Asynchronous Parallel Applications of Block-Iterative Splitting

Jonathan Eckstein
Rutgers University

Joint work with

Theory: Patrick Combettes
North Carolina State University
Stochastic application: Jean-Paul Watson, David L. Woodruff
Sandia National Laboratories, University of California, Davis

Funded in part by US National Science Foundation Grants CCF-1115638, CCF-1617617

RUTGERS

RUTGERS
Rutgers Business School
Newark and New Brunswick

NC STATE UNIVERSITY

Sandia National Laboratories

Simplified Block-Iterative Splitting: Problem Setting

- Hilbert spaces $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n$
- Maximal monotone operators $T_i : \mathcal{H}_i \rightrightarrows \mathcal{H}_i \quad \forall i \in 1..n$
- Continuous linear maps $L_i : \mathcal{H}_0 \to \mathcal{H}_i \quad \forall i \in 1..n$

Problem: find $x \in \mathcal{H}_0$: $\boxed{0 \in \sum_{i=1}^{n} L_i^* T_i (L_i x)}$

- As in previous talk, but expressed as a single inclusion involving only one group of operators

Simplified Block-Iterative Splitting

- Define the *Kuhn-Tucker set*

$$Z = \left\{ (z, w_1, \ldots, w_n) \;\middle|\; w_i \in T_i(L_i z) \;\; \forall i \in 1..n, \;\; \sum_{i=1}^{n} L_i^* w_i = 0 \right\}$$

- Whenever $(z, w_1, \ldots, w_n) \in Z$, the vector z solves our problem

- Given $(x_i, y_i) \in \mathrm{Graph}(T_i) \;\; \forall i \in 1..n$, define

$$\varphi(z, w_1, \ldots, w_n) = \sum_{i=1}^{n} \left\langle L_i z - x_i \,, y_i - w_i \right\rangle$$

$$\Rightarrow \quad \varphi(z, w_1, \ldots, w_n) \le 0 \quad \forall (z, w_1, \ldots, w_n) \in Z$$

(follows from monotonicity of $T_1, \ldots, T_n$)

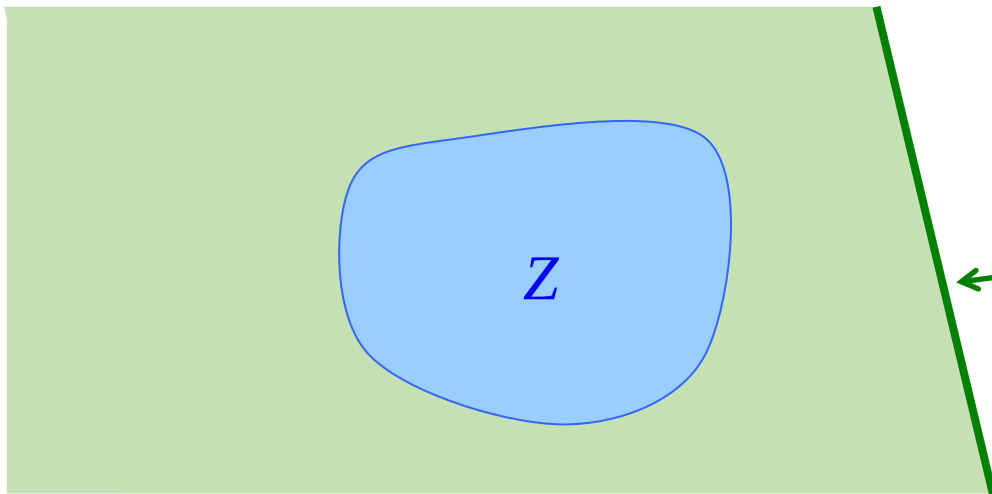
- $\varphi(\cdot)$ is affine on the linear subspace $\mathcal{K}$ given by $\sum_{i=1}^{n} L_i^* w_i = 0$ since quadratic terms are $\sum_{i=1}^{n} \left\langle L_i z, -w_i \right\rangle = \left\langle z, -\sum_{i=1}^{n} L_i^* w_i \right\rangle = \left\langle z, 0 \right\rangle$

- We will operate our algorithm in $\mathcal{K}$ — more restrictive than previous talk; will require projections onto $\mathcal{K}$

Valid Inequalities for Z

Whenever $y_i \in T_i(x_i) \quad \forall i \in 1..n$,

$$\varphi(z, w_1, \ldots, w_n) = \sum_{i=1}^{n} \left\langle L_i z - x_i \,, y_i - w_i \right\rangle \leq 0 \quad \forall (z, w_1, \ldots, w_n) \in Z$$



φ is affine

$H = \left\{ p \,\middle|\, \varphi(p) = 0 \right\}$

$\varphi(p) \leq 0 \quad \forall p \in Z$

But also: these inequalities fully characterize Z within $\mathcal{K}$:

Cutting Off an Arbitrary Point in $\mathcal{K} \setminus Z$

- Take any $\bar{p} = (\bar{z}, \bar{w}_1, \ldots, \bar{w}_n) \in \mathcal{K}$
- For each $i \in 1..n$, compute the unique proximal decomposition $(x_i, y_i) \in \operatorname{Graph}(T_i) \;:\; x + c_i y_i = L_i \bar{z} + c_i \bar{w}_i$ for some $c_i > 0$, hence

$$\varphi(\bar{z}, \bar{w}_1, \ldots, \bar{w}_n) = \sum_{i=1}^{n} \left\langle L_i \bar{z} - x_i \,, y_i - \bar{w}_i \right\rangle$$

$$= \sum_{i=1}^{n} c_i \left\| y_i - \bar{w}_i \right\|^2 = \sum_{i=1}^{n} \left(\frac{1}{c_i} \right) \left\| L_i \bar{z} - x_i \right\|^2 \geq 0$$

- And if $\varphi(\bar{z}, \bar{w}_1, \ldots, \bar{w}_n) = 0$, then $L_i \bar{z} = x_i$ and $\bar{w}_i = y_i \; \forall i$, so $(\bar{z}, \bar{w}_1, \ldots, \bar{w}_n)$ is already in Z since $y_i \in T_i(x_i) \;\; \forall i \in 1..n$

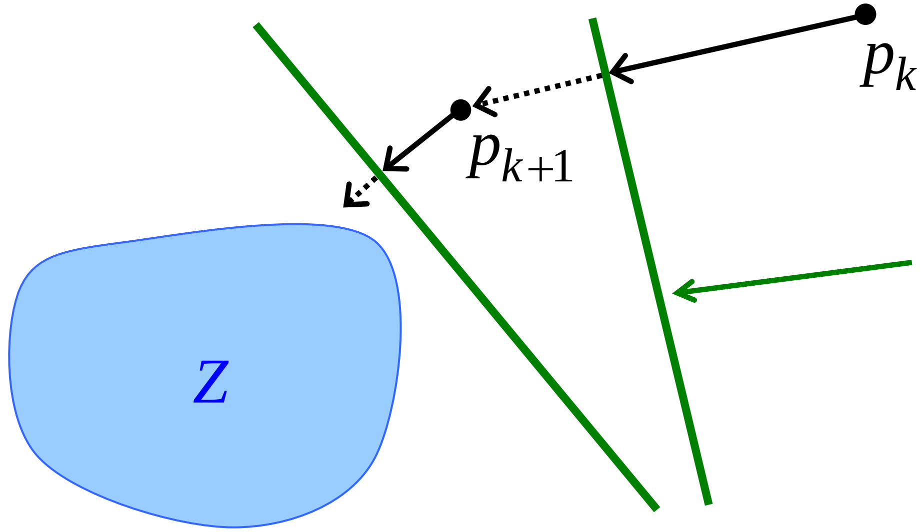
Therefore:

- We may strictly separate any $\bar{p} = (\bar{z}, \bar{w}_1, \ldots, \bar{w}_n) \in \mathcal{K} \setminus Z$ from Z
- Inequalities of the form $\varphi(z, w_1, \ldots, w_n) \leq 0$ fully characterize Z
- Z has to be a closed convex set (can prove in other ways...)

