Parallelizing Partial MUS Enumeration

Wenting Zhao and Mark Liffiton

Department of Computer Science
Illinois Wesleyan University

{wzhao,mliffito}@iwu.edu

http://www.iwu.edu/~mliffito/marco/

ICTAI — November 7, 2016
San Jose, CA

Parallelizing Partial MUS Enumeration [is Easy]

Wenting Zhao and Mark Liffiton

Department of Computer Science
Illinois Wesleyan University

{wzhao,mliffito}@iwu.edu

http://www.iwu.edu/~mliffito/marco/

ICTAI — November 7, 2016
San Jose, CA

Overview

Problem

Analyzing infeasible constraint sets

"Constraint"
= SAT, SMT, CP, LP, IP, MIP, ...
(Implemented/tested w/ SAT & SMT.)

"Analyzing"
Enumerating MUSes/IISes
("Explanations" of infeasibility.)

Overview

Problem

Analyzing infeasible constraint sets

"Constraint"
= SAT, SMT, CP, LP, IP, MIP, ...
(Implemented/tested w/ SAT & SMT.)

"Analyzing"
Enumerating MUSes/IISes
("Explanations" of infeasibility.)

Contributions

1. MARCOs: Parallel algorithm for partial MUS enumeration.
2. Study of performance impact of variants and implementation choices.

Outline

Definitions

"Characteristic Subsets" of an infeasible constraint set C

MUS Minimal Unsatisfiable Subset
aka Irreducible Inconsistent Subsystem (IIS).
$M \subseteq C$ s.t. M is UNSAT and $\forall c \in M : M \setminus \{c\}$ is SAT

MSS Maximal Satisfiable Subset
a generalization of MaxSAT / MaxFS.
$M \subseteq C$ s.t. M is SAT and $\forall c \in C \setminus M : M \cup \{c\}$ is UNSAT

Definitions / Example

"Characteristic Subsets"

MUS Minimal Unsatisfiable Subset

MSS Maximal Satisfiable Subset

Example (Constraint set C, Boolean SAT)

$$C = \{ \underset{1}{(a)} , \underset{2}{(\neg a \vee b)} , \underset{3}{(\neg b)} , \underset{4}{(\neg a)} \}$$

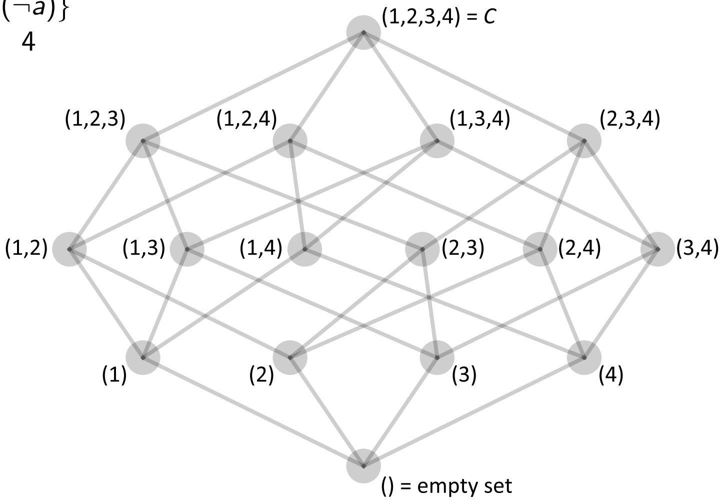
MUSes	MSSes
$\{1,2,3\}$	$\{2,3,4\}$
$\{1,4\}$	$\{1,3\}$
	$\{1,2\}$

Example, Powerset Visualization

Hasse diagram of powerset for:

$$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$$

1 2 3 4

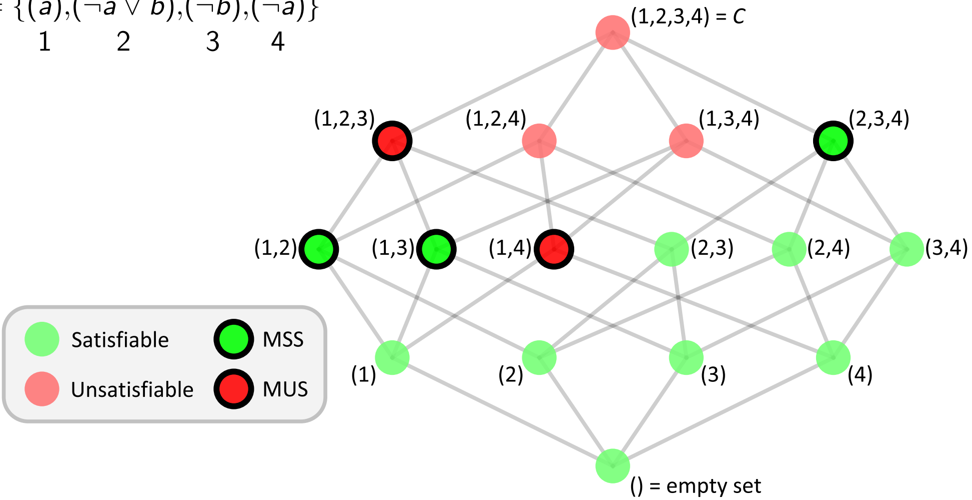


Example, Powerset Visualization

Hasse diagram of powerset for:

$$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$$

1 2 3 4



Earlier Work: *Sequential* MUS Enumeration

MARCO [*CPAIOR* 2013, *Constraints* 2016]

