Distributed L-shaped Algorithms in Julia

Martin Biel

KTH - Royal Institute of Technology

November 16, 2018

Motivation - Stochastic programming

decision x

Motivation - Stochastic programming

decision x $\quad\rightarrow\quad$ observation $\xi(\omega)$

Motivation - Stochastic programming

decision x $\rightarrow$ observation $\xi(\omega)$ $\rightarrow$ recourse y

Motivation - Stochastic programming

$$\text{decision } x \quad \rightarrow \quad \text{observation } \xi(\omega) \quad \rightarrow \quad \text{recourse } y$$

- Determine optimal decision $\hat{x}$ based on predicted outcomes $\{\omega_i\}_{i=1}^{n}$

Motivation - Stochastic programming

decision x $\rightarrow$ observation $\xi(\omega)$ $\rightarrow$ recourse y

- Determine optimal decision $\hat{x}$ based on predicted outcomes $\{\omega_i\}_{i=1}^{n}$
- Numerous industry applications
  - **Power systems**
  - Finance
  - Transportation

Motivation - Stochastic programming

$$\text{decision } x \quad \rightarrow \quad \text{observation } \xi(\omega) \quad \rightarrow \quad \text{recourse } y$$

- Determine optimal decision $\hat{x}$ based on predicted outcomes $\{\omega_i\}_{i=1}^n$
- Numerous industry applications
  - **Power systems**
  - Finance
  - Transportation
- Traditional procedure
  - Formulate deterministically equivalent optimization problem
  - Optimize extended form using standard solvers

Motivation - Limitations of standard approaches

- Industry-scale applications typically involve 10,000+ scenarios

Motivation - Limitations of standard approaches

- Industry-scale applications typically involve 10,000+ scenarios
- Example: 24-hour unit commitment problem [*Petra et al (2014)*]
  - 16,384 scenarios
  - 1.95 billion variables and constraints in the extended form
  - ~ 1 hour computation time on a Titan supercomputer

Motivation - Limitations of standard approaches

- Industry-scale applications typically involve 10,000+ scenarios
- Example: 24-hour unit commitment problem [*Petra et al (2014)*]
  - 16,384 scenarios
  - 1.95 billion variables and constraints in the extended form
  - ~ 1 hour computation time on a Titan supercomputer
- Long computation time required to optimize

Motivation - Limitations of standard approaches

- Industry-scale applications typically involve 10,000+ scenarios
- Example: 24-hour unit commitment problem [*Petra et al (2014)*]
  - 16,384 scenarios
  - 1.95 billion variables and constraints in the extended form
  - ~ 1 hour computation time on a Titan supercomputer
- Long computation time required to optimize
- Memory requirement exceeds the capacity of a single machine

Motivation - Limitations of standard approaches

- Industry-scale applications typically involve 10,000+ scenarios
- Example: 24-hour unit commitment problem [*Petra et al (2014)*]
  - 16,384 scenarios
  - 1.95 billion variables and constraints in the extended form
  - ~ 1 hour computation time on a Titan supercomputer
- Long computation time required to optimize
- Memory requirement exceeds the capacity of a single machine

Parallel algorithms that work on distributed data are required

Contribution

- Framework for formulating and solving stochastic programs

Contribution

- Framework for formulating and solving stochastic programs
- **A collection of L-shaped algorithms**

Contribution

- Framework for formulating and solving stochastic programs
- **A collection of L-shaped algorithms**
- Distributed-memory setting

Contribution

- Framework for formulating and solving stochastic programs
- **A collection of L-shaped algorithms**
- Distributed-memory setting
- Complex functionality using simple abstractions in Julia

Contribution

- Framework for formulating and solving stochastic programs
- **A collection of L-shaped algorithms**
- Distributed-memory setting
- Complex functionality using simple abstractions in Julia

Rapidly formulate and solve real-world problems as stochastic programs

Contribution - Framework

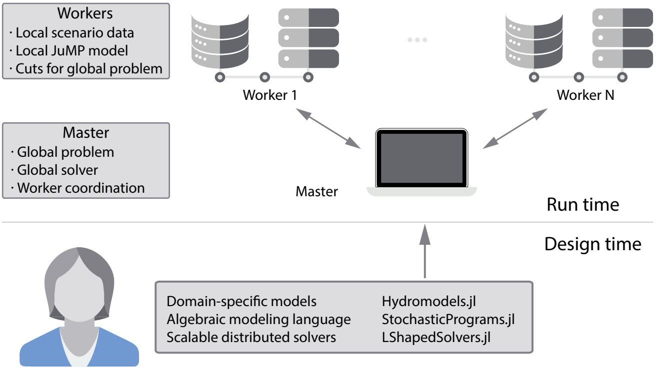


Figure: Overview of software framework.

