Distributed Sampled Convex Problems

David Eckman (dje88) and Calvin Wylie (cjw278)

Abstract

It is well-known that chance constrained programs are in general intractable because of the non-convexity of the feasible region and the requirement of prior information about the distribution of the uncertain parameters. Solving a sampled convex program allows for finding solutions that are feasible for the chance constrained program with high probability with known bounds on the required number of samples. Sampled convex problems are more easily solved because they are characterized by a finite number of constraints and require only that one can obtain i.i.d. samples of the uncertain parameters. However, the required number of samples may still result in a sampled convex program that is expensive to solve exactly. We consider obtaining approximate solutions to these sampled convex programs by distributing the constraints across processors and solving smaller subproblems followed by a consensus on the active constraints. We consider a classical portfolio-optimization problem and study the trade-offs that arise from this method: including wall-clock time, violation probability, and feasibility probability.

Introduction

Chance constrained problems model many real-word applications in areas as diverse as finance, energy, and emergency services [1]. For example, chance constraints can represent an acceptable level of risk in an investment or a fixed level of service that is guaranteed. A typical chance-constrained program CCP_ϵ is given by

$$\begin{aligned} \underset{x \in \mathcal{X}}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \mathbb{P}\{f(x, \delta) \leq 0\} \geq 1 - \epsilon. \end{aligned} \tag{CCP_ϵ}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ is compact, δ is the uncertain parameter drawn from the uncertainty set $\Delta \subseteq \mathbb{R}^d$ according to a probability measure $\mathbb{P}$, and $f : \mathcal{X} \times \Delta \mapsto \mathbb{R}$ is convex in x for any fixed δ. The parameter $0 \leq \epsilon \leq 1$ represents the acceptable probability with which the constraint $f(x, \delta) \leq 0$ can be violated. The problem CCP_ϵ is notoriously intractable for two reasons: (1) that the feasible region is, in general, non-convex, and (2) the distribution $\mathbb{P}$ must be known a priori.

One method for finding solutions with good performance which are also feasible for CCP_ϵ with high probability is to take N iid samples $(\delta_1, \ldots, \delta_N)$ from the uncertainty set Δ

according to $\mathbb{P}$ and solve the corresponding sampled convex program SCP_N:

$$\begin{aligned} \underset{x \in \mathcal{X}}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & f(x, \delta_i) \leq 0 \quad i = 1, \ldots, N. \end{aligned} \tag{SCP_N}$$

Notice that SCP_N is more tractable than CCP_ϵ since it has a finite number of constraints, a convex feasible region, and we need not explicitly know $\mathbb{P}$ but instead only need to be able to obtain iid samples from it.

Let x_N^* be the random solution to SCP_N obtained by taking N samples and let $V(x_N^*) = \mathbb{P}^N\{f(x_N^*, \delta) > 0\}$ be the corresponding violation probability where $\mathbb{P}^N$ refers to the product probability measure. To clarify, if $V(x_N^*) \leq \epsilon$, then x_N^* is feasible for CCP_ϵ, otherwise it is infeasible.

Sampled convex programs do not provide a guarantee that $V(x_N^*) \leq \epsilon$ for a given sample size N. Instead, Calafiore and Campi [2] proved that one can bound the number of samples needed to ensure that $\mathbb{P}^N\{V(x_N^*) > \epsilon\} \leq \beta$. That is, the optimal solution to SCP_N is feasible for CCP_ϵ with probability greater than $1 - \beta$. Their bound of $N \geq 1/(\epsilon\beta) - 1$ is simple to calculate, but is quite loose in terms of asymptotics. The bound was further tightened to choosing N to satisfy

$$\mathbb{P}^N\{V(x_N^*) > \epsilon\} \leq \binom{N}{d}(1-\epsilon)^{N-d}.$$

in [3] and then later by Campi and Garatti in [4] to the solution to a binomial equation:

$$\mathbb{P}^N\{V(x_N^*) > \epsilon\} = \sum_{i=0}^{d-1} \binom{N}{i} \epsilon^i (1-\epsilon)^{N-i}.$$

Note that all of these bounds are distribution-free which means that N can be calculated without prior knowledge of $\mathbb{P}$.

For modest ϵ and β, the required N may be large enough to cause the resulting SCP_N instance to be computationally expensive to solve exactly. Even for instances in which N is modest, but the number of decision variables, n, is large, solving SCP_N may pose challenges. When either the decision variables or constraints number in the thousands, existing software for solving mixed integer programs or semi-definite programs might take too long to run to completion. Without loss of generality, we will assume that the number of constraints is the limiting factor in solving SCP_N directly; if the number of decision variables is the issue, one can consider taking the dual of SCP_N.

One idea for addressing this challenge is to decompose SCP_N into smaller sampled convex programs and solve them in parallel before forming some consensus across the subproblems. Because the subproblems have only a fraction of the constraints of SCP_N, they can be solved faster. However because each subproblem is only considering a subset of the constraints, there is a loss associated with this simplification: namely the solution returned will likely violate some of the constraints of SCP_N and therefore is less likely to be feasible for CCP_ϵ. Furthermore, because the solution violates some of the constraints, its corresponding objective function value will be less than $c^T x_N^*$ (in the case where the objective is to minimize $c^T x$). Still, the speedup in terms of wall-clock time due to parallelism may be worth the loss in the probability that the solution is CCP_ϵ-feasible.

