The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Risk-Sensitive Submodular Optimization

Bryan Wilder
Department of Computer Science
and Center for Artificial Intelligence in Society
University of Southern California
bwilder@usc.edu

Abstract

The conditional value at risk (CVaR) is a popular risk measure which enables risk-averse decision making under uncertainty. We consider maximizing the CVaR of a *continuous submodular* function, an extension of submodular set functions to a continuous domain. One example application is allocating a continuous amount of energy to each sensor in a network, with the goal of detecting intrusion or contamination. Previous work allows maximization of the CVaR of a linear or concave function. Continuous submodularity represents a natural set of *nonconcave* functions with diminishing returns, to which existing techniques do not apply. We give a $(1-1/e)$-approximation algorithm for maximizing the CVaR of a monotone continuous submodular function. This also yields an algorithm for submodular set functions which produces a distribution over feasible sets with guaranteed CVaR. Experimental results in two sensor placement domains confirm that our algorithm substantially outperforms competitive baselines.

Introduction

Decision-making under uncertainty is an ubiquitous problem. Suppose we want to maximize a function $F(\boldsymbol{x}, y)$, where $\boldsymbol{x}$ is a vector of decision variables and y a random variable drawn from a distribution D. A natural approach is to maximize $\mathbb{E}_y\left[F(\boldsymbol{x}, y)\right]$, i.e., to maximize the expected value of the chosen decision. However, decision makers are often *risk-averse*: they would rather minimize the chance of having a very low reward than focus purely on the average. This is a rational behavior when failure can have large consequences. For instance, if a corporation suffers a disastrous loss, they may simply go out of business. Or in many cases, low performance entails safety issues. For instance, if a sensor network for water contamination detects problems instantly in 80% cases, but fails entirely in 20%, the population will inevitably be exposed to an unacceptable health risk. It is much better to have a sensor network which always detects contaminants, even if it requires somewhat more time on average.

Hence, it is natural to move beyond average-case analysis and optimize a risk-aware objective function. One widespread choice is the *conditional value at risk* (CVaR). CVaR takes a tunable parameter α. Roughly, it measures the performance of a decision in the worst α fraction of scenarios. It is known that when the objective F is a concave function, then CVaR can be optimized via a concave program as well. However, many natural objective functions are *not* concave, and no general algorithms are known for nonconcave functions. We focus on *submodular* functions. Submodularity captures diminishing returns and appears in application domains ranging from viral marketing (Kempe, Kleinberg, and Tardos 2003), to machine learning (Kulesza and Taskar 2012), to auction theory (Vondrák 2008). We analyze submodular functions in two settings:

Continuous: Continuous submodularity, which has lately received increasing attention (Bach 2015; Bian et al. 2017; Staib and Jegelka 2017) generalizes the notion of a submodular set function to continuous domains. Many well-known discrete problems (e.g., sensor placement, influence maximization, or facility location) admit natural extensions where resources are divided in a continuous manner. Continuous submodular functions have also been extensively studied in economics as a model of diminishing returns or strategic substitutes (Koçkesen, Ok, and Sethi 2000; Sampson 2016). Our main result is a $(1-\frac{1}{e})$-approximation algorithm for maximizing the CVaR of any monotone, continuous submodular function. No algorithm was previously known for this problem.

Portfolio of discrete sets: Our results for continuous submodular functions also transfer to set functions. We study a setting where the algorithm can select a distribution over feasible sets, which is of interest when the aim is to select a portfolio of sets to hedge against risk (Ohsaka and Yoshida 2017). Similar settings have also been studied in robust submodular optimization (Krause, Roper, and Golovin 2011; Chen et al. 2017; Wilder 2017). We give a black-box reduction from the discrete portfolio problem to CVaR optimization of continuous submodular functions, allowing us to apply our algorithm for the continuous problem. The state of the art for the discrete portfolio setting is an algorithm by Ohsaka and Yoshida (2017) for CVaR influence maximization. Our results are stronger in two ways: (i) they apply to *any* submodular function and (ii) give stronger approximation guarantee. Allowing the algorithm to select a convex combination of sets is provably necessary: Maehara (2015) proved that restricted to single sets, it is NP-hard to compute

any multiplicative approximation to the CVaR of a submodular set function.

We experimentally evaluate our algorithm for sensor resource allocation, focusing on two domains: detecting contagion or rumors in social networks, and detecting contamination in water networks. In both cases, our algorithm substantially outperforms baselines.

Problem description

In this section, we formally define continuous submodularity and the conditional value at risk. We first study the continuous setting and then extend our results to discrete portfolio optimization.

Continuous submodularity: Let $\mathcal{X} = \prod_{i=1}^{n} \mathcal{X}_i$ be a subset of R^n, where each $\mathcal{X}_i$ is a compact subset of R. A twice-differentiable function $F : \mathcal{X} \to R$ is *diminishing returns submodular* (DR-submodular) if for all $\boldsymbol{x} \in \mathcal{X}$ and all $i, j = 1...n$, $\frac{\partial^2 F(\boldsymbol{x})}{\partial x_i \partial x_j} \leq 0$ (Bian et al. 2017). Intuitively, the gradient of F only shrinks as $\boldsymbol{x}$ grows, just as the marginal gains of a submodular set function only decrease as items are added. Continuous submodular functions need not be convex or concave (concavity requires that the Hessian is negative semi-definite, not that the individual entries are nonpositive). We consider *monotone* functions, where $F(\boldsymbol{x}) \leq F(\boldsymbol{y})\ \forall \boldsymbol{x} \preceq \boldsymbol{y}$ ($\preceq$ denotes element-wise inequality). We assume that F lies in $[0, M]$ for some constant M. Without loss of generality, we assume $F(0) = 0$ (normalization).

In our setting F is a function of both the decision variables $\boldsymbol{x}$ and a random parameter y. Specifically, we consider functions $F(\boldsymbol{x}, y)$ where $F(\cdot, y)$ is continuous submodular in $\boldsymbol{x}$ for each fixed y. We allow any DR-submodular F which satisfies some standard smoothness conditions. First, we assume that F is L_1-Lipschitz for some constant L_1 (for concreteness, with respect to the ℓ_2 norm[1]). Second, we assume that F is twice differentiable with L_2-Lipschitz gradient. Third, we assume that F has bounded gradients, $||\nabla F||_2 \leq G$. Only the last condition is strictly necessary; our approach can be extended to any F with bounded gradients via known techniques (Duchi, Bartlett, and Wainwright 2012).