Outline

- Background
- Implementation
- Numerical experiments
- Final remarks

Background - Stochastic programming

Two-stage linear stochastic program

$$
\begin{aligned}
\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \mathbb{E}_\omega[Q(x, \xi(\omega))] \\
\text{s.t.} \quad & Ax = b \\
& x \geq 0
\end{aligned}
$$

where

$$
\begin{aligned}
Q(x, \xi(\omega)) = \min_{y \in \mathbb{R}^m} \quad & q_\omega^T y \\
\text{s.t.} \quad & T_\omega x + Wy = h_\omega \\
& y \geq 0
\end{aligned}
$$

Background - Stochastic programming

Two-stage linear stochastic program

$$
\begin{aligned}
\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \mathbb{E}_\omega[Q(x, \xi(\omega))] \\
\text{s.t.} \quad & Ax = b \\
& x \geq 0
\end{aligned}
$$

where

$$
\begin{aligned}
Q(x, \xi(\omega)) = \min_{y \in \mathbb{R}^m} \quad & q_\omega^T y \\
\text{s.t.} \quad & T_\omega x + Wy = h_\omega \\
& y \geq 0
\end{aligned}
$$

Deterministically equivalent form

$$
\begin{aligned}
\underset{x \in \mathbb{R}^n, y_i \in \mathbb{R}^m}{\text{minimize}} \quad & c^T x + \sum_{i=1}^n \pi_i q_i^T y_i \\
\text{s.t.} \quad & Ax = b \\
& T_i x + W_i y_i = h_i, \qquad i = 1, \dots, n \\
& x \geq 0, y_i \geq 0, \qquad i = 1, \dots, n
\end{aligned}
$$

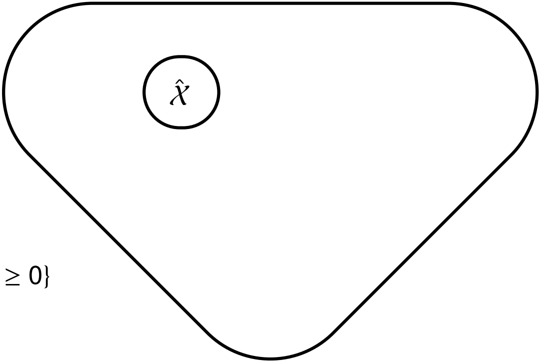
Background - Stochastic programming

$$
\begin{pmatrix}
A & & & & \\
T_1 & W & & & \\
T_2 & & W & & \\
\vdots & & & \ddots & \\
T_n & & & & W
\end{pmatrix}
$$

Figure: L-shaped structure.

Background - L-shaped algorithm

Cutting-plane algorithms



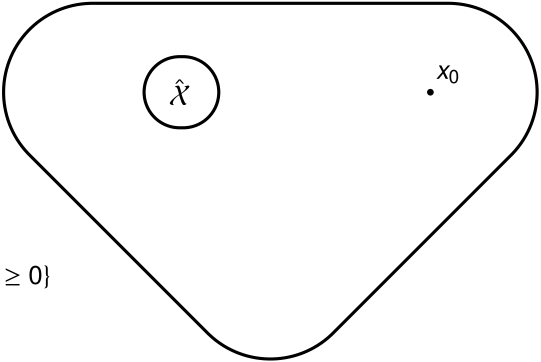
$$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

$$\hat{\mathcal{X}} = \text{Optimal set}$$

Figure: L-shaped cutting planes

Background - L-shaped algorithm

Cutting-plane algorithms



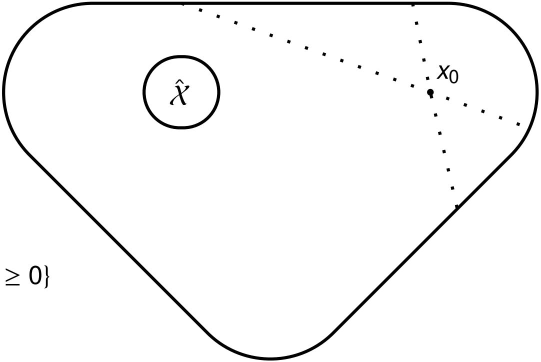
$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$

$\hat{\mathcal{X}} =$ Optimal set

Figure: L-shaped cutting planes

Background - L-shaped algorithm

Cutting-plane algorithms

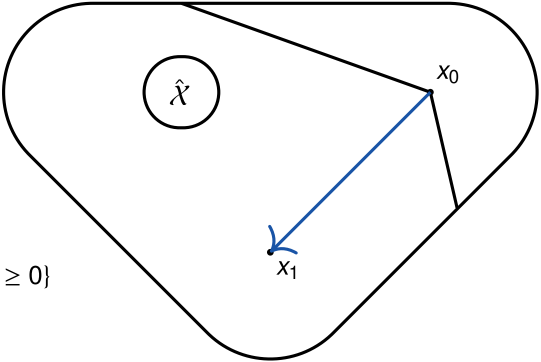


$$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

$$\hat{\mathcal{X}} = \text{Optimal set}$$

Figure: L-shaped cutting planes

Cutting-plane algorithms



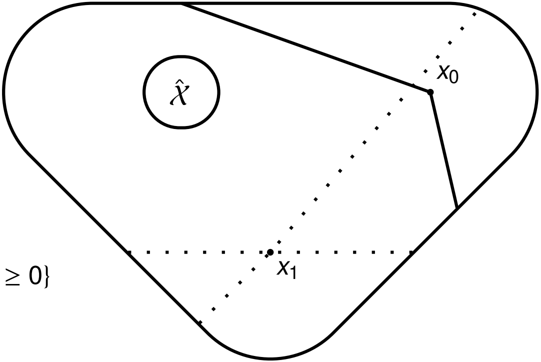
$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$