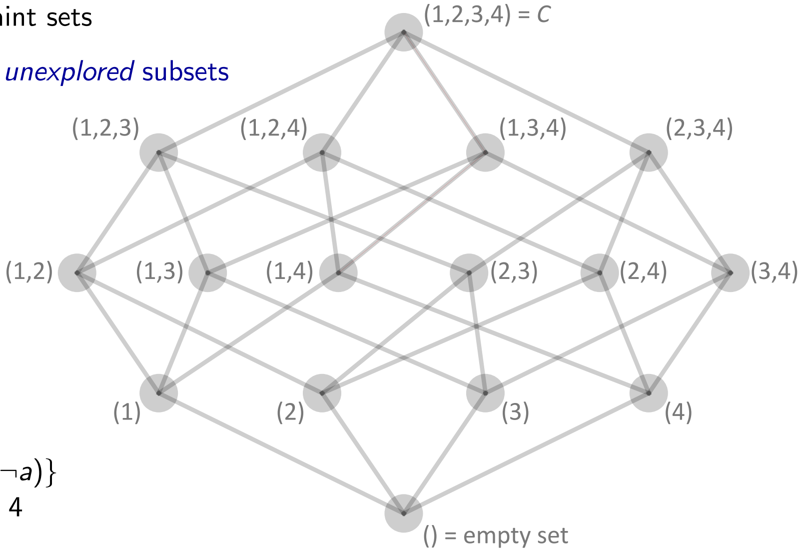
Mapping **R**egions of **Co**nstraint sets

Earlier Work: *Sequential* MUS Enumeration

MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = \top \leftarrow$ formula storing *unexplored* subsets



$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$

1 2 3 4

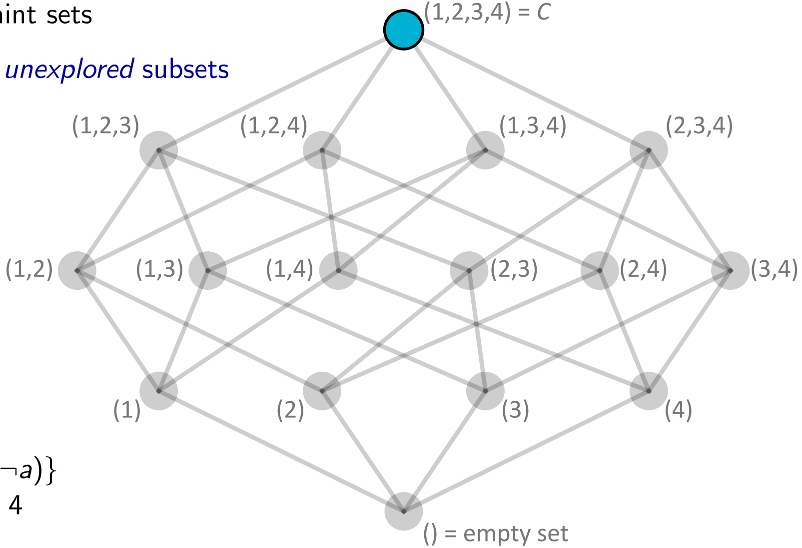
Earlier Work: *Sequential* MUS Enumeration

MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = \top \leftarrow$ formula storing *unexplored* subsets

Seed: $\{1, 2, 3, 4\}$



$$C = \{\underset{1}{(a)}, \underset{2}{(\neg a \vee b)}, \underset{3}{(\neg b)}, \underset{4}{(\neg a)}\}$$