Method

A feasible convex program in n dimensional space will have at most n support constraints at the optimal solution. Therefore the problem of finding an optimal solution to SCP_N can be viewed as the problem of identifying the n supporting constraints at x_N^*.

A recent paper by Carlone et al. [5] studies solving large-scale convex programs in a distributed fashion. It assumes a directed graph topology on the communication network connecting the parallel processors. Initially, each processor is given a subset of the constraints of the master problem with which it solves a convex subproblem. It then identifies the support constraints at that solution and passes them to other processors according to the network topology. Each processor receives the incoming constraints and resolves a convex subproblem with their original set of constraints the new constraints. It was proven that after an almost surely finite number of iterations, all processors will reach a consensus about which constraints are supporting for the solution to the master problem. Note that the objective in [5] is to recover the exact solution to the convex program whereas we are interested in finding an approximate solution in less time.

In practice, this procedure would take far too long to run to completion, especially since there is no bound on the number of iterations needed for consensus. Instead, we will consider a variation of this procedure which only runs for two iterations and the topology of the communication network is a star graph in which all worker processors pass supporting constraints to a master processor. Because there are only two iterations, the solutions returned will likely not be the exact solution to SCP_N. However this approximate solution may be calculated in less time, and thus it may be worth a loss in the probability of being a CCP_ϵ-feasible solution.

Suppose that we have available one "master" processor and p "worker" processors available. For simplicity, suppose that $N = mp$ for some positve integers $m > n$.

We describe our procedure as follows:

Parallel SCP_N Procedure

1. For each processor $1 \leq j \leq p$, solve the sampled convex program

$$\begin{aligned} &\underset{x \in \mathcal{X}}{\text{minimize}} && c^T x \\ &\text{subject to} && f(x, \delta_i) \leq 0 \quad i = m(j-1)+1, \ldots, mj. \end{aligned}$$

   Let $\tilde{x}_j$ be the corresponding solution.

2. For each slave processor $1 \leq j \leq p$, determine the supporting constraints of the optimal solution $\tilde{x}_j$. Assuming that each sampled convex program is fully-supported with probability one, label the sample corresponding to these constraints $\delta_1^j, \ldots, \delta_n^j$. Send these samples to the master processor.

3. On the master processor, solve the sampled convex program

$$\begin{aligned} &\underset{x \in \mathcal{X}}{\text{minimize}} && c^T x \\ &\text{subject to} && f(x, \delta_i^j) \leq 0 \quad i = 1, \ldots, k \text{ and } j = 1, \ldots, p. \end{aligned}$$

   Let $\tilde{x}$ be the corresponding solution.

For each subproblem, we identify the support constraints (of which there should be n unless the problem is degenerate) by checking which constraints are active at $\tilde{x}_j$, i.e. which δ_i satisfy $f(\tilde{x}_j, \delta_i) = 0$.

Changing the number of processors p has interesting effects on the main performance measures: wall-clock time and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$. With more processors, the subproblems are smaller, but the total number of constraints passed to the master program (np) is larger. This will impact the wall-clock time of the procedure in two conflicting ways. And as the number of constraints in the master problem increases, we would expect $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$ to decrease. The main research question of this study will be: given these trade-offs, what is the optimal number of processors to use?

Another way to view the problem is to assume with are willing to commit to the wall-clock time need to solve SCP_N. But instead of solving SCP_N we want to use the time to take more than N samples and apply the distributed procedure. Using that time, what is the best number of processors to use and number of constraints to give to each processor and is the performance of the distributed procedure competitive with simply solving SCP_N.

Example Problem

We consider the classical portfolio optimization problem of selecting an allocation y for investing one dollar over n assets with uncertain returns r_i for $i = 1, \ldots, n$. Of these assets, we assume that the nth asset has a fixed payout; it might, for example, represent a bond with a certain return. Each allocation of assets defines a portfolio. Our objective is to choose a portfolio that maximizes the value-at-risk (VaR), denoted by t, for an acceptable risk level ϵ. Here we refer to VaR as the ϵ-percentile of the distribution of returns for a particular portfolio as opposed to the ϵ-percentile of the distribution of losses.

Therefore we have the following chance constrained program:

$$\begin{aligned} \underset{y,\,t}{\text{maximize}} \quad & t \\ \text{subject to} \quad & \mathbb{P}\left\{\sum_{i=1}^{n-1} r_i y_i + r_n y_n \geq t\right\} \geq 1 - \epsilon \\ & \sum_{i=1}^{n} y_i = 1 \\ & y_i \geq 0 \quad i = 1, \ldots, n. \end{aligned} \tag{Portfolio CCP}$$

We will assume that the vector of uncertain returns $(r_1, \ldots, r_{n-1})$ is distributed according to a multivariate log-normal distribution with mean vector μ and covariance matrix Σ. A log-normal is typical for asset payouts because it is unimodal with support on the positive real line. Even with knowledge of the probability distribution of the returns, the feasible region is non-convex. (Is it really?)

