A primal-dual aggregation algorithm for minimizing Conditional-Value-at-Risk in linear programs

Daniel Espinoza* Eduardo Moreno †

August 19, 2013

Abstract

Recent years have seen growing interest in coherent risk measures, specially on Conditional Value-at-Risk (CVaR). Since CVaR is a convex function, it is suitable as an objective for optimization problems where we desire to minimize risk, specially when the underlying problem can be stated as a linear program. In the case that the underlying distribution has discrete support, this problem can be formulated as a linear programming (LP) problem. Over more general distributions, recent techniques, as the sampled average approximation, allows to well approximate the solution by solving a series of sampled problems. However, there are some cases where the number of samples required to obtain good approximated solutions tends to be very large, specially for extreme risk levels. This has boosted research on algorithms and formulations to solve these large-scale LP problems.

In this paper we propose an automatic primal-dual aggregation scheme to exactly solve these special structured LPs with a very large number of scenarios. The algorithm aggregates scenarios and constraints in order to solve a smaller problem, that is automatically disaggregated using the information of its dual variables. We compare this algorithm with other common approaches found in the literature, as improved formulation of the full problem, cut-generation schemes and other problem-specific approaches available in commercial software. Extensive computational experiments are performed on portfolio and general LP instances. These experiments show that the proposed algorithm result on a speed-up factor of 80 when compared with the best general LP-algorithm, and a speed-up factor of 3-20 when compared with other specialized algorithms.

1 Introduction

Dealing with uncertainty and risk has always been a mayor concern in optimization (see [5, 10, 26, 27, 32]), with many obvious applications. A first mathematical treatment of *risk* came with the work of Von Neumann and Morgenstern [31]; where, under some *rationality assumptions*, an existence of a *risk measure* for agents was assured, and linked risk attitude of an agent to the concavity of the utility function of each agent. A first *algorithmic* way to incorporate risk in decision making, is probably the seminal work of Markowitz [21], where variance of return is used as a proxy of risk, but is introduced more as a *reasonable* way to force diversification in portfolio optimization, than as a rigorous way to handle *risk* in general. However, an important appeal of the approach was the possibility of actually solving those class of problems. Another important development was the introduction of the *Dual Theory of Choice* of Yaari [33, 34], where risk aversion and diminishing returns on wealth can be seen as separated aspects, and introducing a precise notion of *certainty equivalent*. More recently, Artzner et al. [3, 4], motivated again by the portfolio problem, proposed the concept of *Coherent Risk Measures*, in the setting of discrete random variables and proposed, as an example, the *Conditional value at risk*. This was quickly followed by the work of Rockafellar and Uryasev [24, 23], where they present an *extended formulation* for the problem of evaluating CVaR, and furthermore, tested

*Department of Industrial Engineering, Universidad de Chile
†Faculty of Engineering and Science, Universidad Adolfo Ibañez,

their ideas on the classical portfolio problem with sampled data but using CVaR objective function and/or CVaR constraints. The importance of CVaR was further advanced by the result of Kusuoka [16] which characterized, in the continuous setting and under some extra conditions, all coherent, comonotone, law invariant risk measures (also known as *distortion risk measures* [6]) as *combinations* of CVaR measures; which makes CVaR a *basis* of all such risk measures. Interestingly enough, distortion risk measures, are closely related to risk-averse measures introduced early on by Yaari.

The *extended formulation* of CVaR, by Rockafellar and Uryasev, opened up the possibility of using all the approximation machinery from standard stochastic programs (See for example [25] for an in-depth reference, and [14, 20] for examples of their use); these results mean, that in theory, under some mild conditions on the domain of the optimization problem, and on the characteristics of the space of random variables, solving optimization problems with CVaR objectives, can be approximated by solving a series of *sampled* problems. In particular, for most common portfolio problems, this means solving a series of linear programs (LP). Unfortunately, there are examples [18] where using too few scenarios leads to strange behavior, prompting the need to solve these problems using large sets of scenarios. This has bolster research on algorithms and formulations to solve these special-structured LPs. Lim et. al [19] proposed a special-purpose solver for simple portfolio problems with CVaR objective functions. Later on, Ogryczak [22] compared different formulations for the same problem, showing important advantages of using the dual formulation for these LPs. And more recently, Künzi-bay and Mayer [15], proposed a specialized cut-generation approach (based on the L-shaped method [29]) for the problem.

In this paper we propose an automatic, exact, primal-dual aggregation method to solve general linear programs (as opposed to the classical portfolio problem) under a CVaR objective function that is able to deal with very large number of scenarios. We compare our method against general-purpose solvers, and other specialized approaches, under the general set of linear programs found in Netlib [11], as well as on portfolio problems; both exploring the issue of scalability, and sensibility to the *risk-parameter* α of the objective function. These experiments show that the proposed algorithm result on a speed-up factor of 80 when compared with the best general LP-algorithm, and a speed-up factor of 3-20 when compared with other specialized algorithms.

The rest of the paper is organized as follows: Section 2 defines CVaR, introduce linear programs with CVaR objective functions, its representation for the case of finite-support distributions, and the resulting LP problems. Section 3 present three algorithms from the literature to solve the problem, and the description of our algorithm. Section 4 present our computational experiments and results. Finally, in Section 5, we summarize our results and conclusions.