Earlier Work: *Sequential* MUS Enumeration

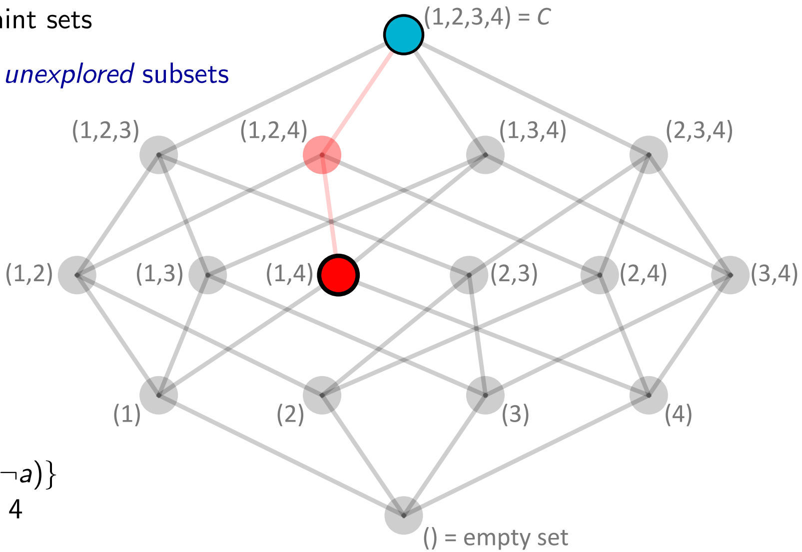
MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = \top \leftarrow$ formula storing *unexplored* subsets

Seed: $\{1, 2, 3, 4\}$

MUS: $\{1, 4\}$



$$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$$

1 2 3 4

Earlier Work: *Sequential* MUS Enumeration

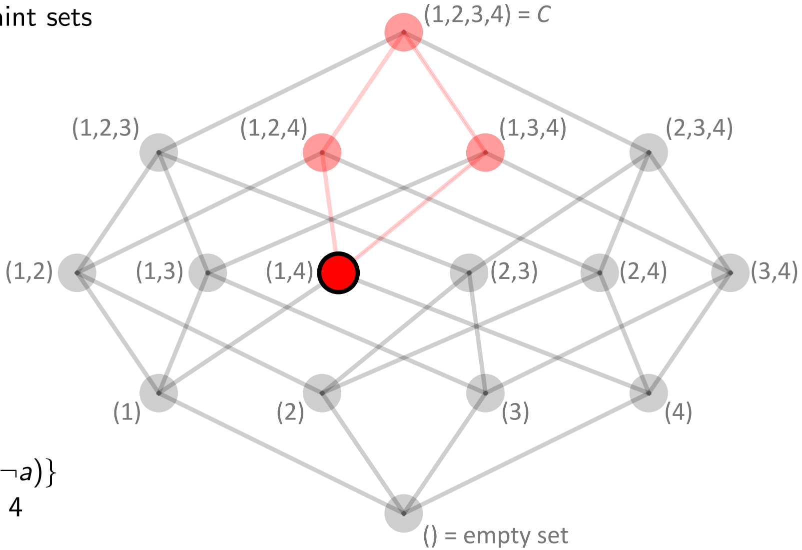
MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = (\neg X_1 \lor \neg X_4)$

Seed: $\{1, 2, 3, 4\}$

MUS: $\{1, 4\}$



$$C = \{(a),(\neg a \lor b),(\neg b),(\neg a)\}$$

1 2 3 4

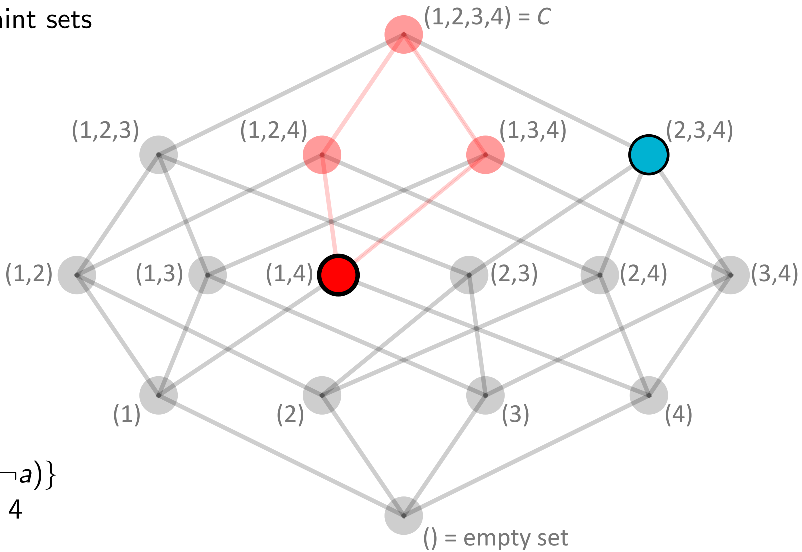
Earlier Work: *Sequential* MUS Enumeration

MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = (\neg X_1 \vee \neg X_4)$