By taking samples $r^{(j)}$ for $j = 1, \ldots, N$, the corresponding sampled convex program is

$$\begin{aligned} \underset{y,t}{\text{maximize}} \quad & t \\ \text{subject to} \quad & \sum_{i=1}^{n-1} r_i^{(j)} y_i + r_n^{(j)} y_n \geq t \quad j = 1, \ldots, N \\ & \sum_{i=1}^{n} y_i = 1 \\ & y_i \geq 0 \quad i = 1, \ldots, n. \end{aligned} \tag{Portfolio SCP$_N$}$$

Portfolio SCP$_N$ has several nice properties:

- The objective function and constraints of Portfolio SCP$_N$ are linear. Therefore we can use the simplex method to solve the sampled convex program and subproblems.
- Because of the fixed asset n, it is possible to easily find a basic feasible solution by taking $y_n = 1$, $y_i = 0$ for $i = 1, \ldots, n-1$ and $t = r_n$.
- The feasible allocations y sit within a unit simplex.
- Is t also bounded by the sum of the VaR for each asset?

Here we take $\epsilon = 0.05$ and $\beta = 1 \times 10^{-5}$ to get a required sample size of $N = 5312$.

Implementation in MPI

To solve our example portfolio optimization problem, we implemented a C interface to solve linear programs in paralell using the GNU Linear Programming Kit (GLPK) [1]. We chose to use the Simplex algorithm to solve our linear programs. As our contraints are very dense, interior point methods would be likely be very slow. To provide paralellism, we coded to the Message Passing Interface (MPI) standard. The code was compiled with the Intel C compiler (icc), and experiments performed on a single 12-core Intel Xeon E5-2620 v3 processor.

Given a linear program with m contraints and n variables, the folklore on the simplex method is that it takes $O(m)$ pivots to reach the optimal solution in the average case (the worst case is always $2^n - 1$). Thus we should observe a roughly linear speedup when decomposing our sampled program into smaller subproblems. Also given a problem where $m >> n$, it may make sense to solve the dual program instead (which can be done through the so-called Dual Simplex Algorithm), which has only n constraints.

Figure 1 shows some experimental timings for the simplex and dual simplex methods applied to our problem with $n = 200$ assets. It shows the expected linear (in fact superlinear) increase in wall clock time with respect to the number of constraints. This trend is promising for us to decompose the Portfolio SCP$_N$ into smaller subproblems to achieve time savings through parallelism.

[1] `https://www.gnu.org/software/glpk/`

However, we do not realize any speedup through solving the dual program. We guess this is because our sampled constraints are on average very similar to each other, and thus not as many pivots are needed to reach an optimal extreme point as would be required in a more "standard" linear program.

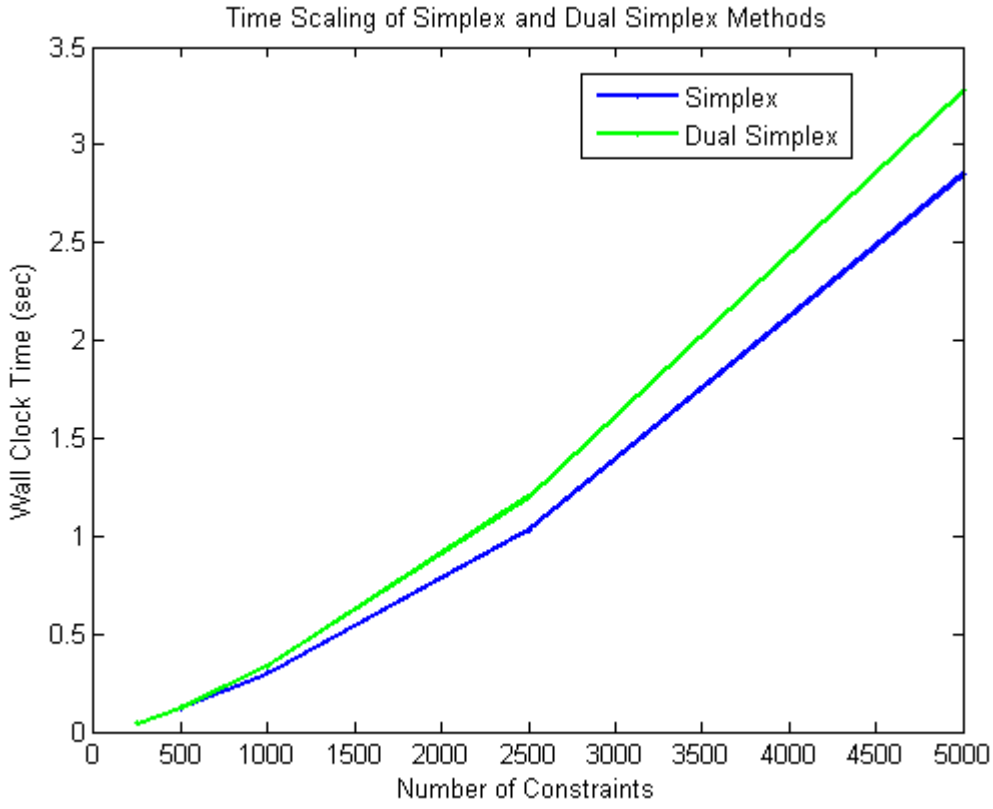

Figure 1: Wall-Clock Times for Simplex and Dual Simplex

Performance Analysis

The program spends time in two main areas: communicating the constraint data through MPI, and solving the linear programs in GLPK. As these are two mature and heavily optimized libraries, manual performance tuning of the routines called would likely not lead to significant speedups. We can however, reason about the overall performance of our program at a high level.

