Stochastic Integer Programming: Parallel distributed-memory algorithms

Deepak Rajan

Collaborators:
Geoffrey Oxberry (LLNL),
Lluís Munguía, Kevin Ryan (Georgia Tech),
Shabbir Ahmed, Santanu Dey (Georgia Tech),
Cosmin Petra (Argonne)

LLNL-PRES-683481
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC
Motivating Example: Stochastic unit commitment

- **Stochastic** unit commitment is used to minimize the day-ahead cost of energy generation.

- **Decisions:**
  - Which generators to operate?
  - How much power to produce at each generator?

- **Why is this hard?**
  - **Discrete** decisions (MIPs are hard)
  - Renewable sources can be **unreliable** (SMIPs are even harder!)

- **Goal:**
  - Meet **uncertain** demand
  - Minimize **expected** cost

Stochastic Mixed Integer Programming: an overview

- Stochastic programming models optimization problems involving uncertainty

- We consider two-stage stochastic mixed-integer programs (SMIPs) with recourse:
  - Mixed-Integer: Some decision variables are integer-valued
  - 1st stage: deterministic “now” decisions
  - 2nd stage: depends on random events and first-stage decisions

\[
\min_x \left\{ c^t x + \mathbb{E}_p[Q(x, \omega)] | A x \leq b, x_j \in \mathbb{Z}, \forall j \in I_1 \right\}
\]
\[
Q(x, \omega) = \min_y \left\{ q^t y | W y \leq h - T x, y_j \in \mathbb{Z}, \forall j \in I_2 \right\}
\]

- Cost function includes first-stage variables and expected value function of second-stage variables that depends on random parameters
Parallel algorithms for SMIPs: Decomposition-based approaches

- Leverage special structure of SMIP formulations to develop decomposition-based schemes
- Decomposition can be applied at many different levels of the optimization process
- Approach 1: Decomposition applied at the linear-algebra level of the algorithm (PIPS-SBB: A SMIP solver that leverages problem structure)
- Approach 2: Decomposition applied at the decision-variable level (Scenario Decomposition and Grouping for Binary Stochastic Programs)
Approach 1: Deterministic equivalents of SMIPs have block-angular structure

- We consider deterministic equivalent formulations of 2-stage SMIPs under the sample average approximation.
- This assumption yields characteristic block-angular structure.

\[
\min_x \{ c^T x + \mathbb{E}_p[Q(x, \omega)] | Ax \leq b, x_j \in \mathbb{Z}, \forall j \in I_1 \}
\]

\[
Q(x, \omega) = \min_y \{ q^T y | Wy \leq h - Tx, y_j \in \mathbb{Z}, \forall j \in I_2 \}
\]
Branch and Bound for SMIPs bottlenecked by LP solves

- Branch & Bound (B&B) is a standard technique for solving MIPs
  - Solves a sequence of Linear Programming (LP) relaxations
- B&B tree search is parallelizable (distributed- or shared-memory)
- LP relaxations solved sequentially or via shared-memory parallelism
- For large-scale instances, LP solves are performance bottleneck
PIPS-SBB addresses performance bottlenecks in memory and LP solve time

- PIPS-SBB is a specialized B&B solver for two-stage SMIPs
- Main Idea: Distributed-memory LP solves can address more memory and reduce LP solve execution time

1st stage distributed redundantly (all processes)

2nd stage is distributed by scenario
PIPS-SBB addresses performance bottlenecks in memory and LP solve time

- PIPS-SBB is a specialized B&B solver for two-stage SMIPs
- Main Idea: Distributed-memory LP solves can address more memory and reduce LP solve execution time
- Uses PIPS-S, a distributed-memory parallel simplex-based stochastic LP solver (Developed at Argonne - Lubin et. al., 2013)
PIPS-SBB addresses performance bottlenecks in memory and LP solve time

- PIPS-SBB is a specialized B&B solver for two-stage SMIPs
- Main Idea: Distributed-memory LP solves can address more memory and reduce LP solve execution time
- Uses PIPS-S, a distributed-memory parallel simplex-based stochastic LP solver (Developed at Argonne - Lubin et. al., 2013)
- PIPS-SBB is extensible; heuristics and branching rules are plugins

