Wei-jiang HONG, Yi-jun LIU, Zhen-bang CHEN, Wei DONG, Ji WANG, 2020. Modified condition/decision coverage (MC/DC) oriented compiler optimization for symbolic execution. *Frontiers of Information Technology & Electronic Engineering*, 21(9):1267-1284. https://doi.org/10.1631/FITEE.1900213

Modified condition/decision coverage (MC/DC) oriented compiler optimization for symbolic execution

Key words: Compiler optimization; Modified condition/decision coverage (MC/DC); Optimization recommendation; Symbolic execution

Corresponding author: Zhen-bang CHEN
E-mail: zbchen@nudt.edu.cn

ORCID: http://orcid.org/0000-0003-0874-3231

Motivation

- Symbolic execution is an effective way to systematically explore the search space of a program.
- The program to be analyzed is usually compiled into a binary or an intermediate representation, on which symbolic execution is carried out. During this process, compiler optimizations influence the effectiveness and efficiency of symbolic execution.
- We provide a compiler optimization recommendation for symbolic execution with respect to modified condition/decision coverage (MC/DC).

Motivation

```
int get_sign (int x) {
    if (x==0)
        return 0;
    if (x<0)
        return −1;
    else
        return 1;
}
```

Symbolic executor (KLEE)

Generate 3 test cases
(e.g. 0, -1, 10)
→ 100% MC/DC

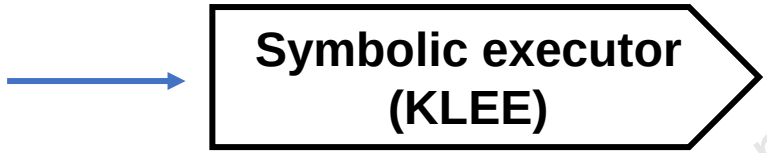
Compiler optimizations options

- Off Instruction combining
- Off Function inlining
- Off Promote memory to register
- Off Scalar replacement of aggregates
- Off Loop rotate
- Off Control flow graph simplification

An example program

Motivation

```
int get_sign (int x) {
    if (x==0)
        return 0;
    if (x<0)
        return −1;
    else
        return 1;
}
```



Generate 2 test cases
(e.g. 0, 10)
→ 50% MC/DC

Compiler optimizations options

- On — Instruction combining
- On — Function inlining
- On — Promote memory to register
- On — Scalar replacement of aggregates
- On — Loop rotate
- On — Control flow graph simplification

