CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases

Pedro Orvalho [1], Mikoláš Janota [2] and Vasco Manquinho [1]

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2]CIIRC, Czech Technical University in Prague, Czechia

FM 24, Milan, Italy

Thursday 12th September, 2024

Motivation

- Debugging is one of the most time-consuming and expensive tasks in software development.

Motivation

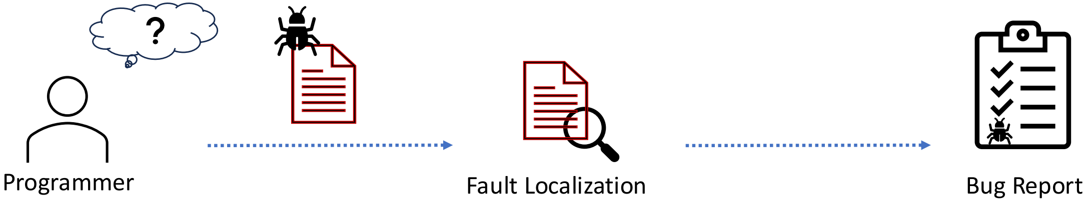
- Debugging is one of the most time-consuming and expensive tasks in software development.
  - In 2000, the total cost of the work done in preparation for Year 2000 Problem likely surpassed 400 Billion US$ [The Guardian, 2019];

Motivation

- Debugging is one of the most time-consuming and expensive tasks in software development.
    - In 2000, the total cost of the work done in preparation for Year 2000 Problem likely surpassed 400 Billion US$ [The Guardian, 2019];
    - In 2024, the estimated global cost of Crowdstrike's error that hit Microsoft systems, is 24 Billion US$ [The Sun UK, 2024].

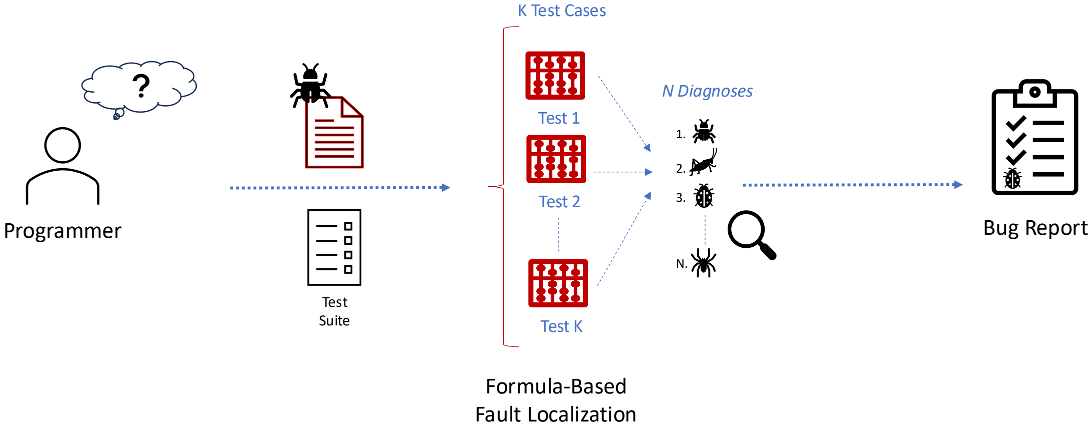
Fault Localization

- Given a buggy program, *fault localization (FL)* involves identifying locations in the program that could cause a faulty behaviour (bug).



Formula-Based Fault Localization (FBFL)

- FBFL methods encode the localization problem into **several optimization problems** to identify a minimal set of bugs (diagnoses).



Current Formula-Based Fault Localization

1: Faulty program example. Faulty lines: {4,6,8}.

```
1  int main(){
2    int f,s,t;
3    scanf("%d%d%d",&f,&s,&t);
4    if (f < s && f >= t)
5      printf("%d",f);
6    if (f > s && s <= t)
7      printf("%d",s);
8    if (f > t && s > t)
9      printf("%d",t);
10   return 0;
11 }
```

Table 1: Test-suite.

