Algorithms For Error Localization On Numeric Constraints

(extended version)

Michel Rueher

University de Nice Sophia Antipolis – I3S / CNRS

NII – Tokyo

06/11/2013

Context

- **Input**
  - Some imperative program with *numeric statements* (over integers or floating-point numbers)
  - *An assertion* to be checked
  - A *counter-example* that violates the assertion

- **Output** : *information on locations of potentially faulty statements*

Goal

- Provide helpful information for error localization on numeric constraint systems
- Two categories of information
  - *How much of an unsatisfiable constraint set can be satisfied ?*

    → MaxSAT, Max CSP, MaxFS
  - *Where in the constraint set the "problem" lies ?*

    → Irreducible / Irredundant Infeasible / Inconsistent Subsystems (IIS),Minimal Unsatisfiable Core (MUC)

Definitions

- **MUS** Minimal Unsatisfiable Subset
  aka Irreducible Inconsistent Subsystem (IIS)
  $M \subseteq C$ is a MUS $\Leftrightarrow$ M is UNSAT and $\forall c \in M : M \setminus \{c\}$ is SAT

- **MSS** Maximal Satisfiable Subset
  a generalization of MaxSAT / MaxFS considering maximality instead of maximum cardinality
  $M \subseteq C$ is a MSS $\Leftrightarrow$ M is SAT and $\forall c \in C \setminus M : M \cup \{c\}$ is UNSAT

- **MCS** Minimal Correction Set
  the complement of some MSS: removal yields a satisfiable MSS (it "corrects" the infeasibility)
  $M \subseteq C$ is a MCS $\Leftrightarrow$ $C \setminus M$ is SAT and $\forall c \in M : (C \setminus M) \cup \{c\}$ is UNSAT

Generic Algorithms

- ***Irreducible infeasible subset (IIS)*** or ***MUS***, (Chinneck - 2008), Deletion Filter (Chinneck et Dravnieks – 1991), Additive Method (Tamiz et al -1996), Additive Deletion Method (Guieu et Chinneck -1999), *The Elastic filter*

- ***Irreducible Conflict Sets (MUS)*** *QUICKXPLAIN* : Divide-and-Conquer for Explanations (Junker-2004)

- ***All MCS / MUS***
  - *Finding Multiple all MUSes:* (Liffiton & Sakallah-2008);
  - Enhancements :
    - MCS : adaptation of QUICKXPLAIN : FastDiag & Enhanced FastDiag (Felfernig et al -2012, Silva et al- 2013)
    - MUSEs: *Marco Polo* - *Finding Multiple MUSes Quickly* (Liffiton & Malik -2013)

Irreducible infeasible subset (IIS)

« An irreducible infeasible subset (IIS) of the constraints, is a (small) subset of constraints that is itself infeasible, but becomes feasible if one or more constraints is removed » (Chinneck - 2008)

- Used in global optimization /operational research
- Correspond to **MUS (Minimal Unsatisfiable Subset)** used in SAT

- Many algorithms:
  - **Deletion filter, Additive method, Additive/Deletion method**
  - **The Elastic filter**
  - Deletion filtering for inequality-constrained QCQP
  - Additive/deletion method for MIPs
  - Deletion filter for NLPs
  - Heuristic for MIN IIS COVER
  - All IIS of LP (based on the construction of a polytope using the simplex method)
  - …

The Deletion Filter Chinneck and Dravnieks (1991)

INPUT: an infeasible set of constraints

FOR each constraint in the set:

1. Temporarily drop the constraint from the set
2. Test the feasibility of the reduced set:

    IF feasible THEN return dropped constraint to the set

    ELSE (infeasible) drop the constraint permanently

OUTPUT: constraints constituting ***a single IIS***

Remarks

- *The only constraints retained in the set are those whose removal renders the set feasible*
- *Efficiency improvement: dynamic reordering*

The Additive Method Tamiz et al. (1995, 1996)

Opposite of the the deletion filter: starting with an empty set of constraints, constraints are added until infeasibility is triggered

C: ordered set of constraints in the infeasible model

T: the current test set of constraints **I**: the set of IIS members identified so far

INPUT: an infeasible set of constraints C

0. $T \leftarrow \varnothing$, $I \leftarrow \varnothing$
1. $T \leftarrow I$

   FOR each constraint c_i in C :

   $T \leftarrow T \cup c_i$

   IF T infeasible **THEN** $I \leftarrow I \cup c_i$

   Go to Step 2.
2. **IF** I feasible **THEN** go to Step 1

   Exit

OUTPUT: I, a single IIS

The Additive Method - Exemple

INPUT: an infeasible set of constraints C
0. T ← ∅, I ← ∅
1. T ← I
   FOR each constraint c_i in C :
      T ← T ∪ c_i
      I**F** T infeasible **THEN** I ← I ∪ c_i
      Go to Step 2.
2. **IF** I feasible **THEN** go to Step 1
   Exit

OUTPUT: I, a single IIS

Consider an IIS {B,F,D} embedded in {A,B,C,D,E,F}.
The members of the IIS are **shown** in boldface:

1. {A}, {A,**B**}, {A,**B**,C}, {A,**B**,C,**D**}, {A,**B**,C,**D**,E} all feasible.
2. {A,**B**,C,**D**,E,**F**} infeasible: $\mathcal{I}$ = {**F**} is feasible.
3. {**F**,A}, {**F**,A,**B**}, {**F**,A,B,C} all feasible.
4. {**F**,A,**B**,C,**D**} infeasible: $\mathcal{I}$= {**F**,**D**} is feasible.
5. {**F**,**D**,A} feasible.
6. {**F**,**D**,A,**B**} infeasible: $\mathcal{I}$ = {**F**,**D**,**B**} is infeasible. Stop.
7. Output: the IIS {F,B,D}