Seed: $\{2, 3, 4\}$



$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$

1 2 3 4

Earlier Work: *Sequential* MUS Enumeration

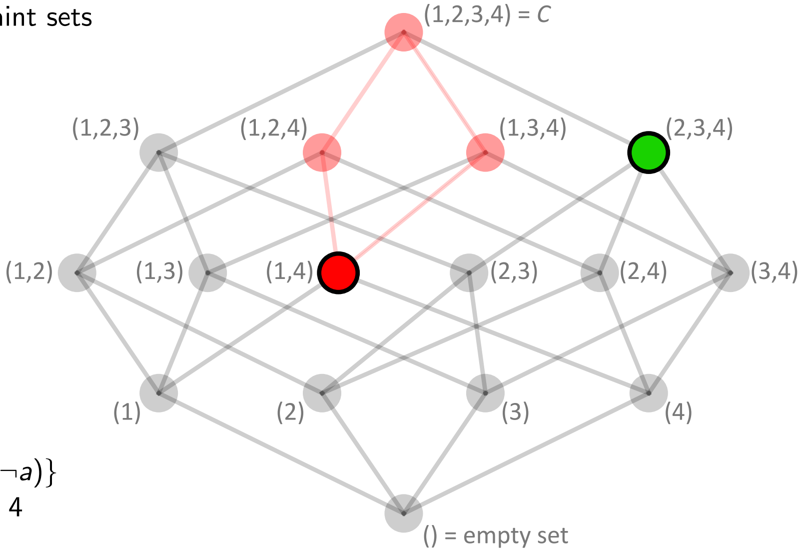
MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = (\neg X_1 \vee \neg X_4)$

Seed: $\{2, 3, 4\}$

MSS: $\{2, 3, 4\}$



$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$
1 2 3 4

Earlier Work: *Sequential* MUS Enumeration

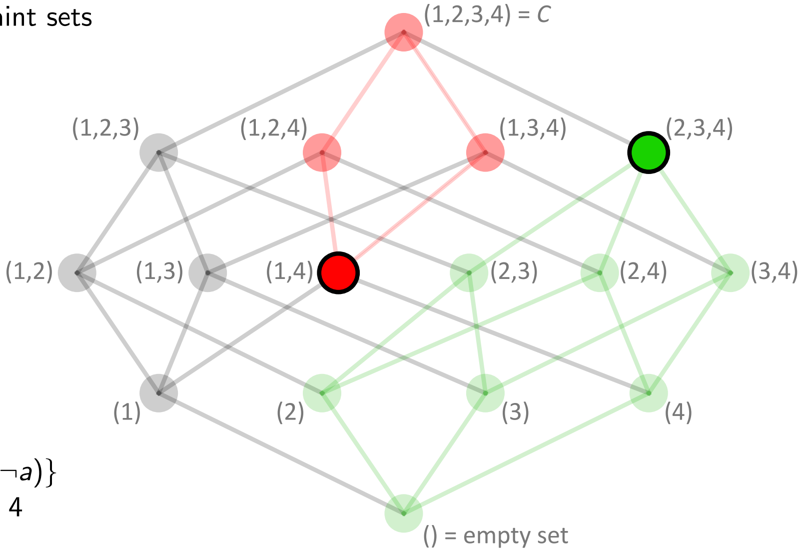
MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = (\neg X_1 \vee \neg X_4) \wedge (X_1)$

Seed: $\{2, 3, 4\}$

MSS: $\{2, 3, 4\}$



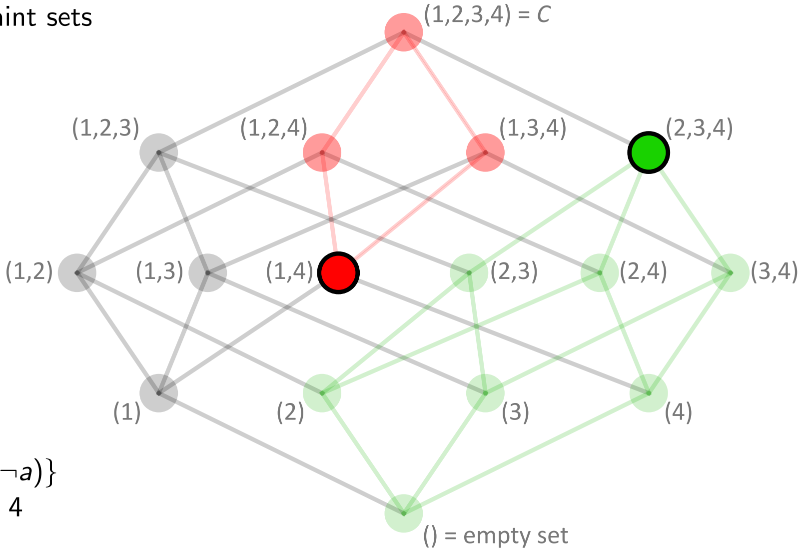
$$C = \{\underset{1}{(a)}, \underset{2}{(\neg a \vee b)}, \underset{3}{(\neg b)}, \underset{4}{(\neg a)}\}$$

Earlier Work: *Sequential* MUS Enumeration

MARCO [*CPAIOR* 2013, *Constraints* 2016]