Conditional value at risk: Intuitively, the CVaR measures performance in the α worst fraction of cases. First, we define the *value at risk* at level $\alpha \in [0, 1]$:

$$\text{VaR}_\alpha(\boldsymbol{x}) = \inf\{\tau \in R : \Pr_y [F(\boldsymbol{x}, y) \leq \tau] \geq \alpha\}.$$

That is, $\text{VaR}_\alpha(\boldsymbol{x})$ is the α-quantile of the random variable $F(\boldsymbol{x}, y)$. CVaR is the expectation of $F(\boldsymbol{x}, y)$, conditioned on it falling into this set of α-worst cases:

$$\text{CVaR}_\alpha(\boldsymbol{x}) = \mathop{\mathbb{E}}_{y} [F(\boldsymbol{x}, y) | F(\boldsymbol{x}, y) \leq \text{VaR}_\alpha(\boldsymbol{x})]\,.$$

[1]We use the ℓ_2 norm for concreteness. However, our arguments easily generalize to any ℓ_p norm.

CVaR is a more popular risk measure than VaR both because it counts the impact of the entire α-tail of the distribution and because it has better mathematical properties (Rockafellar and Uryasev 2000).

Optimization problem: We consider the problem of maximizing $\text{CVaR}_\alpha(\boldsymbol{x})$ over $\boldsymbol{x}$ belonging to some feasible set $\mathcal{P}$. We allow $\mathcal{P}$ to be any downward closed polytope. A polytope is downward closed if there is a lower bound $\boldsymbol{\ell}$ such that $\boldsymbol{x} \succeq \boldsymbol{\ell}\ \forall \boldsymbol{x} \in \mathcal{P}$ and for any $\boldsymbol{y} \in \mathcal{P}$, $\boldsymbol{\ell} \preceq \boldsymbol{x} \preceq \boldsymbol{y}$ implies that $\boldsymbol{x} \in \mathcal{P}$. Without loss of generality, we assume that $\mathcal{P}$ is entirely nonnegative with $\boldsymbol{\ell} = 0$. Otherwise, we can define the translated set $\mathcal{P}' = \{\boldsymbol{x} - \boldsymbol{\ell} : \boldsymbol{x} \in \mathcal{P}\}$ and corresponding function $F'(\boldsymbol{x}, y) = F(\boldsymbol{x} - \boldsymbol{\ell}, y)$. Let $d = \max_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{P}} ||\boldsymbol{x} - \boldsymbol{y}||_2$ be the diameter of $\mathcal{P}$.

We want to solve the problem $\max_{\boldsymbol{x} \in \mathcal{P}} \text{CVaR}_\alpha(\boldsymbol{x})$. It is important to note that $\text{CVaR}_\alpha(\boldsymbol{x})$ need *not* be a smooth DR-submodular function in $\boldsymbol{x}$. However, we would like to leverage the nice properties of the underling F. Towards this end, we note that the above problem can be rewritten in a more useful form (Rockafellar and Uryasev 2000). Let $[t]^+ = \max(t, 0)$. Maximizing $\text{CVaR}_\alpha(\boldsymbol{x})$ is equivalent to solving

$$\max_{\boldsymbol{x} \in \mathcal{P}, \tau \in [0, M]} \tau - \frac{1}{\alpha} \mathbb{E}\left[[\tau - F(\boldsymbol{x}, y)]^+\right] \tag{1}$$

where τ is an auxiliary parameter. For any fixed $\boldsymbol{x}$, the optimal value of τ is $\text{VaR}_\alpha(\boldsymbol{x})$ (Rockafellar and Uryasev 2000). It is known that when $F(\cdot, y)$ is *concave* in $\boldsymbol{x}$, this is a concave optimization problem. However, little is known when F may be nonconcave.

Related work

CVaR enjoys widespread popularity as a risk measure in many domains, ranging from finance (Mansini, Ogryczak, and Speranza 2007) to electricity production (Yau et al. 2011). More broadly, there is a burgeoning interest in methods which move beyond expected performance (Ermon et al. 2011; Yin et al. 2011; Yu and Nikolova 2013; Hoy and Nikolova 2015). Oftentimes, this concern is motivated by safety-critical domains where an algorithm designer must be able to minimize the risk of disastrous events, not just guarantee good results on average. Here, we survey the closest related work, dealing with CVaR optimization.

Rockafellar and Uryasev (2000) introduced CVaR and proposed a linear program for optimizing it. This linear program only applies when utility is linear in the decision variables. Iyengar and Ma (2013) and Hong and Liu (2009) present faster gradient-based algorithms for the linear case. Here, we deal with nonlinear functions. The LP approach can be extended via solving a general concave program when the utilities are concave. Our main contribution is extending the range of optimizable functions to include nonconcave continuous submodular objectives. Another body of work focuses on CVaR in reinforcement learning and MDPs (Prashanth and Ghavamzadeh 2013; Tamar et al. 2015; Chow et al. 2015). Lastly, (Ohsaka and Yoshida 2017) study CVaR for discrete influence maximization; we contrast our

results with theirs when we discuss the discrete portfolio setting.

Preliminaries

We now review techniques for optimizing smooth continuous submodular functions. These do not directly apply to CVaR, but our solution builds on them. An important property is that continuous submodular functions are concave along nonnegative directions. Formally,

Definition 1. A function $F(\boldsymbol{x})$ is *up-concave* if for any $\xi \in [0,1]$ and $\boldsymbol{y} \in \mathcal{P}$, $F(\boldsymbol{x} + \xi \boldsymbol{y})$ is concave in ξ.

All continuous submodular functions are up-concave (Bian et al. 2017). Monotone up-concave algorithms are optimized via a modified Frank-Wolfe algorithm (Bian et al. 2017; Calinescu et al. 2011). Frank-Wolfe is a gradient-based algorithm originally introduced to maximize concave functions. Consider an objective F. Frank-Wolfe algorithms start at an initial point $\boldsymbol{x}^0 \in \mathcal{P}$ and then generate a series of feasible solutions $\boldsymbol{x}^1 ... \boldsymbol{x}^K$ for some number of iterations K. At each step k, the algorithm calculates the gradient at the current point, $\nabla F(\boldsymbol{x}^{k-1})$. It then takes a step towards the point $\boldsymbol{v}^k \in \mathcal{P}$ which lies furthest in the direction of the gradient. That is, $\boldsymbol{v}^k$ is the solution to the linear optimization problem $\arg\max_{\boldsymbol{v} \in \mathcal{P}} \langle \boldsymbol{v}, \nabla F(\boldsymbol{x}^{k-1}) \rangle$. In the standard Frank-Wolfe algorithm for concave functions, the algorithm then updates to a convex combination of $\boldsymbol{x}^{k-1}$ and $\boldsymbol{v}^k$ by setting $\boldsymbol{x}^k = \boldsymbol{x}^{k-1} + \gamma_k \left(\boldsymbol{v}^k - \boldsymbol{x}^{k-1} \right)$ for some step size γ_k. Note that some entries of $\boldsymbol{x}^k$ may be smaller than the corresponding entries of $\boldsymbol{x}^{k-1}$. This is necessary for optimality: the algorithm may need to backtrack if it has made some entry too large.