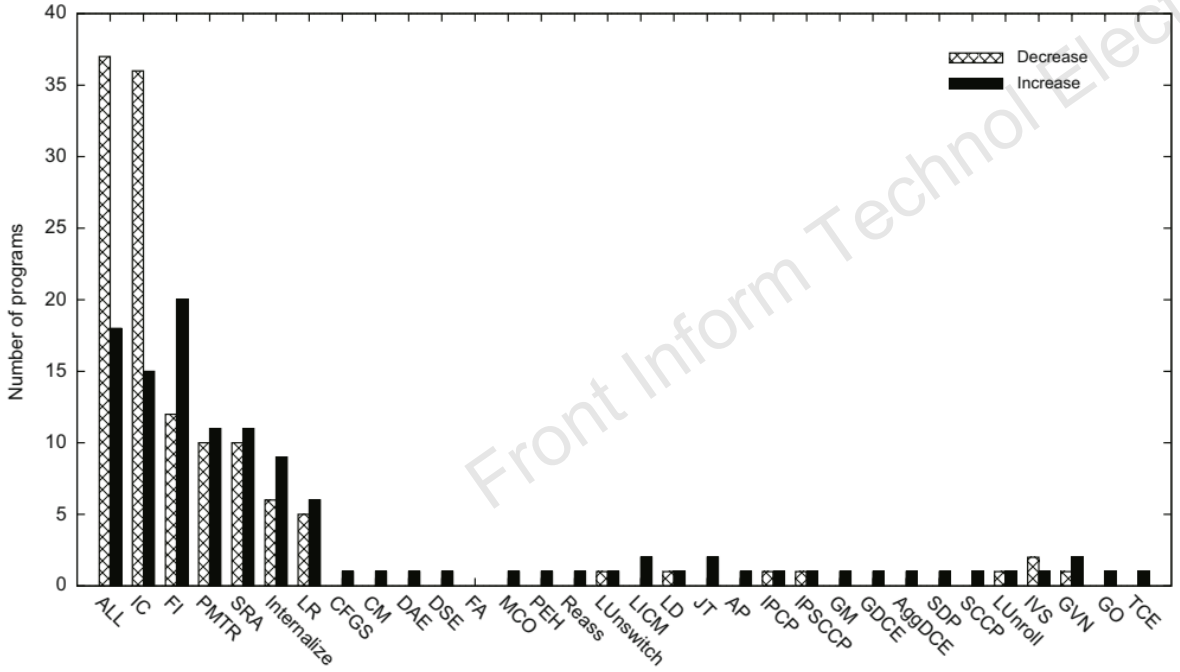
An example program

Method

- Investigate compiler optimization’s influence on MC/DC.
- Train a model to recommend compiler optimization to improve MC/DC.
- Evaluate the model’s effectiveness and efficiency.

i. Investigation

- 83 programs from Coreutils[1]
- We take the MC/DC of not using any compiler optimization as the baseline and count the number of influenced programs (decrease/increase MC/DC by more than 10%) after enabling certain compiler optimization.



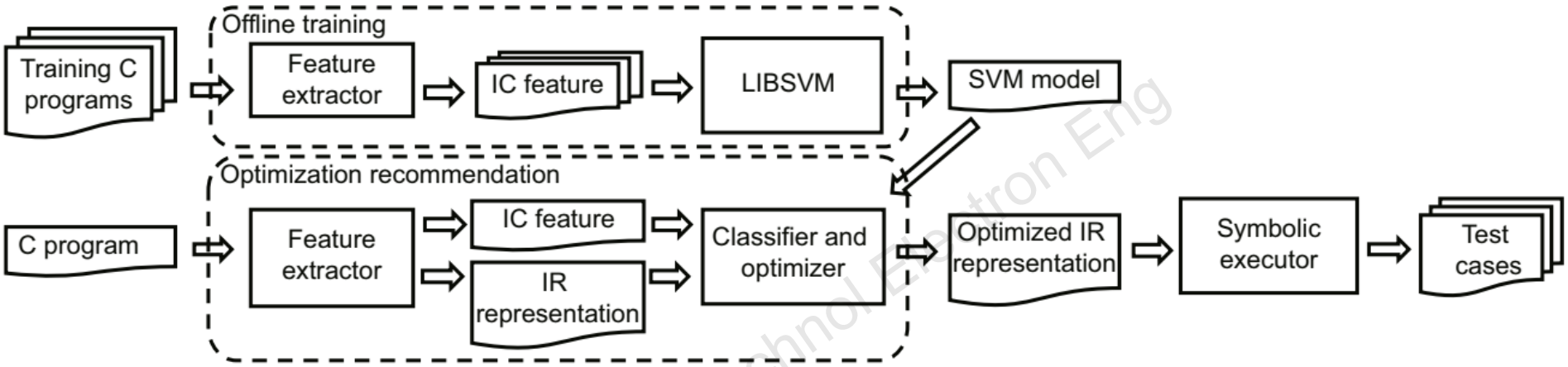
P	#c	P	#c	P	#c
base64	108	id	53	setuidgid	21
basename	11	join	259	shred	207
cat	136	link	6	shuf	130
chcon	134	logname	4	sleep	11
chgrp	22	ls	876	sort	668
chown	21	md5sum	220	split	91
chroot	8	mkdir	13	stat	58
cksum	90	mkfifo	10	stty	927
comm	61	mknod	24	sum	75
cp	165	mktemp	29	sync	3
csplit	171	mv	52	tac	130
cut	209	nice	23	tail	377
date	70	nl	163	tee	76
dd	230	nohup	30	touch	84
df	368	od	32	tr	256
dircolors	177	paste	118	TRUE	47
dirname	11	pathchk	56	tsort	219
echo	50	pinky	92	tty	6
env	59	pr	379	uname	24
expand	101	printenv	14	unexpand	128
expr	178	printf	121	uniq	131
factor	22	ptx	339	unlink	5
FALSE	47	pwd	28	uptime	27
fmt	145	readlink	8	users	32
fold	126	rm	23	wc	225
ginstall	93	rmdir	25	whoami	4
head	259	runcon	34	yes	7
hostid	3	seq	128		

P: program; *#c*: MC/DC condition

- IC (instruction combining) is the dominant one that influences MC/DC.