$\hat{\mathcal{X}} =$ Optimal set

Figure: L-shaped cutting planes

Background - L-shaped algorithm

Cutting-plane algorithms



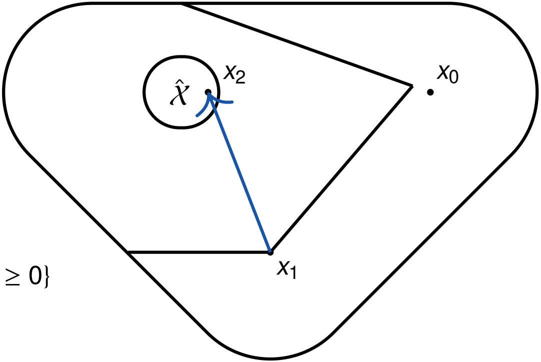
$$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

$$\hat{\mathcal{X}} = \text{Optimal set}$$

Figure: L-shaped cutting planes

Background - L-shaped algorithm

Cutting-plane algorithms



$\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$

$\hat{\mathcal{X}} =$ Optimal set

Figure: L-shaped cutting planes

Background - L-shaped algorithm

Master problem

$$
\begin{aligned}
\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \sum_{i=1}^{n} \theta_i \\
\text{s.t.} \quad & Ax = b \\
& \partial \mathbf{Q} x + \theta_i \geq \mathbf{q}, \qquad i = 1, \dots, n \\
& x \geq 0
\end{aligned}
$$

Subproblems

$$
\begin{aligned}
\underset{y_i \in \mathbb{R}^m}{\text{minimize}} \quad & Q_i^k = q_i^T y_i \\
\text{s.t.} \quad & W y_i = h_i - T_i x_j \\
& y_i \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\partial Q_j &= \pi_i \lambda_{i,j}^T T_i \\
q_j &= \pi_i \lambda_{i,j}^T h_i
\end{aligned}
$$

Background - L-shaped algorithm

Master problem

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \sum_{i=1}^{n} \theta_i \\ \text{s.t.} \quad & Ax = b \\ & \partial \mathbf{Q} x + \theta_i \geq \mathbf{q}, \qquad i = 1, \dots, n \\ & x \geq 0 \end{aligned}$$

Subproblems

$$\begin{aligned} \underset{y_i \in \mathbb{R}^m}{\text{minimize}} \quad & Q_i^k = q_i^T y_i \\ \text{s.t.} \quad & W y_i = h_i - T_i x_j \\ & y_i \geq 0 \end{aligned}$$

$$\begin{aligned} \partial Q_j &= \pi_i \lambda_{i,j}^T T_i \\ q_j &= \pi_i \lambda_{i,j}^T h_i \end{aligned}$$

- One master problem, n subproblems
- Theoretical convergence guarantees
- Convergence can be improved through regularization procedures
- Readily extended to operate in parallel on distributed data

Implementation - StochasticPrograms.jl

- Flexible problem definition
- Deferred model instantiation
- Scenario data injection
- Memory-distributed
- Minimize data passing
  - Lightweight sampler objects to generate data
  - Lightweight model recipes to generate second stage problems
- Interface to structure-exploiting solver algorithms
- Registered Julia package

Implementation - Model recipes

```julia
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

Implementation - Model recipes

```
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

JuMP syntax

Implementation - Model recipes

```
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

$$\begin{aligned} \underset{x_1, x_2 \in \mathbb{R}}{\text{minimize}} \quad & 100x_1 + 150x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 120 \\ & x_1 \geq 40 \\ & x_2 \geq 20 \end{aligned}$$

Implementation - Model recipes

```
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

$$
\begin{aligned}
\underset{y_1, y_2 \in \mathbb{R}}{\text{minimize}} \quad & q_1(\xi)\ y_1 + q_2(\xi)\ y_2 \\
\text{s.t.} \quad & 6y_1 + 10y_2 \leq 60\ x_1 \\
& 8y_1 + 5y_2 \leq 80\ x_2 \\
& 0 \leq y_1 \leq d_1(\xi) \\
& 0 \leq y_2 \leq d_2(\xi)
\end{aligned}
$$

Implementation - Model recipes

```julia
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

$$
\begin{aligned}
\underset{y_1, y_2 \in \mathbb{R}}{\text{minimize}} \quad & q_1(\xi)\; y_1 + \; q_2(\xi)\; y_2 \\
\text{s.t.} \quad & 6y_1 + 10y_2 \leq 60\; x_1 \\
& 8y_1 + 5y_2 \leq 80\; x_2 \\
& 0 \leq y_1 \leq \; d_1(\xi) \\
& 0 \leq y_2 \leq \; d_2(\xi)
\end{aligned}
$$

Implementation - Model recipes