Mapping **R**egions of **Co**nstraint sets

$Map = (\neg X_1 \vee \neg X_4) \wedge (X_1)$



$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$

1 2 3 4

Earlier Work: Parallel MUS *Extraction*

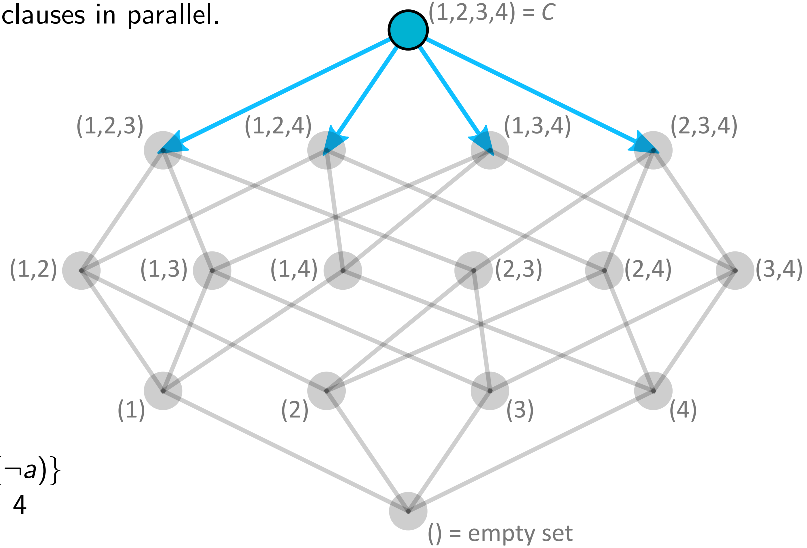
MUSer2-para [Belov, Manthey, & Marques-Silva, *SAT* 2013]

Checks necessity of multiple clauses in parallel.

Earlier Work: Parallel MUS *Extraction*

MUSer2-para [Belov, Manthey, & Marques-Silva, *SAT* 2013]

Checks necessity of multiple clauses in parallel.

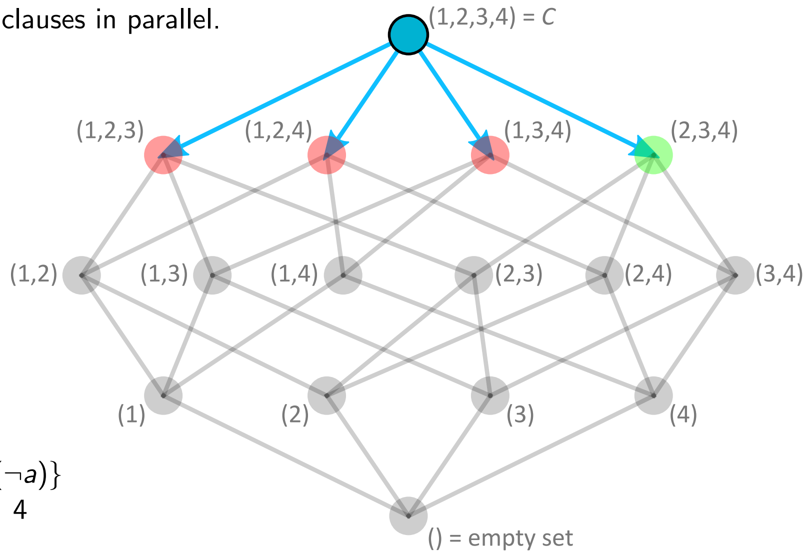


$$C = \{\underset{1}{(a)}, \underset{2}{(\neg a \vee b)}, \underset{3}{(\neg b)}, \underset{4}{(\neg a)}\}$$

Earlier Work: Parallel MUS *Extraction*

MUSer2-para [Belov, Manthey, & Marques-Silva, *SAT* 2013]

Checks necessity of multiple clauses in parallel.



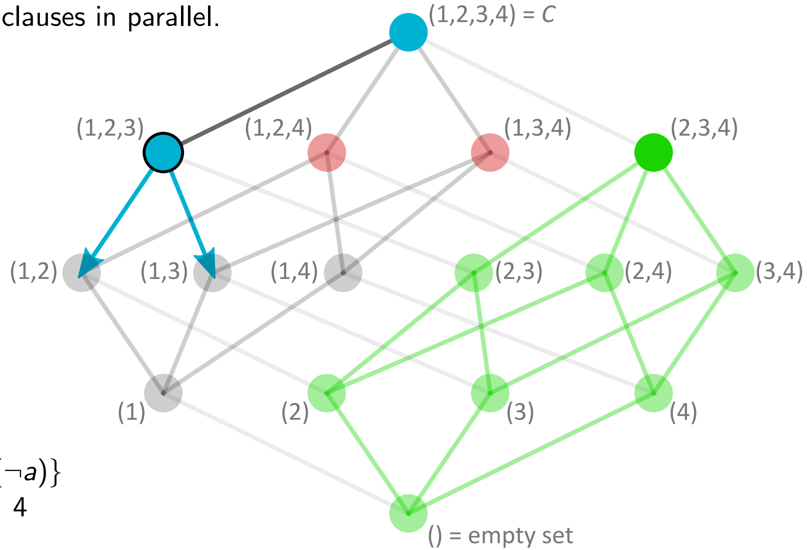
$$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$$

1 2 3 4

Earlier Work: Parallel MUS *Extraction*

MUSer2-para [Belov, Manthey, & Marques-Silva, *SAT* 2013]

Checks necessity of multiple clauses in parallel.