2 Linear programs with CVaR objective functions and discrete distributions

Given $x \in \mathbb{R}^n$, a linear *loss*-function $\hat{z}(x) = -\hat{c}x$, where $\hat{c}$ is a random parameter, and a probability distribution function $F_{\hat{z}}(\lambda) = \mathbb{P}(\hat{z} \leq \lambda)$, the Value-at-Risk (VaR) at level ε is defined as

$$\mathrm{VaR}_\varepsilon(\hat{z}) = \inf\{\lambda | F_{\hat{z}}(\lambda) \geq \varepsilon\}$$

Analogously, the conditional Value-at-Risk (CVaR) at level ε can be defined as

$$\mathrm{CVaR}_\varepsilon(\hat{z}) = \mathbb{E}(\hat{z} | \hat{z} \geq \mathrm{VaR}_\varepsilon(\hat{z})) = \min_t \left[t + \frac{1}{\varepsilon}\mathbb{E}((\hat{z} - t)^+) \right]$$

Note that exact computation of CVaR is not possible except for some particular distribution functions $F_{\hat{z}}$. However, if the underlying distribution is *discrete*, i.e. $\Omega = \{\omega_i\}_{i=1}^N$ and $\mathbb{P}(\hat{\omega} = \omega_i) = p_i$, then; the problem

$$(P) \min \mathrm{CVaR}_\varepsilon(-\hat{c}x) \tag{1a}$$

$$s.t.\ Ax = b \tag{1b}$$

$$x \geq 0, \tag{1c}$$

where $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^m$, $\hat{c} \in \Omega \to \mathbb{R}^n$ is a random variable and $\varepsilon \in]0,1]$; can be reformulated [24] as

$$(P_N)\ \min t + \frac{1}{\varepsilon}\sum_{i=1}^{N} p_i \eta_i \tag{2a}$$

$$s.t.\ Ax = b \tag{2b}$$

$$c^i x + t + \eta_i \geq 0 \quad \forall i \in \{1,\ldots,N\} \tag{2c}$$

$$x, \eta \geq 0 \tag{2d}$$

Note that Problem (2), requires just one extra variable and constraint for each event in Ω. This, together with the continuous advances in solving linear programs [8], which have brought the ability to speed up general LP-computations by a factor of more than 10^6 (and counting) from 1990-2000, have made this kind of models very attractive. However, as we will see, there are still many cases where computing times can be far too long, easily exceeding 20 hours of computation.

Note that Problem (2) is exactly the problem solved when we use a sampled approximation of CVaR for a continuous probability distribution. In this case, c^i is a sampled realization of the random variable $\hat{c}$ and all samples are equiprobable ($p_i = \frac{1}{N}$).

3 Special formulations and special algorithms

These long computation times have lead to several articles dealing on how to speed up the problem of solving these specially structured LPs. In this section we present three such alternatives; we selected them because we feel they are representative of the different schemes proposed so-far, but it is not an exhaustive list. We end this section by describing our proposed primal-dual aggregation method.

3.1 Improved formulations and general algorithms

A common agreement in the mixed-integer programming community, is that nowadays, if you only want to solve a large LP, chances are that the best available algorithm will be an interior point algorithm specialized for LPs [8].

However, Problem (2), specially for large N, have a very special structure; and as was already noted by Ogruczak et al. [22] for portfolio problems, the dual version of (2) can be written as

$$(D_N)\ \max y^t b \tag{3a}$$

$$s.t.\ y^t A + \frac{1}{\varepsilon}\sum_{i=1}^{N} p_i \lambda_i c^i \leq 0 \tag{3b}$$

$$\sum_{i=1}^{N} p_i \lambda_i = \varepsilon \tag{3c}$$

$$0 \leq \lambda_i \leq 1 \quad \forall i \in \{1\ldots N\}. \tag{3d}$$

Note that this problem has only $n+1$ constraints (other than bounds on variables), and $N+m+1$ variables. This observation is crucial, especially when we consider the "*observed* running time of the simplex algorithm in *real instances*", which according to Dantzig [9], and latter Todd [28], was in the order of $\mathcal{O}(r)$ (where r is the number of rows of the LP), and that, more recently is considered to be closer to $\mathcal{O}(r\sqrt{c})$ [12] (where c is the number of columns in the problem). This claim is not meant as a precise statement, nor as an exact complexity result, but as a practical observation. Under this light, the results obtained by Ogruczak et al. [22], are not surprising, but the impact on running time is dramatic. During our numerical results, we replicated the results reported by them, namely, that solving the dual problem using primal simplex greatly

outperform all other alternatives, even when compared against interior point algorithms; this last comparison will be reported in our computational results. Given these results, we will use this dual formulation for all problems without trying the primal version.

3.2 Cut-event algorithm

An alternative to solve Problem (1), is to change the representation of the (convex) and continuous objective function by its (infinite) sub-differential approximation; more precisely, we can solve problem

$$(O) \min f(x) \tag{4a}$$
$$s.t.\, x \in X, \tag{4b}$$

by solving problem

$$(O') \min z \tag{5a}$$
$$s.t.\, x \in X \tag{5b}$$
$$z \geq f(x_o) + d(x - x_o) \quad \forall x_o \in X, d \in \partial f(x_o), \tag{5c}$$

where $\partial f(x_o)$ is the set of sub-differentials of f at x_o. This problem is in general intractable, but in the case of Problem (2), is equivalent to

$$(\text{CVaR}') \min t + \frac{1}{\varepsilon} w \tag{6a}$$
$$s.t.\, Ax = b \tag{6b}$$
$$w + \sum_{i \in C} p_i (c^i x + t) \geq 0 \quad \forall C \subseteq \{1, \ldots, N\} \tag{6c}$$
$$x \geq 0. \tag{6d}$$