```
@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁ + x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    ξ = scenario
    @variable(model, 0 <= y₁ <= ξ.d₁)
    @variable(model, 0 <= y₂ <= ξ.d₂)
    @objective(model, Min, ξ.q₁*y₁ + ξ.q₂*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```

$$\begin{aligned} \underset{y_1, y_2 \in \mathbb{R}}{\text{minimize}} \quad & \boxed{q_1(\xi)}\, y_1 + \boxed{q_2(\xi)}\, y_2 \\ \text{s.t.} \quad & 6y_1 + 10y_2 \leq 60\ x_1 \\ & 8y_1 + 5y_2 \leq 80\ x_2 \\ & 0 \leq y_1 \leq \boxed{d_1(\xi)} \\ & 0 \leq y_2 \leq \boxed{d_2(\xi)} \end{aligned}$$

Implementation - LShapedSolvers.jl

- Collection of L-shaped algorithms

Implementation - LShapedSolvers.jl

- Collection of L-shaped algorithms
- Eight variants in total
  - Three different regularization procedures
  - Distributed variants of each algorithm
  - Numerous tweakable parameters

Implementation - LShapedSolvers.jl

- Collection of L-shaped algorithms
- Eight variants in total
  - Three different regularization procedures
  - Distributed variants of each algorithm
  - Numerous tweakable parameters
- Trait-based implementation

Implementation - LShapedSolvers.jl

- Collection of L-shaped algorithms
- Eight variants in total
  - Three different regularization procedures
  - Distributed variants of each algorithm
  - Numerous tweakable parameters
- Trait-based implementation
- Interfaces to StochasticPrograms.jl

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data
- A `Channel` administers data on one process

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data
- A `Channel` administers data on one process
- The `Channel` data is readable and writable from all processes.
- Calling `put` on a remote channel involves data passing

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data
- A `Channel` administers data on one process
- The `Channel` data is readable and writable from all processes.
- Calling `put` on a remote channel involves data passing
- Calling `fetch` on a remote channel involves data fetching

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data
- A `Channel` administers data on one process
- The `Channel` data is readable and writable from all processes.
- Calling `put` on a remote channel involves data passing
- Calling `fetch` on a remote channel involves data fetching
- A remote call returns a `Future` to the result

Implementation - Distributed computations in Julia

- Distributed computing in Julia revolves around two primitives:
  - Remote references: administer which node data resides on
  - Remote calls: schedule execution tasks on the distributed data
- A `Channel` administers data on one process
- The `Channel` data is readable and writable from all processes.
- Calling `put` on a remote channel involves data passing
- Calling `fetch` on a remote channel involves data fetching
- A remote call returns a `Future` to the result
- A process can `wait` for data to arrive on a remote reference

Implementation - Distributed L-shaped channels

Master node

- `Decisions`: Master solutions ($\mathcal{D}$)
- `CutQueue`: Optimality cuts from workers ($\mathcal{C}$)

Worker nodes

- `Worker`: Local subproblems ($\mathcal{S}$)
- `Work`: Index into `Decisions` ($\mathcal{W}$)

Implementation - Distributed L-shaped channels

Master node

- `Decisions`: Master solutions ($\mathcal{D}$)
- `CutQueue`: Optimality cuts from workers ($\mathcal{C}$)

Worker nodes

- `Worker`: Local subproblems ($\mathcal{S}$)
- `Work`: Index into `Decisions` ($\mathcal{W}$)

- Master determines first stage decisions and shedules worker tasks
- Workers solve subproblems given first stage decisions and generate optimality cuts
- The amount of cuts needed to proceed is governed by a asynchronicity parameter κ
- Timestamps used to synchronize and check convergence

Implementation - Distributed L-shaped tasks

Master node

```
function do_work!(master::Master,
                  cuts::CutQueue,
                  decisions::Decisions,
                  workers::Vector{Work})
    x₀ = initialize()
    put!(decisions, 0, x₀)
    send_work(workers, 1)
    while true
        wait(cuts)
        (t,Q,cut) = take!(cuts)
        add_cut!(master,cut)
        if added_cuts(master,t) ≥ κ*nscenarios(master)
            # Enough information to resolve master
            xₜ₊₁ = solve(master)
            # Send new work to remote nodes
            put!(decisions, t+1, xₜ₊₁)
            send_work(workers, t+1)
        end
        if added_cuts(master,t) == nscenarios(master) && converged(master)
            return :Optimal
        end
    end
end
```

Implementation - Distributed L-shaped tasks

Worker nodes

```julia
function do_work!(worker::Worker,
                  work::Work,
                  cuts::CutQueue,
                  decisions::Decisions)
    subproblems::Vector{SubProblem} = fetch(worker)
    while true
        wait(work)
        t::Int = take!(work)
        if t == -1
            # Worker finished
            return
        end
        x = fetch(decisions,t)
        # Update and solve all local subproblems
        @sync for subproblem in subproblems
            @async begin
                update_subproblem!(subproblem,x)
                cut = subproblem()
                Q = cut(x)
                # Send optimality cut to master, with timestamp
                # of decision and objective value
                put!(cuts,(t,Q,cut))
            end
        end
    end
end
```

