Computational Approaches to Mixed Integer Second Order Cone Optimization (MISOCP)

Aykut Bulut[1]
Ted Ralphs[2]

[1]The MathWorks, Inc.
[2] COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University

INFORMS Annual Meeting 2017,
23 October 2017

Outline

MISOCP Definition

- We are interested in solving Mixed Integer Second Order Conic Optimization (MISOCP) problems.
- MISOCP is a generalization of Mixed Integer Linear Optimization (MILP).
- MISOCP can be formulated as follows,

$$
\begin{aligned}
\min \quad & c^\top x \\
s.t. \quad & Ax = b \\
& x \in \mathbb{L}^1 \times \cdots \times \mathbb{L}^k \\
& x_i \in \mathbb{R}_+ \qquad\qquad i \in I \\
& x_j \in \mathbb{Z}_+ \qquad\qquad j \in J.
\end{aligned}
\tag{MISOCP}
$$

Motivation: Evaluation of Solution Approaches for MISOCP

- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Motivation: Evaluation of Solution Approaches for MISOCP

- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Motivation: Evaluation of Solution Approaches for MISOCP

- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Motivation: Evaluation of Solution Approaches for MISOCP

- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Motivation: Evaluation of Solution Approaches for MISOCP

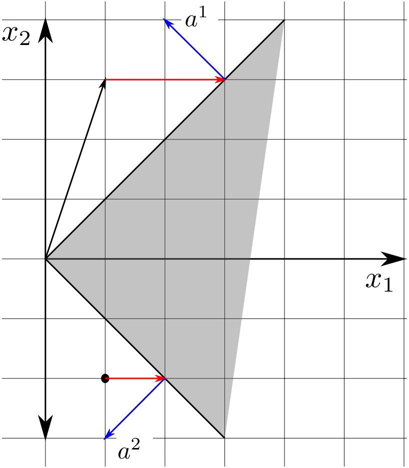
- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Motivation: Evaluation of Solution Approaches for MISOCP

- Choice of branch and bound subproblem: LP vs SOCP?
- What does an LP–based branch and cut for MISOCP look like?
- How to balance cutting and branching for LP–based approach?
- Do MILP cuts help in case of LP–based subproblems?
- Do disjunctive conic cuts help?
- Which branching strategy to use?

Separating Infeasible Directions/Solutions

Figure: Separation Example



Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

1: Solve SOCP.
2: Relax all integrality and conic constraints to create root node LP.
3: **while** there are nodes to process **do**
4:     Pick a node.
5:     Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:     Decide whether to constrain or branch.
7:     **while** cutting is preferred **do**
8:         Add cuts to the LP and solve.
9:         Decide whether to constrain or branch.
10:     **end while**
11:     Branch, remove current node.
12: **end while**

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1:  Solve SOCP.
2:  Relax all integrality and conic constraints to create root node LP.
3:  while there are nodes to process do
4:      Pick a node.
5:      Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:      Decide whether to constrain or branch.
7:      while cutting is preferred do
8:          Add cuts to the LP and solve.
9:          Decide whether to constrain or branch.
10:     end while
11:     Branch, remove current node.
12: end while
```

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1: Solve SOCP.
2: Relax all integrality and conic constraints to create root node LP.
3: while there are nodes to process do
4:     Pick a node.
5:     Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:     Decide whether to constrain or branch.
7:     while cutting is preferred do
8:         Add cuts to the LP and solve.
9:         Decide whether to constrain or branch.
10:    end while
11:    Branch, remove current node.
12: end while
```

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1:  Solve SOCP.
2:  Relax all integrality and conic constraints to create root node LP.
3:  while there are nodes to process do
4:      Pick a node.
5:      Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:      Decide whether to constrain or branch.
7:      while cutting is preferred do
8:          Add cuts to the LP and solve.
9:          Decide whether to constrain or branch.
10:     end while
11:     Branch, remove current node.
12: end while
```

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

1: Solve SOCP.
2: Relax all integrality and conic constraints to create root node LP.
3: **while** there are nodes to process **do**
4:     Pick a node.
5:     Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:     Decide whether to constrain or branch.
7:     **while** cutting is preferred **do**
8:         Add cuts to the LP and solve.
9:         Decide whether to constrain or branch.
10:     **end while**
11:     Branch, remove current node.
12: **end while**

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1:  Solve SOCP.
2:  Relax all integrality and conic constraints to create root node LP.
3:  while there are nodes to process do
4:      Pick a node.
5:      Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:      Decide whether to constrain or branch.
7:      while cutting is preferred do
8:          Add cuts to the LP and solve.
9:          Decide whether to constrain or branch.
10:     end while
11:     Branch, remove current node.
12: end while
```