Although (6) still has an exponential number of constraints, it has the flavor of the *sub-tour elimination polytope* (SEP) of the TSP, which can be solved quite efficiently, even on very large scale problems [2] by using standard cut-generation methods. In the stochastic programming literature, this is known as the L-shaped method [29], and this idea was proposed by Künzi-Bay and Mayer [15] for minimizing CVaR. However, taking into account the observations of the previous section, in our implementation, we solve the dual problem

$$(\text{CutEv}) \max b'y \tag{7a}$$
$$s.t.\, A'y + \sum_{C \subseteq \{1,\ldots,N\}} \mu_C \mathbb{E}(\hat{c}|C)\mathbb{P}(C) \leq 0 \tag{7b}$$
$$\sum_{C \subseteq \{1,\ldots,N\}} \mu_C \mathbb{P}(C) = 1 \tag{7c}$$
$$\sum_{C \subseteq \{1,\ldots,N\}} \mu_C = \frac{1}{\varepsilon} \tag{7d}$$
$$\mu \geq 0, \tag{7e}$$

where $\mathbb{P}(C) = \sum_{i \in C} p_i$ and $\mathbb{E}(\hat{c}|C) = \frac{1}{\mathbb{P}(C)} \sum_{i \in C} p_i c^i$. Note that this problem can be solved by column generation, using primal simplex after each re-optimization step. This method is what we call **CutEv** in our computational section.

Another very interesting feature of this approach, is that at every step (i.e. when we solve a *partial* version of (7)) we obtain a lower bound of the original problem, but not only that, taking the candidate (feasible) x^*-solution, by evaluating $\text{CVaR}_\varepsilon(-\hat{c}x^*)$ we obtain a valid upper bound for the problem; thus allowing us to stop our algorithm if the proven gap is small enough. Moreover, computing this value is a side-effect of finding the most violated *column* to add to the problem.

3.3 Other specialized algorithms

Given the relevance of portfolio optimization, other specialized algorithms have been proposed in the literature, and some of them have been made into commercial solvers. For example Lim et al. [19] shows a Proximal Bundle Algorithm [13] for this problem, and they also propose an exact three-phase algorithm to solve (1). The explanation of these algorithms is out of the scope of this paper; however, Stan Uryasev kindly provided us with a copy of his software for comparison purposes, and was thus included in our computational tests.

3.4 Primal-dual aggregation method (PDAgg)

Looking at (3), under the assumption of $N \ggg m$, then, most λ variables will be at one of the bounds in any (extreme) optimal solution; i.e., they will have the same value. Moreover, in our experiments, the number of λ variables that are basic is very small, usually, up to three. Thus, if we could *guess* the different values these variables will assume in the optimal solution, $\{\lambda_k^*\}_{k=1}^K$, then, we could re-write (3) using only $n+1$ constraints (other than bounds), and $m+K$ variables. Our algorithm will start by assuming that all λ variables will have the same value, obtain a candidate x^*, evaluate $\text{CVaR}_\varepsilon(-\hat{c}x^*)$, and then identify the set of λ variables that *support* this value, and allow the approximated problem to support the same set of values, and iterate the process. Interestingly, this idea of forcing a set of λ variables to have the same value in the dual, exactly correspond in the primal to aggregate that set of scenarios into a single scenario. This idea of forcing extra constraints (in our case fix many λ variables to have the same value) into a large problem to solve it more efficiently was proposed, in a different context, by Bienstock and Zuckerberg [7].

More precisely, let us consider $\mathcal{N} = \{N_k\}_{k=1}^K$ a partition of $\{1,\dots,N\}$, and the problems

$$(\underline{D}_{\mathcal{N}}) \quad \max\, y^t b \tag{8a}$$

$$s.t.\; y^t A + \frac{1}{\varepsilon}\sum_{k=1}^{K} \bar{p}_k \lambda_k \bar{c}^k \leq 0 \tag{8b}$$

$$\sum_{k=1}^{K} \bar{p}_k \lambda_k = \varepsilon \tag{8c}$$

$$0 \leq \lambda_k \leq 1 \quad \forall k \in \{1 \dots K\}, \tag{8d}$$

where $\bar{p}_j = \mathbb{P}(N_k)$ and where $\bar{c}^k = \mathbb{E}(\hat{c}|N_k)$. Also, given $\tilde{x} \in \mathbb{R}^n$, consider the problem

$$(\overline{D}_{\tilde{x}}) \max\, -\frac{1}{\varepsilon}\sum_{i=1}^{N} p_i \lambda_i c^i \tilde{x} \tag{9a}$$

$$s.t. \sum_{i=1}^{N} p_i \lambda_i = \varepsilon \tag{9b}$$

$$0 \leq \lambda_i \leq 1 \, \forall i = 1,\dots,N. \tag{9c}$$

It is easy to see that for any partition $\mathcal{N}$ of $\{1,\dots,N\}$ $z_{\underline{D}_{\mathcal{N}}}$ is a lower bound of (3), since (8) is obtained from (3) by adding the constraints $\lambda_i = \lambda_j, \forall i,j : \exists N_k \in \mathcal{N}, i,j \in N_k$. Also, given $\tilde{x}$ satisfying $Ax = b, x \geq 0$; $z_{\overline{D}_{\tilde{x}}}$ is an upper bound for (3), since (9) is obtained from (2) by adding the constraint $x = \tilde{x}$. Furthermore, (9) is a continous knapsack problem, whose solution is computable in time $\mathcal{O}(N\log(N))$. Moreover, any extreme optimal solution of (9) has at most three different values for λ_i; we call this (induced) *optimal partition* $\mathcal{N}_{\tilde{x}}^*$. Interestingly enough, if we take $\tilde{x}$ as a dual optimal solution for (8b), and we have that $\mathcal{N}_{\tilde{x}}^*$ is *representable* in $\mathcal{N}$, then $z_{\underline{D}_{\mathcal{N}}} = z_{\overline{D}_{\tilde{x}}}$, which implies that we have solved (3). This is the key idea that motivates our algorithm, so we will state it formally:

Lemma 1. *Given $\mathcal{N} := \{N_1,\dots,N_k\}$ a partition of $\{1,\dots,N\}$, $(y^*,\lambda^*,x^*,\mu^*)$ an optimal extreme primal-dual solution of (8), and $\mathcal{N}_{x^*}^* := \{N_1^*,\dots,N_I^*\}$ the partition of the λ-space induced by an extreme optimal*

solution of $(\overline{D}_{x^*})$*; then, if for each* $i = 1, \ldots, I$ *there exists* $K_i \subseteq \{1, \ldots, k\}$ *satisfying* $N_i^* = \bigcup_{j \in K_i} N_j$*. Then,* $z_{\underline{D}_{\mathcal{N}}} = z_{\overline{D}_{x^*}}$.

Proof. The proof is direct, as we only need to group the terms inducing N_i^* accordingly, and by using the fact that x^* is a dual-optimal multiplier of (8b). □

Algorithm 1 PDAgg($\mathcal{N}, \delta$)

Require: $\mathcal{N}$ partition of $\{1, \ldots, N\}$, $\delta \geq 0$.

1: **loop**
2: Solve $(\underline{D}_{\mathcal{N}})$. Let $\tilde{x}$ be a dual-optimal solution of (8b) and $z_{\underline{D}_{\mathcal{N}}}$ its objective value.
3: Solve $(\overline{D}_{\tilde{x}})$. Let $\mathcal{N}^*$ be an induced optimal partition of $(\overline{D}_{\tilde{x}})$, and $z_{\overline{D}_{\tilde{x}}}$ its optimal objective value.
4: **if** $z_{\underline{D}_{\mathcal{N}}} - z_{\overline{D}_{\tilde{x}}} \leq \delta$ **then**
5: **return** Solution $(\tilde{x})$
6: **else**
7: Let $\mathcal{N} \leftarrow \{N_i \cap N_j : N_i \in \mathcal{N},\, N_j \in \mathcal{N}_{\tilde{x}}^*\}$.

Algorithm 1 summarizes the algorithm. Note also that finiteness of the algorithm is guaranteed because at every step the number of elements in the partition $\mathcal{N}$ strictly increases; thus bounding the number of iterations of the overall algorithm.

4 Computational Results

4.1 Definitions and settings:

In order to benchmark the performance of the proposed methodology, we implemented the following algorithms and techniques to solve Problem (2):

CPX: We formulate Problem (2) and we solve it using CPLEX Barrier algorithm.

CPX-Dual: As suggested in [22], we formulate the dual of the problem, detailed in (3), and we solve this dual formulation using Primal Simplex algorithm.

CutEv: *Cut-event* formulation, presented in (7) and solved using column generation.

PDAgg: Our proposed methodology, presented in Algorithm 1.

We compare previous methodologies over two set of instances: general LP problems and a set of large portfolio problems. For general LP problems, we use the Netlib [11] LP problem collection. In order to introduce uncertainty in the objective function, we modify the original objective function by multiplying independently each coefficient of the objective function with a random variable χ. We test two random variables:

- $\chi \sim U[0, 1]$, i.e. χ is a uniform distribution in [0,1].
- $\chi \sim \begin{cases} \mathcal{N}(1, 0.4) & \text{with probability } 0.95 \\ \exp(10) & \text{otherwise} \end{cases}$, i.e. 95% of the time, χ follows a normal distribution, and in 5% of the time, it follows an exponential distribution. This is one of the distributions used in [18].

Furthermore, we test different risk levels for CVaR, namely $\varepsilon \in \{0.001, 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.99 \text{ and } 0.999\}$, and ten different sample sizes $|\mathcal{N}|$ from 10.000 to 100.000 samples in increments of 10.000. This give use a total of 200 configurations for each Netlib instance.

All our code is implemented in the C programming language. All runs where made using a single thread with an address space limit and data limit of 2Gb and 20 hours running time limit. The machines were running Linux 2.6.18 under x86_64 architecture, with two quad-core Intel® Xeon® E5620 processors and with 48Gb of RAM. To obtain better timing measures, the machines where configured (in BIOS) with the following technologies disabled: Intel® Turbo Boost Technology, Intel® Hyper-Threading Technology, and Intel® Virtualization Technology (VT-x). These settings, allowed us to run up to eight instances in a machine without (much) interference between processes.

A final detail is in the termination condition of the algorithm. Since **CutEv** and **PDAgg** provide an upper and lower bound for the problem at each iteration, we stop whenever the relative gap between these bounds is less than 10^{-6}. The value 10^{-6} was chosen because it is the default reduced cost tolerance for CPLEX simplex algorithm, which should make all approaches comparable.

4.2 Overall comparisons:

Table 3 (in Appendix A) shows the geometric mean running time between the 200 configurations for each instance. In this table, we only include the 56 instances that can be solved by the four algorithms within our prescribed limits.

It can be seen that formulating the full problem and solving them using CPLEX barrier algorithm (Column **CPX** in the table) is the slowest, with an average running time of 29.957. Also, we can see that the observed behavior of **CPX-dual** for portfolio problems [22] also applies to general CVaR minimization problems, with an average running time of 8.279 seconds. Additionally, it can be seen that our algorithm (Column **PDAgg**) and the *cut-event* formulation (Column **CutEv**) are much faster than formulating the full problem, with an average of 0.373 seconds and 1.127 seconds respectively.