Assuming that the number of constraints is held constant, as Figure 1 suggests we should see a linear decrease in the wall-clock time of the simplex algorithm when distributing the constraints across different processors. The time required for the final solve on the master processor (using active constraints) from the subproblems) should be small, as we can warm-start this optimization with an optimal basis from a subproblem.

However, as we increase the number of processors, the time to communicate the initial constraints (through an MPI_Scatter) and the active constraints (through MPI_Gather) will

increase. Through profiling, we verified that there is a roughly $O(p^2)$ relationship between communication time and the number of processors p.

This suggests that we should initially see decreases in wall-clock time as we add processors, but at some point the communication time will start to dominate the savings gained by splitting our problem. Figure 2 shows this experimentally.

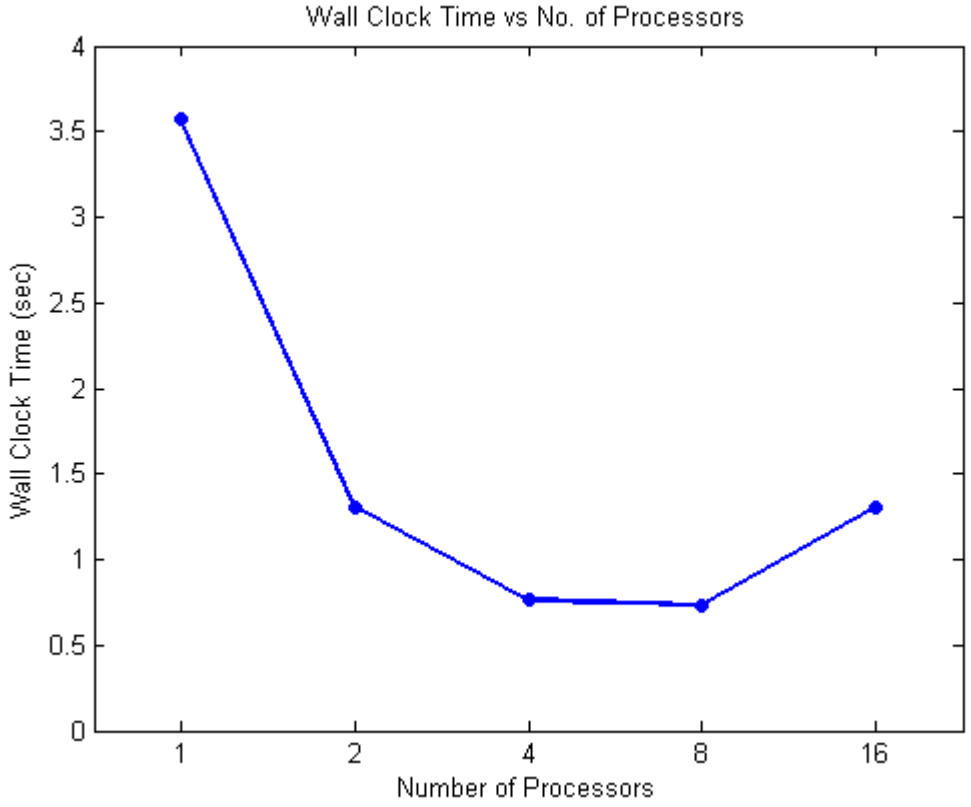

Figure 2: Wall-Clock Times by Number of Processors

Experiments

Let $\tilde{x} = (\tilde{y}, \tilde{t})$ denote the solution returned by running our parallel SCP_N procedure. Note that even though we know the distribution $\mathbb{P}$, calculating $V(\tilde{x})$ exactly is intractable. Instead we resort to estimating via Monte Carlo simulation: obtaining R i.i.d. realizations of r and calculating

$$\hat{V}(\tilde{x}) = \frac{1}{R} \sum_{j=1}^{R} \mathbf{1} \left\{ \sum_{i=1}^{n-1} r_i^{(j)} \tilde{y}_i + r_n^{(j)} \tilde{y}_n < \tilde{t} \right\}.$$

We then M macro-replications of the procedure, with corresponding solutions $\tilde{x}^{(k)}$ for

$k = 1, \ldots, M$, and calculate the following point estimates of $\mathbb{E}[V(\tilde{x})]$ and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$:

$$\widehat{\mathbb{E}[V(\tilde{x})]} := \frac{1}{M} \sum_{k=1}^{M} \hat{V}(\tilde{x}^{(k)}),$$

and

$$\widehat{\mathbb{P}\{V(\tilde{x}) > \epsilon\}} := \frac{1}{M} \sum_{i=1}^{M} \mathbf{1}\left\{\hat{V}(\tilde{x}) > \epsilon\right\}.$$

We use exact binomial tests (Clopper-Pearson) to form approximate confidence intervals for these two performance measures.