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1:  Solve SOCP.
2:  Relax all integrality and conic constraints to create root node LP.
3:  while there are nodes to process do
4:      Pick a node.
5:      Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:      Decide whether to constrain or branch.
7:      while cutting is preferred do
8:          Add cuts to the LP and solve.
9:          Decide whether to constrain or branch.
10:     end while
11:     Branch, remove current node.
12: end while
```

LP–based Branch And Cut

Algorithm 1 LP–based Branch and Cut Algorithm (bb-lp)

```
1:  Solve SOCP.
2:  Relax all integrality and conic constraints to create root node LP.
3:  while there are nodes to process do
4:      Pick a node.
5:      Solve LP, if LP solution is feasible, update bounds and go to line 3.
6:      Decide whether to constrain or branch.
7:      while cutting is preferred do
8:          Add cuts to the LP and solve.
9:          Decide whether to constrain or branch.
10:     end while
11:     Branch, remove current node.
12: end while
```

Whether to Cut or Branch

Let $\bar{x}$ be LP subproblem solution.

- If $\bar{x}$ is conic infeasible,
  - Generate Outer Approximation (OA) cuts for at least α iterations.
  - Generate OA cuts for at most γ iterations if $\bar{x}$ is integer infeasible too.
  - After α iterations, generate OA cuts if last improvement in the LP bound was greater than β times difference of LP bound to current upper bound.