4.3 Comparing PDAgg and CutEv algorithms:

Since these last two algorithms are much faster than both algorithms for the monolithic formulation, we focus on the results of **PDAgg** and **CutEv**. In this case, a total of 83 Netlib's instance where solved by both algorithms within our prescribed limits. We only exclude instances `80bau3b, fit2p, fit2d, ship12l` and `ship12`, because some configuration of these problems exceed the memory limit of 2Gb.

4.3.1 The effect of the stopping criteria:

Table 1: Comparison of **PDAgg** and **CutEv** on Netlib instances under different stopping criteria.

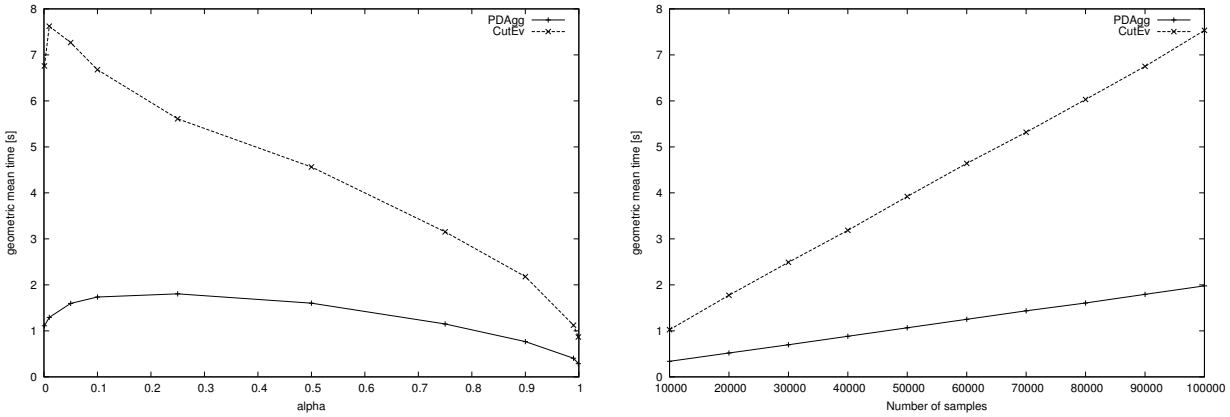
		PDAgg			**CutEv**		
	# Inst.	10^{-2}	10^{-4}	10^{-6}	10^{-2}	10^{-4}	10^{-6}
Global	16600	0.392	0.776	1.017	0.459	1.507	2.714
$>$ 1s	9541	1.360	3.663	5.293	1.555	6.949	14.900
$>$ 10s	5411	2.828	10.365	15.710	2.946	19.276	48.111
$>$ 100s	2303	5.013	30.894	50.716	4.027	51.408	160.112
$>$ 1000s	573	9.471	114.207	194.492	5.025	176.010	680.784

In Table 1 we show the geometric mean running times of both methods over the 16600 problems solved by each algorithm, under different stopping criteria (namely, a relative gap less than 10^{-2}, 10^{-4} and 10^{-6}). In order to improve the detail of the results, we also show these averages over subset of problems where at least one of the algorithms requires more than 1, 10, 100 and 1000 seconds to achieve the final gap of 10^{-6}.

As we can see, both methods requires a similar time to solve instances with a gap of 10^{-2} with a slight advantage for the **CutEv** algorithm, specially on the harder problems. However, our algorithm outperforms the *cut-event* approach when we require more precision.

4.3.2 The effect of the sampling size and the risk parameter:

Figure 1 shows the geometric mean running times of both algorithms on the instances grouped by different levels of α and by different sample sizes. It can be seen that our algorithm outperforms *cut-events* approach, specially when the number of samples increases, moreover, is clear that the grow rate of running time with respect to N for **PDAgg** is smaller than for **CutEv**, which is an indication of a better scalability (on size) for our algorithm. Something interesting can be seen when we study the effect of α on the running time. Both algorithms are slower for small values of α, and faster for large values of α. This was somewhat unexpected, and it may be because for larger α values, we are closer to optimizing plain expected value. But again, it must be noted that our algorithm greatly outperform **CutEv** specially for small values of α, which is the most common case, specially in problems coming from the risk-averse optimization literature.



(a) Time versus α (b) Time versus N

Figure 1: Geometric mean running times of **PDAgg** and **CutEv** on Netlib instances.

4.3.3 Portfolio problems:

The purpose of this second set of instances is to compare our algorithm with other specialized algorithms for portfolio optimization. A large portfolio problem is constructed following [30]: we use monthly closing price of 4553 stocks from January 2008 to December 2012, obtained using CRSP Monthly Stock data from Wharton Research Data Service. From this set, we randomly select a sample of 500 stocks and we compute expected returns and the covariance matrix using the methodology of [17]. Using these data, we generate $100'000$ scenarios from a multivariate normal distribution with the corresponding parameters.

For this experiment, we additionally benchmark against the algorithm of [19], implemented in AORDA Portfolio Safeguard [1]. The experiments with this last algorithm was run in a similar architecture but in a Microsoft Windows environment, and with the same stopping criteria (precision of 10^{-6}).

Table 2 shows the computation times on these instances. The results are similar to the obtained on Netlib's instances. For this particular problem, the *cut-event* formulation and the specialized algorithm are faster than our algorithm for α equal to 0.25, 0.5 and 0.75. However, our algorithm outperforms the other two algorithms for more extreme values of α. This is particularly interesting because for portfolio problem, the common desired value of α is near to zero, where our algorithm is more than 10 times faster than the other algorithms.