The Elastic Filter – Linear constraints (1)

- **LP solvers**
  - Adding *nonnegative artificial variables* (s_i) to all inequality constraints
  - LP Phase 1 minimizes $W = \sum s_i$, via standard LP: If $W^* \neq 0$, no solution exists

- **Elastic Filter**: *nonnegative artificial variables* (s_i) are added to all equality and ≥ constraints

  → so a solution always exits in the space of the original plus artificial variables, but not in the space of just the original variables

  If $W^* \neq 0$ then at least one of s_i cannot be forced to zero: the corresponding constraint remains violated in the original variable space

- Rules for adding elastic variables are as follows:

non-elastic constraint	elastic version
$\Sigma_j a_{ij} x_j \geq b_i$	$\Sigma_j a_{ij} x_j + s_i \geq b_i$
$\Sigma_j a_{ij} x_j \leq b_i$	$\Sigma_j a_{ij} x_j - s_i \leq b_i$
$\Sigma_j a_{ij} x_j = b_i$	$\Sigma_j a_{ij} x_j + s'_i - s'_i = b_i$

The Elastic Filter – Linear constraints (2)

Intuition: *remove elastic variable until the system becomes inconsistent*

INPUT: an infeasible set of constraints

1. Make all constraints elastic by incorporating nonnegative elastic variables s_i
2. Solve the model using the elastic objective function
3. IF feasible THEN
    - Enforce the constraints in which any $s_i > 0$ by permanently removing their elastic variable(s)
    - Go to step 2

ELSE (infeasible)

    Exit

OUTPUT: the set of de-elasticized enforced constraints contains at least one IIS

The Elastic Filter – Linear constraints (3)

INPUT: an infeasible set of constraints
1. Make all constraints elastic by incorporating nonnegative elastic variables s_i
2. Solve the model using the elastic objective function
3. IF feasible THEN
   - Enforce the constraints in which any $s_i > 0$ by permanently removing their elastic variable(s)
   - Go to step 2

ELSE (infeasible) Exit
OUTPUT: the set of de-elasticized enforced constraints contains at least one IIS

Consider the example in which the IIS {B,D,F} appears in the set of constraints {A,B,C,D,E,F,G}

Assume that the solver stretches just one constraint in the IIS at each iteration. The members of the IIS are shown in boldface and elasticized constraints are underscored:

1. {A,**B**,C,**D**,E,**F**,G} is feasible, B stretches, so is de-elasticized
2. {A,**B**,C,**D**,E,**F**,G} is feasible, F stretches, so is de-elasticized
3. {A,**B**,C,**D**,E,**F**,G} is feasible, D stretches, so is de-elasticized
4. {A,**B**,C,**D**,E,**F**,G} is infeasible
5. Output: the set {**B**,**F**,**D**}

Number of iteration : at most the size of the smallest IIS in the input set

Speed-ups

- Combining the Additive Method and the Deletion Filter
- Treating Constraints in Groups
- The depth first binary search filter

Combining the Additive Method and the Deletion Filter (Guieu and Chinneck 1999)

*Run the additive method until unfeasibility is first detected and then change to the deletion filter to obtain a **Minimal** Unsatisfiable Subset*

C: ordered set of constraints in the infeasible model, T: test set of constraints

INPUT: an infeasible set of constraints C

0. Set T = ∅
1. FOR each constraint c_i in C:
   Set T = T ∪ c_i
   IF T infeasible THEN go to Step 2
2. FOR each constraint t_i to $t_{|T|-1}$ in T :
   Temporarily drop the constraint t_i
   Test the feasibility of the reduced set:
   IF feasible THEN return dropped constraint to T
   ELSE (infeasible) T←T \ t_i

OUTPUT: T is an IIS

QUICKXPLAIN: overview (1)

- Computes ***Preferred conflict set / Preferred relaxation***

- ***Generic method*** for CP, SAT, or DL (description logic) solver : Assumes only monotonic satisfiability property: *if* $\mathcal{S}$ *is a solution of a set of constraints* $\mathcal{C}$ *then it is also a solution of all subsets* $\mathcal{C}_i$ *of* $\mathcal{C}$

- ***Divide-and-Conquer*** strategy

QUICKXPLAIN: overview (2)

- **Input** : P = ($\mathcal{B}$, C, $\prec$)
  - B : Background /mandatory constraints
  - $\prec$: total order on set of constraints C

    $c_1 \prec c_2$ iff constraint c_1 is preferred to c_2

- **Output** :
  - ***Preferred conflict set*** *of* P : MUS (Minimal Unsatisfiable Subset)
  - ***Preferred relaxation*** *of* P : MSS (Maximal Satisfiable Subset)

QUICKXPLAIN : Preferred relaxations & Conflicts

- R $\subseteq$ C is a ***relaxation*** of P = ($\mathcal{B}$, C) iff $\mathcal{B}$ $\cup$ R holds
  - *A* relaxation R *of* P= ($\mathcal{B}$, C, $\prec$) *is a* ***preferred relaxation*** *of* P *iff there is no other relaxation* R* *of* P *s.t.* R* $\prec_{lex}$ R
  - If $\prec$ is a strict total order and B is consistent, then P has a unique preferred relaxation
  - *No preferences : maximal relaxations* ≡ *preferred relaxations*
- A subset C of C is a ***conflict*** of a problem P =($\mathcal{B}$, C) iff $\mathcal{B}$ U C has no solution
  - *A* conflict C *of* P= ($\mathcal{B}$, C, $\prec$) *is a* ***preferred conflict*** *of* P *iff there is no other conflict* C* *of* P *s.t.* C* $\prec_{antilex}$ C
  - A ***preferred conflict*** C is minimal (irreducible) meaning that each proper subset of C has a solution.