- If $\bar{x}$ is integer infeasible, and a fixed number of nodes passed since last, generate MILP cuts.

Whether to Cut or Branch

Let $\bar{x}$ be LP subproblem solution.

- If $\bar{x}$ is conic infeasible,
  - Generate Outer Approximation (OA) cuts for at least α iterations.
  - Generate OA cuts for at most γ iterations if $\bar{x}$ is integer infeasible too.
  - After α iterations, generate OA cuts if last improvement in the LP bound was greater than β times difference of LP bound to current upper bound.
- If $\bar{x}$ is integer infeasible, and a fixed number of nodes passed since last, generate MILP cuts.

Whether to Cut or Branch

Let $\bar{x}$ be LP subproblem solution.

- If $\bar{x}$ is conic infeasible,
  - Generate Outer Approximation (OA) cuts for at least α iterations.
  - Generate OA cuts for at most γ iterations if $\bar{x}$ is integer infeasible too.
  - After α iterations, generate OA cuts if last improvement in the LP bound was greater than β times difference of LP bound to current upper bound.
- If $\bar{x}$ is integer infeasible, and a fixed number of nodes passed since last, generate MILP cuts.

Whether to Cut or Branch

Let $\bar{x}$ be LP subproblem solution.

- If $\bar{x}$ is conic infeasible,
  - Generate Outer Approximation (OA) cuts for at least α iterations.
  - Generate OA cuts for at most γ iterations if $\bar{x}$ is integer infeasible too.
  - After α iterations, generate OA cuts if last improvement in the LP bound was greater than β times difference of LP bound to current upper bound.