Example: branching rule plugins

Branching Rule (abstract base class) →
- Pseudo-cost branching
- Minimum index branching
- Maximum fractional-part branching
- (insert your favorite branching rule here!)
PIPS-SBB uses many MIP methods to improve performance

- MIPs are hard, no matter how fast we can solve LP relaxations
- But we have a good bag of methods
  - Primal Heuristics (quick and fast algorithms for finding solutions)
  - Cutting planes (Finding new inequalities that cut off part of the problem domain)
  - Branching strategies (How do we partition the problem?)
  - Node selection strategies (Which subproblem do we solve next?)
  - Presolving (Analysis of the problem structure to find simplifications)

- All of these algorithms must be implemented in a way that:
  - accounts for the block-angular data distribution
  - preserves block-angular structure
- Implementing each class of methods requires substantial work
MIP solvers have improved a lot over time (~30,000x over ~25 yrs)

- MIP solvers are highly complex software, which have been tuned and improved over decades.
- We will mine this work for further improvements to PIPS-SBB

Experimental performance results

- We test PIPS-SBB on SSLP instances, from the SIPLIB library
  - First introduced in Ntaimo and Sen (2004)
  - See http://www2.isye.gatech.edu/~sahmed/siplib/sslp/sslp.html

- SSLP instances model server locations under uncertainty

- Instances are coded as SSLP $m.n.s$, where $s$ represents #scenarios

- Larger number of scenarios means bigger problems
  - LP relaxations of all SSLP instances fit in memory
  - PIPS-SBB can handle much larger LP relaxations

- PIPS-SBB run on Sierra cluster at LLNL:
  - Each node: Intel Xeon EP X5660, 2.8 GHz, 12 MB cache
  - 12 cores/node, 24 GB RAM/node
  - Infiniband QDR interconnect
PIPS-SBB: Comparison against state-of-the-art MIP solver

Performance comparison against CPLEX 12.6.1

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Cores</th>
<th>RelGap (Time)</th>
<th>Best Solution Time</th>
<th>Solution Quality</th>
<th>CP RelGap (Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.45.5</td>
<td>2</td>
<td>1.36%</td>
<td>1488s</td>
<td>1.07%</td>
<td>(4s)</td>
</tr>
<tr>
<td>15.45.10</td>
<td>2</td>
<td>7.93%</td>
<td>2129s</td>
<td>7.26%</td>
<td>(1s)</td>
</tr>
<tr>
<td>15.45.15</td>
<td>2</td>
<td>5.25%</td>
<td>2392s</td>
<td>4.84%</td>
<td>(12s)</td>
</tr>
<tr>
<td>5.25.50</td>
<td>1</td>
<td>(12.34s)</td>
<td>12s</td>
<td>0%</td>
<td>(1s)</td>
</tr>
<tr>
<td>5.25.100</td>
<td>1</td>
<td>(41.63s)</td>
<td>41s</td>
<td>0%</td>
<td>(1s)</td>
</tr>
<tr>
<td>10.50.50</td>
<td>5</td>
<td>1.48%</td>
<td>923s</td>
<td>1.31%</td>
<td>(81s)</td>
</tr>
<tr>
<td>10.50.100</td>
<td>10</td>
<td>1.74%</td>
<td>194s</td>
<td>1.56%</td>
<td>(442s)</td>
</tr>
<tr>
<td>10.50.500</td>
<td>50</td>
<td>1.57%</td>
<td>2792s</td>
<td>-7.32%</td>
<td>(M) 10.13%</td>
</tr>
<tr>
<td>10.50.1000</td>
<td>100</td>
<td>1.60%</td>
<td>2397s</td>
<td>-11.19%</td>
<td>14.47%</td>
</tr>
<tr>
<td>10.50.2000</td>
<td>100</td>
<td>24.00%</td>
<td>2384s</td>
<td>-0.73%</td>
<td>20.33%</td>
</tr>
</tbody>
</table>

Time limit: 1 hour

But we require a bigger bag of tricks!

These require cutting planes!