QUICKXPLAIN : constructive definitions

Preferred relaxation of P

Elements of C are enumerated in increasing order $c_1, \ldots, c_n$

$R_0 := \varnothing$ and

$$R_i = \begin{cases} R_{i-1} \cup \{c_i\} & \text{if } B \cup R_{i-1} \cup \{c_i\} \text{ has } \textbf{\textit{a}} \text{ solution} \\ R_{i-1} & \text{otherwise} \end{cases}$$

Preferred conflict

constructed in the reverse order: $C_n := C$ and

$$C_i = \begin{cases} C_{i+1} - \{c_i\} & \text{if } B \cup C_{i+1} - \{c_i\} \text{ has } \textbf{\textit{no}} \text{ solution} \\ C_{i+1} & \text{otherwise} \end{cases}$$

QUICKXPLAIN: Divide-and-Conquer

Intuition: Split C until we obtain sub-problems of the form P ' = (B, {α}, $\prec$), where all but one constraint are in the background, then we know that B ∪ {α} is inconsistent

Algorithm QUICKXPLAIN($\mathcal{B}$, C, $\prec$)

1. **if** *isConsistent*($\mathcal{B}$ ∪ C) **return** 'no conflict';
2. **else if** C = ∅ **then return** ∅;
3. **else return** QUICKXPLAIN'($\mathcal{B}$, $\mathcal{B}$, C, $\prec$);

Algorithm QUICKXPLAIN'($\mathcal{B}$, Δ, C, $\prec$)

4. **if** Δ ≠ ∅ and not *isConsistent*($\mathcal{B}$) **then return** ∅;
5. **If** C = {α} **then return** {α};
6. let $\alpha_1, \ldots, \alpha_n$ be an enumeration of C that respects $\prec$;
7. let k be *split(n)* where 1 ≤ k < n;
8. C_1 := {α_1,...,α_k} and C_2 := {α_{k+1},...,α_n};
9. Δ_2 := QUICKXPLAIN'($\mathcal{B}$ ∪ C_1, C_1, C_2, $\prec$);
10. Δ_1 := QUICKXPLAIN'($\mathcal{B}$ ∪ Δ_2, Δ_2, C_1, $\prec$);
11. **return** Δ_1 ∪ Δ_2;

QUICKXPLAIN: example

Consider again the set of constraints {A,B,C,D,E,F} where {B,F,D} is a MUS. No mandatory constraints and $\prec$ is lex

Q'(ø,ø, {A,B,C,D,E,F}, $\prec$}
C1= {A,B,C} C2 = {D,E,F}
    Q'({A,B,C},{A,B,C},{,D,E,F}, $\prec$}
    C1 = {D,E} C2= {F}
        Q'({A,B,C,D,E},{D,E},{F}, $\prec$}
        C1 = {F} C2 = {ø}
            Q'({A,B,C,D,E,F},{F},{ø}, $\prec$}
            {A,B,C,D,E,F} UNSAT, Δ2={ø}
            Q' {{A,B,C,D,E}, {ø}{F}} $\prec$}
            Δ1= {F}
        Δ= {F}
        Q'({A,B,C,F}, {F}, {D,E},$\prec$}
            C1={D}, C2={E}
            Q' ({A,B,C,F,D}, {D}, {E},$\prec$}
            {A,B,C,F,D} UNSAT Δ2={ø}
            Q' ({A,B,C,F}, {ø}, {D},$\prec$}, Δ1= {D}
            Δ={D}
        Δ= {D,F}
    Q'({D,F},{D,F},{A,B,C}, $\prec$}
        C1={A,B},C2={C}
        Q'({D,F,A,B},{A,B},{C}, $\prec$}
        {D,F,A,B}, UNSAT Δ2=ø
        Q'({D,F},ø, {A,B},$\prec$}
            C1={A}, C2={B}
            Q'(D,F,A}, {A}, {B} , $\prec$}
            Δ1={B}
            Q'(D,F,A,B}, {B} , A}, $\prec$}
            Δ2={ø}
            Δ ={B}
    Δ= {D,F,B}

Algorithm QUICKXPLAIN'($\mathcal{B}$, Δ, C, $\prec$)

4. **if** Δ ≠ ∅ and not *isConsistent($\mathcal{B}$)* **then return** ∅;
5. **If** C = {α} **then return** {α};
6. let $\alpha_1, \ldots, \alpha_n$ be an enumeration of C that respects $\prec$;
7. let k be *split(n)* where 1 ≤ k < n;
8. C_1 := {α_1,...,α_k} and C_2 := {α_{k+1},...,α_n};
9. Δ_2 := QUICKXPLAIN'($\mathcal{B} \cup C_1$, C_1, C_2, $\prec$);
10. Δ_1 := QUICKXPLAIN'($\mathcal{B} \cup \Delta_2$, Δ_2, C_1, $\prec$);
11. **return** $\Delta_1 \cup \Delta_2$;

QUICKXPLAIN : complexity

Number of Consistency Checks (*k is size of* conflicts, *n* size of C)

Method	Split	Best Case	Worst Case
1.	*split(n)=n/2*	*log(n/2) +2k*	*2k·log(n/k) +2k*
2.	*split(n)* = n − 1	*2k*	*2n*
3.	*split(n)* = 1	*k*	*n + k*

Accelerates if k << n

QUICKXPLAIN : Multiple Preferred Explanations

Set up a choice point when c_i is removed

from C_{i+1}:

- The left branch removes c_i from C_{i+1} and determines preferred conflicts not containing c_i
- The right branch removes other constraints from C_{i+1} such the removal of c_i leads to a solution