- If $\bar{x}$ is integer infeasible, and a fixed number of nodes passed since last, generate MILP cuts.

Whether to Cut or Branch

Let $\bar{x}$ be LP subproblem solution.

- If $\bar{x}$ is conic infeasible,
  - Generate Outer Approximation (OA) cuts for at least α iterations.
  - Generate OA cuts for at most γ iterations if $\bar{x}$ is integer infeasible too.
  - After α iterations, generate OA cuts if last improvement in the LP bound was greater than β times difference of LP bound to current upper bound.
- If $\bar{x}$ is integer infeasible, and a fixed number of nodes passed since last, generate MILP cuts.

DisCO Solver

- A branch and cut framework to solve MISOCP. Extends COIN-OR's High-Performance Parallel Search (CHiPPS) framework for conic problems.
- Uses *conic* OSI to manipulate SOCP subproblems.
- Default behavior is LP–based branch and cut using CLP and *conic* CGL.
- Cplex, Mosek and Ipopt can be used through *conic* OSI interface.
- Reads problems in CBF and Mosek's extended MPS format.

DisCO Solver

- A branch and cut framework to solve MISOCP. Extends COIN-OR's High-Performance Parallel Search (CHiPPS) framework for conic problems.
- Uses *conic* OSI to manipulate SOCP subproblems.
- Default behavior is LP–based branch and cut using CLP and *conic* CGL.
- Cplex, Mosek and Ipopt can be used through *conic* OSI interface.
- Reads problems in CBF and Mosek's extended MPS format.

DisCO Solver