Table 2: Computation times on portfolio instances

α	**PDAgg**	**CutEv**	**AORDA**
0.001	5.92	85.07	54.51
0.01	7.93	99.22	98.48
0.05	24.09	103.56	113.39
0.10	117.14	102.28	125.91
0.25	561.91	106.50	144.32
0.50	584.92	136.98	177.12
0.75	465.65	127.21	107.55
0.90	28.69	47.07	41.91
0.99	2.60	7.09	9.11
0.999	0.43	2.28	1.84

4.3.4 Detailed results for PDAgg:

Table 4 (in Appendix A) shows the running details of our algorithms on the portfolio problems and on the 83 Netlib instances using 100'000 scenarios. We present the geometric mean running time (in seconds) required to solve the problem, the size of the final partition (column $|\mathcal{N}|$) and the average number of iterations required (column Iter), among the different distributions and risk-levels. We also include the size of each original problem, in terms of the number of variables, constraints, and the number of non-zero coefficients of the objective function, which represent the dimension of the sample space. It can be seen that the efficiency of our algorithm is partially explained because the required number of iterations is small, resulting in a partition of small size, hence each sub-problem can be solved quickly. In fact, instead of solving a problem with 100'000 scenarios, our algorithm finishes in a geometric average of 5.97 iterations using only 43.87 aggregated scenarios. Note that for several problems, particularly the problems with a low-dimensional sample space, the algorithm requires only two iterations to solve problem, obtaining the optimal partition of the scenarios in a few seconds.

5 Conclusions

We show a primal-dual aggregation technique to exactly solve minimization of CVaR problems under discrete probabilistic distributions, with a very large number of scenarios. This framework also comprises sampled approximation of CVaR minimization problems under arbitrary probability distributions. Computational experiments show that this method is much more efficient in running time than other general and specialized algorithms for this problem. Moreover, our algorithm showed better scalability on the number of scenarios that the competing algorithms. This might be due because the optimal solution for these problems usually requires a far smaller number of *representative* scenarios to prove (ε-)optimality. An interesting open question is how to find a better selection of these representative scenarios, maybe allowing to obtain even smaller aggregations, and then further improving the performance of this algorithm. We also note that this primal-dual aggregation technique could also be applied to solve some other special-structured two-stage stochastic programming problems, opening a new direction for future research. Finally, the proposed framework (as well as the cut-event approach) is easily extendible for dealing with problems with several CVaR constraints and/or objective function; further extending its applicability.

References

[1] American Optimal Decisions, I. (ed.): Portfolio Safeguard (PSG) in Windows Shell Environment: Basic Principles. AORDA, Gainesville, FL. (2011)

[2] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., Espinoza, D.G., Goycoolea, M., Helsgaun, K.: Certification of an optimal tsp tour through 85,900 cities. Operations Research Letters **37**(1), 11 – 15 (2009). DOI 10.1016/j.orl.2008.09.006

[3] Artzner, P., Delbaen, F., Eber, J.M., Heath, D.: Thinking coherently: Generalised scenarios rather than var should be used when calculating regulatory capital. Risk **10**, 68–71 (1997)

[4] Artzner, P., Delbaen, F., Eber, J.M., Heath, D.: Coherent measures of risk. Mathematical finance **9**(3), 203–228 (1999)

[5] Avriel, M., Williams, A.: The value of information and stochastic programming. Operations Research **18**(5), 947–954 (1970)

[6] Bertsimas, D., Brown, D.: Constructing uncertainty sets for robust linear optimization. Operations research **57**(6), 1483–1495 (2009)

[7] Bienstock, D., Zuckerberg, M.: Solving lp relaxations of large-scale precedence constrained problems. In: F. Eisenbrand, F. Shepherd (eds.) Integer Programming and Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 6080, pp. 1–14. Springer Berlin / Heidelberg (2010)

[8] Bixby, R.E.: Solving real-world linear programs: A decade and more of progress. Operations Research **50**, 3–15 (2002). DOI 10.1287/opre.50.1.3.17780

[9] Dantzig, G.B.: Linear programming and extensions. Princeton landmarks in mathematics and physics. Princeton University Press (1963)

[10] Dantzig, G.B.: The diet problem. Interfaces **20**(4), 43–47 (1990)

[11] Gay, D.M.: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Bulletin **13**, 10–12 (1985). Data available at http://www.netlib.org/netlib/lp

[12] Gu, Z.: private communication (2013)

[13] Kiwiel, K.C.: Methods of descent for nondifferentiable optimization, *Lecture Notes in Mathematics*, vol. 1133. Springer-Verlag Berlin (1985)

[14] Kleywegt, A., Shapiro, A., Homem-de Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM Journal on Optimization **12**(2), 479–502 (2002)

[15] Künzi-Bay, A., Mayer, J.: Computational aspects of minimizing conditional value-at-risk. Computational Management Science **3**(1), 3–27 (2006)

[16] Kusuoka, S.: On law invariant coherent risk measures. In: Advances in mathematical economics, pp. 83–95. Springer (2001)

[17] Ledoit, O., Wolf, M.: Honey, I shrunk the sample covariance matrix. UPF Economics and Business Working Paper **691** (2003)

[18] Lim, A.E., Shanthikumar, J.G., Vahn, G.Y.: Conditional value-at-risk in portfolio optimization: Coherent but fragile. Operations Research Letters **39**(3), 163 – 171 (2011). DOI 10.1016/j.orl.2011.03.004