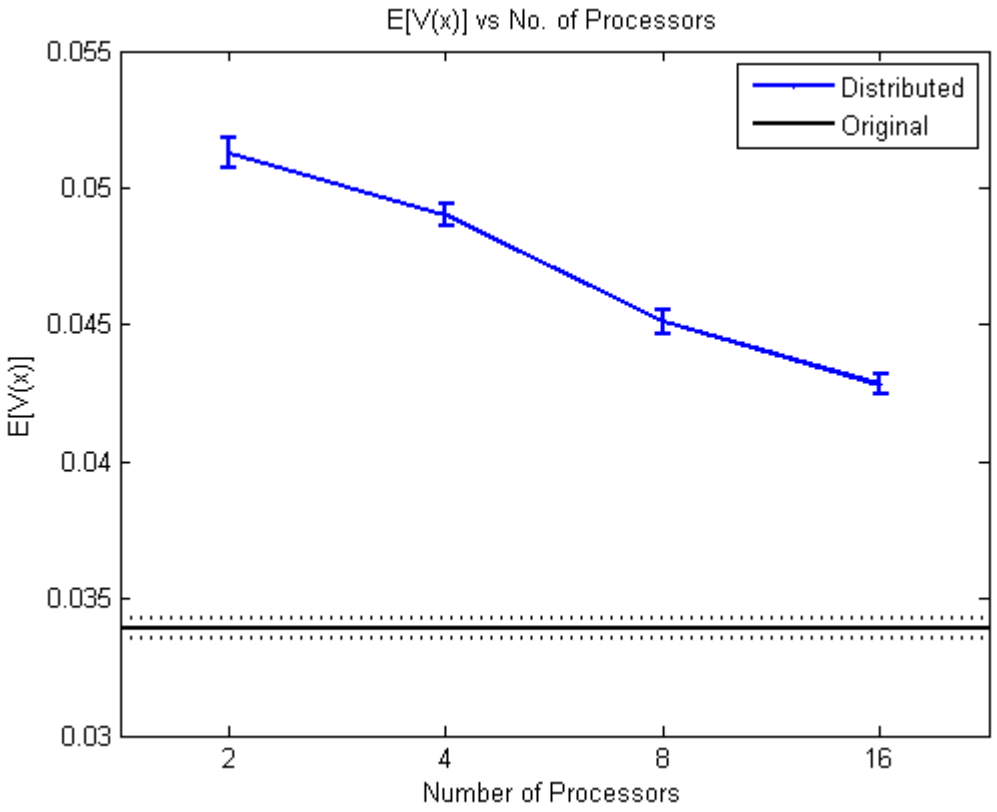

Figure 3: $\mathbb{E}[V(\tilde{x})]$ by Number of Processors

Further Experiments

Our previous experimental results suggested that the parallel SCP_N procedure could run in a fraction of the time as the serial procedure, but returned solutions with higher expected violation probabilty and probability of being infeasible for CCP_ϵ. This suggested that the time savings could be used to run the parallel procedure on more than N samples. In the experiments that followed, we increased the number of samples for each processor, typically

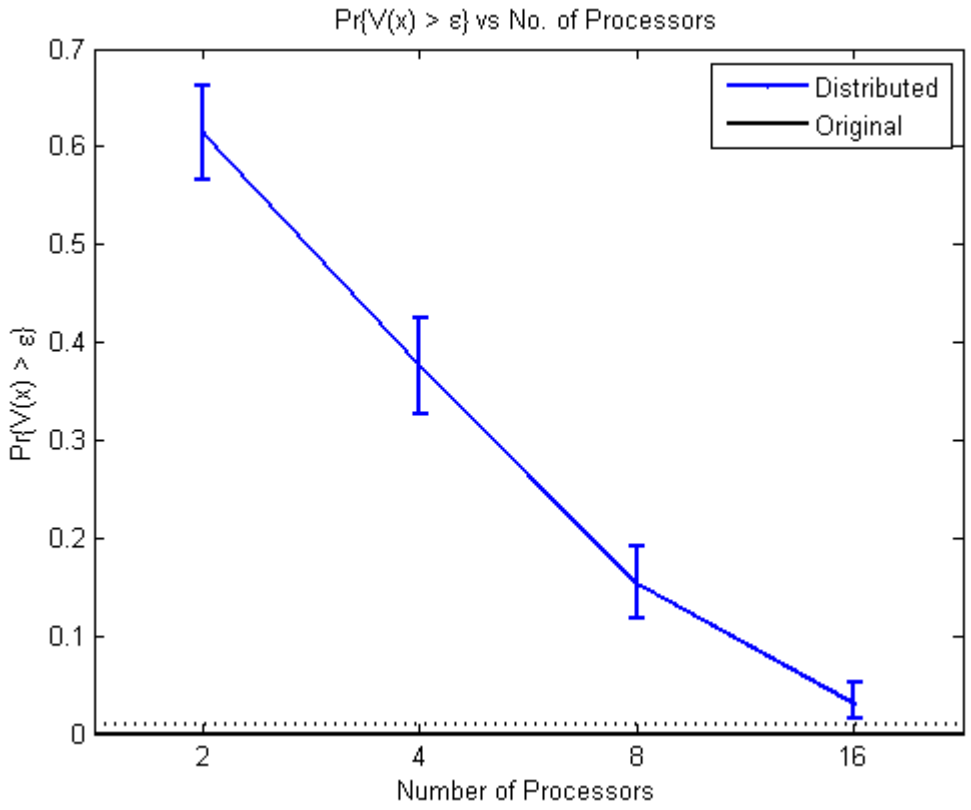

Figure 4: $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$ by Number of Processors

by powers of two. For example, with $N = 5312$, running the parallel procedure with 8 processors means that each processor receives 664 samples. Here we run the same procedure on 8 processors, except with 1328 and 2656 constraints for each processor where now the total number of samples generated was 10624 and 21248.