Generic Projection Method for a Closed Convex Set Z

This structure suggests that we can use the following general recipe for finding a point in a closed convex set Z:

- Given $p_k \in \mathcal{K}$, find separating hyperplane H_k between p_k and Z
- Project p_k onto H_n, possibly with an overrelaxation factor $\lambda_k \in [\varepsilon, 2-\varepsilon]$, giving p_{k+1}, and repeat...



φ_k is affine

$$H_k = \left\{ p \,\middle|\, \varphi_k(p) = 0 \right\}$$

$$\varphi_k(p) \le 0 \quad \forall p \in Z$$

$$\varphi_k(p_k) > 0$$

- Fejér monotone: non-increasing distance to all points in Z
- Separators are "sufficiently deep" $\Rightarrow$ (weak) convergence to some point in Z

One Way to Use this Idea (Similar to E and Svaiter 2009)

Here is one possible algorithm, for fixed $0 < c_{\min} \leq c_{\max}$

Starting with an arbitrary $(z^0, w_1^0, \ldots, w_n^0) \in \mathcal{K}$:

1. For $i = 1, \ldots, n$, pick any $c_{i,k} \in [c_{\min}, c_{\max}]$ and find the unique $(x_i^k, y_i^k) \in \mathrm{Graph}(T_i)$: $x_i^k + c_{i,k} y_i^k = L_i z^k + c_{i,k} w_i^k$ (prox operation)
   (Decomposition Step)

2. Define $\varphi_k(z, w_1, \ldots, w_n) = \sum_{i=1}^{n} \left\langle L_i z - x_i^k, y_i^k - w_i \right\rangle$

3. Compute $(z^{k+1}, w_1^{k+1}, \ldots, w_n^{k+1}) \in \mathcal{K}$ by projecting $(z^k, w_1^k, \ldots, w_n^k)$ onto the halfspace $\varphi_k(z, w_1, \ldots, w_n) \leq 0$
   (possibly with some overrelaxation) (Coordination Step)

Computing the Projection

Generic formula for projecting $\overline{p}$ onto $\left\{ p \,\middle|\, \langle a, p \rangle \leq b \right\}$:

$$p^{+} = \overline{p} - \left(\frac{\max\left\{ \langle a, p \rangle - b, 0 \right\}}{\|a\|^{2}} \right) a$$

In the case of the halfspace $\left\{ p \in \mathcal{K} \,\middle|\, \varphi_k(p) \leq 0 \right\} \subset \mathcal{K}$,

$$a = \mathrm{proj}_{\mathcal{K}} \left(\sum_{i=1}^{n} y_i^k, x_1^k, \ldots, x_n^k \right)$$

- Difference from last talk:

  There could be problems if $\mathrm{proj}_{\mathcal{K}}$ is difficult to compute
- But often it is straightforward
- For example, suppose $\mathcal{H}_0 = \mathcal{H}_1 = \cdots = \mathcal{H}_n$ the L_i are all identity matrices, so the problem is $0 \in \sum_{i=1}^{n} T_i(x)$. Then

$$\mathrm{proj}_{\mathcal{K}} \left(v, x_1^k, \ldots, x_n^k \right) = \left(v, x_1^k - \overline{x}, \ldots, x_n^k - \overline{x} \right), \text{where } \overline{x} = \tfrac{1}{n} \sum_{i=1}^{n} x_i$$

Making the Method More General

- At each iteration k, we do not process all the operators $i = 1, \ldots, n$, but just some subset $I_k \subseteq \{1, \ldots, n\}$
  - For the others, we just recycle $(x_i^k, y_i^k) = (x_i^{k-1}, y_i^{k-1})$
- We also consider lags:

  Find $(x_i^k, y_i^k) \in \mathrm{Graph}(T_i):\ x_i^k + c_{i,k} y_i^k = L_i z^{d_k(i)} + c_i w_i^{d_k(i)}$

  where $d_k(i) \le k$ is some possibly earlier iteration.
- We also allow errors

$$(x_i^k, y_i^k) \in \mathrm{Graph}(T_i):\ x_i^k + c_{i,k} y_i^k = L_i z^{d_k(i)} + c_i w_i^{d_k(i)} + e_i^k$$

- Still have valid cuts for Z because $(x_i^k, y_i^k) \in \mathrm{Graph}(T_i)\quad \forall i \in 1..n$
- But are they sufficiently deep to force convergence to Z? In some cases they might not cut off $(z^k, w_1^k, \ldots, w_n^k)$ at all…

Full Algorithm (Still Not as General as Previous Talk)

For $k = 1, 2, \ldots$

Find $(x_i^k, y_i^k) \in \mathrm{Graph}(T_i):\; x_i^k + c_{i,k} y_i^k = L_i z^{d_k(i)} + c_i w_i^{d_k(i)} + e_i^k \qquad i \in I_k$

$(x_i^k, y_i^k) = (x_i^{k-1}, y_i^{k-1}) \qquad i \in 1..n \setminus I_k$

$(u_1^k, \ldots, u_n^k) = \mathrm{proj}_{\mathcal{L}}(x_1^k, \ldots, x_n^k)$, where $\mathcal{L} = \left\{ (w_1, \ldots, w_n) \,\middle|\, \sum_{i=1}^n L_i^* w_i = 0 \right\}$

$v^k = \sum_{i=1}^n L_i^* y_i^k$

$$\theta_k = \frac{\max\left\{ \sum_{i=1}^n \left\langle L_i z - x_i^k, y_i^k - w_i \right\rangle, 0 \right\}}{\left\| v^k \right\|^2 + \sum_{i=1}^n \left\| u_i^k \right\|^2}$$

Pick any $\lambda \in [\varepsilon, 2 - \varepsilon]$

$z^{k+1} = z^k - \lambda_k \theta_k v^k$

$w_i^{k+1} = w_i^k - \lambda_k \theta_k u_i^k \;\; \forall i$

Convergence of the More General Method

The cuts are sufficiently deep (on average) and the method does converge (weakly) under the following assumptions:

- Quasicyclic control: (there is a bound to how long we can ignore any given operator) there exists some integer $M \geq 0$ such that

$$\left(\bigcup_{k=\ell}^{\ell+M} I_k \right) = \{1, \ldots, n\} \quad \forall \ell \geq 0$$

  This control rule borrowed from set intersection methods

- Bounded lags: there exists an integer $D \geq 0$ such that

$$\max\{0, k-D\} \leq d_k(i) \leq k \quad \forall k \geq 0, \forall i \in I_k$$

- Relative error criterion: there exists $B \geq 0, \sigma \in [0,1[$ such that

$$\left(\forall k \geq 0, \forall i \in I_k\right) \quad \left\| e_i^k \right\| \leq B \quad \left\langle e_i^k, L_i z^{d_k(i)} - x_i^k \right\rangle \geq -\sigma \left\| L_i z^{d_k(i)} - x_i^k \right\|^2$$

$$\left\langle e_i^k, y_i^k - w_i^{d_k(i)} \right\rangle \leq \sigma \left\| y_i^k - w_i^{d_k(i)} \right\|^2$$

Implications

This algorithm (and the more general one in the previous talk) have some unique features among splitting methods

- The sets I_k mean that we can adjust the balance between effort expended solving subproblems (the prox operations) and the effort expended on coordination
  - In most n-way splitting methods, every operator must be preocessed before you perform a coordination step
- Together, the I_k and the lags permit a kind of asynchronous parallel operation: at each iteration, you process some set of subproblem calculations that may have been initiated at earlier iterations.

If the operation $\mathrm{proj}_{\mathcal{L}}$ is problematic, use the more general method of the previous talk instead

An Example Application:
A Non-Random Asynchronous n-Block ADMM-Like Algorithm

Problem, for $f_1, \ldots, f_n$ closed proper convex:

$$\boxed{\begin{array}{ll} \min & \sum_{i=1}^{n} f_i(t_i) \\ \text{ST} & \sum_{i=1}^{n} M_i t_i = b \end{array}}$$

Dual formulation (assuming standard regularity conditions):

$$\min_x \sum_{i=1}^{n} f_i^*\left(-M_i^* x\right) + \langle x, b \rangle \quad \rightleftarrows \quad 0 \in \sum_{i=1}^{n} (-M_i) \partial f_i^*\left(-M_i^* x\right) + b$$

One possible way to apply our algorithm: for any $b_1 + \cdots + b_n = b$,

$$T_i(x) = (-M_i) \partial f_i^*\left(-M_i^* x\right) + b_i \quad \forall i$$

We then use the framework above with $L_i = \mathrm{Id} \;\; \forall i \in 1..n$ and $e_i^k \equiv 0$

A Non-Random Asynchronous ADMM-Like Algorithm

Workers' loop: ($b_i - w_i$ is the "target" value for $M_i t_i$)

Wait to receive command (z, i, w_i, μ) from "controller"

$$t_i \in \operatorname*{Arg\,min}_t \left\{ f_i(t) + \langle z, M_i t \rangle + \tfrac{\mu}{2} \left\| M_i t - (b_i - w_i) \right\|^2 \right\}$$

$$x_i = z + \mu\big(M_i t_i - (b_i - w_i)\big) \qquad y_i = b_i - M_i t_i$$

Send (i, x_i, y_i, t_i) back to controller

- Looks like augmented Lagrangian iteration with multiplier z, penalty μ, and constraint $M_i t_i = b_i - w_i$, and like ADMM subproblem
- Many workers operating in parallel, asynchronously

Controller starts with

- $z, w_1, \ldots w_n : \sum_{i=1}^n w_i = 0$
- A set $\Omega = 1..\omega_{\max}$ of available workers

A Non-Random Asynchronous ADMM-Like Algorithm: Controller

"Controller" loop (leaving out iteration indices for simplicity):

While Ω nonempty

  Pick $i \in \{1,\ldots,n\}$ and remove some ω from Ω (*)

  Pick $\mu \in \left[\mu_{\min}, \mu_{\max}\right]$ and send (z, i, w_i, μ) to ω

Wait for at least one worker to complete a task

For each worker ω with a completed task

  Receive (i, x_i, y_i, t_i) from ω

  Insert ω into Ω

$$v \leftarrow \sum_{i=1}^{n} y_i = b - \sum_{i=1}^{n} M_i t_i \qquad \bar{x} \leftarrow \tfrac{1}{n}\sum_{i=1}^{n} x_i \qquad u_i \leftarrow x_i - \bar{x} \quad \forall i$$

$d \leftarrow \|v\|^2 + \sum_{i=1}^{n} \|u_i\|^2$ $\qquad \lambda \leftarrow$ arbitrary choice $\in [\varepsilon, 2-\varepsilon]$

$$\theta \leftarrow \lambda \max\left\{0, \left\langle z - x_i, y_i - w_i \right\rangle\right\}$$

$$x \leftarrow x - (\theta / d)\, v$$

$$w_i \leftarrow w_i - (\theta / d)\, u_i \quad \forall i$$

More about the Algorithm, Parallel Implementation

- $\|v\|^2$ measures the constraint violation and $\sum_{i=1}^{n}\|u_i\|^2$ measures the "disagreement" about the dual variables

- The controller algorithm description above assumes a global memory space
- There is a more general version of the controller that accounts for partitioned memory: some subsystems i can only be processed on certain processors
  - Details too complicated to show here, but conceptually similar
- The implementation style is aimed at multicore or HPC hardware rather than distributed sensor networks etc. on graphs
- The controller should not have to be a serial bottleneck – the controller functions may also be distributed

Convergence Result

Proposition: In the ADMM-like algorithm above, suppose

- There is a bound on the ratio of the longest to shortest possible subproblem solution time.
- Once in every $\overline{M} > 0$ executions of line (*), each possible value of i is selected at least once

Then z converges to an optimal dual solution and the t_i are asymptotically optimal for the primal:

$$\sum_{i=1}^{n} M_i t_i \to b \qquad \text{and} \qquad \limsup\left(\sum_{i=1}^{n} f_i(t_i)\right) \leq f_{\text{opt}}$$

Commentary

Several asynchronous ADMM-like methods have been suggested for an arbitrary number of blocks n. However, they each have some combination of the following features:

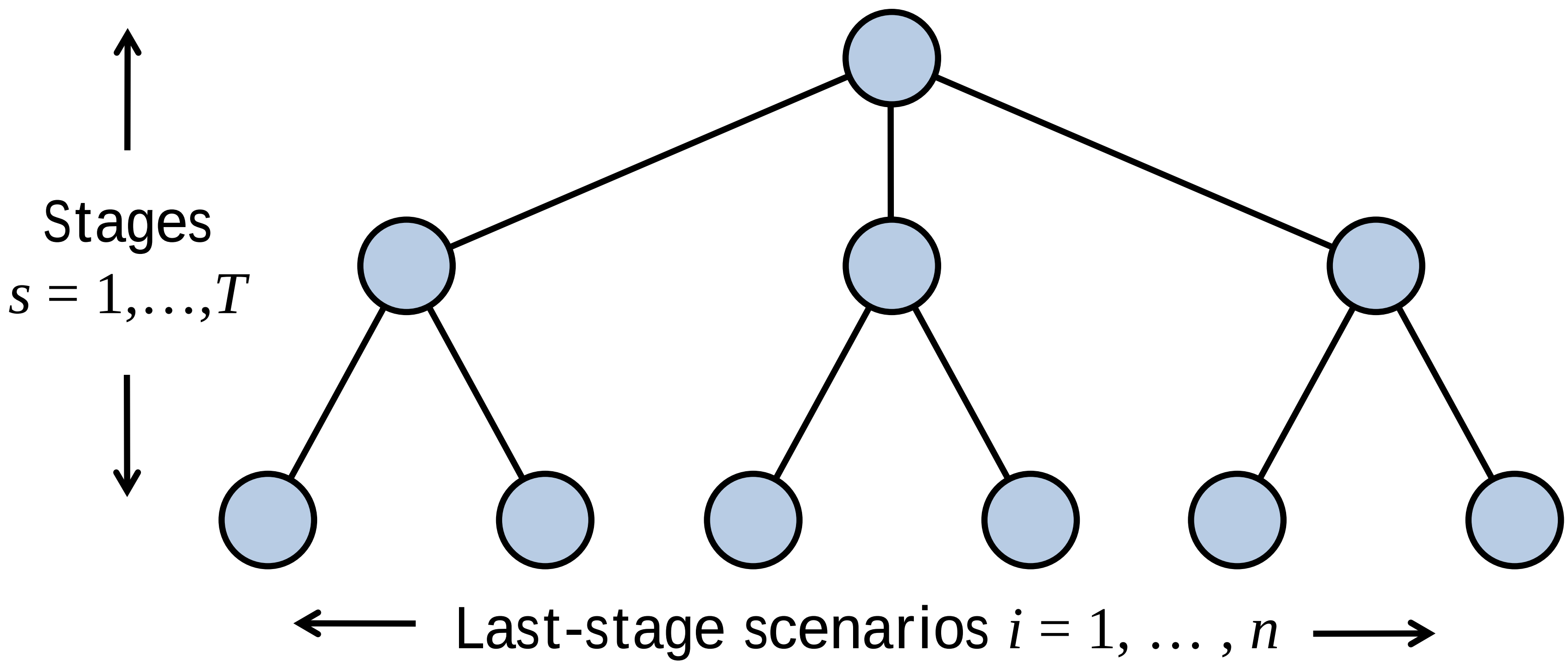
- They require randomness in the activation of blocks and convergence is in expectation (not along every sample path?)
- Convergence is ergodic (in the long-term average of the iterates)
- They require restrictive assumptions about the problem

This new method has “plain” convergence and does not require randomness or restrictive assumptions (only some standard convex-analytic regularity)

- We also have huge freedom in choosing the proximal parameters (stepsizes) inherited from the projective splitting framework – can vary by both iteration and block

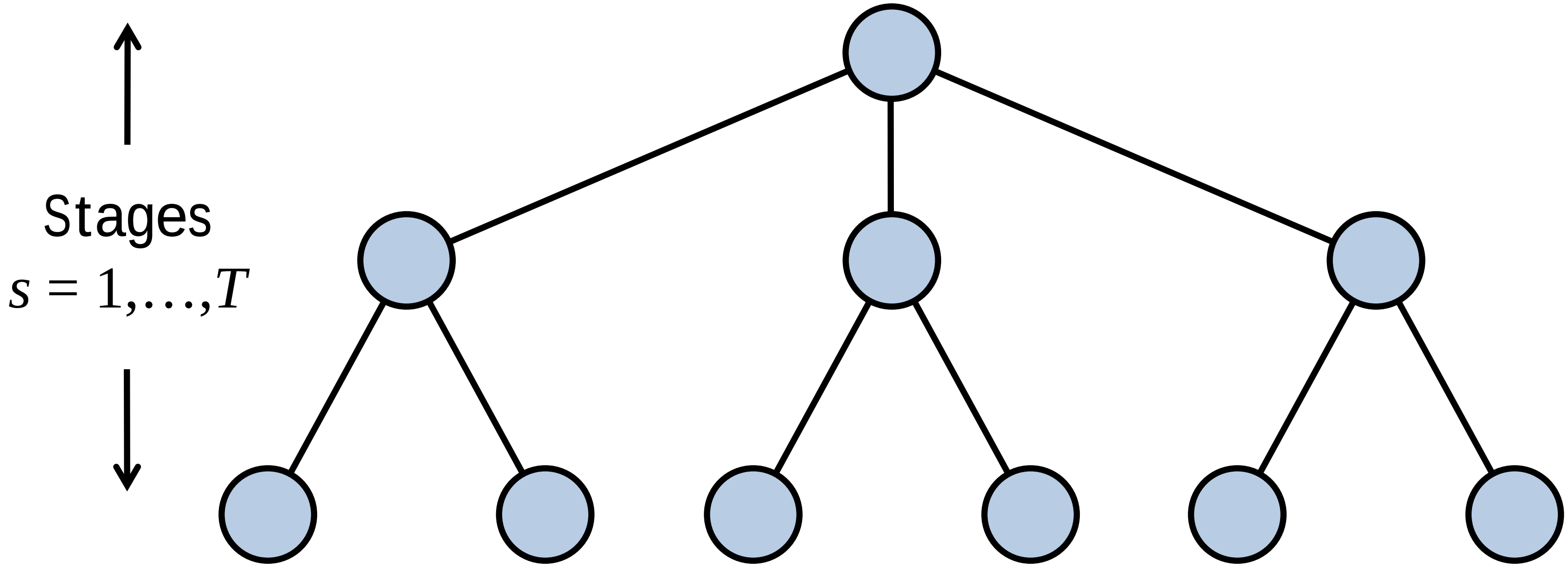
A Related Application: Convex Stochastic Programming

- Consider a standard stochastic programming scenario tree:



- π_i is the probability of last-stage scenario i
- Will use “scenario” as a shorthand for “last-stage scenario”