[19] Lim, C., Sherali, H.D., Uryasev, S.: Portfolio optimization by minimizing conditional value-at-risk via nondifferentiable optimization. Computational Optimization and Applications **46**(3), 391–415 (2010)

[20] Linderoth, J., Shapiro, A., Wright, S.: The empirical behavior of sampling methods for stochastic programming. Annals of Operations Research **142**(1), 215–241 (2006)

[21] Markowitz, H.: Portfolio selection. Journal of Finance **7**(1), 77–91 (1952)

[22] Ogryczak, W., Śliwiński, T.: On solving the dual for portfolio selection by optimizing conditional value at risk. Computational Optimization and Applications **50**, 591–595 (2011). DOI 10.1007/s10589-010-9321-y

[23] Rockafellar, R., Uryasev, S.: Conditional value-at-risk for general loss distributions. Journal of Banking & Finance **26**(7), 1443–1471 (2002)

[24] Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. Journal of Risk **2**, 21–41 (2000)

[25] Shapiro, A., Dentcheva, D., Ruszczynski, A.: Lectures on Stochastic Programming: Modeling and Theory. MPS-SIAM Series on Optimization. SIAM-Society for Industrial and Applied Mathematics (2009)

[26] Stigler, G.J.: The cost of subsistence. Journal of Farm Economics **27**(2), 303–314 (1945)

[27] Tintner, G.: Stochastic linear programming with applications to agricultural economics. In: Proceedings of the Second Symposium in Linear Programming, vol. 1, pp. 197–228. National Bureau of Standards Washington, D. C (1955)

[28] Todd, M.J.: The many facets of linear programming. Mathematical Programming **91**(3), 417–436 (2002)

[29] Van Slyke, R.M., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. SIAM Journal on Applied Mathematics **17**(4), 638–663 (1969)

[30] Vielma, J.P., Ahmed, S., Nemhauser, G.L.: A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. INFORMS Journal on Computing **20**(3), 438–450 (2008)

[31] Von Neumann, J., Morgenstern, O.: Theory of games and economic behavior. Princeton University Press (1944)

[32] Wets, R.J.B.: Programming under uncertainty: the equivalent convex program. SIAM Journal on Applied Mathematics **14**(1), 89–105 (1966)

[33] Yaari, M.E.: Some remarks on measures of risk aversion and on their uses. Journal of Economic Theory **1**(3), 315 – 329 (1969). DOI 10.1016/0022-0531(69)90036-2

[34] Yaari, M.E.: The dual theory of choice under risk. Econometrica **55**(1), 95–115 (1987)

A Detailed tables of results

Table 3: Results of all algorithms on running Netlib instances

Name	**PDAgg**	**CPX-Dual**	**CutEv**	**CPX**
25fv47	2.979	227.412	7.897	2823.874
adlittle	1.773	45.729	9.471	66.804
afiro	0.045	0.413	0.162	1.364
agg2	2.076	116.208	31.119	295.196
agg3	1.073	88.085	11.633	303.347
bandm	0.996	33.523	5.044	127.979
beaconfd	0.212	6.993	0.401	45.023
boeing1	3.088	82.197	55.529	741.207
boeing2	0.769	19.458	5.040	108.206
bore3d	0.080	9.219	0.283	21.642
brandy	0.058	0.377	0.161	1.775
capri	0.371	2.358	1.436	6.860
degen2	3.916	668.789	19.976	882.728
e226	1.837	62.300	18.394	187.984
etamacro	0.242	11.452	0.431	44.608
fffff800	0.250	1.921	0.427	3.235
finnis	1.042	69.745	2.186	1040.445
ganges	0.557	8.265	3.468	76.688
greenbea	10.656	89.550	6.473	1203.021
greenbeb	4.576	127.795	4.441	1046.617
grow15	0.188	3.632	0.542	22.783
grow22	0.442	5.593	0.922	41.402
grow7	0.171	1.347	0.406	6.174
israel	0.380	19.627	1.265	62.674
kb2	0.051	0.382	0.145	1.479
lotfi	0.110	2.053	0.231	2.758
nesm	2.366	190.897	2.936	5050.060
perold	0.452	3.215	0.582	3.774
pilot	8.530	39.183	15.557	30.690
pilot.ja	0.848	6.457	1.124	12.999
pilot.we	18.668	316.110	64.417	227.635
pilot4	0.205	1.952	0.325	2.855
recipe	0.068	3.260	0.256	17.871
sc105	0.036	0.227	0.136	0.872
sc205	0.039	0.233	0.141	0.915
sc50a	0.034	0.175	0.134	0.745
sc50b	0.035	0.174	0.134	0.846
scagr25	0.614	159.899	1.712	640.479
scagr7	0.179	24.360	0.458	55.400
scfxm1	0.078	3.024	0.197	4.568
scfxm2	0.119	5.901	0.270	12.914
scfxm3	0.172	9.188	0.364	25.703
scorpion	1.130	54.413	3.756	392.443
scsd1	9.582	225.561	47.585	3432.556
sctap1	5.742	386.091	110.350	865.274
seba	1.088	46.020	3.502	411.507

Table 3 – *Continued from previous page*

Name	**PDAgg**	**CPX-Dual**	**CutEv**	**CPX**
share1b	0.130	2.587	0.284	14.688
share2b	0.188	3.500	0.346	19.146
stair	0.112	0.731	0.267	1.365
standata	0.066	0.985	0.181	3.114
standgub	0.066	1.030	0.184	3.074
standmps	0.057	1.001	0.183	4.692
stocfor1	0.164	3.318	0.401	11.764
tuff	0.058	1.080	0.181	1.861
vtp.base	0.058	1.078	0.163	2.418
wood1p	0.240	5.645	0.483	9.805
woodw	0.627	5.983	0.856	14.044
Total	0.373	8.279	1.127	29.957