This update rule does not work for up-concave functions because the objective is not concave along negative directions. Hence, the update for the modified Frank-Wolfe algorithm is $\boldsymbol{x}^k = \boldsymbol{x}^{k-1} + \gamma_k \boldsymbol{v}^k$, which only increases each coordinate. Because the algorithm is unable to backtrack, it achieves a $(1 - 1/e)$-approximation instead of the global optimum which is achievable for fully concave functions. The process is analogous to the greedy algorithm for submodular set functions, which successively includes elements based on their current marginal gain. The continuous Frank-Wolfe algorithm instead successively increases entries in the solution vector based on the current gradient.

Algorithmic approach

We now introduce the RASCAL (Risk Averse Submodular optimization via Conditional vALue at risk) algorithm for continuous submodular CVaR optimization. RASCAL solves Problem 1, which is a function of both the decision variables $\boldsymbol{x}$ and the auxiliary parameter τ. Roughly, τ should be understood as a threshold maintained by the algorithm for what constitutes a "bad" scenario: at each iteration, RASCAL tries to increase $F(\boldsymbol{x}, y)$ for those scenarios y such that $F(\boldsymbol{x}, y) \leq \tau$.

Before describing the optimization algorithm more formally, we deal with the challenge that the expectation in Problem 1 cannot generally be evaluated in closed form. We replace the expectation with the average of a set of sampled scenarios. Suppose that we draw a set of samples $y_1 ... y_s$ i.i.d from D. Call the set of samples $\mathcal{Y}$. Then we can estimate $\mathbb{E}\left[[\tau - F(\boldsymbol{x}, y)]^+ \right] \approx \frac{1}{s} \sum_{y \in \mathcal{Y}} [\tau - F(\boldsymbol{x}, y)]^+$. With sufficiently many samples, this approximation will be accurate to any desired level of accuracy:

Lemma 1. *Take* $s = O\left(\frac{nM^2}{\epsilon^2} \log \frac{1}{\delta} \log \frac{L_1}{\epsilon} \right)$ *samples and let* $\widehat{CVaR_\alpha}$ *be the empirical CVaR on the samples. Then,* $|CVaR_\alpha(\boldsymbol{x}) - \widehat{CVaR_\alpha}(\boldsymbol{x})| \leq \frac{\epsilon}{3}$ *holds for all* $\boldsymbol{x} \in \mathcal{P}$ *with probability at least* $1 - \delta$.

The proof is in the supplement. As a minor technicality, we assume that $F(\boldsymbol{x}, y_i)$ takes a distinct value for each $\boldsymbol{x}$ and $y_i \in \mathcal{Y}$ so that an exact α-quantile exists. This is without loss of generality since we can always add an arbitrarily small "tie breaker" value r_i, using $F(\boldsymbol{x}, y_i) + r_i$ instead.

We can now formally introduce RASCAL (Algorithm 1). RASCAL maximizes the objective $H(\boldsymbol{x}, \tau) = \tau - \frac{1}{\alpha s} \sum_{y \in \mathcal{Y}} [\tau - F(\boldsymbol{x}, y)]^+$. Maximizing H is equivalent to maximizing the sampled CVaR. RASCAL is a coordinate ascend style algorithm. Each iteration first makes a Frank-Wolfe style update to $\boldsymbol{x}$ (lines 4-6). This step assumes access to a linear optimization oracle LO which maximizes a given linear function over $\mathcal{P}$. RASCAL then sets τ to its

Algorithm 1 RASCAL

Require: K, u, s, LO
1: $\mathcal{Y} \leftarrow s$ samples i.i.d. from D
2: $\boldsymbol{x}^0 \leftarrow 0$, $\tau \leftarrow 0$
3: **for** $k = 1 ... K$ **do**
4: $\quad \tilde{\nabla} \leftarrow \text{SMOOTHGRAD}(\boldsymbol{x}^{k-1}, \tau, u)$
5: $\quad \boldsymbol{v} \leftarrow LO(\tilde{\nabla})$
6: $\quad \boldsymbol{x}^k \leftarrow \boldsymbol{x}^{k-1} + \frac{1}{K} \boldsymbol{v}$
7: $\quad \tau \leftarrow \text{SMOOTHTAU}(\boldsymbol{x}^{k-1}, u)$
8: **end for**
9: **return** $\boldsymbol{x}^K$
10:
11: **function** SMOOTHGRAD($\boldsymbol{x}$, τ, u)
12: $\quad I_y(\tau) \leftarrow \max(\min(\frac{F(\boldsymbol{x}, y) - \tau}{u}, 1), 0) \, \forall y \in \mathcal{Y}$
13: $\quad$ **return** $\sum_{y \in \mathcal{Y}} I_y(\tau) \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, y)$
14: **end function**
15:
16: **function** SMOOTHTAU($\boldsymbol{x}$, u)
17: $\quad \mathcal{B} = \{F(\boldsymbol{x}, y) | y \in \mathcal{Y}\} \cup \{F(\boldsymbol{x}, y) + u | y \in \mathcal{Y}\}$
18: $\quad$ Sort $\mathcal{B}$ in ascending order, obtaining $\mathcal{B} = \{b_1 ... b_{|\mathcal{B}|}\}$.
19: $\quad i^* = \min\{i = 1 ... |\mathcal{B}| : g(b_i) > \alpha s\}$
20: $\quad A \leftarrow \{y \in \mathcal{Y} : b_{i^*-1} < F(\boldsymbol{x}, y) < b_{i^*}\}$
21: $\quad C \leftarrow \{y \in \mathcal{Y} : F(\boldsymbol{x}, y) \leq b_{i^*-1}\}$
22: $\quad$ Return the τ which solves the linear equation

$$\sum_{y \in A} \frac{F(\boldsymbol{x}, y) - \tau}{u} + |C| = \alpha s$$

23: **end function**

optimal value given the current $\boldsymbol{x}$ (line 7). This approach is motivated by the unique properties of H. It can be shown that H is jointly up-concave in the variable $(\boldsymbol{x}, \tau)$. However, H is not monotone in τ. Indeed, H is decreasing in τ for $\tau > \text{VaR}_\alpha(\boldsymbol{x})$. The Frank-Wolfe algorithm relies crucially on monotonicity; nonmonotonicity is much more difficult to handle.