	Input			**Output**
t0	1	2	3	3
t1	-1	-2	-3	-1
t2	1	2	1	2

Current Formula-Based Fault Localization

2: Faulty program example. Faulty lines: {4,6,8}.

```c
1  int main(){
2    int f,s,t;
3    scanf("%d%d%d",&f,&s,&t);
4    if (f < s && f >= t)
5      printf("%d",f);
6    if (f > s && s <= t)
7      printf("%d",s);
8    if (f > t && s > t)
9      printf("%d",t);
10   return 0;
11 }
```

	BugAssist	SNIPER
#Diagnoses t_0	8	8
#Diagnoses t_1	21	21
#Diagnoses t_2	9	9
#Total Unique Diagnoses	32	1297
Final Diagnosis	{3,9}	{4,6,8}

Table 2: Number of diagnoses (faulty statements) generated by BugAssist [Jose et al., 2011] and SNIPER [Lamraoui et al., 2016] per test.

Current Formula-Based Fault Localization

	BugAssist	SNIPER
#Diagnoses t_0	8	8
#Diagnoses t_1	21	21
#Diagnoses t_2	9	9
#Total Unique Diagnoses	32	1297
Final Diagnosis	$\{3,9\}$	$\{4,6,8\}$

Table 3: Number of diagnoses (faulty statements) generated by BugAssist [Jose et al., 2011] and SNIPER [Lamraoui et al., 2016] per test.

Current Limitations

FBFL tools especially for programs with multiple faults:

Current Formula-Based Fault Localization

	BugAssist	SNIPER
#Diagnoses t_0	8	8
#Diagnoses t_1	21	21
#Diagnoses t_2	9	9
#Total Unique Diagnoses	32	1297
Final Diagnosis	{3,9}	{4,6,8}

Table 3: Number of diagnoses (faulty statements) generated by BugAssist [Jose et al., 2011] and SNIPER [Lamraoui et al., 2016] per test.

Current Limitations

FBFL tools especially for programs with multiple faults:

- **do not ensure a minimal diagnosis** across all failing tests (e.g., BugAssist);

Current Formula-Based Fault Localization

	BugAssist	SNIPER
#Diagnoses t_0	8	8
#Diagnoses t_1	21	21
#Diagnoses t_2	9	9
#Total Unique Diagnoses	32	1297
Final Diagnosis	{3,9}	{4,6,8}

Table 3: Number of diagnoses (faulty statements) generated by BugAssist [Jose et al., 2011] and SNIPER [Lamraoui et al., 2016] per test.

Current Limitations

FBFL tools especially for programs with multiple faults:

- **do not ensure a minimal diagnosis** across all failing tests (e.g., BugAssist);
- may produce an overwhelming number of **redundant sets of diagnoses** (e.g., SNIPER).

Our Work

- We formulate the FL problem as a **single optimization problem**;

Our Work

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;

Our Work

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;
- We implement this MBD approach in a publicly available tool called CFaults.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.
- For each component $c \in \mathcal{C}$, $h(c) = 0$ **if** c **is unhealthy, otherwise,** $h(c) = 1$.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.
- For each component $c \in \mathcal{C}$, $h(c) = 0$ **if** c **is unhealthy, otherwise,** $h(c) = 1$.
- $\mathcal{P}$ is described by a CNF formula, where $\mathcal{F}_c$ denotes the encoding of component c:

$$\mathcal{P} \triangleq \bigwedge\nolimits_{c \in \mathcal{C}} (\neg h(c) \vee \mathcal{F}_c) \tag{1}$$

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, to be encodable in CNF as a set of unit clauses.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, to be encodable in CNF as a set of unit clauses.
- In this work, **the failing test cases represent the set of observations**.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, to be encodable in CNF as a set of unit clauses.
- In this work, **the failing test cases represent the set of observations**.
- A system $\mathcal{P}$ is considered **faulty if there exists an inconsistency with a given observation o when all components are declared healthy**:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C}} h(c) \vDash \bot \tag{2}$$

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \not\models \perp \tag{3}$$

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \perp \tag{3}$$

- A diagnosis Δ is minimal iff no subset of Δ, $\Delta' \subsetneq \Delta$, is a diagnosis, and Δ is of minimal cardinality if there is no other diagnosis $\Delta'' \subseteq \mathcal{C}$ with $|\Delta''| < |\Delta|$.