Table 4: Results of **PDAgg** algorithm on Netlib and portfolio instances with 100'000 scenarios

Name	Ncols	Nrows	Obj	Time	# Sets	# Iter
25fv47	1571	821	727	4.61	271.35	9.70
adlittle	97	56	82	4.03	3445.20	15.60
afiro	32	27	5	0.11	4.60	2.90
agg	163	488	131	1.18	598.60	11.45
agg2	302	516	231	5.24	1566.20	12.80
agg3	302	516	231	2.43	727.30	11.15
bandm	472	305	165	2.18	1013.45	12.95
beaconfd	262	173	101	0.46	25.00	5.90
bnl1	1175	643	1008	8.11	739.40	11.10
bnl2	3489	2324	2125	25.55	930.80	12.45
boeing1	384	351	380	6.58	1840.40	13.45
boeing2	143	166	143	1.84	1124.25	13.00
bore3d	315	233	96	0.17	2.00	2.00
brandy	249	220	2	0.12	3.00	2.50
capri	353	271	19	0.77	1592.75	11.95
cycle	2857	1903	602	38.64	6358.50	18.90
czprob	3523	929	3504	539.68	3600.20	15.30
d2q06c	5167	2171	3257	381.67	4659.35	16.55
d6cube	6184	415	6184	215.99	1133.45	13.15
degen2	534	444	471	8.23	1298.25	12.40
degen3	1818	1503	1584	83.53	2448.45	13.65
e226	282	223	189	4.11	2374.20	14.25
etamacro	688	400	80	0.44	18.05	5.15
fffff800	854	524	8	0.45	78.15	8.30
finnis	614	497	404	2.13	163.90	9.95
fit1d	1026	24	1026	2.78	52.40	6.40
fit1p	1677	627	1026	1.77	9.00	4.00
ganges	1681	1309	109	1.16	538.10	10.10
greenbea	5405	2392	622	11.62	57.65	6.15
greenbeb	5405	2392	622	5.35	17.65	4.80
grow15	645	300	45	0.29	74.75	4.50
grow22	946	440	66	0.66	53.95	6.70

Table 4 – *Continued from previous page*

Name	Ncols	Nrows	Obj	Time	# Sets	# Iter
grow7	301	140	21	0.38	90.05	7.70
israel	142	174	89	0.82	324.50	10.65
kb2	41	43	5	0.12	3.80	2.80
lotfi	308	153	8	0.28	21.20	7.05
maros	1443	846	392	8.99	1852.15	14.35
maros-r7	1443	846	392	18.07	72.30	5.80
modszk1	1620	687	990	4.87	411.80	11.70
nesm	2923	662	700	3.16	20.85	5.10
perold	1376	625	8	0.50	2.70	2.35
pilot	3652	1441	53	9.28	967.45	11.90
pilot.ja	3652	1441	53	0.94	3.20	2.60
pilot.we	2789	722	92	30.72	6773.30	18.00
pilot4	1000	410	4	0.28	5.85	3.70
pilot87	4883	2030	652	43.59	798.20	11.65
pilotnov	2172	975	72	3.93	1974.05	13.50
portfolio	500	1	500	30.29	7596.30	17.80
recipe	180	91	89	0.16	2.00	2.00
sc105	103	105	1	0.09	2.00	2.00
sc205	203	205	1	0.09	2.00	2.00
sc50a	48	50	1	0.08	2.00	2.00
sc50b	48	50	1	0.09	2.00	2.00
scagr25	500	471	475	1.26	73.50	6.50
scagr7	140	129	133	0.42	17.65	4.85
scfxm1	457	330	23	0.16	3.50	2.75
scfxm2	914	660	46	0.21	3.60	2.80
scfxm3	1371	990	69	0.28	4.05	2.90
scorpion	358	388	282	2.33	782.95	12.65
scrs8	1169	490	847	6.29	1209.60	11.10
scsd1	760	77	760	26.98	2800.65	15.00
scsd6	1350	147	1350	84.83	3429.45	15.10
scsd8	2750	397	2750	72.07	1609.50	13.05
sctap1	480	300	360	15.91	2284.55	13.35
sctap2	1880	1090	1410	37.58	1534.25	13.05
sctap3	2480	1480	1860	55.98	1805.80	13.20
seba	1028	515	522	2.43	268.60	9.80
share1b	225	117	31	0.29	15.40	6.15
share2b	79	96	36	0.43	22.40	7.65
shell	1775	536	1344	6.12	97.05	9.90
ship04l	2118	402	2118	8.82	146.10	9.45
ship04s	1458	402	1458	5.76	140.55	9.30
ship08l	4283	778	4283	27.14	584.95	11.10
ship08s	2387	778	2387	16.21	518.90	10.60
stair	467	356	1	0.16	2.00	2.00
standata	1075	359	7	0.14	3.70	2.85
standgub	1184	361	7	0.14	3.70	2.85
standmps	1075	467	7	0.11	2.00	2.00
stocfor1	111	117	27	0.39	51.40	7.90
stocfor2	2031	2157	1149	2.60	19.50	4.65
tuff	587	333	3	0.11	2.00	2.00

Table 4 – *Continued from previous page*

Name	Ncols	Nrows	Obj	Time	# Sets	# Iter
vtp.base	203	198	6	0.13	3.30	2.65
wood1p	2594	244	1	0.28	2.00	2.00
woodw	8405	1098	4	0.68	9.60	4.05