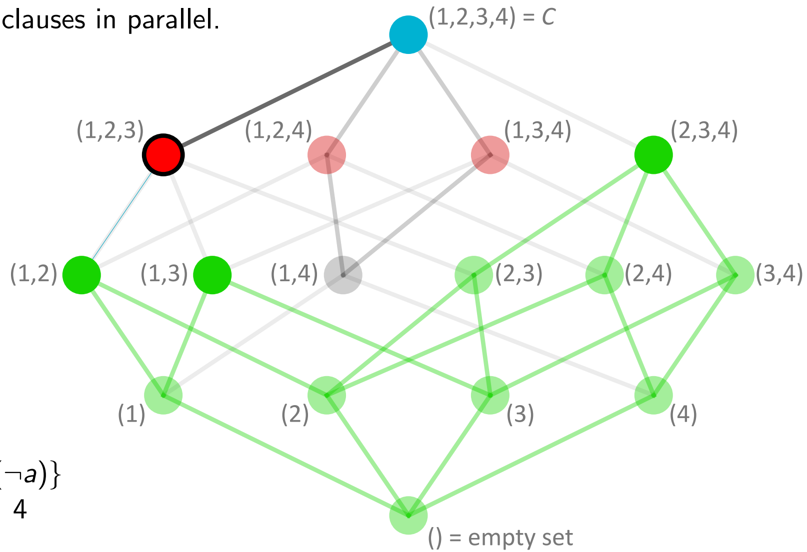


$$C = \{\underset{1}{(a)},\underset{2}{(\neg a \vee b)},\underset{3}{(\neg b)},\underset{4}{(\neg a)}\}$$

Earlier Work: Parallel MUS *Extraction*

MUSer2-para [Belov, Manthey, & Marques-Silva, *SAT* 2013]

Checks necessity of multiple clauses in parallel.



$$C = \{(a),(\neg a \vee b),(\neg b),(\neg a)\}$$

1 2 3 4

Goals of This Work

1. Parallel MUS enumeration
2. Constraint-agnostic
3. Flexible: easily able to adopt improvement in MUS enumeration
4. Scales well on multi-core machines
   - Ideal: *Perfect scaling*. k times speedup on k-core machine.

MARCOs

MARCOs: **Ma**pping **R**egions of **Co**nstraint sets **S**imultaneously

- Parallelization of MARCO
- Master-worker architecture
- Limits / avoids two enemies of scaling:
  1. Communication between threads
  2. Duplicate / redundant work

MARCOs

MARCOs: **Ma**pping **R**egions of **Co**nstraint sets **S**imultaneously

- Parallelization of MARCO
- Master-worker architecture
- Limits / avoids two enemies of scaling:
  1. Communication between threads
  2. Duplicate / redundant work

Basic Algorithm:

1. Run k parallel copies of MARCO
2. Filter duplicate results

MARCOs

MARCOs: **Ma**pping **R**egions of **Co**nstraint sets **S**imultaneously

- Parallelization of MARCO
- Master-worker architecture
- Limits / avoids two enemies of scaling:
  1. Communication between threads
  2. Duplicate / redundant work

Basic Algorithm:

1. Run k parallel copies of MARCO
2. Filter duplicate results

Optional:

- Share results between workers
- Randomize solver in each worker

MARCOs

MARCOs: **Ma**pping **R**egions of **Co**nstraint sets **S**imultaneously

- Parallelization of MARCO
- Master-worker architecture
- Limits / avoids two enemies of scaling:
  1. Communication between threads
  2. Duplicate / redundant work

Basic Algorithm:

1. Run k parallel copies of MARCO
2. Filter duplicate results

Optional:

- Share results between workers
- Randomize solver in each worker

Alternative: parallelize MARCO using MUSer2-para

Experimental Setup

- Implementation
  - MUS extraction: MUSer2[-para] [Belov & Marques-Silva, *JSAT* 2012]
  - SAT solver: MiniSAT v2.2
  - Platform: Python w/ `multiprocessing` library