- A branch and cut framework to solve MISOCP. Extends COIN-OR's High-Performance Parallel Search (CHiPPS) framework for conic problems.
- Uses *conic* OSI to manipulate SOCP subproblems.
- Default behavior is LP–based branch and cut using CLP and *conic* CGL.
- Cplex, Mosek and Ipopt can be used through *conic* OSI interface.
- Reads problems in CBF and Mosek's extended MPS format.

DisCO Solver

- A branch and cut framework to solve MISOCP. Extends COIN-OR's High-Performance Parallel Search (CHiPPS) framework for conic problems.
- Uses *conic* OSI to manipulate SOCP subproblems.
- Default behavior is LP–based branch and cut using CLP and *conic* CGL.
- Cplex, Mosek and Ipopt can be used through *conic* OSI interface.
- Reads problems in CBF and Mosek's extended MPS format.

DisCO Solver

- A branch and cut framework to solve MISOCP. Extends COIN-OR's High-Performance Parallel Search (CHiPPS) framework for conic problems.
- Uses *conic* OSI to manipulate SOCP subproblems.
- Default behavior is LP–based branch and cut using CLP and *conic* CGL.
- Cplex, Mosek and Ipopt can be used through *conic* OSI interface.
- Reads problems in CBF and Mosek's extended MPS format.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Experimental Details

- Problem set contains CBLIB2014 problems, 6 Steiner Tree Problems and 41 randomly generated problems.
- Experiments are conducted in COR@L Lab, each node has 16 processors at 2 GHz and 32 GB of memory.
- Memory allowed for serial runs is limited to 2GB.
- Memory limit for parallel runs is 2GB per process.