MUS (Minimal Unsatisfiable Subset) **- MCS**(Minimal Correction Set) **duality**

The set of MCSes ⇔ all the irreducible hitting sets of the MUSes

The set of MUSes ⇔ The set of all irreducible hitting sets of the MCSes

H is a hitting set of Ω if $H \subseteq D$ and $\forall S \in \Omega, H \cap S \neq \varnothing$

H is a minimal hitting sets if no element can be removed without losing the property of being a hitting set

Given an unsatisfiable constraint system C:

1. A subset M of C is an MCS of C iff M is an irreducible hitting set of MUSes(C)
2. A subset U of C is an MUS of C iff U is an irreducible hitting set of MCSes(C)

Intuition: A MCS *must at least remove one constraint* from each MUS

A MUS can be made satisfiable by removing any one constraint from it

→ every MCS contains at least one constraint from each MUS.

Computing all MCS / MUSes: CAMUS

Liffiton & Sakallah-2007

Easier to find ***satisfiable subsets*** of constraints than unsatisfiable ones

- **CAMUS** approach

1. Computing all MCS in order of increasing size
2. Computing all MUSES from MCS in order of decreasing size

Computing all MCS : *CAMUS*

Liffiton & Sakallah-2007

All_MCSes(ϕ)

1. ϕ′ ← AddYVars(ϕ) *% Adds y_i selector variables*
2. MCSes ← ∅
3. k ← 1
4. **while** (SAT(ϕ′))
5. ϕ′$_k$ ← ϕ′ ⋀ **AtMost**({¬y_1, ¬y_2, . . . , ¬y_n}, k)
6. **while** (newMCS ← IncrementalSAT(ϕ′$_k$)) *%All MCS of size K*
7. MCSes ← MCSes ∪ {newMCS}
8. ϕ′$_k$ ← ϕ′$_k$ ⋀ **BlockingClause**(newMCS) % Excludes super sets for *% for size k*
9. ϕ′ ← ϕ′ ⋀ **BlockingClause**(newMCS) *% Excludes super set % for size > k*
10. **end while**
11. k←k+1
12. **end while**
13. **return** MCSes

- *Incremental solver (MiniSAT) can be used in loop (l. 6) because constraints are only added but not external loop(l.4) since incrementing k relaxes constraints*
- *The set of yi variables assigned to false indicates the clauses in MCS*

Computing all MCS – *Example* / Liffiton & Sakallah-2007

- $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$
- $\phi = (x_1) \wedge (\neg x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$
- $\phi' = (\neg y_1 \vee x_1) \wedge (\neg y_2 \vee \neg x_1) \wedge (\neg y_3 \vee \neg x_1 \vee x_2) \wedge (\neg y_4 \vee \neg x_2) \wedge (\neg y_5 \vee \neg x_1 \vee x_3) \wedge (\neg y_6 \vee \neg x_3)$
- **K = 1**

  $\neg y_1 \wedge \neg x_1 \wedge (\neg x_1 \vee x_2) \vee \neg x_2 \vee (\neg x_1 \vee x_3) \wedge \neg x_3$: SAT $(\neg x_1 \wedge \neg x_2 \wedge x_3) \rightarrow$ **MCS : (C_1)**

  Adding : $\neg\neg y_1$, so $(\neg y_1 \vee x_1)$ reduces to x_1

  $x_1 \wedge \neg y_2 \wedge (\neg x_1 \vee x_2) \vee \neg x_2 \vee (\neg x_1 \vee x_3) \wedge \neg x_3$: UNSAT

  $x_1 \wedge \neg x_1 \wedge \neg y_3 \wedge \neg x_2$ ….: UNSAT

  …
- **K = 2**

  $\phi' = (\neg y_1 \vee x_1) \wedge (\neg y_2 \vee \neg x_1) \wedge (\neg y_3 \vee \neg x_1 \vee x_2) \wedge (\neg y_4 \vee \neg x_2) \wedge (\neg y_5 \vee \neg x_1 \vee x_3) \wedge (\neg y_6 \vee \neg x_3) \wedge \mathbf{y_1}$

  $= x_1 \wedge (\neg y_2 \vee \neg x_1) \wedge (\neg y_3 \vee \neg x_1 \vee x_2) \wedge (\neg y_4 \vee \neg x_2) \wedge (\neg y_5 \vee \neg x_1 \vee x_3) \wedge (\neg y_6 \vee \neg x_3)$

  $= x_1 \wedge \neg y_2 \wedge \neg y_3 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$:UNSAT

  $x_1 \wedge \neg x_1 \wedge$ …. :UNSAT

  …
- **K = 3**

  $x_1 \wedge \neg y_2 \wedge \neg y_3 \wedge \neg y_4 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$:UNSAT

  $x_1 \wedge \neg y_2 \wedge (\neg x_1 \vee x_2) \wedge \neg y_4 \wedge \neg y_5 \wedge \neg x_3$: SAT $(x_1, \neg y_2, x_2, \neg y_4, \neg y_5, \neg x_3)$: → **MCS : ($C_2$,$C_4$, C_5)**

  …

Computing one MUS from a set of MCSes

CAMUS - Liffiton & Sakallah-2007

Irreducible hitting set

Consider MCSes ={{C_1,C_2,C_3},{C_2,C_4}} : {C_1,C_2} is a hitting set but is not Irreducible

PropagateChoice(MCSes, thisClause, thisMCS)

for each clause ∈ thisMCS

    for each testMCS ∈ MCSes

        if (clause ∈ testMCS) then testMCS ← testMCS - {clause}

for each testMCS ∈ MCSes

    if (thisClause ∈ testMCS) then MCSes ← MCSes - {testMCS}

MaintainNoSupersets(MCSes) *% removes any set in MCSes that is **now** a % superset of some other*