Figure 6 shows the wall-clock time results for the instances we tested. Observe that the lowest point on each colored curve corresponds to the case of running the original problem of $N = 5312$ constraints. From Figure 6, we can see the increases in wall-clock time for a fixed number of processors is again superlinear. In addition, doubling the number of constraints per processor (for $p = 4$, 8, and 16) appears to outperform the serial procedure in terms of wall-clock time and quadrupling the number of constraints per processor (for $p = 4$ and 8) has comparable times to the serial procedure.

Figures 7 and 8 show $\mathbb{E}[V(\tilde{x})]$ and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$, respectively, for these same instances. On these plots, the highest point on each colored curve corresponds to the case of running the original problem. As expected, both performance measures improve as the number of constraints per processor are increased. It was appears that doubling the number of constraints per processor is enough for the parallel procedure to match or surpass the performance of the serial procedure.

Finally, the wall-clock time and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$ are shown together in Figure 9. Each colored line represents a frontier for a fixed number of processors where the left-point on

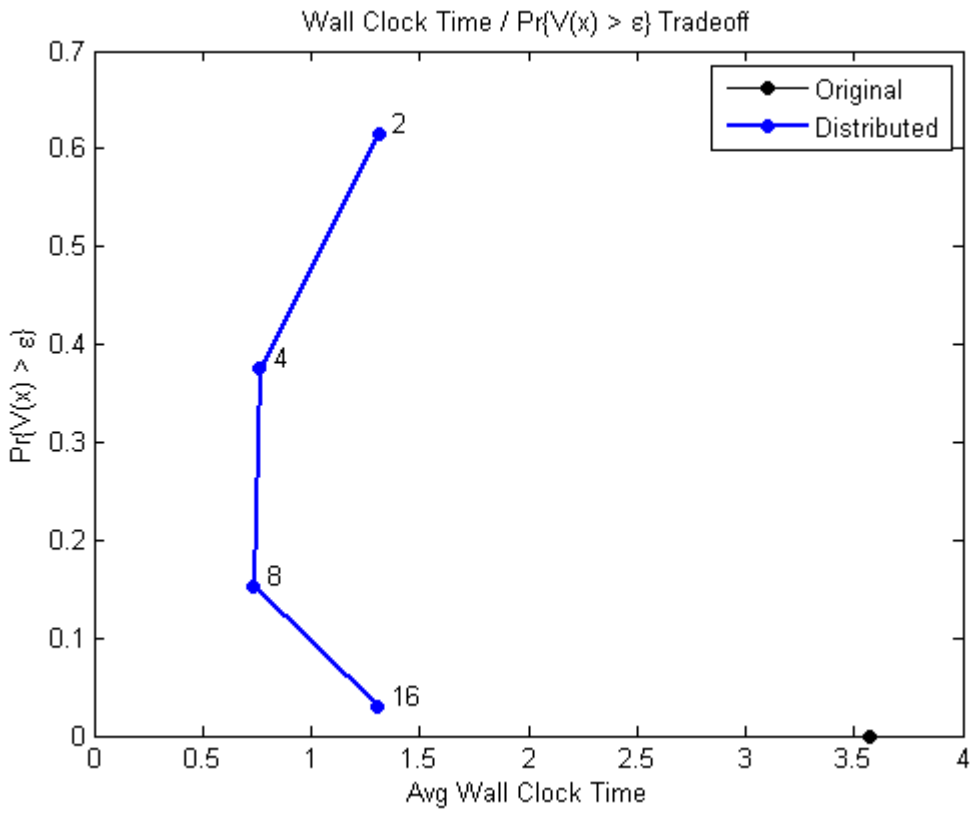

Figure 5: Trade-off between Time and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$

each curve is the original problem. The number of constraints per processor increases as one moves to the right along each curve. Based on this plot, an "optimal" number of configuration appears to be 8 processors with 1328 constraints per processor (the middle red point) because it attains the $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$ of the serial solution in less than half the time.

Conclusions

Open Extensions

If we had more time, there are a number of avenues we would have wanted to explore:

- Experiment on an actual financial data set with forecasts for asset returns.
- Compare the quality of the solution returned by our procedure with other solutions found by solving over convex inner approximations of the CCP_ϵ feasible region.
- Extend our procedure to run multiple iterations of the worker $\rightarrow$ master sequence in hopes of better recovering the optimal active constraints.
- Test for large problem sizes and other problems whose solution methods are of harder complexity than linear programming.