- Time limit is 7100 seconds.
- Cplex 12.7 is used to solve SOCP problems.

Table: SOCP-based Branch and Bound Variations Experimented

Parameters	referred as
default	disco-socp
strong branching	disco-socp-strong
disjunctive cuts in root	disco-socp-dc-all
only best disjunctive cut	disco-socp-dc-best
parallel bb-socp with OpenMPI	disco-socp-mpi

Table: OA Branch and Cut Variations Experimented

Parameters	referred as
$\alpha \leftarrow 1, \beta \leftarrow 0.001, \gamma \leftarrow 50$	disco-lp
strong branching	disco-lp-strong
$\alpha \leftarrow 2$	disco-lp-2
$\alpha \leftarrow 4$	disco-lp-3
$\beta \leftarrow 0.01$	disco-lp-4
$\beta \leftarrow 0.0001$	disco-lp-5
$\gamma \leftarrow 20$	disco-lp-6
$\gamma \leftarrow 100$	disco-lp-7
add all disjunctive cuts at root node	disco-lp-dc-all
add only best disjunctive cut at root node	disco-lp-dc-best
no MILP cuts	disco-lp-nomilpcuts
parallel version with OpenMPI	disco-lp-mpi

Figure: disco-socp, CPU Time, Branching Strategies

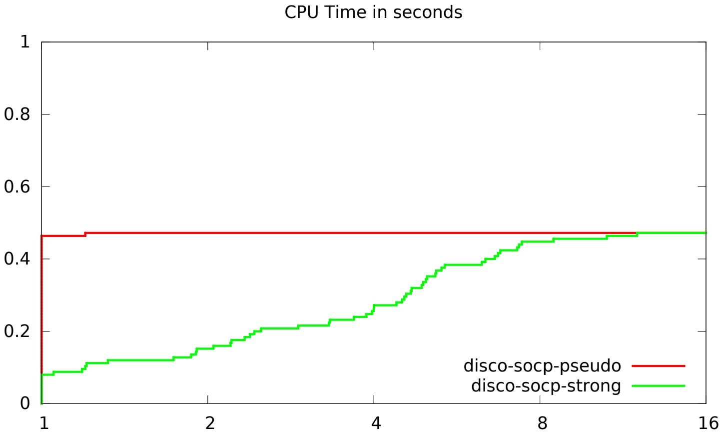


Figure: disco-socp, Number of Nodes, Branching Strategies

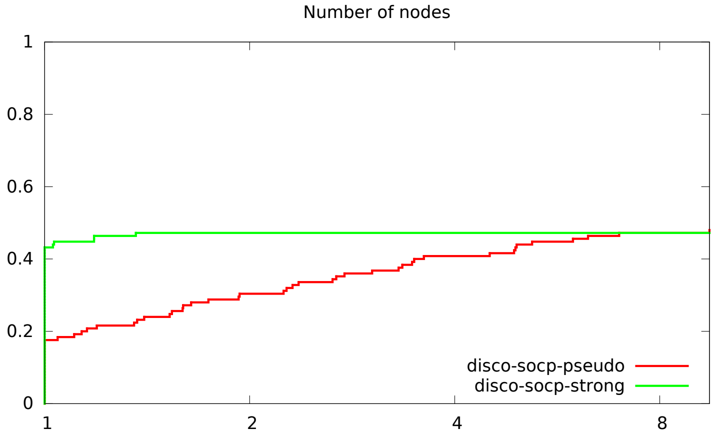


Figure: disco-lp, CPU Time, Branching Strategies

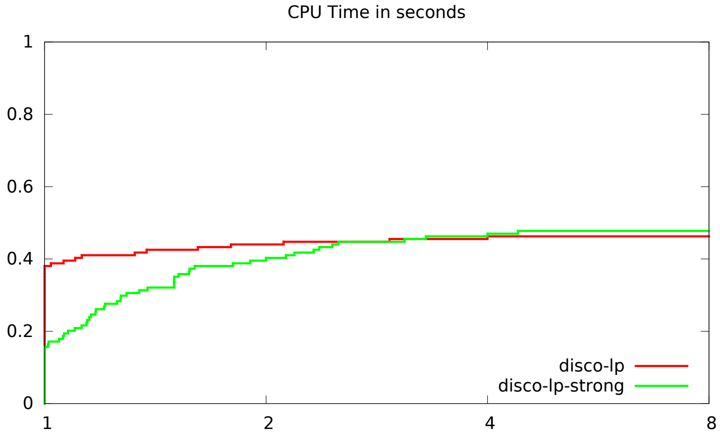


Figure: disco-lp, Number of Nodes, Branching Strategies

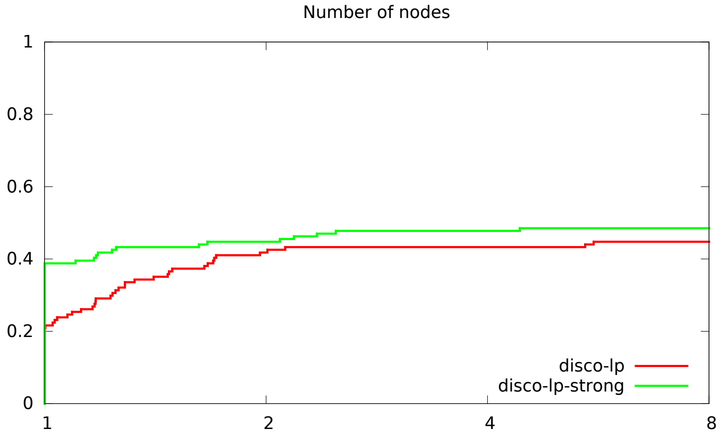


Figure: disco-lp, CPU Time, OA Cut Parameters

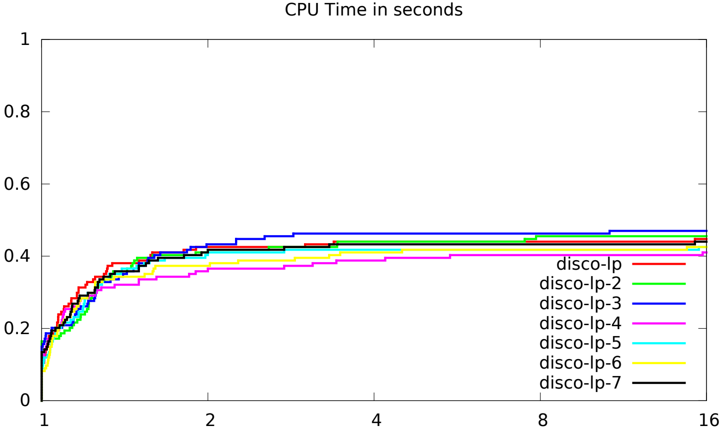


Figure: disco-lp, Number of Nodes, OA Cut Parameters

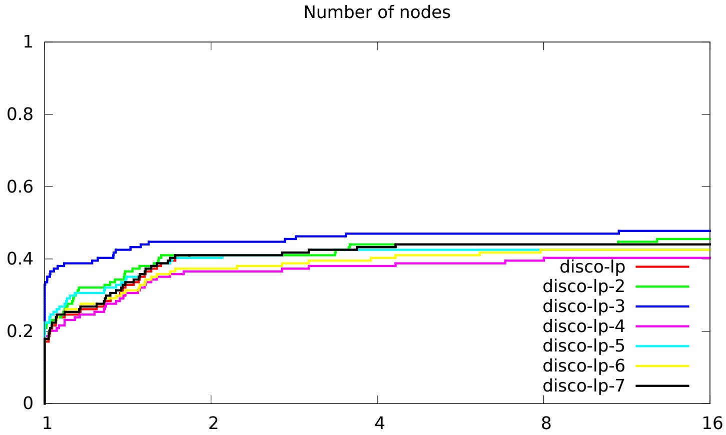