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \perp \tag{3}$$

- A diagnosis Δ is minimal iff no subset of Δ, $\Delta' \subsetneq \Delta$, is a diagnosis, and Δ is of minimal cardinality if there is no other diagnosis $\Delta'' \subseteq \mathcal{C}$ with $|\Delta''| < |\Delta|$.
- A diagnosis is redundant if it is not subset-minimal [Ignatiev et al., 2019].

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;
- The soft clauses consists of unit clauses that aim to **maximize the set of healthy components**, i.e.,:

$$\bigwedge_{c \in \mathcal{C}} h(c);$$

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;
- The soft clauses consists of unit clauses that aim to **maximize the set of healthy components**, i.e.,:

$$\bigwedge_{c \in \mathcal{C}} h(c);$$

- This encoding enables enumerating subset **minimal diagnoses, considering a single observation**;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;
- The hard clauses, ϕ_h, in our MaxSAT formulation correspond to:

$$\phi_h = \bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \wedge o_i);$$

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;
- The hard clauses, ϕ_h, in our MaxSAT formulation correspond to:

$$\phi_h = \bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \wedge o_i);$$

- The soft clauses are formulated as:

$$\phi_s = \bigwedge_{c \in \mathcal{C}} h(c).$$

Model-Based Diagnosis with Multiple Test Cases

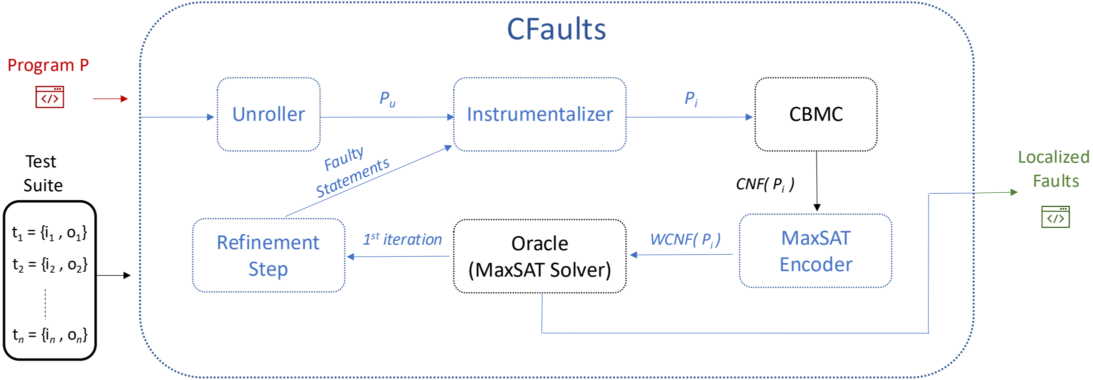
- The set of unhealthy components ($h(c) = 0$), corresponds to a **subset-minimal aggregated diagnosis**.

Model-Based Diagnosis with Multiple Test Cases

- The set of unhealthy components ($h(c) = 0$), corresponds to a **subset-minimal aggregated diagnosis**.
- This diagnosis is a subset-minimal of components that, when declared unhealthy (deactivated), make the system consistent with all observations, as follows:

$$\bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \land o_i) \land \bigwedge_{c \in \mathcal{C} \setminus \boldsymbol{\Delta}} h(c) \land \bigwedge_{c \in \boldsymbol{\Delta}} \neg h(c) \not\models \perp \tag{4}$$

CFaults



Program P
Test Suite
$t_1 = \{i_1, o_1\}$
$t_2 = \{i_2, o_2\}$
$t_n = \{i_n, o_n\}$
CFaults
Unroller
P_u
Instrumentalizer
P_i
CBMC
Faulty Statements
CNF(P_i)
Refinement Step
1^{st} iteration
Oracle (MaxSAT Solver)
WCNF(P_i)
MaxSAT Encoder
Localized Faults