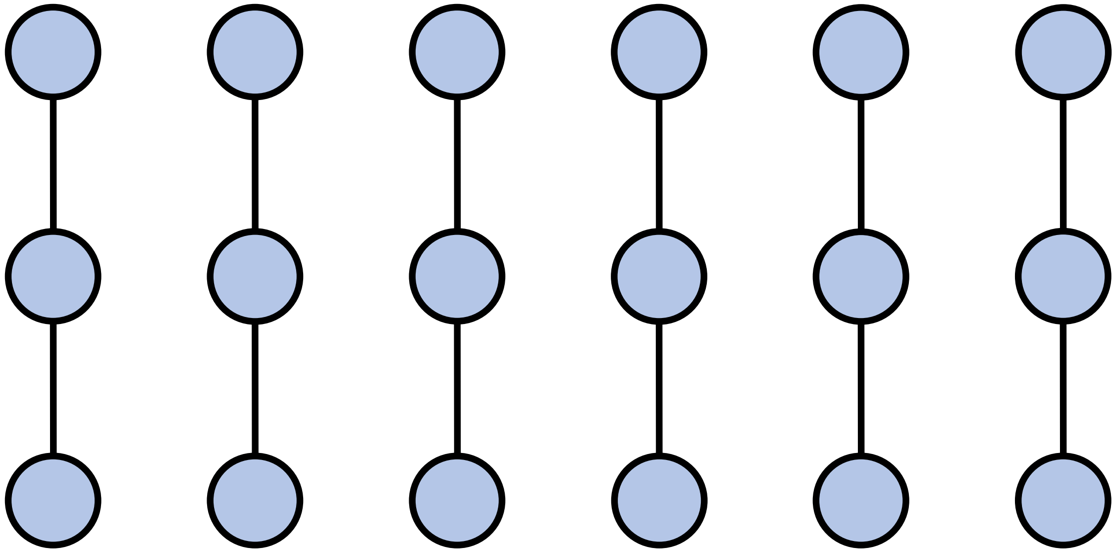
Convex Stochastic Programming



- System walks randomly from the root to some leaf
- At each node there are decision variables, for example
  - o How much of an investment to buy or sell
  - o How much to run a power generator, etc…
- … and constraints that depend on earlier decisions
- Model alternates decisions and uncertainty resolution

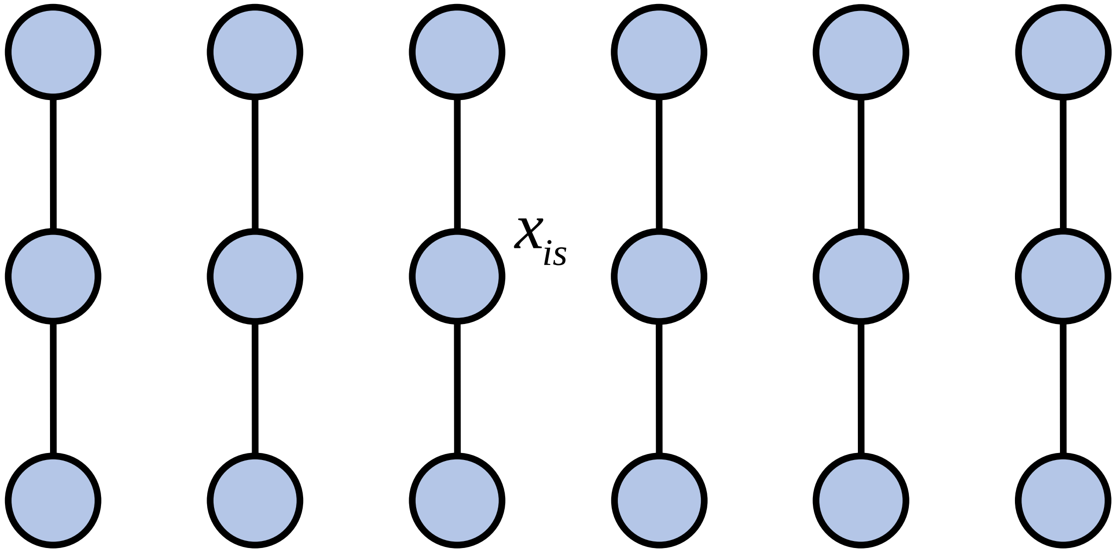
Problem Formulation and Notation

- Replicate decision variables: n copies at every stage



Problem Formulation and Notation

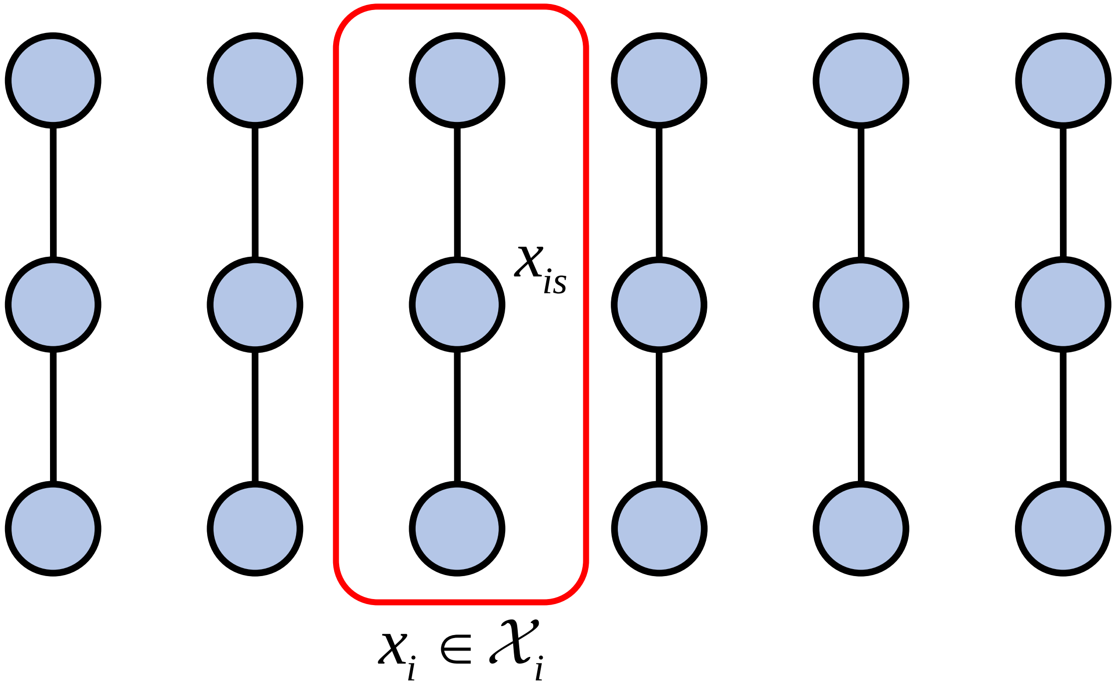
- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s