Implementation - Distributed L-shaped

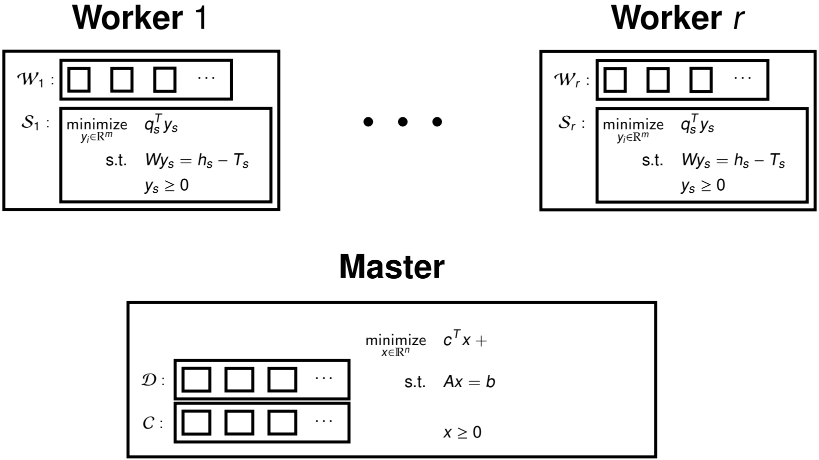


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

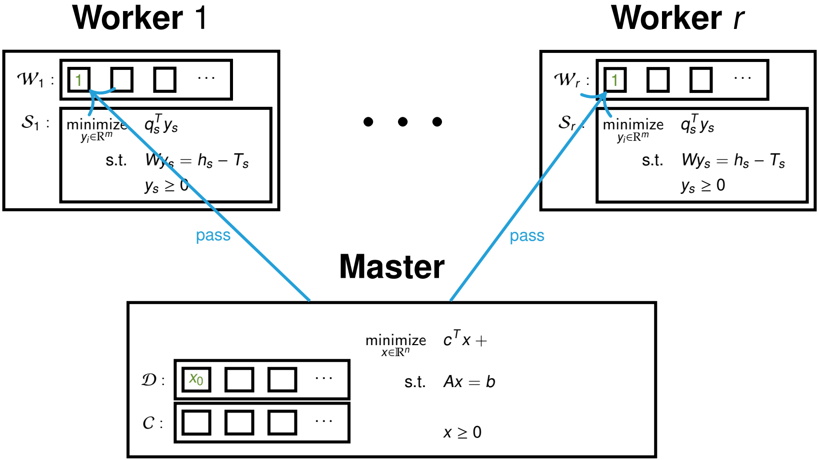


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

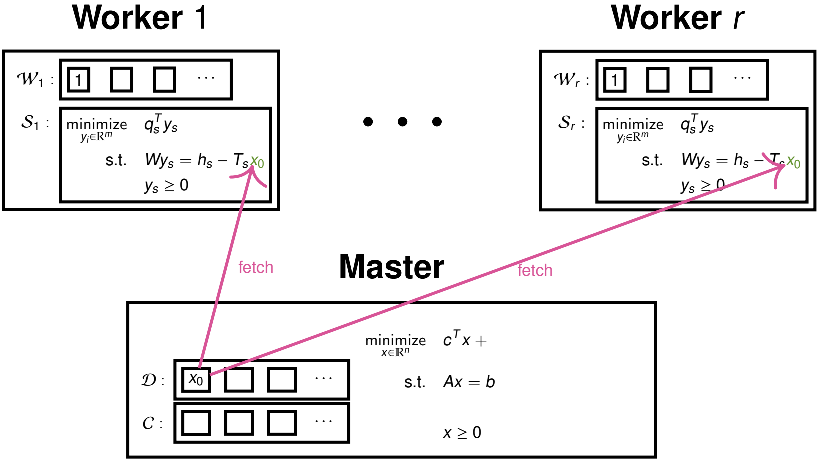


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

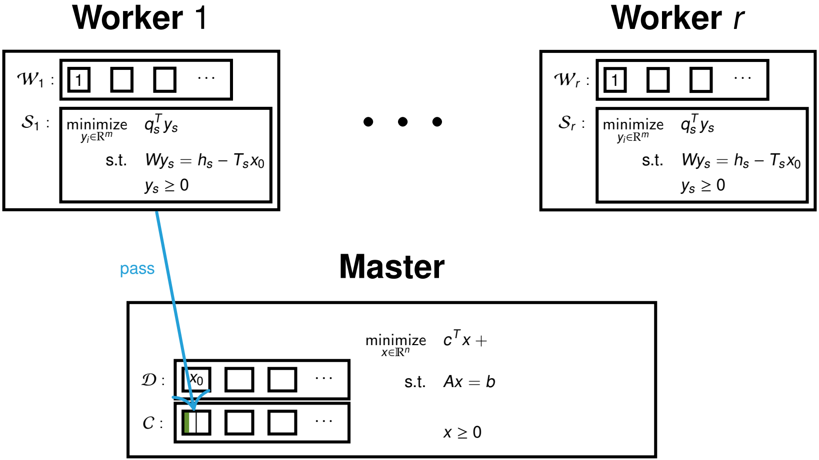


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

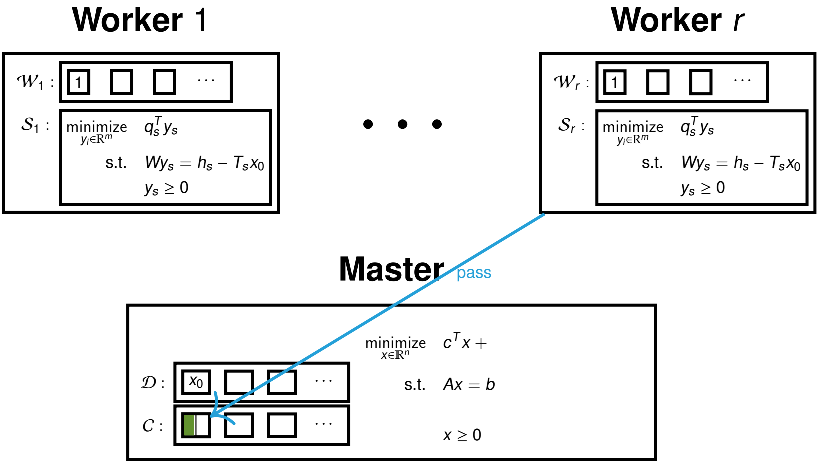


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

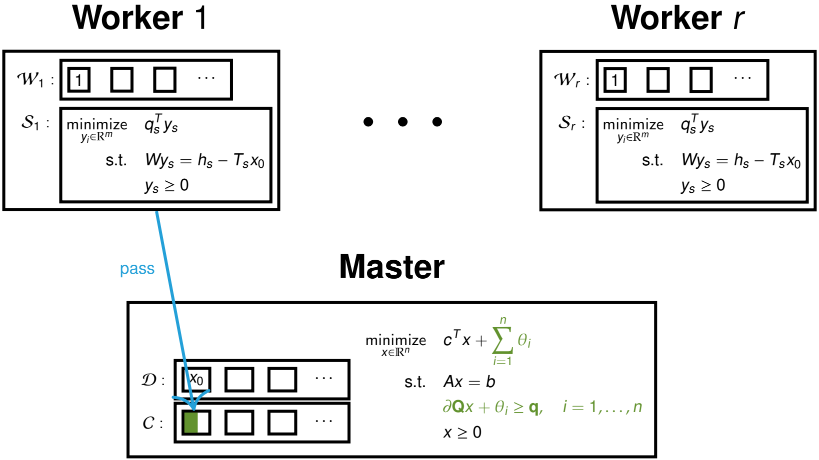


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

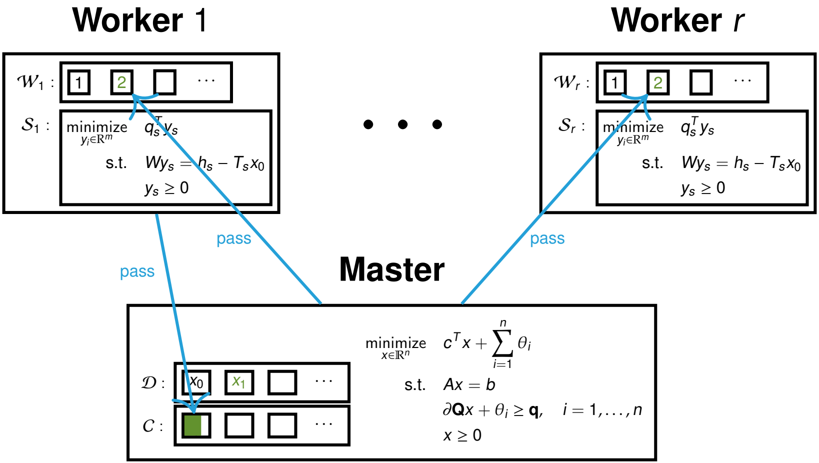