Instead, we exploit a unique form of structure in H. Specifically, H is monotone in $\boldsymbol{x}$, but only up-concave (not fully concave). Conversely, while H is nonmonotone in τ, we can easily solve the one-dimensional problem $\max_{\tau \in [0,M]} H(\boldsymbol{x}, \tau)$ for any fixed $\boldsymbol{x}$ (we explain how later). Our approach makes use of both properties: the Frank-Wolfe update leverages monotone up-concavity in $\boldsymbol{x}$, while the update to τ leverages easy solvability of the one-dimensional subproblem.

In order to make this approach work, two ingredients are necessary. First, we need access to the gradient of H in order to implement the Frank-Wolfe update for $\boldsymbol{x}$. Unfortunately, H is not even differentiable everywhere. We instead present a smoothed estimator SMOOTHGRAD which restores differentiability at the cost of introducing a controlled amount of bias. Second, we need to solve the one-dimensional problem of finding the optimal value of τ. We in fact introduce a subroutine SMOOTHTAU which solves a smoothed version of the optimal τ problem.

Smoothed gradient: We now calculate the gradient of the objective with respect to $\boldsymbol{x}$, $\nabla_{\boldsymbol{x}} H(\boldsymbol{x}, \tau)$. Essentially, H counts the value of all scenarios y for which $F(\boldsymbol{x}, y) \leq \tau$. If $F(\boldsymbol{x}, y) \neq \tau \, \forall y \in \mathcal{Y}$ then

$$\nabla_{\boldsymbol{x}} H(\boldsymbol{x}, \tau) = \frac{1}{\alpha s} \sum_{y \in \mathcal{Y}: F(\boldsymbol{x}, y) \leq \tau} \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, y).$$

Unfortunately, if there is a $y \in \mathcal{Y}$ such that $F(\boldsymbol{x}, y) = \tau$, then H may not be differentiable at $\boldsymbol{x}$. To see this, consider the directional derivatives from two different directions. From a nonpositive direction, $F(\boldsymbol{x}, y)$ is always below τ and hence will count towards the gradient. From a positive direction, $F(\boldsymbol{x}, y)$ may lie above τ in which case its contribution will be zero. Frank-Wolfe algorithms require differentiability (in fact, they require a Lipschitz gradient). This is not a minor technical point: if the gradient can radically change over small regions, then gradient-based updates may prove fruitless. Thus, RASCAL uses a smoothed gradient estimate over the region from τ to $\tau + u$ for some small $u > 0$:

$$\text{SMOOTHGRAD}(\boldsymbol{x}, \tau) = \frac{1}{u} \int_{z=0}^{u} \nabla_{\boldsymbol{x}} H(\boldsymbol{x}, \tau + z) dz$$

The intuition is that we average over a small window of τ values so that the contribution of a given scenario to the gradient does not suddenly drop to 0 if $\boldsymbol{x}$ increases slightly. Note that as we have sampled a finite set of s scenarios, the set of points at which H is not differentiable has measure 0. Hence, the integral exists. We now show how to exactly evaluate the integral (see Algorithm 1, lines 11-14 for pseudocode). We have

$$\begin{aligned}
&\frac{1}{u} \int_{z=0}^{u} \nabla_{\boldsymbol{x}} H(\boldsymbol{x}, \tau + z) dz \\
&= \frac{1}{u} \int_{z=0}^{u} \sum_{y \in \mathcal{Y}} 1\left[F(\boldsymbol{x}, y) \leq \tau + z\right] \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, y) dz \\
&= \sum_{y \in \mathcal{Y}} \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, y) \int_{z=0}^{u} \frac{1}{u} 1\left[F(\boldsymbol{x}, y) \leq \tau + z\right] dz
\end{aligned}$$

where $1[\cdot]$ is the indicator function. Now value of the inner integral is just $\max(\min(\frac{F(\boldsymbol{x}, y) - \tau}{u}, 1), 0)$. Call this value $I_y(\tau)$. By the above, $\text{SMOOTHGRAD}(\boldsymbol{x}, \tau) = \sum_{y \in \mathcal{Y}} I_y(\tau) \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, y)$. This can be computed in time $O(s(T_1 + T_2))$, where T_1 is the time to evaluate F and T_2 is the time to differentiate it.

Finding the optimal $\boldsymbol{\tau}$: The update SMOOTHTAU sets τ to its optimal value over a smoothed window of size u (in order to match SMOOTHGRAD). Specifically, we find the τ minimizing $\frac{1}{u} \int_{z=0}^{u} H(\boldsymbol{x}, \tau) dz$. Recall that for the unsmoothed H, the optimal setting for τ is $\text{VaR}_\alpha(\boldsymbol{x})$, i.e., the value such that F takes value at most τ in an α-fraction of scenarios. An analogous property holds for the smoothed version:

Lemma 2. *Define $g(\tau) = \sum_{y \in \mathcal{Y}} I_y(\tau)$. (a) τ maximizes $\frac{1}{u} \int_{z=0}^{u} H(\boldsymbol{x}, \tau) dz$ if $g(\tau) = \alpha s$. (b) g is piecewise linear and monotone decreasing.*

In Lemma 2(a), the condition $g(\tau) = \alpha s$ expresses that an α-fraction of the scenarios weighted by $I_y(\tau)$ should have $F(\boldsymbol{x}, y) \leq \tau$. The key property for efficiently finding the τ which satisfies this condition is given in Lemma 2(b): g is piecewise linear and monotone decreasing in τ. This follows since it is the sum of functions which share these properties (the I_y). SMOOTHTAU (Algorithm 1, lines 16-23) uses these properties as follows. The breakpoints of g are $F(\boldsymbol{x}, y)$ and $F(\boldsymbol{x}, y) + u$ for each $y \in \mathcal{Y}$ (line 17). Let these breakpoints $\mathcal{B} = \{b_1 ... b_{2s}\}$ be sorted in ascending order. We can find the τ such that $g(\tau) = \alpha s$ by first finding the interval such that $g(b_i) \leq \alpha s \leq g(b_{i+1})$ (line 19). Within this interval, g is linear and hence we can solve exactly for the desired point (lines 20-22). This process takes time $O(sT_1)$.