We solve the small instances, albeit with ~10x worse performance

On big instances we do better!
PIPS-SBB: Importance of specialized SMIP-structure aware algorithms

Performance comparison against General (structure oblivious) PIPS-SBB

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Cores</th>
<th>RelGap (Time) Stochastic PIPS-SBB</th>
<th>RelGap (Time) General PIPS-SBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.45.5</td>
<td>2</td>
<td>1.36 %</td>
<td>4.23%</td>
</tr>
<tr>
<td>15.45.10</td>
<td>2</td>
<td>7.93 %</td>
<td>8.39%</td>
</tr>
<tr>
<td>15.45.15</td>
<td>2</td>
<td>5.25 %</td>
<td>8.26%</td>
</tr>
<tr>
<td>5.25.50</td>
<td>1</td>
<td>(12.34s)</td>
<td>289.71%</td>
</tr>
<tr>
<td>5.25.100</td>
<td>1</td>
<td>(41.63s)</td>
<td>65.42%</td>
</tr>
<tr>
<td>10.50.50</td>
<td>5</td>
<td>1.48%</td>
<td>27.13%</td>
</tr>
<tr>
<td>10.50.100</td>
<td>10</td>
<td>1.74 %</td>
<td>28.60%</td>
</tr>
<tr>
<td>10.50.500</td>
<td>50</td>
<td>1.57 %</td>
<td>29.13%</td>
</tr>
<tr>
<td>10.50.1000</td>
<td>100</td>
<td>1.60 %</td>
<td>∞</td>
</tr>
<tr>
<td>10.50.2000</td>
<td>100</td>
<td>24.00 %</td>
<td>∞</td>
</tr>
</tbody>
</table>

Time limit: 1 hour

Even small instances cannot be solved unless we use structure information.

On big instances we need structure information to even get a feasible solution.
PIPS-SBB: Scaling results

MIP performance of PIPS-SBB is similar to LP performance of PIPS-S
Approach 1: Summary and Future Directions

- We developed a distributed memory branch-and-bound SMIP solver called PIPS-SBB
  - Based on parallel distributed memory simplex solver PIPS-S
- PIPS-SBB can address more memory than CPLEX
- PIPS-SBB: node processing rates and performance scales to 50 cores for large instances
- Structure-aware MIP methods outperform general methods for solving SMIPs

Future directions
- Further increasing parallelism by parallelizing tree search
- Utilize distributed-memory interior-point methods for stochastic LPs (PIPS-IPM) to increase scalability
- Implement more structure-aware versions of general MIP methods (cuts, cuts, cuts!)
Approach 2: Scenario Decomposition and Grouping schemes for Binary Stochastic Programs (First-Stage Binary)

- We consider deterministic equivalent formulations of 2-stage SMIPs under the sample average approximation
- This assumption yields block-angular structure with non-anticipativity constraints, where we assume

- Relaxing non-anticipativity constraints, we can decompose by scenario to get
Decomposition algorithms for SMIPs are not new...

- Dual Decomposition (Caroe and Schultz, 1999)
- Benders Decomposition (Benders, 1962), L-Shaped (Van Slyke and Wets, 1969)
- Branch and Fix (Alonso-Ayuso, 2003)
- Disjunctive Decomposition (Ntaimo and Sen, 2005, 2008)
- **Scenario Decomposition (Ahmed 2013)**
- Many others…
Scenario Decomposition algorithm

- Finitely convergent to optimality for binary first-stage
- Can easily include Lagrangian multiplier updates

\[ B > LB, \text{ GOTO 1} \]
Parallel Scenario Decomposition algorithm: Synchronous implementation

- Easily parallelizes to a synchronous algorithm

\[ B > LB, \text{ GOTO 1} \]
Asynchronous Scenario Decomposition algorithm

- Significant improvement over synchronous algorithm
  - Worker processes do not sit idle
  - Works well for instances where seen solve/eval work is unbalanced

- Incorporates performance improvements aimed at reducing upper bound evaluation time

- Includes lower bound improvements
  - Optimality cuts originally proposed in Santoso et. al. (2005)
  - **New**: Scenario grouping that optimizes improvements in lower bounds
Asynchronous Scenario Decomposition algorithm: Comparison against state-of-the-art for SSLP instances

- Scenario Decomposition scheme significantly outperforms general MIP solvers for large number of scenarios
- Asynchronous algorithm 3x better than Synchronous algorithm