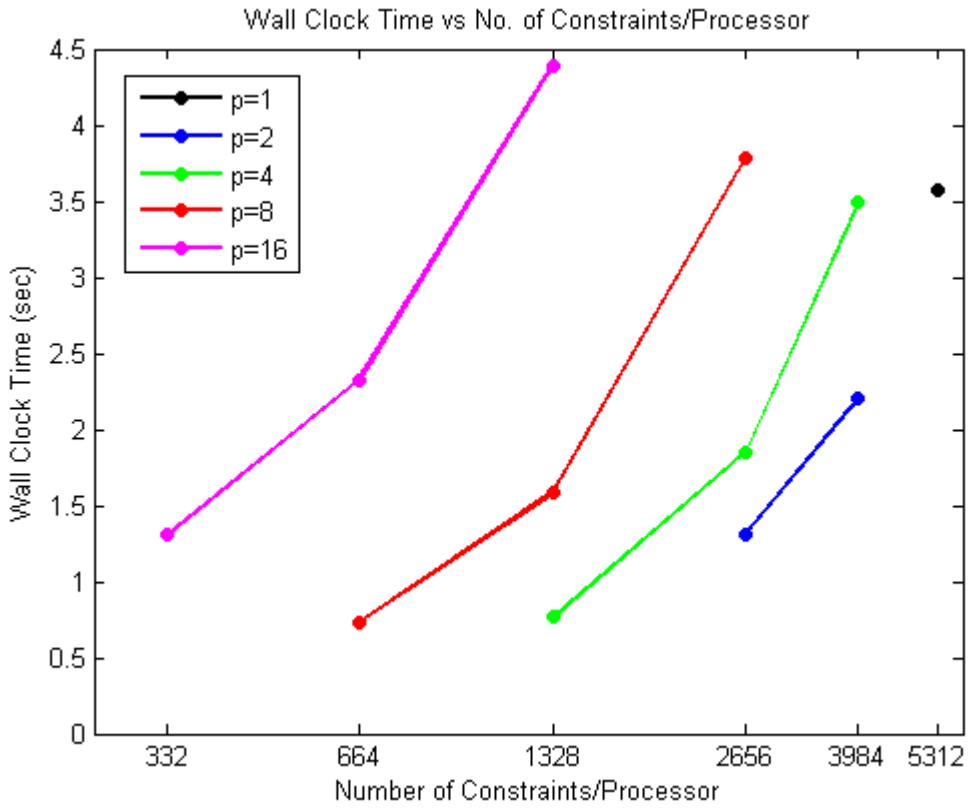

Figure 6: Wall-Clock Times by Number of Constraints/Processor

References

[1] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*, Princeton University Press, 2009.

[2] G. Calafiore and M. C. Campi, *Uncertain convex programs: randomized solutions and confidence levels*, Mathematical Programming, 102 (2005), pp. 25–46.

[3] ——, *The scenario approach to robust control design*, IEEE Transactions on Automatic Control, 51 (2006), pp. 742–753.

[4] M. C. Campi and S. Garatti, *The exact feasibility of randomized solutions of uncertain convex programs*, SIAM Journal on Optimization, 19 (2008), pp. 1211–1230.

[5] L. Carlone, V. Srivastava, F. Bullo, and G. C. Calafiore, *Distributed random convex programming via constraints consensus*, SIAM Journal on Control and Optimization, 52 (2014), pp. 629–662.