Consider {{C_1,C_2,C_3},{C_2,C_4},{C_1, C_3,C_4,C_5}}, after selection of $C_1$of first MCS:

1. Remove C_2 and C_3 in other MCS → {{C_4},{C_1,C_4,C_5}}
2. Remove MCS where C_1 occurs → {{C_4}}

Computing one MUS from a set of MCSes

CAMUS - Liffiton & Sakallah-2007

Irreducible hitting set

Consider MCSes ={{C_1,C_2,C_3},{C_2,C_4}} : {C_1,C_2} is a hitting set but is not Irreducible

PropagateChoice(MCSes, thisClause, thisMCS)
for each clause ∈ thisMCS
    for each testMCS ∈ MCSes
        if (clause ∈ testMCS) then testMCS ← testMCS - {clause}
for each testMCS ∈ MCSes
    if (thisClause ∈ testMCS) then MCSes ← MCSes - {testMCS}
MaintainNoSupersets(MCSes) *% removes any set in MCSes that is **now** a % superset of some other*

Consider {{C_1,C_2,C_3},{C_2,C_4},{C_1, C_3,C_4,C_5}}, after selection of $C_1$of first MCS:

1. Remove C_2 and C_3 in other MCS → {{C_4},{C_1,C_4,C_5}}
2. Remove MCS where C_1 occurs → {{C_4}}

Computing all MUSes: *CAMUS*

Liffiton & Sakallah-2007

Intuition: recursive algorithm that branches at the two choice points

```
All_MUSes(MCSes, currentMUS)
1     if (MCSes = ∅)
2           print(currentMUS)
3           return
4     end if
5     for each selClause ∈ RemainingClauses(MCSes) % try all clause choices
6           newMUS ← currentMUS ∪ selClause
7           for each selMCS ∈ MCSes such that selClause ∈ selMCS
                                                    % try all MCS choices
8                 newMCSes ← MCSes
9                 PropagateChoice(newMCSes, selClause, selMCS)
10                AllMUSes(newMCSes, newMUS)
11          end for
12    end for
13    return
```

Computing all MUSes – *CAMUS: Optimizations & Relaxations*

Liffiton & Sakallah-2007

- ***Optimizations***
  - Elimination of ***duplicates***:

    MCSes:{{C_1,C_2},{C_1,C_2}} → Outputs twice MUS {C_1}

    Using hash table to store an intermediate state → returning immediately from AllMUSes if the current state matches an entry already in the table
  - Optimizing ***MaintainNoSupersets***
  - ***Grouping clauses:*** assigning a *single clause-selector variable yi* per per high-level statement (and it is added it to every constraint generated from that statement) ***→ search can enable or disable entire statements/blocks***

- ***Relaxations***

  Set of MCSes and MUSes can be exponentially large: MUSes is an anytime algorithm but if **All_MCSes** is stopped early computed hitting set may not be unsatisfiable

  → truncating MCSes: A subset P ⊆ C is a PCS if there exists some MCS M such that P ⊆ M

  - *Reduce size* of All_MCSe'**s** output (each PCS "represents" all of the MCSes that are supersets of it)
  - *Reduce the number of MUSes* computed by All_MUSes

Marco Polo : Finding Multiple MUSes Quickly

(Liffiton , Malik 2013)

- Constraint-agnostic (there exists a solving method capable of returning SAT or UNSAT)
- As efficient as the best single-MUS algorithm
- Good anytime behaviour
- **Polo** : Efficiently explores the **power set** $\mathcal{P}(C)$

  → **Power set Logic :** general technique of maintaining a "map" of $\mathcal{P}(C)$ as a propositional logic formula.
- **Marco** : **Mapping Regions** of **Constraint** sets — the MUS enumeration algorithm

POLO Framework (1)

Any function f : $\mathcal{P}(C)$ → {0,1} can be represented by a propositional formula over |C| variables

- **POLO** maintains a particular function :

  f : $\mathcal{P}(C)$ → {0,1} that tracks "unexplored" subsets

  $C' \subseteq C$ such that f(C') = 1 iff the satisfiability of C' is unknown and it remains to be checked

  f is stored as a propositional formula and can be viewed as a "map" of $\mathcal{P}(C)$ showing which "regions" have been explored and which have not

- **Solver for Map**:
  - any engine for obtaining a model of a Boolean formula can be used
  - an incremental interface or Binary Decision Diagram (**BDD**) engines will be most efficient