Figure: disco-lp, CPU Time without MILP Cuts

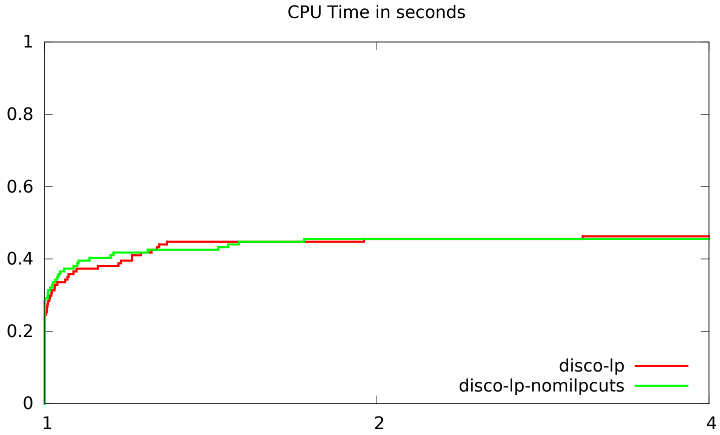


Figure: disco-lp, Number of Nodes without MILP Cuts

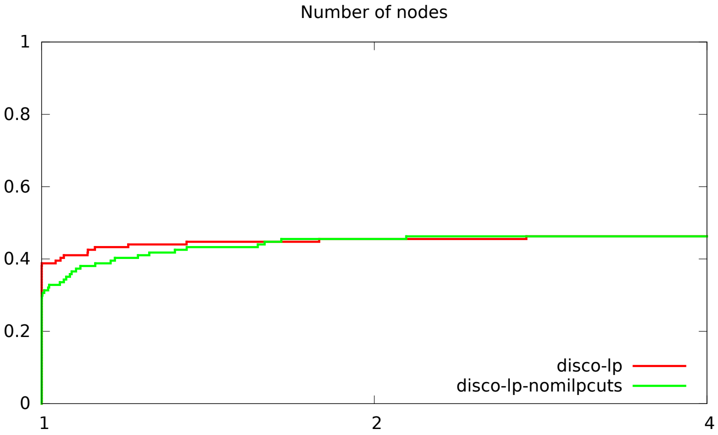


Figure: disco-socp, CPU Time with Disjunctive Cuts

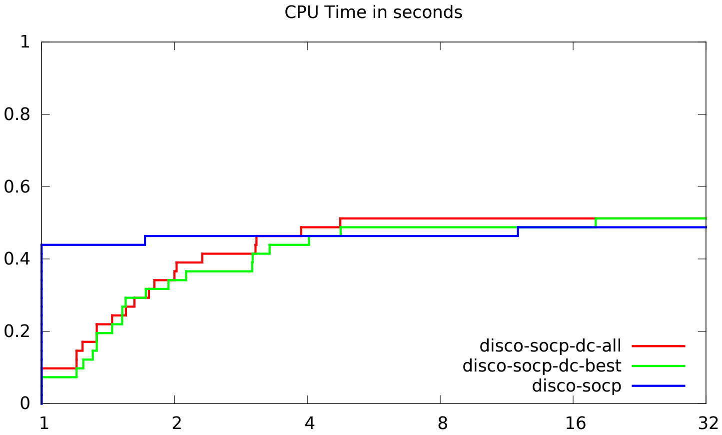


Figure: disco-socp, Number of Nodes with Disjunctive Cuts

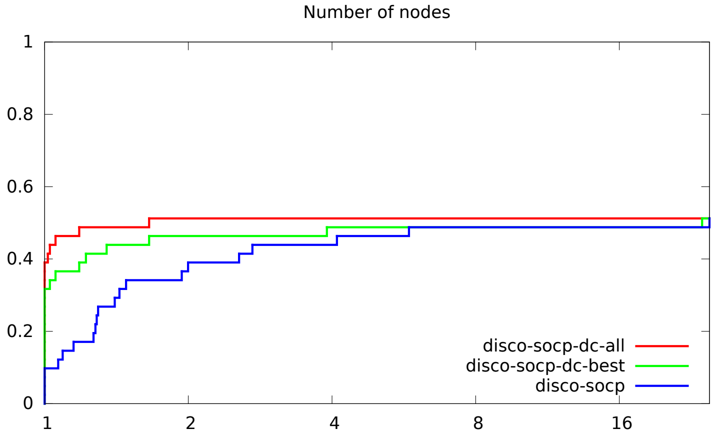


Figure: disco-lp, CPU Time with Disjunctive Cuts

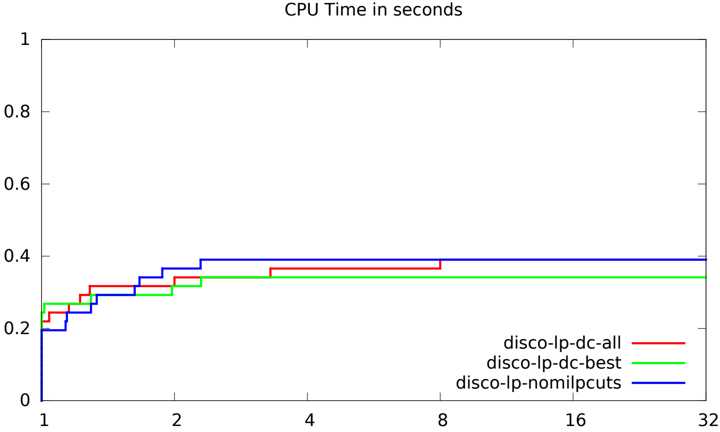


Figure: disco-lp, Number of Nodes with Disjunctive Cuts

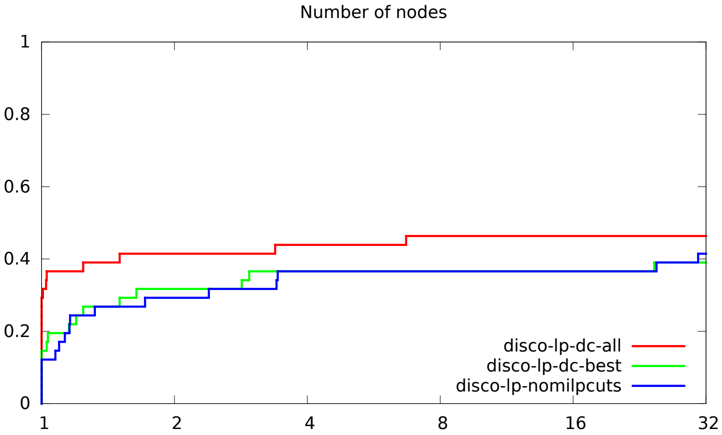


Figure: disco-socp, CPU Time, Parallel Runs

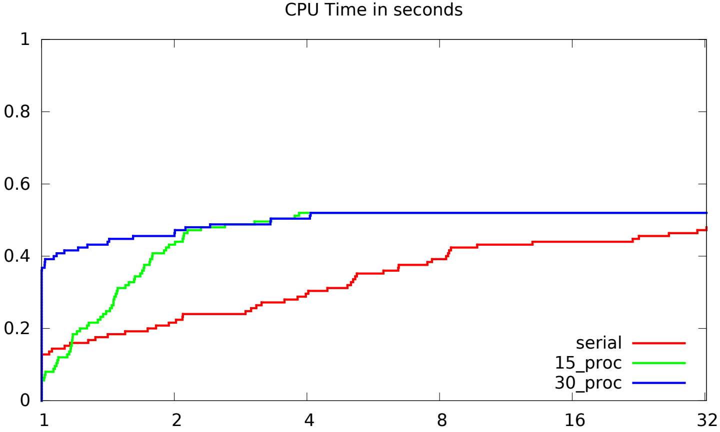


Figure: disco-socp, Number of Nodes, Parallel Runs

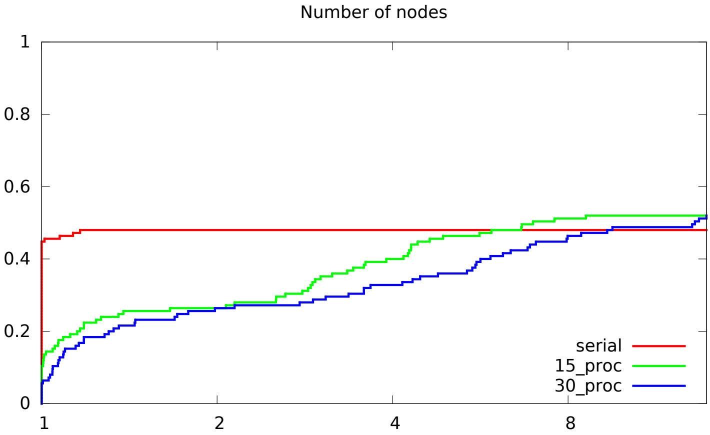


Figure: disco-lp, CPU Time, Parallel Runs

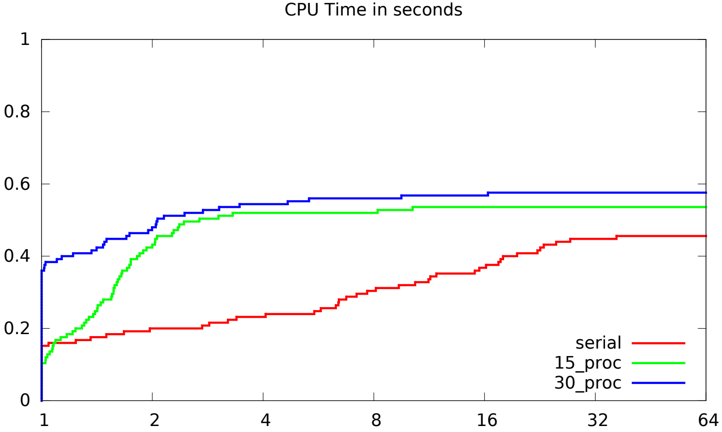


Figure: disco-lp, Number of Nodes Processed, Parallel Runs

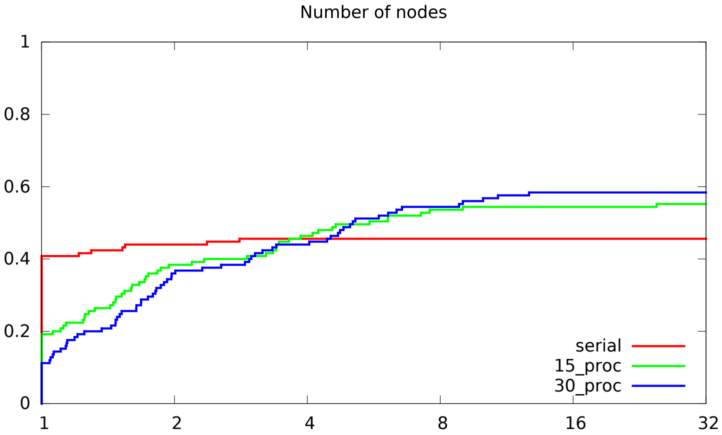


Figure: disco-lp versus disco-socp, CPU Time

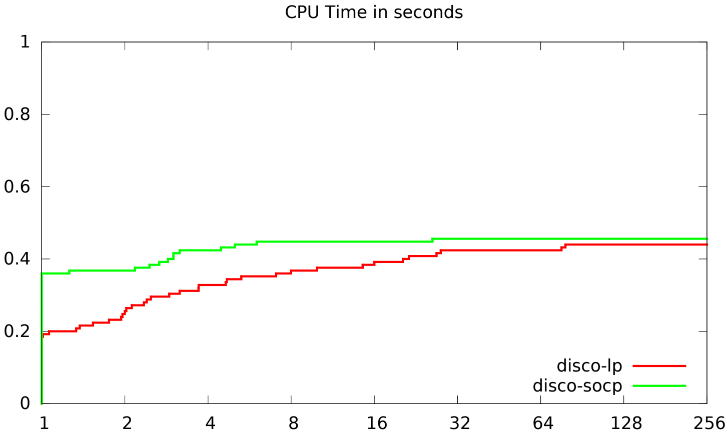


Figure: disco-lp versus disco-socp, Number of Nodes

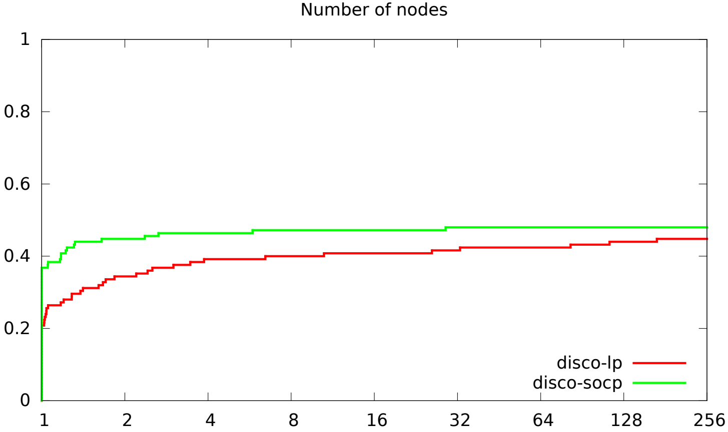


Figure: disco-lp versus disco-socp, Problems with Low Dimensional Cones

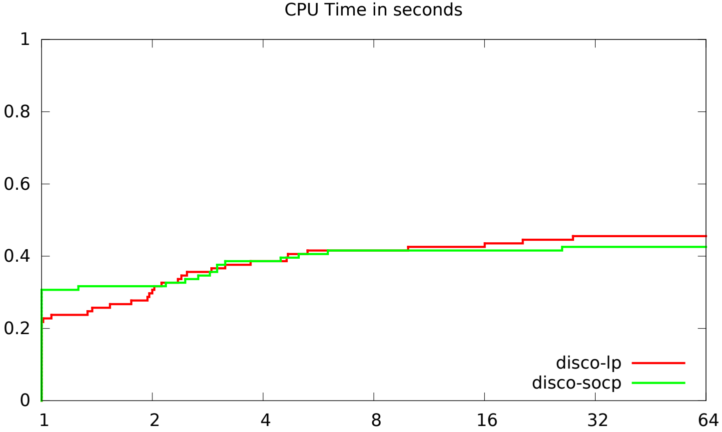


Figure: disco-lp versus disco-socp, CPU Time, Parallel Runs

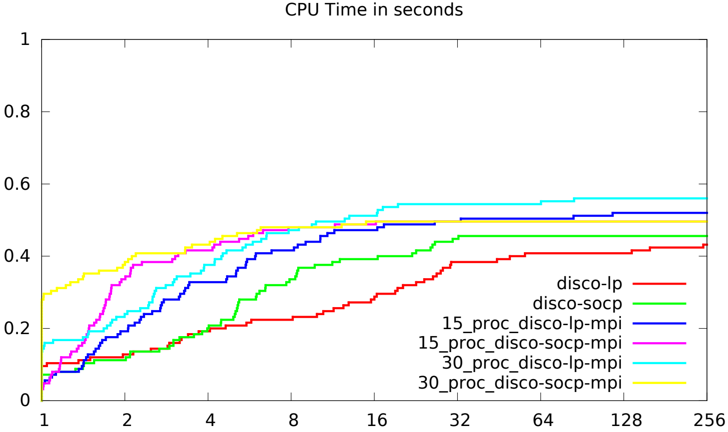
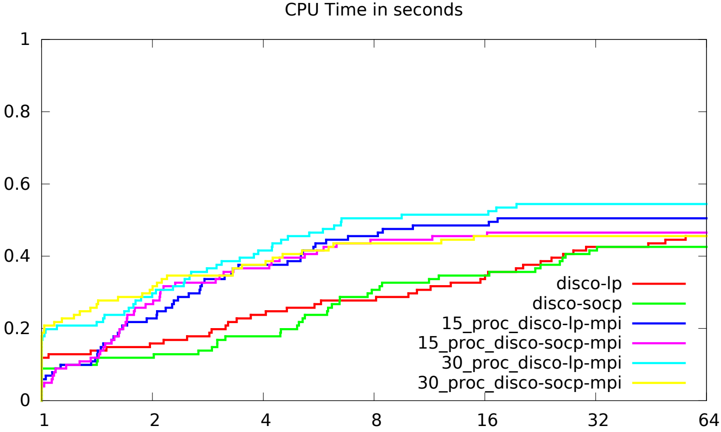


Figure: disco-lp versus disco-socp, Problems with Low Dimensional Cones



Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Conclusion