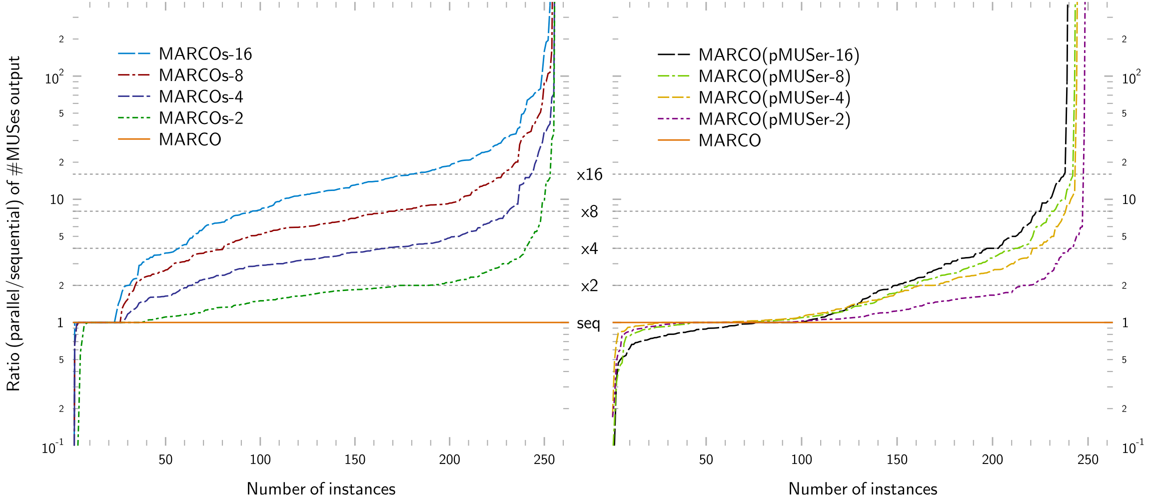
Experimental Setup

- Implementation
  - MUS extraction: MUSer2[-para] [Belov & Marques-Silva, *JSAT* 2012]
  - SAT solver: MiniSAT v2.2
  - Platform: Python w/ `multiprocessing` library
- Benchmarks
  - 309 Boolean CNF instances selected from several sources
    - limited representation per "family" to 5 instances
  - Filtered to 263 instances for which:
    - some algorithm found at least one MUS
    - sequential MARCO does *not* complete

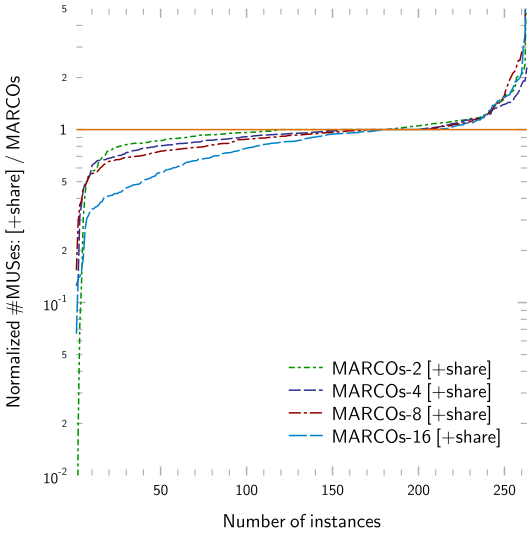
Experimental Setup

- Implementation
  - MUS extraction: MUSer2[-para] [Belov & Marques-Silva, *JSAT* 2012]
  - SAT solver: MiniSAT v2.2
  - Platform: Python w/ `multiprocessing` library
- Benchmarks
  - 309 Boolean CNF instances selected from several sources
    - limited representation per "family" to 5 instances
  - Filtered to 263 instances for which:
    - some algorithm found at least one MUS
    - sequential MARCO does *not* complete
- Experiments
  - CPU: Intel Xeon E5-2680 v2 [Amazon EC2 'c3.8xlarge' instances]
  - Cores/threads: $k = 1, 2, 4, 8$, or 16 per execution
  - RAM limit: $3500 \cdot k$ MB
  - Time limit: 10min / 600sec

Experimental Results: Scaling (with Randomization, no Results-Sharing)