Problem Formulation and Notation

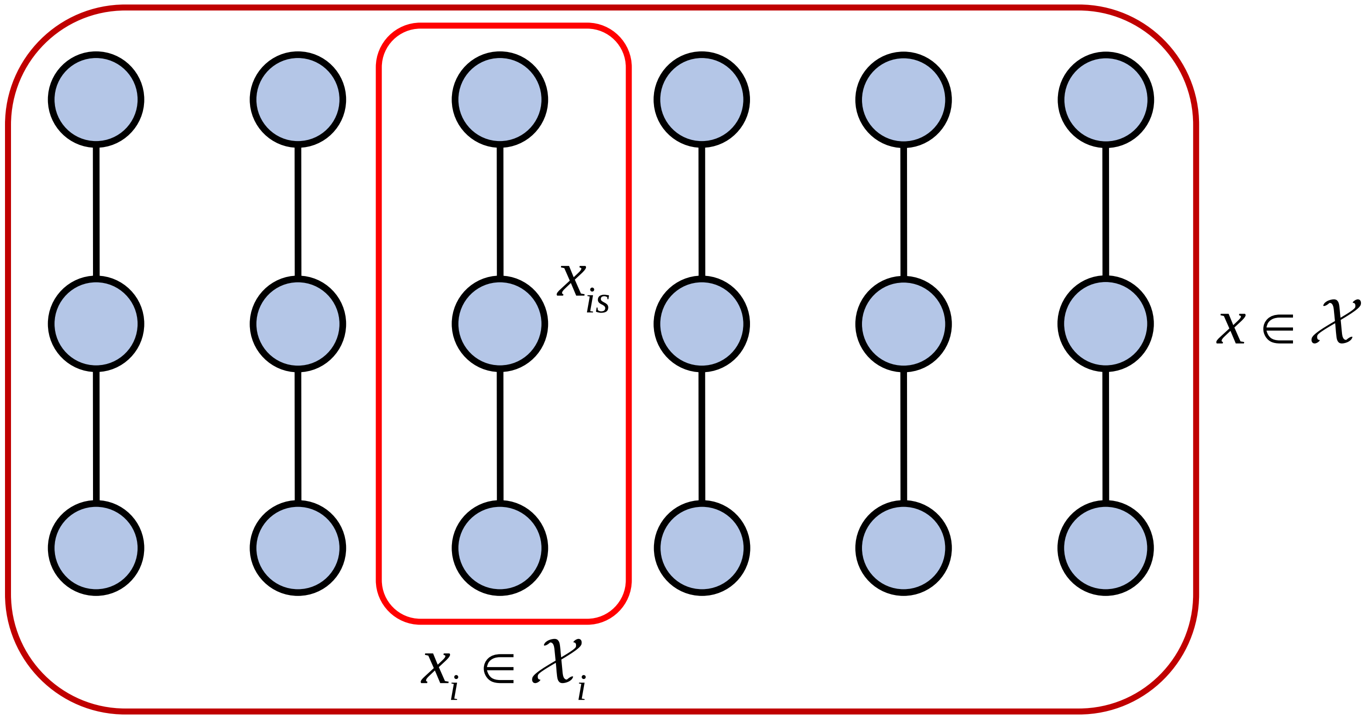
- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s
- $\mathcal{X}_i$ is the space of all variables pertaining to scenario i; elements are $x_i = (x_{i1}, \ldots, x_{iT})$

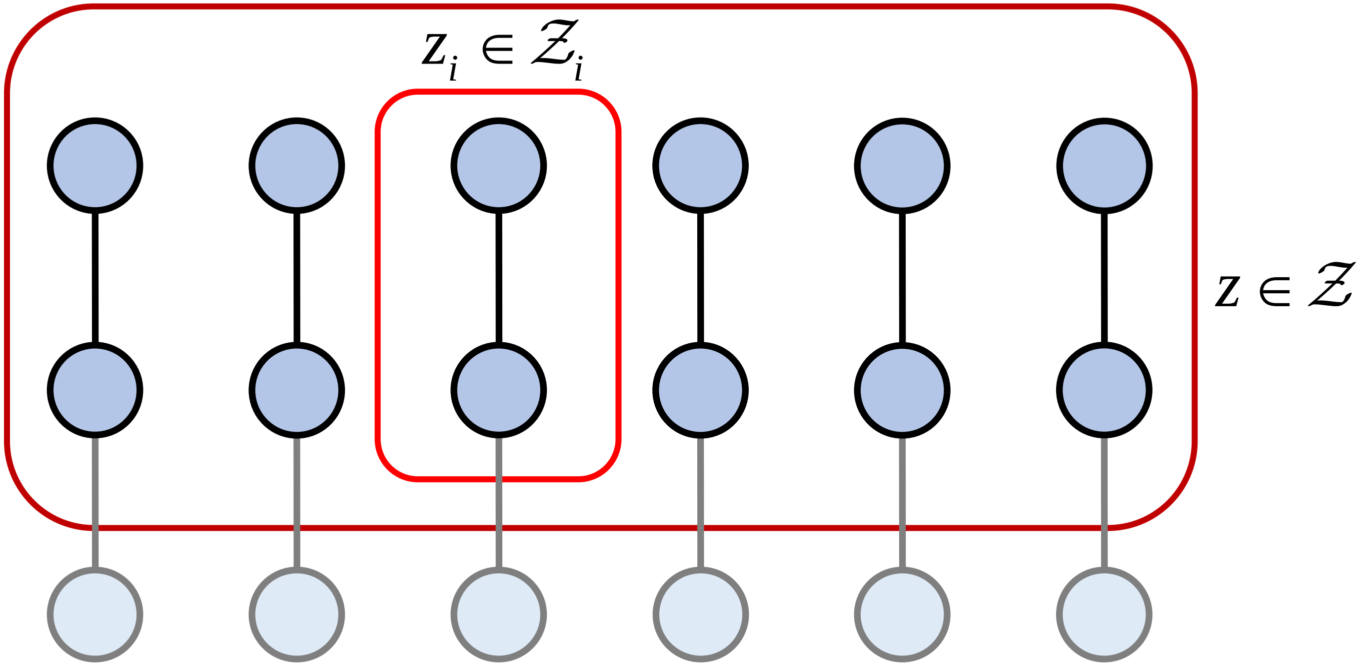
Problem Formulation and Notation

- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s
- $\mathcal{X}_i$ is the space of all variables for scenario i; elements are
  $x_i = (x_{i1}, \ldots, x_{iT})$
- $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ is space of all decision variables; elements are
  $x = (x_1, \ldots, x_n) = \big((x_{11}, \ldots, x_{1T}), \ldots, (x_{n1}, \ldots, x_{nT})\big)$