Program unrolling

- An unrolled program is the **original program expanded** *m* **times**;
- It **encodes the execution of all failing tests** within the program;

```
float _input_f0[3] = {1, 2, 3};
char _out_0[2] = "3";
int _ioff_f0 = 0, _ooff_0 = 0;
// ... inputs and outputs for the other tests
int main(){
  scope_0:{
    int f_0, s_0, t_0;
    f_0 = _input_f0[_ioff_f0++];
    s_0 = _input_f0[_ioff_f0++];
    t_0 = _input_f0[_ioff_f0++];
    if ((f_0 < s_0) && (f_0 >= t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, f_0);
    if ((f_0 > s_0) && (s_0 <= t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, s_0);
    if ((f_0 > t_0) && (s_0 > t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, t_0);
    goto scope_1;
  }
  // ... scope_1 and scope_2
  final_step:
  assert(strcmp(_out_0, "3") != 0  // other assertions);
}
```

Program unrolling

For each scope, CFAULTS:

- generates **fresh variables and functions**;
- establishes **variables representing the inputs and outputs**;
- **embeds an assertion capturing all the specifications**.

```c
float _input_f0[3] = {1, 2, 3};
char _out_0[2] = "3";
int _ioff_f0 = 0, _ooff_0 = 0;
// ... inputs and outputs for the other tests
int main(){
  scope_0:{
    int f_0, s_0, t_0;
    f_0 = _input_f0[_ioff_f0++];
    s_0 = _input_f0[_ioff_f0++];
    t_0 = _input_f0[_ioff_f0++];
    if ((f_0 < s_0) && (f_0 >= t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, f_0);
    if ((f_0 > s_0) && (s_0 <= t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, s_0);
    if ((f_0 > t_0) && (s_0 > t_0))
        _ooff_0 = printInt(_out_0, _ooff_0, t_0);
    goto scope_1;
  }
  // ... scope_1 and scope_2
  final_step:
  assert(strcmp(_out_0, "3") != 0  // other assertions);
}
```

Program Intrumentalization

3: Program statements.

```
int i;
int n;
int s;

s = 0;
n = _input_f0[_ioff_f0++];

if (n == 0)
    return 0;

for (i=1; i < n; i++){
    s = s + i;
}
```

4: Program statements relaxed.

```
//main scope
bool _rv1, _rv2, _rv3, _rv5;
bool _rv6[UNWIND],..., _rv8[UNWIND];
int _los; // loop1 offset

//test scope
bool _ev4;
int i,n,s;
_los=1;

if (_rv1) s = 0;
if (_rv2) n = _input_f0[_ioff_f0++];
if ( _rv3 ? (n == 0) : _ev4)
    return 0;

for (_rv5 ? (i = 1) : 1;
     !_rv6[_los] || (i<n);
     _rv8[_los] ? i++ : 1, _los++){
    if (_rv7[_los]) s = s + i;
}
```

MaxSAT Encoder

- CFaults generates a weighted partial MaxSAT formula **aiming to minimize the necessary code alterations**;

MaxSAT Encoder

- CFaults generates a weighted partial MaxSAT formula **aiming to minimize the necessary code alterations**;
- The soft clauses are the relaxation variables used to instrument the C program, expressed as

$$\mathcal{S} = \bigwedge_{c \in \mathcal{C}} (rv_c);$$

MaxSAT Encoder

- CFaults generates a weighted partial MaxSAT formula **aiming to minimize the necessary code alterations**;
- The soft clauses are the relaxation variables used to instrument the C program, expressed as
$$\mathcal{S} = \bigwedge_{c \in \mathcal{C}} (rv_c);$$
- We assign **a hierarchical weight to each relaxation variable** based on the height of its sub-AST (abstract syntax tree);

MaxSAT Encoder

- CFAULTS generates a weighted partial MaxSAT formula **aiming to minimize the necessary code alterations**;
- The soft clauses are the relaxation variables used to instrument the C program, expressed as

$$\mathcal{S} = \bigwedge_{c \in \mathcal{C}} (rv_c);$$

- We assign **a hierarchical weight to each relaxation variable** based on the height of its sub-AST (abstract syntax tree);
- CFAULTS **enumerates all MaxSAT solutions** to identify all subset-minimal diagnoses.