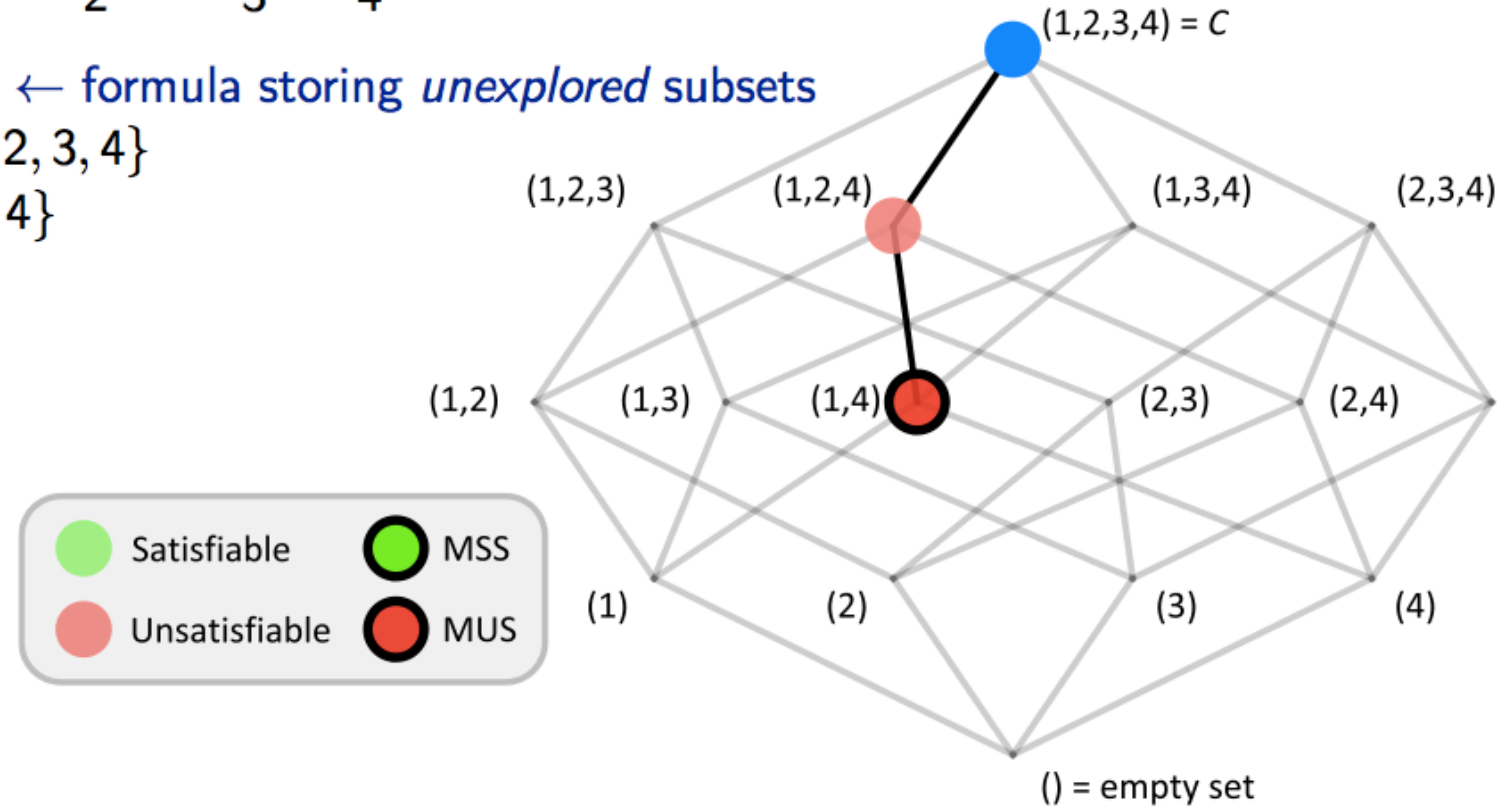
POLO Framework (2)

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

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

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

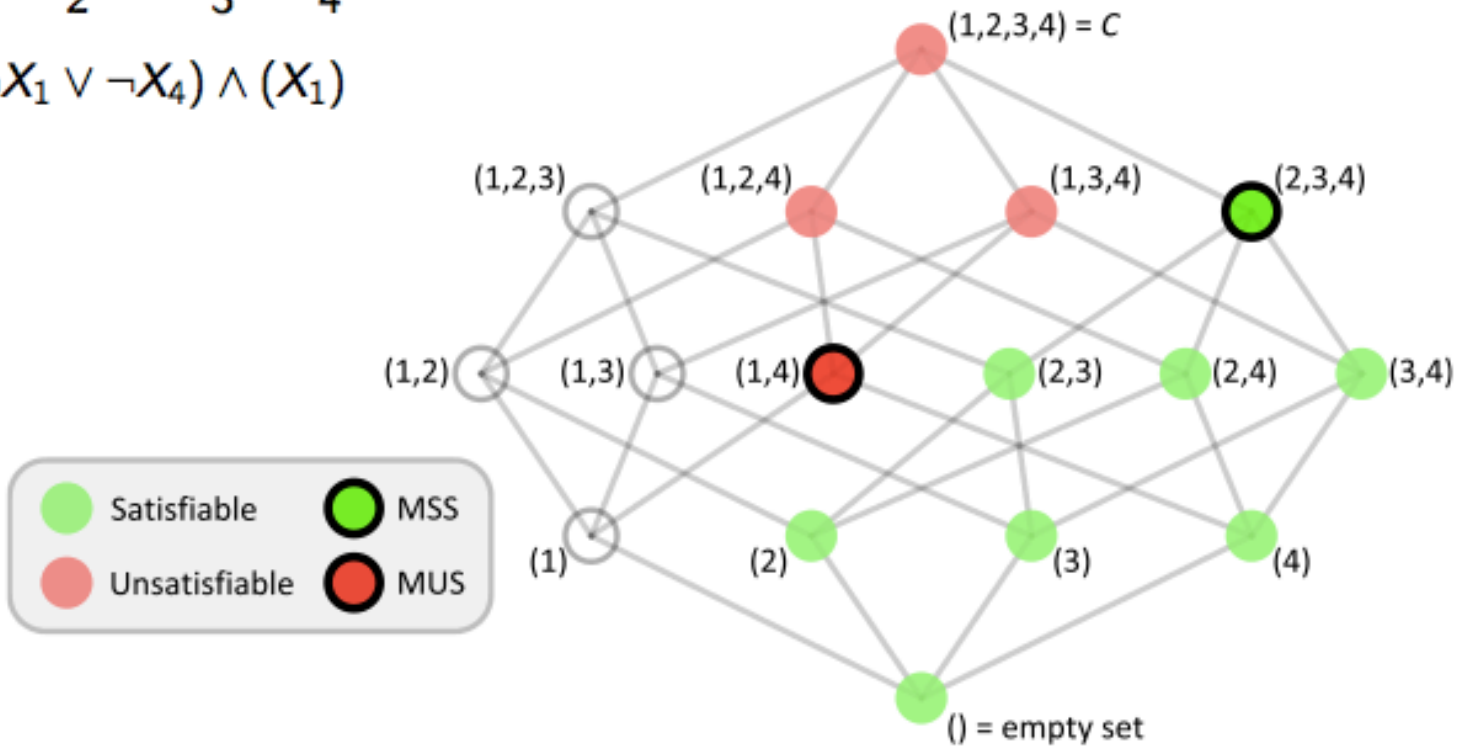
MUS: $\{1, 4\}$



POLO Framework (3)