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

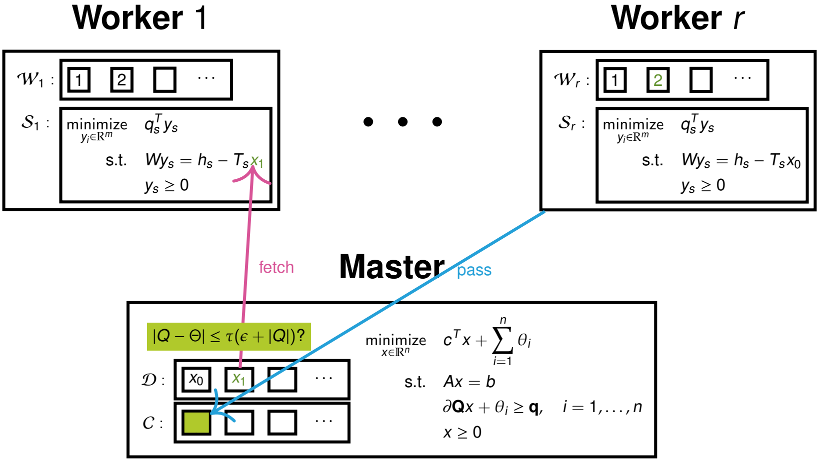


Figure: Distributed L-shaped procedure

Implementation - Distributed L-shaped

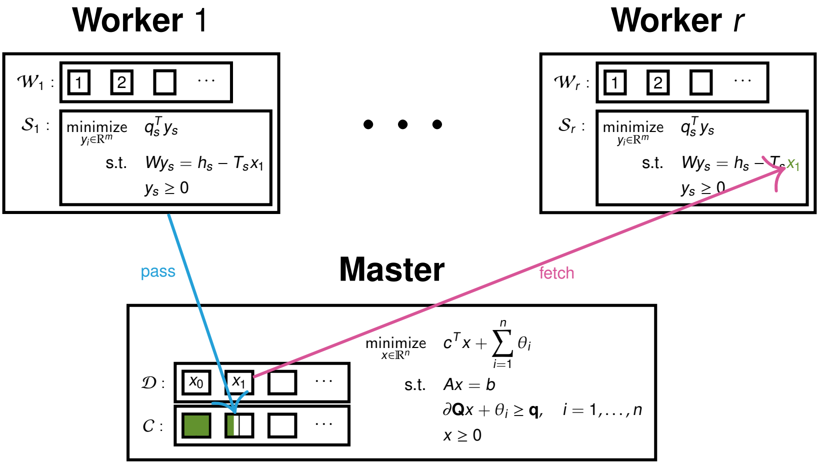


Figure: Distributed L-shaped procedure

Numerical experiments - Day-ahead problem

- Optimal order strategies on a deregulated electricity market
- From the perspective of a hydropower producer
- First stage: Hourly electricity volume bids for the upcoming day
- Second stage: Optimize production when market price is known
- Market data taken from NordPool, used to sample scenarios
- Physical data on hydroelectric plants in river Skellefteälven