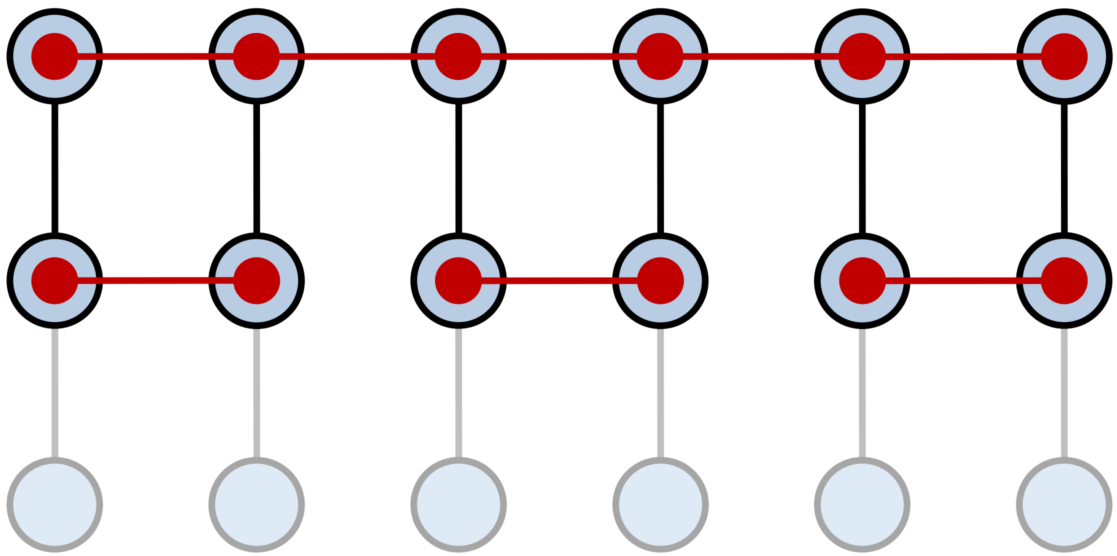
Problem Formulation and Notation

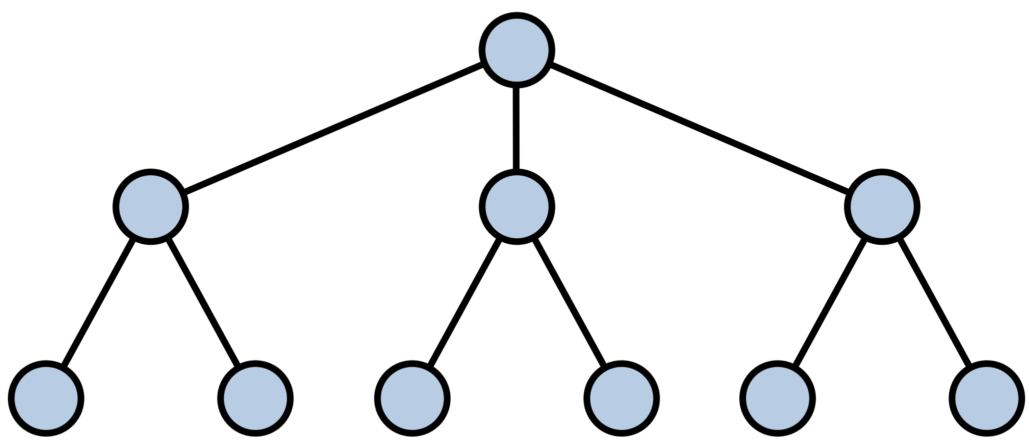


- $\mathcal{Z}_i$ is $\mathcal{X}_i$ without the last stage; elements $z_i = (z_{i1}, \dots, z_{i,T-1})$
- $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$ is the space of all variables except the last stage: elements $z = (z_1, \dots, z_n) = \big((z_{11}, \dots, z_{1,T-1}), \dots, (z_{n1}, \dots, z_{n,T-1})\big)$

Nonanticipativity Subspace

- $\mathcal{N} \subset \mathcal{Z}$ is the subspace of $\mathcal{Z}$ meeting the *nonanticipativity constraints* that $z_{is} = z_{js}$ whenever scenarios i and j are indistinguishable at stage s





Projecting onto the Nonanticipativity Space

- Following Rockafeller and Wets (1991), we use the following probability-weighted inner product on $\mathcal{Z}$:

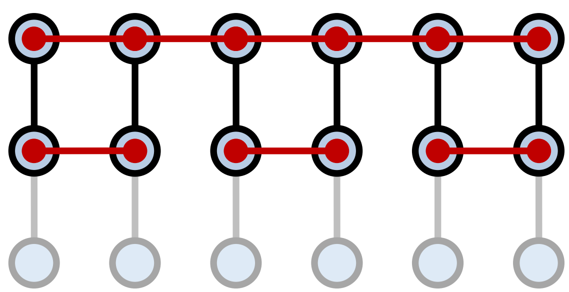
$$\left\langle (z_1,\ldots,z_n),(q_1,\ldots,q_n) \right\rangle = \sum_{i=1}^{n} \pi_i \left\langle z_i, q_i \right\rangle$$

- With this inner product, the projection map $\mathrm{proj}_{\mathcal{N}} : \mathcal{Z} \to \mathcal{N}$ is given by

$$\mathrm{proj}_{\mathcal{N}}(q) = z, \quad \text{where}$$

$$z_{is}^{k+1} = \frac{1}{\left(\sum_{j\in S(i,s)} \pi_j\right)} \sum_{j\in S(i,s)} \pi_j q_{js}^{k+1} \qquad i = 1,\ldots,n, \;\; s = 1,\ldots,T-1$$

and $S(i,s)$ is the set of scenarios indistinguishable from scenario i at time s.



Formulation Continued

- $h_i : \mathcal{X}_i \to \mathbb{R} \cup \{+\infty\}$ is the cost function for scenario i
  - Includes all constraints within scenario i (infeasible points have $h_i(x_i) = +\infty$)
  - Assume that h_i is convex
- $M_i : \mathcal{X}_i \to \mathcal{Z}_i$ is the linear map $(x_{i1}, \ldots, x_{iT}) \mapsto (x_{i1}, \ldots, x_{i,T-1})$ (just drops last stage from scenario i)
- $M : \mathcal{X} \to \mathcal{Z}$ takes $(x_1, \ldots, x_n) \mapsto (M_1 x_1, \ldots, M_n x_n)$ (just drops last stage from full decision vector)

We may formulate a convex stochastic program as

$$\boxed{\begin{array}{ll} \min\limits_{x} & \sum_{i=1}^{n} \pi_i h_i(x_i) \\ \text{ST} & Mx \in \mathcal{N} \end{array}}$$

Formulation Continued

Further define $f:\mathcal{X}\to\mathbb{R}\cup\{+\infty\}$ and $g:\mathcal{Z}\to\mathbb{R}\cup\{+\infty\}$ by

- $f(x)=\sum_{i=1}^{n}\pi_i h_i(x_i)$
- $g(z)=\begin{cases}0, & z\in\mathcal{N}\\ +\infty, & z\notin\mathcal{N}\end{cases}$ (the convex indicator function of $\mathcal{N}$)

Then our stochastic program is just

$$\boxed{\min_{x\in\mathcal{X}} f(x)+g(Mx)}$$

Progressive Hedging (Rockafellar and Wets 1991)

- Apply the ADMM (alternating direction method of multipliers) and obtain, with iterates $\{x^k\} \subset \mathcal{X}, \{z^k\} \subset \mathcal{N}, \{w^k\} \subset \mathcal{N}^\perp$,

$$\boxed{\begin{aligned} &x_i^{k+1} \in \operatorname*{Arg\,min}_{x_i} \left\{ h_i(x_i) + \left\langle M_i x_i, w^k \right\rangle + \frac{\rho}{2} \left\| M_i x_i - z_i^k \right\|^2 \right\} \quad i = 1, \ldots, n \\ &z^{k+1} = \operatorname{proj}_{\mathcal{N}}(Mx^{k+1}) \\ &w^{k+1} = w^k + \rho(Mx^{k+1} - z^{k+1}) \end{aligned}}$$

- Minimize each scenario separately, but with a linear-quadratic perturbation on all variables except the last stage
- Average the results into a nonanticipative z
- Update Lagrange multiplier estimates w and repeat

- Note: Rockafellar and Wets present a derivation from first principles, but it is also an application of the ADMM

Progressive Hedging is Naturally Parallel...