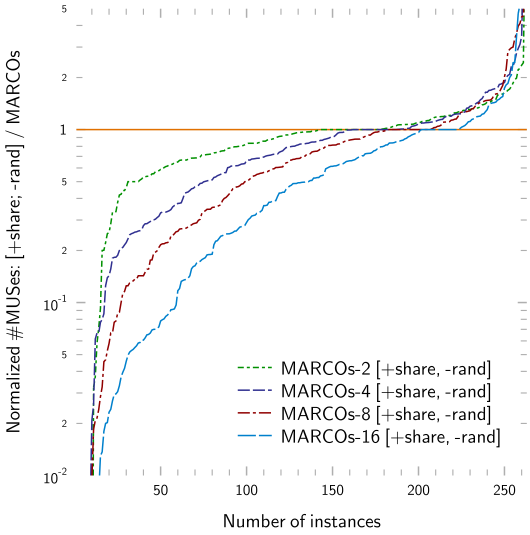


Experimental Results: Effect of Adding Results-Sharing



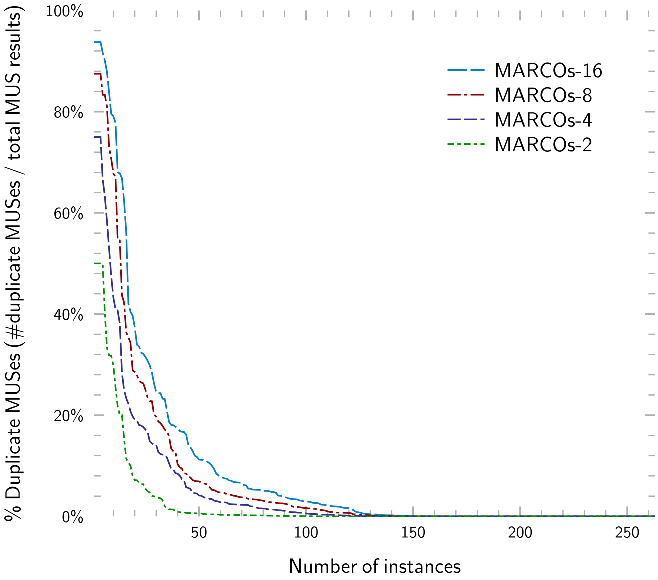
- Results-sharing has very little effect on performance
- Effect is overall negative

Experimental Results: Effect of Removing Randomization



- Randomization is critical
- Results-sharing is not sufficient to avoid duplicate work

Experimental Results: Duplicate MUSes



- Search space is massive
- Randomization alone provides results with little to no overlap (for *partial* MUS enumeration)

Conclusion

Contributions

1. MARCOs Algorithm:
   - Parallelization of MARCO
   - Achieves substantial fraction of perfect scaling
   - Easily integrates improvements to MARCO
2. Performance Study:
   - Results-sharing typically unimportant due to massive search space
   - Randomization provides sufficient duplicate protection

Conclusion

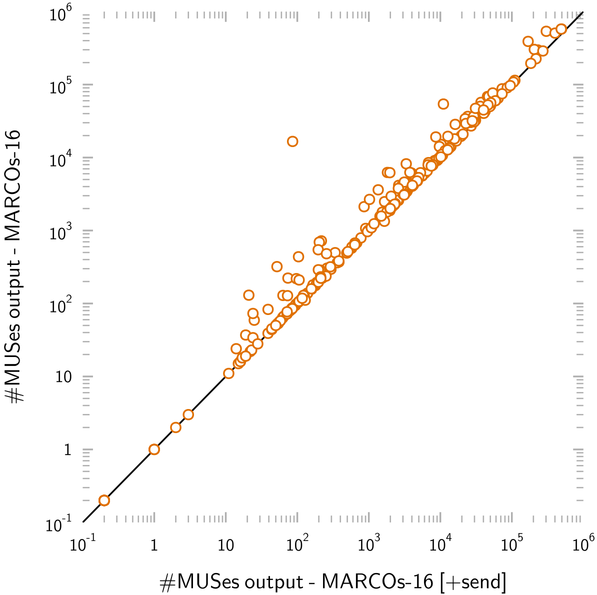
Contributions

1. MARCOs Algorithm:
   - Parallelization of MARCO
   - Achieves substantial fraction of perfect scaling
   - Easily integrates improvements to MARCO
2. Performance Study:
   - Results-sharing typically unimportant due to massive search space
   - Randomization provides sufficient duplicate protection

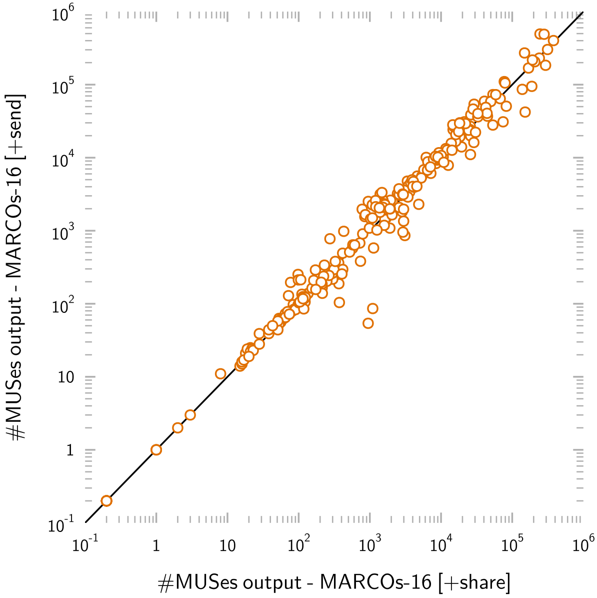
Thank you.

Source code: `http://www.iwu.edu/~mliffito/marco/`

Performance cost of communication



Performance benefit of using shared results



Combined performance chart