- Full model included in `HydroModels.jl`

Numerical experiments - Convergence

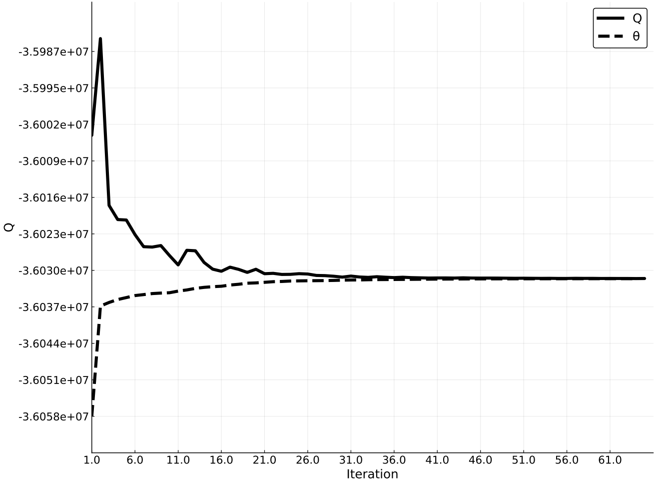


Figure: L-shaped convergence for a day-ahead problem with 10 price scenarios.

Numerical experiments - Single node

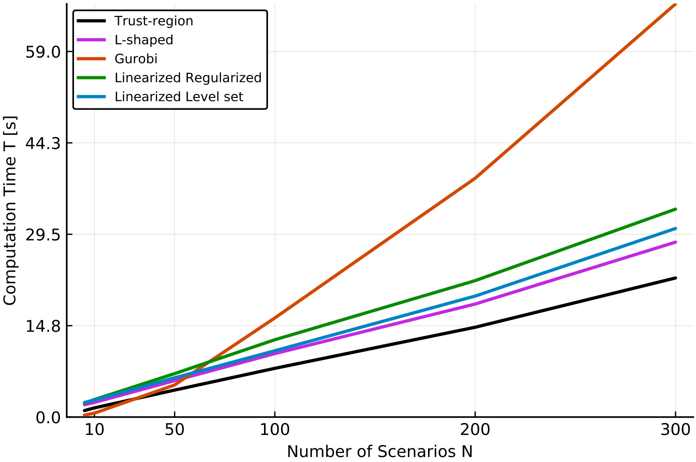


Figure: Median computation time required to solve day-ahead problems.

Numerical experiments - Strong scaling

- Day-ahead problem with 1000 price scenarios.
- Results in 2.5 million variables and 1.4 million constraints.
- Solving the extended form required 350+ seconds.

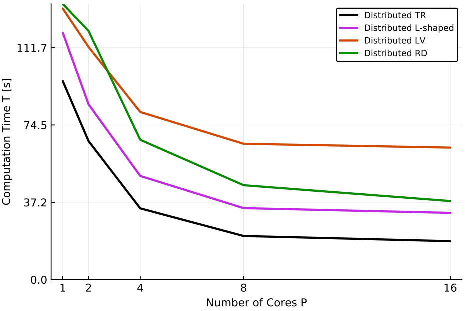


Figure: Computation time.