- The minimization step (subproblem) naturally decomposes by scenario
- The remaining calculations take comparatively little time and may also be parallelized (only communication is for the summations required by $\mathrm{proj}_{\mathcal{N}}$, and is simple/efficient)

...But Also Naturally Synchronous

- If some scenarios take longer than others, the algorithm cannot proceed until the slowest one completes
- You must solve all n subproblems between successive coordination steps

Setup to Apply Asynchronous Splitting Method

Problem setup for stochastic programming

- $\mathcal{H}_0 = \mathcal{N}$ (run algorithm in nonanticipativity subspace)
- $\mathcal{H}_i = \mathcal{Z}_i$, but with inner product multiplied by π_i
- $L_i : \mathcal{N} \to \mathcal{Z}_i$ selects the subvector relevant to scenario i
- $f_i(\tilde{x}_i) = \min_{x_{iT}} \left\{ \pi_i h_i \left((\tilde{x}_i, x_{iT}) \right) \right\}$ minimizes scenario i's cost over the last-stage variables
  - Remember, scenario-infeasible points have $h_i(x_i) = +\infty$
- Then our stochastic program is just

$$\min_{x \in \mathcal{H}_0} \sum_{i=1}^{n} f_i(L_i x)$$

- Apply the method from earlier in the talk for $0 \in \sum_{i=1}^{n} L_i^* T_i(L_i x)$
- Conveniently, it turns out that $\mathcal{L} = \mathcal{N}^\perp$, so $\mathcal{K} = \mathcal{N} \times \mathcal{N}^\perp$

A New, Asynchronous Alternative

Subproblem: (many operating in parallel, asynchronously)

Parameters sent from "controller":

- $i \in 1..n$: which scenario to solve
- $z_i = (z_{i1}, \ldots, z_{i,T-1})$: scenario i "target" values, except last stage
- w_i : multipliers (same dimensions as z_i)
- $\rho > 0$: scalar penalty parameter

Receive $i \in 1..n,\ \ z_i, w_i \in \mathcal{Z}_i,\ \ \rho > 0$ from controller

$$x_i \in \operatorname*{Arg\,min}_{x_i} \left\{ h_i(x_i) + \left\langle M_i x_i, z_i \right\rangle + \frac{\rho}{2} \left\| M_i x_i - z_i \right\|^2 \right\}$$

$$y_i = w_i + \rho(M_i x_i - z_i)$$

Return $i,\ \ \tilde{x}_i \doteq M_i x_i,\ \ y_i$ to controller

Looks like progressive hedging subproblem
\+ part of multiplier update

A New, Asynchronous Alternative: "Controller" Setup

The controller maintains working variables:

- $z = (z_1, \ldots, z_n) \in \mathcal{N}$
- $w = (w_1, \ldots, w_n) \in \mathcal{N}^\perp$
- $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n) \in \mathcal{Z}$ (the tildes mean no last-stage variables)
- $y = (y_1, \ldots, y_n) \in \mathcal{Z}$

At each iteration we also compute step direction vectors:

- $u = (u_1, \ldots, u_n) \in \mathcal{N}^\perp$
- $v = (v_1, \ldots, v_n) \in \mathcal{N}$

Scalar parameters:

- Primal-dual scaling factor $\gamma > 0$ (improves conditioning; fixed?)
- Subproblem penalty parameters $\rho \in [\underline{\rho}, \overline{\rho}],\ 0 < \underline{\rho} \le \overline{\rho}$ (varying)
- Overrelaxation factors $\lambda \in [\varepsilon, 2 - \varepsilon]$ (varying)

A New, Asynchronous Alternative: "Controller"

repeat

  while there is a free worker ω **do**

    Choose a scenario i and $\rho \in [\underline{\rho}, \overline{\rho}]$

    Dispatch i, z_i, w_i, ρ to worker ω

  wait for at least one worker to complete its task

  for each worker ω with a completed task

    Receive $i, \tilde{x}_i, y_i$ from ω

  $u \leftarrow \tilde{x} - \mathrm{proj}_{\mathcal{N}}(\tilde{x})$

  $v \leftarrow \mathrm{proj}_{\mathcal{N}}(y)$

  $\tau \leftarrow \|u\|^2 + \gamma\|v\|^2 = \sum_{i=1}^{n} \pi_i \|u_i\|^2 + \gamma \sum_{i=1}^{n} \pi_i \|v_i\|^2$

  $\phi \leftarrow \langle z - \tilde{x}, w - y \rangle = \sum_{i=1}^{n} \pi_i \left(z_i - \tilde{x}_i\right)^{\mathsf{T}} \left(w_i - y_i\right)$

  if $\phi > 0$ **then**

    Choose some $\lambda \in [\varepsilon, 2 - \varepsilon]$

    $z \leftarrow z + (\gamma\lambda\phi / \tau)v$

    $w \leftarrow w + (\lambda\phi / \tau)u$

until termination detected

Partial Resemblance to PH

- Subproblem has recognizable pieces of the PH subproblem optimization step and multiplier update
- Controller has $\mathrm{proj}_{\mathcal{N}}$ operations
- But otherwise the controller algorithm comes from our splitting framework

- Unlike progressive hedging, the algorithm runs asynchronously
  - Only a single subproblem needs to complete between cycles of the controller (more is OK too)

- In our description, the controller looks centralized/serial, but it could be distributed with careful implementation

Conclusion / Summary / Ongoing Work

- A general decomposition method for monotone inclusions
- Gives freedom to...
  - Strike arbitrary balance between computing and coordination
  - Not have to reevaluate every operator between each pair of successive coordination steps
  - Implement asynchronously without requiring randomness
- Numerous possible applications:
  - Asynchronous ADMM-like method without randomness (shown above)
  - Asynchronous stochastic programming decomposition

Some Early Computational Results from JP Watson

- Contingency-constrained AC optimal power flow instances
- Two-stage stochastic programs with n scenarios
- We run an asynchronous algorithm essentially the same as described in this talk (but for stochastic programming)
- Compared to progressive hedging (PH)
- Use n processors, one per scenario ($\approx$ an ADMM block)
- These are very early results, lots left to do

Problem	n	**PH Time**	**Async Time**
case6ww	11	0:00:02	0:00:02
case57	79	0:00:12	0:00:09
case118	117	0:02:03	0:01:40
case300	322	0:02:54	0:02:19

References Part 1

"The mothership"

- Patrick L. Combettes and Jonathan Eckstein. "Asynchronous block-iterative primal-dual decomposition methods for monotone inclusions". *Mathematical Programming*, online July 2016.

References Part 2

- Jonathan Eckstein. "A simplified form of block-iterative operator splitting, and an Asynchronous Algorithm Resembling the Multi-Block ADMM".
  - Convergence analysis for simplified framework in this talk...
  - But weaker initialization conditions than the "mothership"
  - And an asynchronous ADMM-like method generalizing the one in this talk

More realistic applications coming "soon"...