Experimental Results

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];
- TCAS, **from Siemens**, comprises **41 versions** of a program with introduced faults;

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];
- TCAS, **from Siemens**, comprises **41 versions** of a program with introduced faults;
- C-Pack-IPAs is a set of **introductory programming assignments**. It consists of ten programming assignments, comprising **486 faulty programs**.

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];
- TCAS, **from Siemens**, comprises **41 versions** of a program with introduced faults;
- C-Pack-IPAs is a set of **introductory programming assignments**. It consists of ten programming assignments, comprising **486 faulty programs**.
- All the experiments were conducted using:

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];
- TCAS, **from Siemens**, comprises **41 versions** of a program with introduced faults;
- C-Pack-IPAs is a set of **introductory programming assignments**. It consists of ten programming assignments, comprising **486 faulty programs**.
- All the experiments were conducted using:
  - a memory limit of **32GB**;

Experimental Setup

- CFaults has been evaluated using two benchmarks of C programs: TCAS [Do et al., 2005] and C-Pack-IPAs [Orvalho et al., 2022];
- TCAS, **from Siemens**, comprises **41 versions** of a program with introduced faults;
- C-Pack-IPAs is a set of **introductory programming assignments**. It consists of ten programming assignments, comprising **486 faulty programs**.
- All the experiments were conducted using:
  - a memory limit of **32GB**;
  - a timeout of **3600 seconds** (1 hour).

Experimental Setup

BugAssist and SNIPER:

- are either **unavailable or no longer maintained**, prototypes of their algorithms were implemented;

Experimental Setup

BugAssist and SNIPER:

- are either **unavailable or no longer maintained**, prototypes of their algorithms were implemented;
- in this experiment, **handle ANSI-C programs**, as their algorithms are built on top of CFaults's unroller and instrumentalizer modules.

Results

Benchmark: **TCAS**

	Valid Diagnosis	Memouts	Timeouts
BugAssist	41 (100.0%)	0 (0.0%)	0 (0.0%)
SNIPER	7 (17.07%)	34 (82.93%)	0 (0.0%)
CFaults	41 (100.0%)	0 (0.0%)	0 (0.0%)
CFaults-Refined	41 (100.0%)	0 (0.0%)	0 (0.0%)

Table 4: BugAssist, SNIPER and CFaults fault localization results on TCAS.

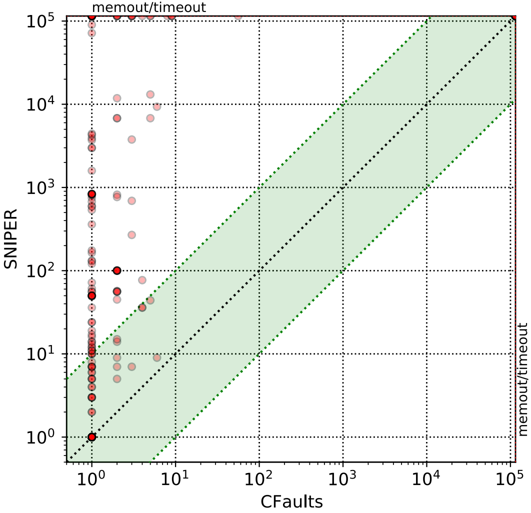
Results

Benchmark: **C-Pack-IPAs**

	Valid Diagnosis	Memouts	Timeouts
BugAssist	454 (93.42%)	0 (0.0%)	32 (6.58%)
SNIPER	446 (91.77%)	4 (0.82%)	36 (7.41%)
CFaults	483 (99.38%)	1 (0.21%)	2 (0.41%)
CFaults-Refined	482 (99.18%)	1 (0.21%)	3 (0.62%)