- Use SOCP–based subproblems for instances with large cones.
- LP–based subproblems might perform better for instances with low dimensional cones.
- Use LP–based subproblems for instances that are difficult and have large branch and bound trees.
- Strong branching might help with LP-based subproblems on hard instances.
- Right cut parameters are crucial in case of LP–based subproblems.
- Disjunctive cuts might help depending on the instance. They might perform better with LP–based subproblems.
- MILP cuts does not help much.

Clone, Try, Contribute

https://github.com/aykutbulut
https://github.com/coin-or

References

Pietro Belotti, Julio C. Góez, Imre Pólik, Ted K. Ralphs, and Tamás Terlaky.
On families of quadratic surfaces having fixed intersections with two hyperplanes.
Discrete Applied Mathematics, 161(16–17):2778–2793, 2013.
ISSN 0166-218X.

Pietro Belotti, Julio C Góez, Imre Pólik, Ted K Ralphs, and Tamás Terlaky.
A conic representation of the convex hull of disjunctive sets and conic cuts for integer second order cone optimization.
In *Numerical Analysis and Optimization*, pages 1–35. Springer, 2015.

Henrik A. Friberg.
Cblib 2014: A benchmark library for conic mixed-integer and continuous optimization.
Optimization Online, 2014.
URL `http://www.optimization-online.org/DB_FILE/2014/03/4301.pdf`.

Julio C. Góez.
Mixed Integer Second Order Cone Optimization Disjunctive Conic Cuts: Theory and experiments.
PhD thesis, Lehigh University, Bethlehem, PA, September 2013.