal{O}}),$$

which is feasible in the right-hand side and satisfies the inequality. Neither one of the divisions is by 0 since $0 < \mathbf{1}_{\mathcal{O}}^{\mathsf{T}}\boldsymbol{p} < 1$.

□