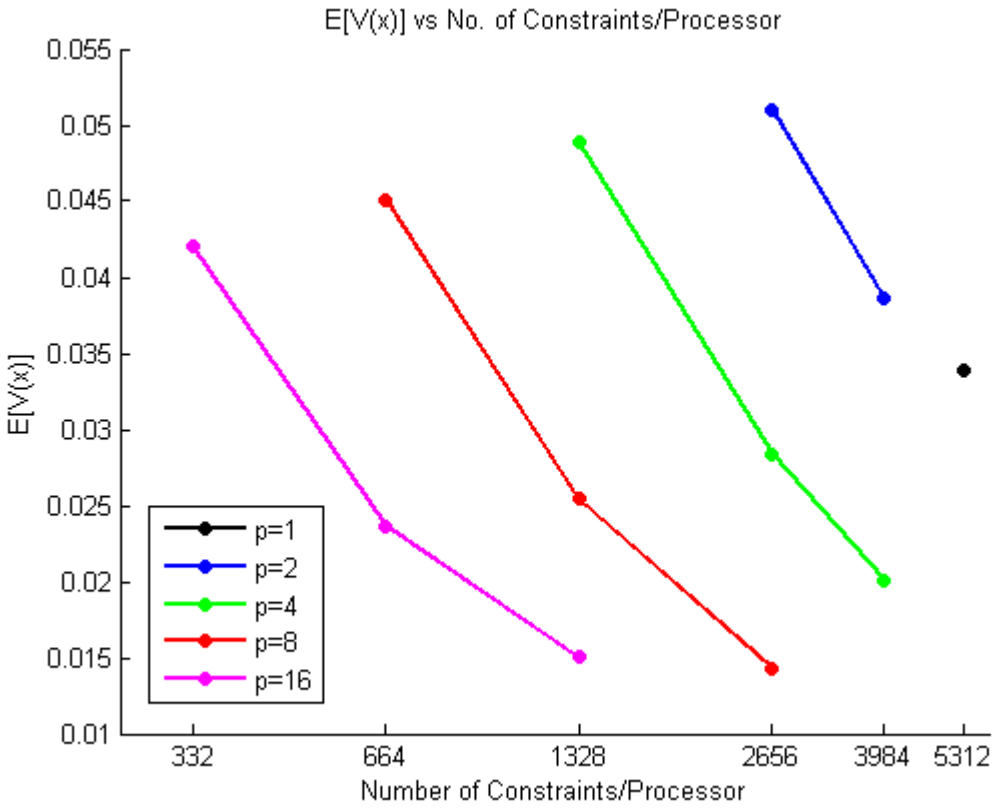

Figure 7: $\mathbb{E}[V(\tilde{x})]$ by Number of Constraints/Processor

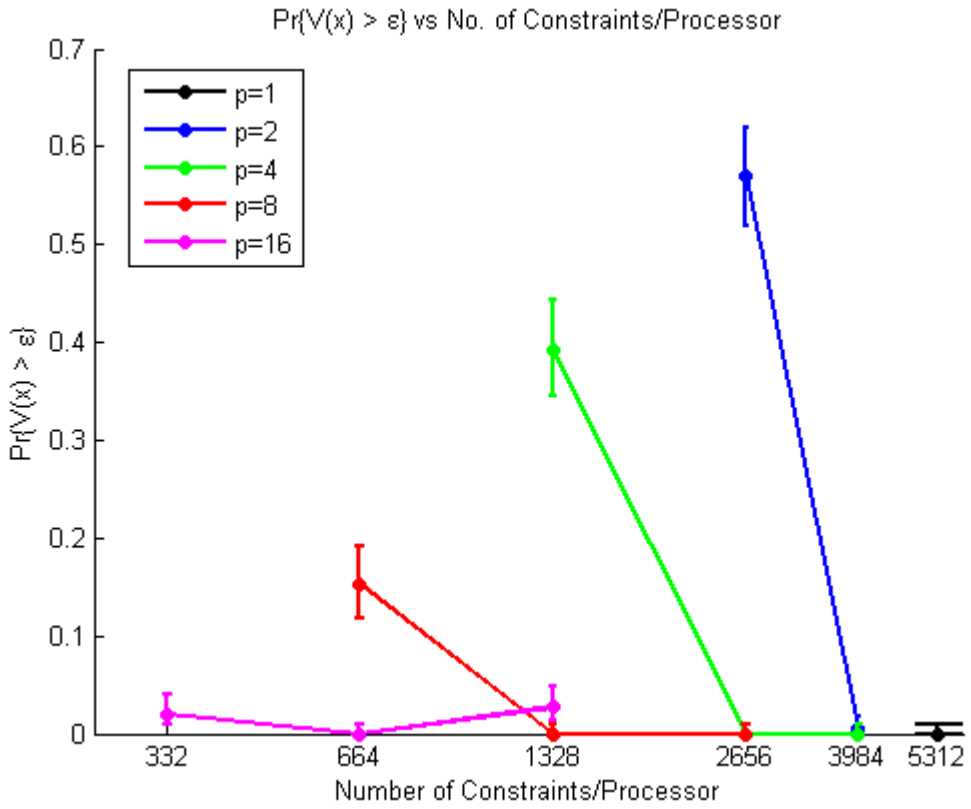

Figure 8: $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$ by Number of Constraints/Processor

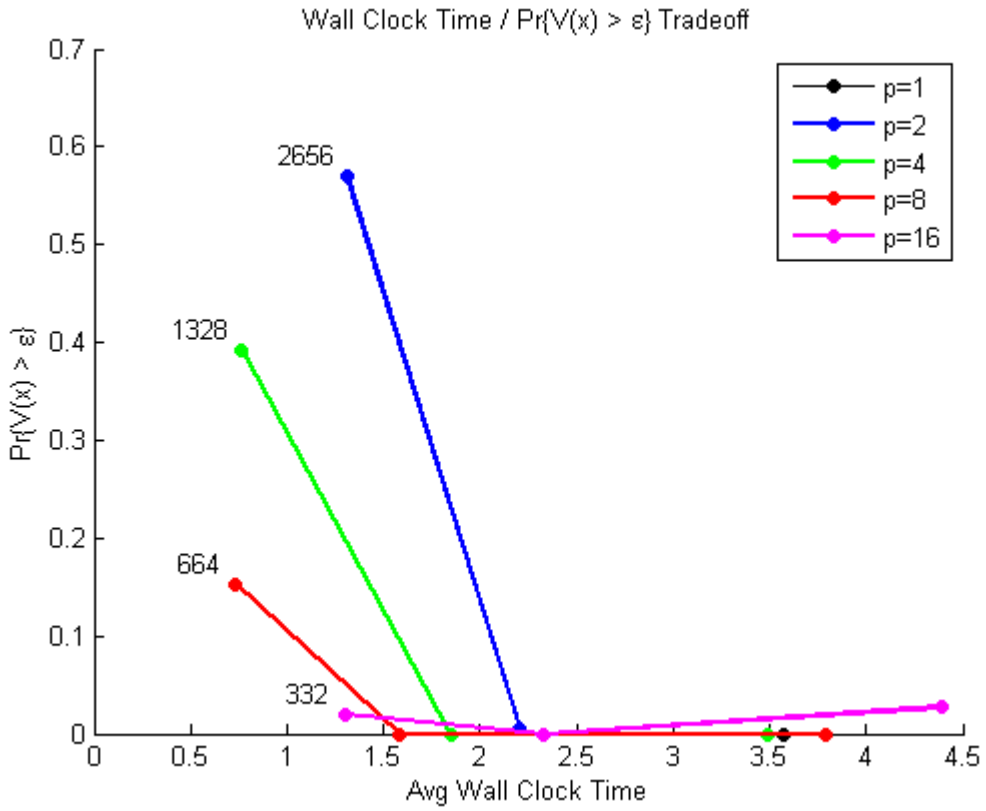

Figure 9: Trade-off between Time and $\mathbb{P}\{V(\tilde{x}) > \epsilon\}$