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

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



MARCO algorithm for enumerating MSSes & MUSes of a constraint set (1)

grow(seed, C)
input: unsatisfiable constraint set C
input: satisfiable subset seed $\subset$ C
output: an MSS of C
1. **for c $\in$ C \ seed:**
2. **if** seed $\cup$ {c} **is satisfiable:**
3. seed = seed $\cup$ {c}
4. **return** seed

shrink(seed, C)
input: unsatisfiable constraint set C
input: unsatisfiable subset seed $\subseteq$ C
output: an MUS of C
1. **for c $\in$ seed:**
2. **if** seed \ {c} **is unsatisfiable:**
3. seed = seed \ {c}
4. **return** seed

Models corresponding to any subset of M are eliminated by requiring that later models of Map include at least one constraint not in M:

blockDown(M) $\equiv \bigvee_{i : C_i \notin M} x_i$

Supersets of M are blocked by requiring models to exclude at least one of its constraints:

blockUp(M) $\equiv \bigvee_{i : C_i \in M} \neg x_i$

MARCO algorithm for enumerating MSSes & MUSes of a constraint set (2)

MARCO enumerates MUSes by repeatedly selecting an unexplored subset $C' \in \mathcal{P}(C)$ from the map, checking whether C' is satisfiable, minimizing or maximizing it into an MUS or an MSS, and marking a region of the map as explored based on that result

input: unsatisfiable constraint set $C = \{C_1, C_2, C_3, \ldots, C_n\}$

output: MSSes and MUSes of C as they are discovered

Map ← BoolFormula(nvars = $|C|$) ▷ *Empty formula over $|C|$ Boolean variables*

while Map **is satisfiable:**

    m ← **getModel(Map)**
    seed ← $\{C_i \in C : m[x_i] = \text{True}\}$ ▷ *Project the assignment* m *onto* C
    if seed **is satisfiable:**
        MSS←grow(seed, C)
        yield MSS
        Map ← Map ∧ **blockDown(MSS)**
    else:
        MUS ← **shrink(seed,** C)
        yield MUS
        Map ← Map ∧ **blockUp(MUS)**

MARCO : example

C ={ (a), (¬a v b), (¬b), (¬a)}
C_1 C_2 C_3 C_4

Initialization: Map ← [empty formula over $\{x_1,x_2,x_3,x_4\}$]
Iteration 1: Map = ⊤ : **SAT**
getModel → $[x_1,x_2,x_3,x_4]$
seed ← $\{C_1, C_2, C_3, C_4\}$: **UNSAT**
shrink → MUS: $\{C_1, C_2, C_3\}$
Map ← Map∧$(\neg x_1 \vee \neg x_2 \vee \neg x_3)$
Iteration 2: Map = $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$: **SAT**
getModel → $[\neg x_1,x_2,x_3,x_4]$
seed ← $\{C_2, C_3, C_4\}$: **SAT**
grow → MSS: $\{C_2, C_3, C_4\}$ — equiv. MCS: {C1}
Map ← Map ∧ (x1)
Iteration 3: Map = $(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1)$: **SAT**
getModel → $[x_1, \neg x_2,x_3,x_4]$
seed ← $\{C_1, C_3, C_4\}$: **UNSAT**
shrink → MUS: $\{C_1, C_4\}$
Map ← Map ∧ $(\neg x_1 \vee \neg x_4)$

…still 3 iterations required to ensure completeness

Computing all MCS – SMT context (1)
Morgado,Liffiton, Marques-Silva 2012

- **Output of MaxSMT** : assignment A (consistent with theory T) that minimize the number of falsified clauses of ϕ

  → *any MaxSMT solution indicates an MCS* :

  - the constraints not satisfied by that solution must be an MCS,
  - any such solution is a smallest MCS.
- **Two problems**:
  - **AllMinMCS** problem: finding all the minimum size MCSes of ϕ.
  - **AllMCS** problem: finding all the MCSes of ϕ (independent of their size)
- ALL MCS Algorithms based on *3 SMT algorithms*: MSU1-SMT, MSU3-SMT, FM-SMT

Computing all MCS – SMT context (2)

Morgado,Liffiton, Marques-Silva 2012

ALL-MCS-MSU3-SMT(ϕ)

$\phi_W \leftarrow \phi$ % ϕ_W : working formula

$RV \leftarrow \varnothing$ % RV : Set of relaxation variables

$\lambda \leftarrow 0$ % λ: Lower bound on true relaxation variables

while true

do (st,A) ← **MSU3-SMT** (ϕ, ϕ_W , RV , λ)

if st = **true** % A : satisfying assignment (solution to MaxSMT)

then ReportMCS(ϕ, A)

BlockMCSbyRV(ϕ_W , RV , A)

else exit

Computing all MCS – Speed up (1)

FastDiag (Felfernig et al 2012)

mimics QUICKXPLAIN's → *approach same complexity*

If |largest MCS| << |R| BFD can outperform linear search

Function BFD(R,T ⊆R,hasD)

% R: set of clauses one aims to maximally satisfy (reference set

% T : the clauses that can be extracted when attempting to satisfy the clauses in R

% hasD :indicates whether a valid correction set already exists.