<table>
<thead>
<tr>
<th>Scenario</th>
<th>CPLEX 12.5 Time Limit: 3 hrs</th>
<th>Nodes/Cores: 1/12</th>
<th>Cores = #scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50_1000</td>
<td>142</td>
<td>63</td>
<td>(0.17%)</td>
</tr>
<tr>
<td>sslp_10_50_2000</td>
<td>312</td>
<td>143</td>
<td>(0.30%)</td>
</tr>
<tr>
<td>sslp_15_45_5</td>
<td>4*</td>
<td>1*</td>
<td>2</td>
</tr>
<tr>
<td>sslp_15_45_10</td>
<td>17*</td>
<td>6*</td>
<td>2</td>
</tr>
<tr>
<td>sslp_15_45_15</td>
<td>29*</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>
SSCh instances: Experimental performance results

- First introduced in Alonso-Ayuso (2003)
- SSCh instances supply chain problems under uncertainty
- 9 instances are coded as c1-c10
- 67-78 binary first stage variables, 3,000 continuous second stage variables, 23 equi-probable scenarios
- 5 previously unsolved instances ("hard" instances, rest "easy")
Asynchronous Scenario Decomposition algorithm: Comparison against state-of-the-art for SSCh instances

- Adding cuts (AS+Cut) solves all “easy” instances
- Solves additional “hard” instance

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Time Limit</th>
<th>Cuts</th>
<th>Resource</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>c10</td>
<td>139,738</td>
<td>3hrs</td>
<td>(9,649)</td>
<td>181</td>
<td>OPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,732)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>201,454</td>
<td></td>
<td>(12,202)</td>
<td>(5,901)</td>
<td>(9,832)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(3,523)</td>
<td></td>
<td>(18,427)</td>
</tr>
<tr>
<td>c6</td>
<td>231,368</td>
<td></td>
<td>(10,514)</td>
<td>(4,828)</td>
<td>(10,273)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(10,273)</td>
<td></td>
<td>(8,825)</td>
</tr>
<tr>
<td>c8</td>
<td>100,523</td>
<td></td>
<td>(5,071)</td>
<td>2,545</td>
<td>(3,106)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(13,842)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time limit: 3hrs
Nodes/Cores: 2/24
Scenario Grouping as pre-processing: Motivation

- ISSUE: Scenario Decomposition Lower Bound may be too weak
- IDEA: ‘Group’ Scenarios by relaxing only a subset of non-anticipativity constraints
- QUESTIONS:
  - Why does Grouping work?
ISSUE: Scenario Decomposition Lower Bound may be too weak

IDEA: ‘Group’ Scenarios by relaxing only a subset of non-anticipativity constraints

QUESTIONS:
• Why does Grouping work?
• What metric should we use to ‘group’ scenarios?

ANSWER: Group to maximize lower bound improvement
• New: Framed as an scenario partitioning problem, given maximum group size (P)
• Solve problem as an Integer Program

ALGORITHM:
• Decompose and solve scenario problems
• Choose max partition size P, solve for optimal partition
• Group scenarios based on optimal partition
• Solve grouped problem using scenario decomposition algorithm of choice
Asynchronous Scenario Decomposition algorithm: Comparison against state-of-the-art for SSCh instances

- Optimal Scenario Grouping as pre-processing solves all instances
- Even random partitioning helps (somewhat)

| 9 | (1,732) | 20 | 13 | 41 |

Time limit: 3hrs
Nodes/Cores: 1/12
Approach 2: Summary and Future Directions

- Asynchronous scenario decomposition algorithm significantly more effective than synchronous implementation and extensive formulation
- Optimal scenario grouping scheme used as preprocessing step
- Solved previously unsolved instances, demonstrated effectiveness on standard test instances
- Future directions: Extend scenario grouping scheme to a finitely convergent scheme for all SMIPs, not just with binary first-stage
Other Recent/Ongoing Stochastic Optimization Projects at LLNL

- Topology Optimization under Uncertainty (PI: Geoffrey Oxberry), funded by LLNL LDRD
- The Value of Energy Storage and Demand Response for Renewable Integration in California (PI: Tom Edmunds), funded by California Energy Commission
- Evaluation of Robust Unit Commitment using High Performance Computing (PI: Liang Min), funded by HPC4energy

Acknowledgments/Auspices

- Prepared by LLNL under Contract DE-AC52-07NA27344
- LLNL LDRD Funding (15-ERD-041)
- SCIP team (LP relaxation diagram)
- Cosmin Petra (ANL) supported by DOE OE and DOE ASCR M²ACS project
Thank you!