Theoretical analysis

We now prove that by taking appropriate choices for the smoothing parameter u and the number of steps K, RASCAL efficiently obtains a provably approximate solution. Our main theoretical result is as follows:

Theorem 1. *For any $\epsilon > 0$, by taking $u = \frac{\epsilon}{3\left(1 + \frac{1}{\alpha}\right)}$, RASCAL outputs a solution $\boldsymbol{x} \in \mathcal{P}$ satisfying CVaR$_\alpha(\boldsymbol{x}) \geq (1 - 1/e)OPT - \epsilon$ with probability at least $1 - \delta$. There are $K = O\left(\frac{L_2 d^2}{\alpha \epsilon} + \frac{L_1 G d^2}{\alpha^2 \epsilon^2}\right)$ iterations, requiring $O(sK)$ total evaluations of F, $O(sK)$ evaluations of ∇F, and K calls to LO.*

The rest of this section is devoted to proving Theorem 1. We start out by introducing a surrogate objective that we consider for the sake of analysis. Let

$$\tilde{H}(\boldsymbol{x},\tau) = \frac{1}{u}\int_{z=0}^{u} H(\boldsymbol{x},\tau+z)dz.$$

This is the smoothed version of the objective, which SMOOTHGRAD computes the gradient for. Let $\tau(x) = \max_\tau \tilde{H}(\boldsymbol{x},\tau)$ be the optimal setting for τ under $\boldsymbol{x}$. Note that this is with respect to the smoothed objective $\tilde{H}$, so $\tau(\boldsymbol{x})$ is not necessarily $\text{VaR}_\alpha(\boldsymbol{x})$. We first show that H and $\tilde{H}$ are close:

Lemma 3. $|\tilde{H}(\boldsymbol{x},\tau) - H(\boldsymbol{x},\tau)| \leq \frac{u\left(1+\frac{1}{\alpha}\right)}{2} \;\; \forall \boldsymbol{x},\tau$

The main idea is to show that H is Lipschitz with respect to τ, so we do not change the value of the function too much by changing τ slightly. This lemma essentially bounds the bias introduced by SMOOTHGRAD.

Now we turn to the main step: showing that the coordinate ascent strategy makes an appropriate amount of progress towards the optimum at each iteration. Note that at the end of each iteration k, RASCAL sets $\tau^k \leftarrow \tau(\boldsymbol{x}^k)$. This is because SMOOTHTAU exactly computes the optimal setting for τ with respect to the smoothed objective $\tilde{H}$. Let $\tilde{\boldsymbol{x}}^* = \max_{\boldsymbol{x}\in\mathcal{P}} \tilde{H}(\boldsymbol{x},\tau(\boldsymbol{x}))$ be the point achieving the optimal value of $\tilde{H}$. Since RASCAL always sets τ to its optimal value in SMOOTHTAU, the gap from optimality at the end of iteration k is exactly

$$\Delta^k := \tilde{H}(\tilde{\boldsymbol{x}}^*,\tau(\tilde{\boldsymbol{x}}^*)) - \tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k))$$

Our aim is to show that the gap Δ^k decreases by a factor of $(1-\gamma_k)$ at each iteration (up to a small amount of additive loss). We start out by providing an upper bound on Δ^k in terms of the current gradient.

Lemma 4. *At each iteration $k = 1...K$,*

$$\tilde{H}(\tilde{\boldsymbol{x}}^*,\tau(\tilde{\boldsymbol{x}}^*)) - \tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)) \leq \max_{\boldsymbol{v}\in\mathcal{P}} \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)),\boldsymbol{v}\rangle.$$

The proof uses the underlying up-concavity of F combined with the concavity-preserving properties of CVaR. The intuition is that any concave function is upper bounded by its linearization at a given point (though the bound is weaker than for concave functions (Lacoste-Julien and Jaggi 2015) because F is only up-concave). Lemma 4 gives us a benchmark to track progress: it suffices to show that the improvement in iteration k is at least $\gamma_k \max_{\boldsymbol{v}\in\mathcal{P}} \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)),\boldsymbol{v}^k\rangle$ since this implies that we make up at least a γ_k fraction of the current gap from optimality.

We now express the actual improvement that is made. At iteration k, the Frank-Wolfe update moves from $\boldsymbol{x}^{k-1}$ to $\boldsymbol{x}^{k-1} + \gamma_k \boldsymbol{v}^k$. Integrating over the transition between these two points gives

$$\tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)) - \tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})) = \int_{\xi=0}^{1} \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1}+\xi\gamma_k\boldsymbol{v},\tau(\boldsymbol{x}^{k-1}+\xi\gamma_k\boldsymbol{v})),\gamma_k\boldsymbol{v}\rangle d\xi. \quad (2)$$

What we would like is for the gradient to stay relatively constant as we move from $\boldsymbol{x}^{k-1}$ to $\boldsymbol{x}^{k-1}+\gamma_k\boldsymbol{v}^k$. This is because we chose $\boldsymbol{v}^k$ to lie in the direction of $\nabla_{\boldsymbol{x}}\tilde{H}$ at the starting point $\boldsymbol{x}^{k-1}$. If the gradient changes very sharply along the way, then we may not not actually improve the objective value very much.

There are two obstacles to showing that the gradient is smooth enough. The first is that the value of τ in Equation 2 may change with ξ. We can deal with this as follows. Note that that since $\boldsymbol{v}^k$ is nonnegative, $\boldsymbol{x}^{k-1} + \xi\gamma_k\boldsymbol{v}^k \succeq \boldsymbol{x}^{k-1}$ holds for all $\xi \in [0,1]$. It is easy to see that $\tau(\boldsymbol{x})$ is monotone increasing in $\boldsymbol{x}$. Thus, $\tau(\boldsymbol{x}^{k-1}+\xi\gamma_k\boldsymbol{v}) \geq \tau(\boldsymbol{x}^k)$. By looking at the expression for $\nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x},\tau)$, we can see if that if we increase the value of τ, then the gradient can only increase because more scenarios can contribute. Formally,

Lemma 5. *If $\boldsymbol{x}_2 \succeq \boldsymbol{x}_1$, $\nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}_2,\tau(\boldsymbol{x}_2)) \succeq \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}_2,\tau(\boldsymbol{x}_1))$.*