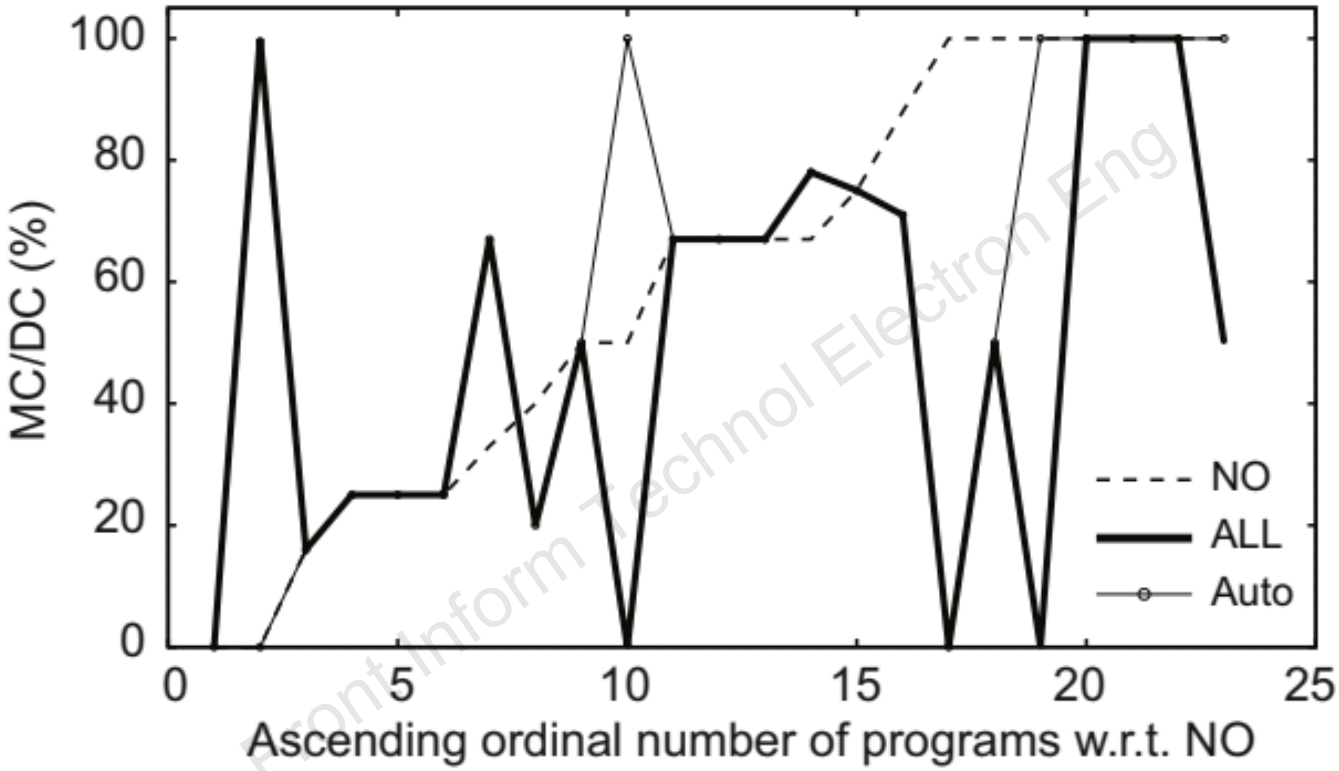
[1] https://www.gnu.org/software/Coreutils/Coreutils.html

ii. Training



- Extract the features of the program w.r.t. IC to obtain a vector of 43 dimensions.
- Label each program's features according to whether applying IC improves MC/DC during the investigation process.
- Use SVM to train a model that recommends two different compiler configurations: using all compiler optimization / using all compiler optimization except for IC.

iii. Evaluation



- The recommendation method's line (auto) is above those of ALL and NO on most (78.26%) programs, which implies the effectiveness of this method on the NECLA benchmark[1].

[1] https://www.neclabs.com/research/system/systems_SAV-website/benchmarks.php

Conclusions

- We provide a general framework for the empirical influence study of compiler optimization on symbolic execution w.r.t. MC/DC. Empirical studies indicate that instruction combining (IC) optimization is the dominant component influencing the effectiveness and efficiency of symbolic execution.

- We propose a method to extract the program features w.r.t. IC optimization, based on which an optimization recommendation method for symbolic execution is proposed to improve the coverage. Experimental results indicate that the recommendation method is effective.