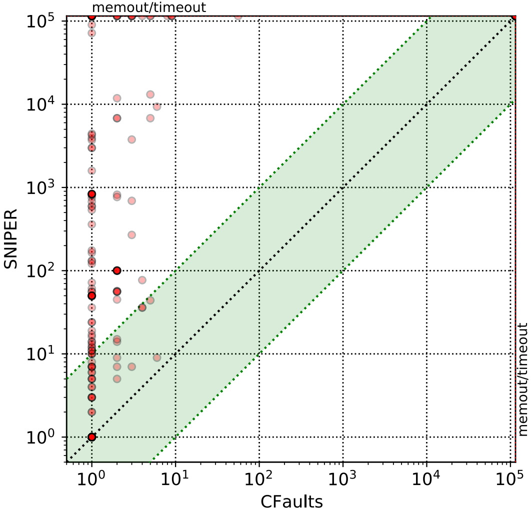
Table 5: BugAssist, SNIPER and CFaults fault localization results on C-Pack-IPAs.

Diagnoses Enumerated



1. CFAULTS needs to enumerate all MaxSAT solutions due to the weighted MaxSAT formula;

Diagnoses Enumerated



1. CFAULTS needs to enumerate all MaxSAT solutions due to the weighted MaxSAT formula;
2. SNIPER generates significantly more diagnoses.

Takeaway Message

- We tackle the FL problem in C using **Model-Based Diagnosis (MBD) with multiple failing test cases**, formulating it as a **unified optimization problem**;

Takeaway Message

- We tackle the FL problem in C using **Model-Based Diagnosis (MBD) with multiple failing test cases**, formulating it as a **unified optimization problem**;
- We only **generate subset-minimal aggregated diagnosis** to identify all faulty program components;

Takeaway Message

- We tackle the FL problem in C using **Model-Based Diagnosis (MBD) with multiple failing test cases**, formulating it as a **unified optimization problem**;
- We only **generate subset-minimal aggregated diagnosis** to identify all faulty program components;
- We present CFaults, **a fault localization tool for ANSI-C programs**, that:

Takeaway Message

- We tackle the FL problem in C using **Model-Based Diagnosis (MBD) with multiple failing test cases**, formulating it as a **unified optimization problem**;
- We only **generate subset-minimal aggregated diagnosis** to identify all faulty program components;
- We present CFaults, **a fault localization tool for ANSI-C programs**, that:
  - allows **refinement of localized faults** to pinpoint the bugs' location more precisely;

Takeaway Message

- We tackle the FL problem in C using **Model-Based Diagnosis (MBD) with multiple failing test cases**, formulating it as a **unified optimization problem**;
- We only **generate subset-minimal aggregated diagnosis** to identify all faulty program components;
- We present CFaults, **a fault localization tool for ANSI-C programs**, that:
  - allows **refinement of localized faults** to pinpoint the bugs' location more precisely;
  - is **fast and only produces subset-minimal diagnoses**, unlike other SOTA FBFL tools.

References

Reiter, Raymond (1987)
A Theory of Diagnosis from First Principles.
Artif. Intell. 1987.

Do, Hyunsook and Elbaum, Sebastian G. and Rothermel, Gregg (2005)
Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact.
Empir. Softw. Eng. 2005.

Manu Jose and Rupak Majumdar (2011)
Cause clue clauses: error localization using maximum satisfiability.
PLDI 2011.

Lamraoui, Si-Mohamed and Nakajima, Shin (2016)
A Formula-based Approach for Automatic Fault Localization of Multi-fault Programs.
J. Inf. Process. 24(1), 88 – 98.

References

Ignatiev, Alexey and Morgado, António and Weissenbacher, Georg and Marques-Silva, João (2019)
Model-Based Diagnosis with Multiple Observations.
IJCAI 2019.

Orvalho, P. and Janota, M. and Manquinho, V. (2022)
C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments.
arXiv:2206.08768.

The Guardian - Year 2000 Problem
https://www.theguardian.com/commentisfree/2019/dec/31/millennium-bug-face-fears-y2k-it-systems
The Guardian 2019.

The Sun UK - Crowdstrike Meltdown
https://www.thesun.co.uk/tech/27223882/microsoft-crowdstrike-meltdown-trillions-cost-world-economy.
The Sun UK.

CFaults

Thank you!

https://github.com/pmorvalho/cfaults