Applying Lemma 5 to Equation 2 gives

$$\tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)) - \tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})) \geq \int_{\xi=0}^{1} \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1}+\xi\gamma_k\boldsymbol{v},\tau(\boldsymbol{x}^{k-1})),\gamma_k\boldsymbol{v}\rangle d\xi$$

The second obstacle is that $\nabla_{\boldsymbol{x}}\tilde{H}$ might change sharply as we vary $\boldsymbol{x}$ from $\boldsymbol{x}^{k-1}$ to $\boldsymbol{x}^{k-1}+\gamma_k\boldsymbol{v}^k$. However, this is exactly what SMOOTHGRAD is designed to avoid. Formally, the gradient of $\tilde{H}$ is Lipschitz:

Lemma 6. *If $\forall y \in \mathcal{Y}$, $F(\cdot,y)$ is L_1-Lipschitz and $\nabla_{\boldsymbol{x}}F(\cdot,y)$ is L_2 Lipschitz with $||\nabla_{\boldsymbol{x}}F||_2 \leq G$, then $\nabla_{\boldsymbol{x}}\tilde{H}$ is $\frac{1}{\alpha}\left(L_2 + \frac{L_1 G}{u}\right)$ −Lipschitz.*

This gives us the tools to finish the proof. Let $C = \frac{1}{\alpha}\left(L_2 + \frac{L_1 G}{u}\right)$ be the Lipschitz constant of $\nabla_{\boldsymbol{x}}\tilde{H}$. The Cauchy-Shwartz inequality and Lemma 6 yield

$$\begin{aligned}&\langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1}+\xi\gamma_k\boldsymbol{v},\tau(\boldsymbol{x}^{k-1})),\boldsymbol{v}\rangle \\ &\geq \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})),\boldsymbol{v}\rangle - \xi\gamma_k C||\boldsymbol{v}||_2^2\end{aligned}$$

and hence

$$\begin{aligned}&\tilde{H}(\boldsymbol{x}^k,\tau(\boldsymbol{x}^k)) - \tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})) \\ &\geq \gamma_k \int_0^1 \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})),\boldsymbol{v}\rangle - \xi\gamma_k C||\boldsymbol{v}||_2^2 d\xi \\ &= \gamma_k \langle \nabla_{\boldsymbol{x}}\tilde{H}(\boldsymbol{x}^{k-1},\tau(\boldsymbol{x}^{k-1})),\boldsymbol{v}\rangle - \frac{\gamma_k^2 C||\boldsymbol{v}||_2^2}{2} \\ &\geq \gamma_k\Delta^{k-1} - \frac{\gamma_k^2 C d^2}{2}\end{aligned}$$

and by rearranging we obtain

$$\Delta^k \leq (1-\gamma_k)\Delta^{k-1} - \frac{\gamma_k^2 C d^2}{2}.$$

This is exactly what we wanted to show: the gap shrinks by a factor $(1-\gamma_k)$ each iteration, up to a small amount of additive loss. From here, the proof proceeds by fairly standard arguments which may be found in the supplement.

Discrete portfolio optimization

We may also want to optimize the CVaR of a submodular *set* function, as opposed to the continuous functions that we have dealt with so far. We study the portfolio optimization problem (Ohsaka and Yoshida 2017) where the decision maker may select any *distribution* over feasible sets. Equivalently, they select a decision which is a convex combination of feasible decisions but which is not guaranteed to lie in the original feasible set itself (Chen et al. 2017). This is a natural setting for CVaR optimization because the decision maker essentially hedges their bets between multiple options.

Formally, we are given a collection of submodular set functions $f(\cdot, y)$ on a ground set X, where y is a random variable. There is a collection of feasible sets $\mathcal{I}$. For instance, $\mathcal{I}$ could be all size-k subsets. In general, our algorithm works when $\mathcal{I}$ is any matroid. The algorithm selects a distribution q over the sets in $\mathcal{I}$. The objective is to maximize $\text{CVaR}_\alpha\left(\sum_{S\in\mathcal{I}} q_S f(S, y)\right)$.

We provide a black-box reduction from this problem to the continuous submodular CVaR optimization problem considered earlier. Since RASCAL solves the continuous problem, we immediately obtain efficient algorithms for a range of portfolio problems. Formally,

Theorem 2. *Given access to an α-approximation algorithm for the continuous CVaR problem, there is an algorithm which obtains value at least $\alpha OPT - \epsilon$ for the discrete portfolio CVaR problem.*

A proof is deferred to the supplement. The main idea is to translate from the discrete to continuous settings via the multilinear extension (Calinescu et al. 2011). The multilinear extension F of a submodular set function f is a continuous function defined on the hypercube $[0, 1]^{|X|}$ which agrees with f at the vertices. We apply the promised continuous CVaR algorithm to the multilinear extensions $F(\cdot, y)$ and then use known rounding techniques (Chekuri, Vondrak, and Zenklusen 2010) to convert the fractional solution to a distribution over integral points which preserves the fractional solution's CVaR value. However, some additional technical steps are needed to make this strategy work (e.g., we need to maintain multiple copies of the decision variables to get the optimal approximation ratio).

We note that this result strengthens that of Ohsaka and Yoshida (2017) in two respects. First, their result applies only to influence maximization, while ours applies to any submodular function. Second, they obtain the additive approximation $OPT - \frac{1}{e}$ when the objective values are rescaled by n (the total number of nodes in the graph for influence maximization) to the interval $[0, 1]$. Hence, their bound does not apply when $OPT \leq \frac{1}{e}n$, which is very possible since CVaR counts worst-case outcomes. We have only an arbitrarily small ϵ of additive loss, which allows for stronger guarantees when OPT is small.

Experiments

We show experimental results for the sensor resource allocation problem, where the goal is to use a limited sensing budget to quickly detect a contagion spreading through a network (Leskovec et al. 2007; Soma and Yoshida 2015; Bian et al. 2017). We are given a graph $G = (V, E)$ with $|V| = n$. A contagion starts at a random node y and spreads over time according to a given stochastic process. Let t_v be the time at which the contagion reaches each node v. t_v is a random variable which depends on both the source node y and the stochastic contagion process. The vector $\boldsymbol{t}$ collects t_v for all $v \in V$. We assume that $t_v < \infty\ \forall v \in V$ (every node is eventually reached). If this does not hold, we can cut the process off after some large time horizon. Let t_∞ be the maximum possible value of t_v.