1 **if** hasD ∧ SAT(R) then return ∅
2 **if** |T|=1 **then return** T
3 m ← |T|/2
4 $(T_1, T_2) \leftarrow (T_{1..m}, T_{m+1..|T|})$
5 $D_2 \leftarrow$BFD($R \backslash T_1, T_2, T_1 \neq \emptyset$)
6 $D_1 \leftarrow$BFD($R \backslash D_2, T_1, D_2 \neq \emptyset$)
7 return $D_1 \cup D_2$ // Clauses of MCS

Basic FASTDIAG (BFD)

Computing all MCS – Speed up (2)
Enhanced FastDiag (Silva et al, 2013)

Three techniques to reduce the number of calls to a SAT solver and to simplify each call:

- finding a *disjoint set of unsatisfiable cores*
- Exploiting *backbone literals*
- Exploiting clauses satisfied by satisfying assignments for computing one MCS

Computing all MCS – Speed up (3)
Enhanced FastDiag (Silva et al, 2013)

Intuition (see property 1 & 2)

- F: an unsatisfiable formula F, μ an assignment μ

  → μ induces a partition {S , U } of F where S = satisfied clauses, U = falsified clauses

  → *There exist an MSS* W *and an MCS* M *of* F *such that* $S \subseteq W$ *and* $U \supseteq M$

- If F is partitioned into $\{G, C_1, \ldots, C_r\}$, such that G is satisfiable, and $C_1, \ldots, C_r$ are disjoint unsatisfiable cores

  → any assignment μ that satisfies G partitions each unsatisfiable core C_i into two disjoint sets S_i and U_i of satisfied and falsified clauses

  Let $S = G \cup \cup^{r}_{i=1} S_i$, with $S \subseteq F$, the clauses satisfied by some assignment μ

  → $S \subseteq W$ where W is an MSS of F

  S is to be iteratively updated with additional clauses.

  Let S_q qth update : $S \equiv S_0 \subset S_1 \subset \ldots \subset S_q \subset \ldots \subseteq W$ (each set S_q is satisfiable, and so is a subset of an MSS W of F . The set of falsified clauses from unsatisfiable core C_1 is U_1 (with $U_1 \neq \varnothing$, since C_1 is unsatisfiable)

  Consider each clause c in U_1, the clauses in S_q must be satisfied. If $S_q \cup \{c\}$ is satisfiable, then $S_{q+1} = S_q \cup \{c\}$; otherwise, since $S_q \cup \{c\} \models \bot$, c *must* be included in any MCS M of F, such that the MSS W = F \ M contains S_q

Computing all MCS – Speed up (4)
Enhanced FastDiag (Silva et al, 2013)

Computing disjoint unsatisfiable cores

Function DISJOINT_UNSATISFIABLE_CORES(F)
Output: μ: Assignment; F: Subformula; U: Disjoint cores (unsatisfiable formula)

```
1    (U, st ) ← (∅, false)
2    while not st do
3          (st, μ, V) ← SAT(F)
                  %  st: outcome (true or false) ,
                  % μ: the computed model (if the st is true)
                  % V: computed unsatisfiable subformula (if st is false)
4          if  not st then
5                F←F \ V
6     U ← U ∪ {V}
7     return (μ, F, U)
```

For enumerating MCSes, function DISJOINT_UNSATISFIABLE_CORES can be used recursively until a computed unsatisfiable core is empty

Computing all MCS – Speed up (6)
Enhanced FastDiag (Silva et al, 2013)

*Exploiting **backbone** literals*

M : an MCS of F

$\forall c \in M, c = (l_1 \vee l_2 \vee \ldots \vee l_k), (F \setminus M) \cup \{c\} \models \bot$

$\rightarrow F \setminus M \models \bar{c}$, i.e. $\forall l_i \in c, F \setminus M \models \neg l_i$

Thus, $\neg l_i$ is a *backbone literal* of F\M

L : set of clauses to be included in some MCS of F

$\rightarrow$ for every MCS M of F such that $L \subseteq M$, the complements of the literals of each of the clauses of L represent backbone literals of F \ M

$\rightarrow$ *any* algorithm for MCS computation can exploit backbone literals, by adding the negation of each clause that is decided to be included in a MCS of F

Computing all MCS – Speed up (6)
Enhanced FastDiag (Silva et al, 2013)

Function EFD(R,T $\subseteq$R,hasD)
    Global: S,B,U_1,...,U_r
if *hasD* **then**
    (st, μ) ← SAT(R ∪ S ∪ B)
    if st **then**
        UpdateSatisfiedClauses(μ)
        return ∅
if|T|=1 **then**
    UpdateBackboneLiterals(T)
    return T
m←|T|/ 2
(T_1, T_2) ← ($T_{1..m}$, $T_{m+1..|T|}$)
D_2 ← **EFD**(R\T_1, T_2\S, T_1≠ ∅)
D_1 ← **EFD**(R\D_2, T_1 \S, D_2 ≠ ∅)
return D_1 ∪ D_2 // Clauses of MCS

On going work at I3S

- Experiments
  - Simple tests: Slice, MinMax, Foo
  - Different versions of Trityp (determines the type of a triangle, computation of perimeter)
  - TCAS
- Efficient computation of MCS / MUS with a *limited cardinality*
- Efficient computation of MCS / MUSes with a *limited number of failing conditions*
- *Efficient consistency checking* for LP, MIP , floating point numbers