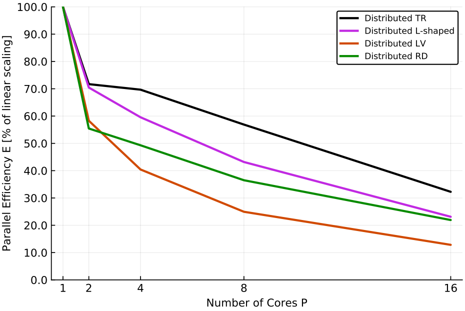


Figure: Parallel efficiency.

Numerical experiments - Load imbalance

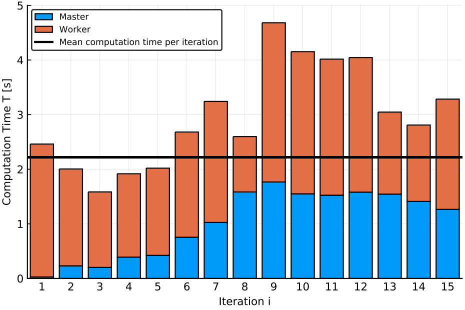


Figure: 4 workers.

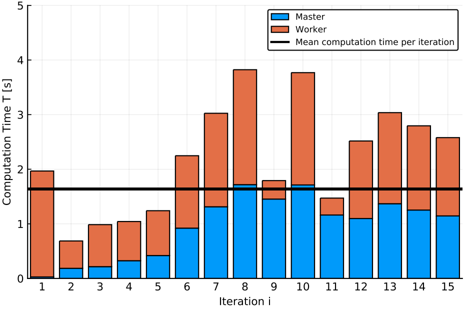


Figure: 16 workers.

Numerical experiments - Load imbalance

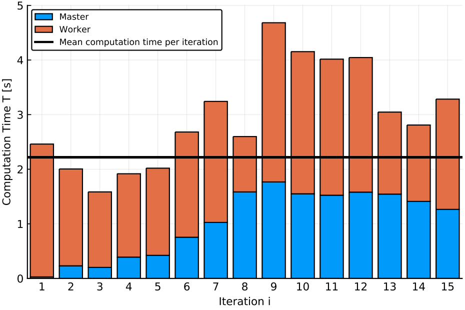


Figure: 4 workers with $\kappa = 1$.

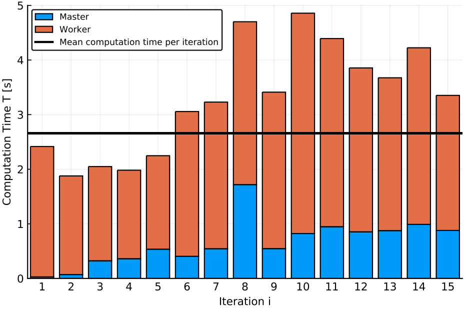


Figure: 4 workers with $\kappa = 0.5$.

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance
- Performance of regularized variants affected by flat objective

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance
- Performance of regularized variants affected by flat objective

Outlook on future work

- Evaluate on other applied problems
  - Larger scale
  - Less flat

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance
- Performance of regularized variants affected by flat objective

Outlook on future work

- Evaluate on other applied problems
  - Larger scale
  - Less flat
- Multi-node testing

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance
- Performance of regularized variants affected by flat objective

Outlook on future work

- Evaluate on other applied problems
  - Larger scale
  - Less flat
- Multi-node testing
- Algorithmic improvements

Final Remarks

Discussion

- L-shaped algorithms outperform solving the extended form directly
- Scalability affected by load imbalance
- Performance of regularized variants affected by flat objective

Outlook on future work

- Evaluate on other applied problems
  - Larger scale
  - Less flat
- Multi-node testing
- Algorithmic improvements
- Bundling procedures to reduce load imbalance

Final Remarks

Julia as an alternative to MPI

- Complexity versus implementation effort
  - Abstractions for distributed computing are simple and efficient
  - High-level features for modeling optimization problems

Final Remarks

Julia as an alternative to MPI

- Complexity versus implementation effort
  - Abstractions for distributed computing are simple and efficient
  - High-level features for modeling optimization problems
- MPI communicators can be used through `MPI.jl`

Final Remarks

Julia as an alternative to MPI

- Complexity versus implementation effort
  - Abstractions for distributed computing are simple and efficient
  - High-level features for modeling optimization problems
- MPI communicators can be used through `MPI.jl`
- Prototype on laptop, run the same code on a supercomputer

Final Remarks

Summary

- Memory-distributed stochastic programs

Final Remarks

Summary

- Memory-distributed stochastic programs
- L-shaped algorithms that run in parallel on distributed data

Final Remarks

Summary

- Memory-distributed stochastic programs
- L-shaped algorithms that run in parallel on distributed data
- Simple Julia abstractions enable complex parallel algorithms

Final Remarks

Summary

- Memory-distributed stochastic programs
- L-shaped algorithms that run in parallel on distributed data
- Simple Julia abstractions enable complex parallel algorithms
- Framework for formulating and solving stochastic programs

Final Remarks

Summary

- Memory-distributed stochastic programs
- L-shaped algorithms that run in parallel on distributed data
- Simple Julia abstractions enable complex parallel algorithms
- Framework for formulating and solving stochastic programs
- The full framework is open-source and freely available on Github

`https://github.com/martinbiel`