The decision maker has a budget B (e.g., energy) to spend on sensing resources. x_v represents the amount of energy allocated to the sensor at node v. When contagion reaches v at time t_v, the sensor detects with probability $1-(1-p)^{x_v}$ and otherwise fails. Essentially, investing an extra unit of energy in sensor v buys an extra chance to detect the contagion with probability p. Fix a vector of times $\boldsymbol{t}$, and order the nodes $v_1 ... v_n$ so that $t_{v_1} \leq t_{v_2} \leq ... \leq t_{v_n}$. The objective F for source y is expected amount of detection time that is saved by the sensor placements:

$$F(\boldsymbol{x}, \boldsymbol{t}) = t_\infty - \sum_{i=1}^{n} t_{v_i}\left(1-(1-p)^{x_i}\right)\prod_{j<i}(1-p)^{x_j}$$

where the summation counts the probability that sensor i succeeds but all $j < i$ fail. Is is known that F is DR-submodular (Bian et al. 2017). Previous work maximizes $\mathbb{E}_{\boldsymbol{t}}\left[F(\boldsymbol{x}, \boldsymbol{t})\right]$, the expected utility over the random source node and diffusion process. Here, we consider instead $\text{CVaR}_\alpha(\boldsymbol{x})$, where the scenarios are all possible time vectors $\boldsymbol{t}$. Essentially, we want to perform well when the contagion starts in hard to detect portions of the network or spreads in an unlikely way. We take the CVaR with respect to $\boldsymbol{t}$ but not the success or failure of the sensors because no algorithm can successfully detect contagion when almost all sensors fail *and* the source and diffusion pattern are worst-case.

Domains: We consider two sensing domains. In both, the source node is uniformly random. First, contagion spreading according to the continuous time independent cascade model (CTIC). This models applications like detecting news or a disease in a social network. The CTIC is variant of the independent cascade model proposed by Gomez-Rodriguez et al. (2012) which better reflects the temporal dynamics of real-world social processes. Each edge (u, v) has propagation time $\rho_{u,v}$ drawn from an exponential distribution with mean λ. The contagion starts at y ($t_y = 0$). Letting $\delta(v)$ be v's neighbors, $t_v = \min_{u\in\delta(v)} t_u + \rho_{u,v}$. That is, t_v is the first time contagion spreads from a neighbor to v.

We show experiments on several networks. First, *netscience*[2]: a collaboration network of network science researchers with 1461 nodes. Second, *euroroad*: a network of European cities and roads between them, with 1,174 nodes. Third, synthetic Watts-Strogatz networks (parameters k = 2, p = 0.1). These allow us to test our algorithm on a similar graphs as n grows. For all networks, $\lambda = 5$, $p = 0.01$, and we simulate 1000 scenarios (random source nodes and propagation times).

[2]http://www-personal.umich.edu/ mejn/netdata/

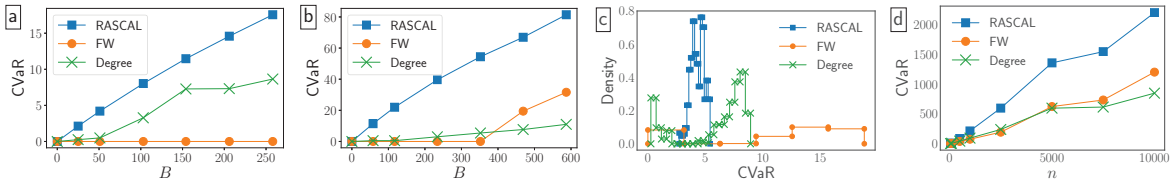

Figure 1: Results for the continuous time independent cascade model. (a) netscience as B varies (b) euroroad as B varies (d) histogram of values for netscience with $B = 0.1n$. (d) Watts-Strogatz networks as n varies

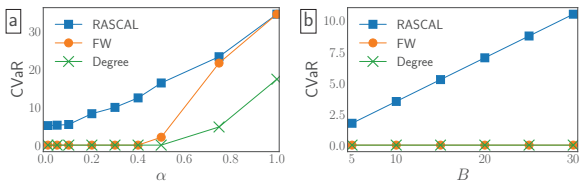

Figure 2: Results for BWSN

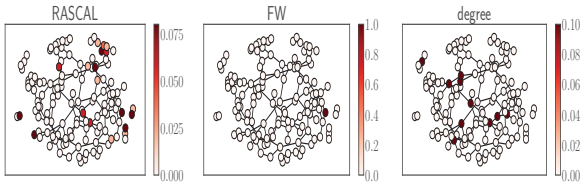

Figure 3: Example allocations for BWSN

Second, we consider detecting contamination in a water network via the Battle of Water Sensor Networks (BWSN). BWSM (Ostfeld et al. 2008) simulates the spread of contamination through a 126-node water network consisting of junctions, tanks, pumps, and the links between them. The network is a real water distribution network from an anonymous location, and the t values are provided by EPANET, a highly realistic water distribution simulator designed by the U.S. Environmental Protection Agency. We use $p = 0.001$ and simulate 1000 random scenarios (source node and t values).

Baselines: No previous work directly addresses our setting. We consider two competitive baselines. First, *FW*, which uses the Frank-Wolfe algorithm of Bian et al. (2017) to maximize the expected reward. Maximizing expected value is default approach to decision making under uncertainty. Second, *degree*, a heuristic for producing risk-averse solutions. Specifically, degree allocates one unit of budget to each of the B nodes with highest degree. This disperses the budget throughout the network, hedging against unlikely outcomes.

Results for CTIC: Figure 1 shows results under the CTIC. Figures 1(a) and 1(b) show the CVaR of each algorithm on the netscience and euroroad networks as the budget B varies on the x axis. RASCAL substantially outperforms both FW and Degree. This indicates that maximizing expected value is not a sufficient proxy for risk-aversion under uncertainty. In fact, FW obtains *zero* value for many values of B, indicating that its sensor selection is useless in the 10% worst cases. Degree often performs better than FW, indicating some benefit to heuristically hedging against possible contagion sources. However, RASCAL's principled optimization still results in much higher performance. Figure 1(c) shows a histogram of each algorithm's value across the different scenarios on netscience. RASCAL's reward distribution is tightly concentrated, which is desirable from the perspective of risk aversion. By contrast, FW and degree have more bimodal distributions, with the potential for both very low and high reward. Lastly, Figure 1(d) shows the CVaR obtained by each algorithm for Watts-Strogatz networks as the network size n grows on the x axis. RASCAL again obtains much higher value across the board. RASCAL scales easily to 10,000 nodes, running in under 1 minute.

Results for BWSN: We now examine our second domain, water network sensor management. Figure 2 shows the CVaR obtained by each algorithm. Figure 2(a) shows α on the x axis, varying the decision maker's degree of risk aversion. Throughout, $B = 10$. RASCAL substantially outperforms FW and degree until $\alpha = 0.6$, at which point FW becomes competitive. However, for $\alpha \leq 0.4$, both FW and degree obtain zero value. This indicates that even when the decision maker is not severely risk averse (e.g., preferring to focus on the worst 50% of scenarios), they can substantially benefit from using our principled approach to optimizing CVaR. It is natural to ask whether the baselines are competitive when there are more resources available, allowing them to cover a larger portion of the network. Figure 2(b) shows the results as the budget B is varied on the x axis with $\alpha = 0.1$. FW and degree still obtain a CVaR of zero even when the budget is tripled to $B = 30$. By contrast, RASCAL's value steadily grows as it makes productive use of the additional resources.

Lastly, Figure 3 shows an example of the allocation produced by each algorithm for $B = 10$, $\alpha = 0.1$. RASCAL disperses its resources throughout the network. It places some resources on central nodes, but also spends a portion of the budget on outlying parts of the network where contagions will not be detected by centrally placed sensors. On the other hand, FW concentrates is *entire* budget on one central node. Degree, by design, disperses its budget more widely. However, it spends the budget largely on central nodes, instead of balancing between central and outlying nodes like RASCAL. We conclude that RASCAL successfully balances different scenarios to find risk-averse solutions.

Acknowledgments: Wilder was supported by a NSF Graduate Fellowship.

References

Bach, F. 2015. Submodular functions: from discrete to continous domains. *arXiv preprint arXiv:1511.00394*.

Bian, A. A.; Mirzasoleiman, B.; Buhmann, J. M.; and Krause, A. 2017. Guaranteed non-convex optimization: Submodular maximization over continuous domains. In *AISTATS*.

Calinescu, G.; Chekuri, C.; Pál, M.; and Vondrák, J. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40(6):1740–1766.

Chekuri, C.; Vondrak, J.; and Zenklusen, R. 2010. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*.

Chen, R.; Lucier, B.; Singer, Y.; and Syrgkanis, V. 2017. Robust optimization for non-convex objectives. In *NIPS*.

Chow, Y.; Tamar, A.; Mannor, S.; and Pavone, M. 2015. Risk-sensitive and robust decision-making: a CVaR optimization approach. In *NIPS*, 1522–1530.

Duchi, J. C.; Bartlett, P. L.; and Wainwright, M. J. 2012. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization* 22(2):674–701.

Ermon, S.; Conrad, J.; Gomes, C. P.; and Selman, B. 2011. Risk-sensitive policies for sustainable renewable resource allocation. In *IJCAI*, 1942–1948.

Gomez-Rodriguez, M.; Leskovec, J.; and Krause, A. 2012. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5(4):21.

Hong, L. J., and Liu, G. 2009. Simulating sensitivities of conditional value at risk. *Manag. Sci* 55(2):281–293.

Hoy, D., and Nikolova, E. 2015. Approximately optimal risk-averse routing policies via adaptive discretization. In *AAAI*.

Iyengar, G., and Ma, A. K. C. 2013. Fast gradient descent method for mean-CVaR optimization. *Annals of Operations Research* 205(1):203–212.

Kempe, D.; Kleinberg, J.; and Tardos, É. 2003. Maximizing the spread of influence through a social network. In *KDD*.

Koçkesen, L.; Ok, E. A.; and Sethi, R. 2000. The strategic advantage of negatively interdependent preferences. *Journal of Economic Theory* 92(2):274–299.

Krause, A.; Roper, A.; and Golovin, D. 2011. Randomized sensing in adversarial environments. In *IJCAI*.

Kulesza, A., and Taskar, B. 2012. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning* 5(2–3):123–286.

Lacoste-Julien, S., and Jaggi, M. 2015. On the global linear convergence of Frank-Wolfe optimization variants. In *NIPS*.

Leskovec, J.; Krause, A.; Guestrin, C.; Faloutsos, C.; VanBriesen, J.; and Glance, N. 2007. Cost-effective outbreak detection in networks. In *KDD*, 420–429.

Maehara, T. 2015. Risk averse submodular utility maximization. *Operations Research Letters* 43(5):526–529.

Mansini, R.; Ogryczak, W.; and Speranza, M. G. 2007. Conditional value at risk and related linear programming models for portfolio optimization. *Annals of operations research* 152(1):227–256.

Ohsaka, N., and Yoshida, Y. 2017. Portfolio optimization for influence spread. In *WWW*, 977–985.

Ostfeld, A.; Uber, J. G.; Salomons, E.; Berry, J. W.; Hart, W. E.; Phillips, C. A.; Watson, J.-P.; et al. 2008. The battle of the water sensor networks (BWSN). *J. Water Resour. Plan. Manag.* 134(6):556–568.

Prashanth, L., and Ghavamzadeh, M. 2013. Actor-critic algorithms for risk-sensitive MDPs. In *NIPS*, 252–260.

Rockafellar, R. T., and Uryasev, S. 2000. Optimization of conditional value-at-risk. *Journal of risk* 2:21–42.

Sampson, T. 2016. Assignment reversals: Trade, skill allocation and wage inequality. *J. Econ. Theory* 163:365–409.

Soma, T., and Yoshida, Y. 2015. A generalization of submodular cover via the diminishing return property on the integer lattice. In *NIPS*, 847–855.

Staib, M., and Jegelka, S. 2017. Robust budget allocation via continuous submodular functions. In *ICML*.

Tamar, A.; Chow, Y.; Ghavamzadeh, M.; and Mannor, S. 2015. Policy gradient for coherent risk measures. In *NIPS*.

Vondrák, J. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, 67–74.

Wilder, B. 2017. Equilibrium computation and robust optimization in zero sum games with submodular structure. In *AAAI*.

Yau, S.; Kwon, R. H.; Rogers, J. S.; and Wu, D. 2011. Financial and operational decisions in the electricity sector. *Int J Prod Econ* 134(1):67–77.

Yin, Z.; Jain, M.; Tambe, M.; and Ordónez, F. 2011. Risk-averse strategies for security games with execution and observational uncertainty. In *AAAI*.

Yu, J. Y., and Nikolova, E. 2013. Sample complexity of risk-averse bandit-arm selection. In *IJCAI*, 2576–2582.