DetReduce: Minimizing Android Test Suites for Regression Testing

Wontae Choi*, **Koushik Sen** @ UC Berkeley, George Necula*

Wenyu Wang** @ UIUC

** Currently at Google*

*** Work done while the author was at UC Berkeley*

Motivation

- Surge in apps for smartphones and tablets
  - More mobile phone apps than desktops
- Mobile apps have complex Graphical User Interfaces (GUI)
- Testing of mobile apps focus on GUI

Observation

- Many automated GUI testing tools
  - Learning-based
  - Model-based
  - Fuzzing
  - Static analysis based

Observation

- Our experience with automated GUI testing tools
  - SwiftHand [OOPSLA'13] and Monkey
  - The good:
    - achieve good coverage and find bugs
  - The bad:
    - Runs for several hours
    - Generates a large test suite
      - Unreadable, not easy to reuse

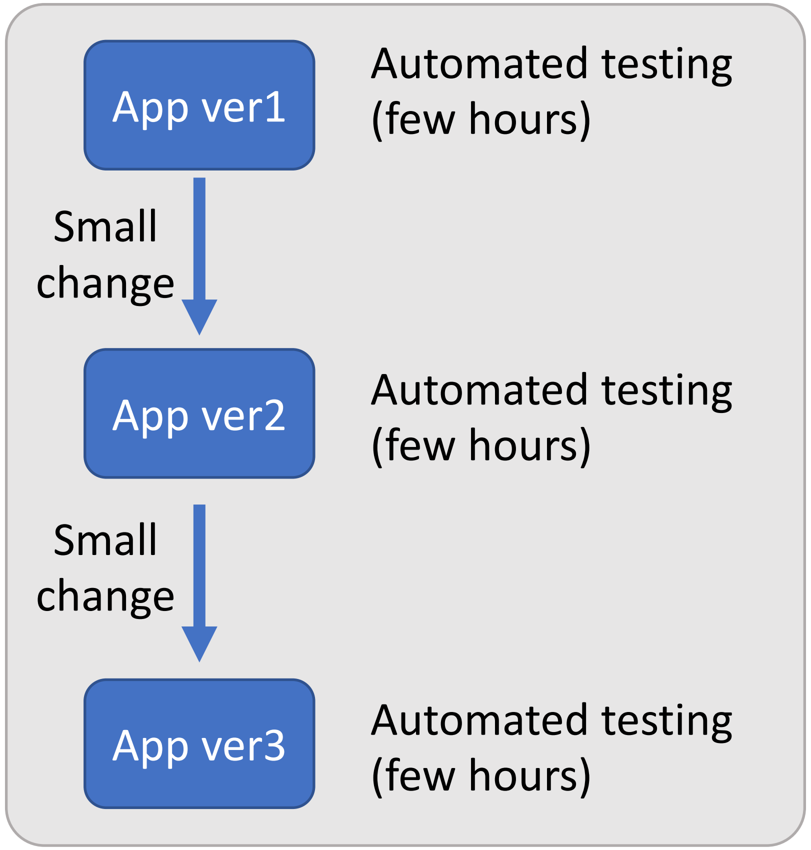
Programmers don't like this

Problem Statement

Can we generate a **small** regression test suite by **minimizing** a machine generated large test suite?

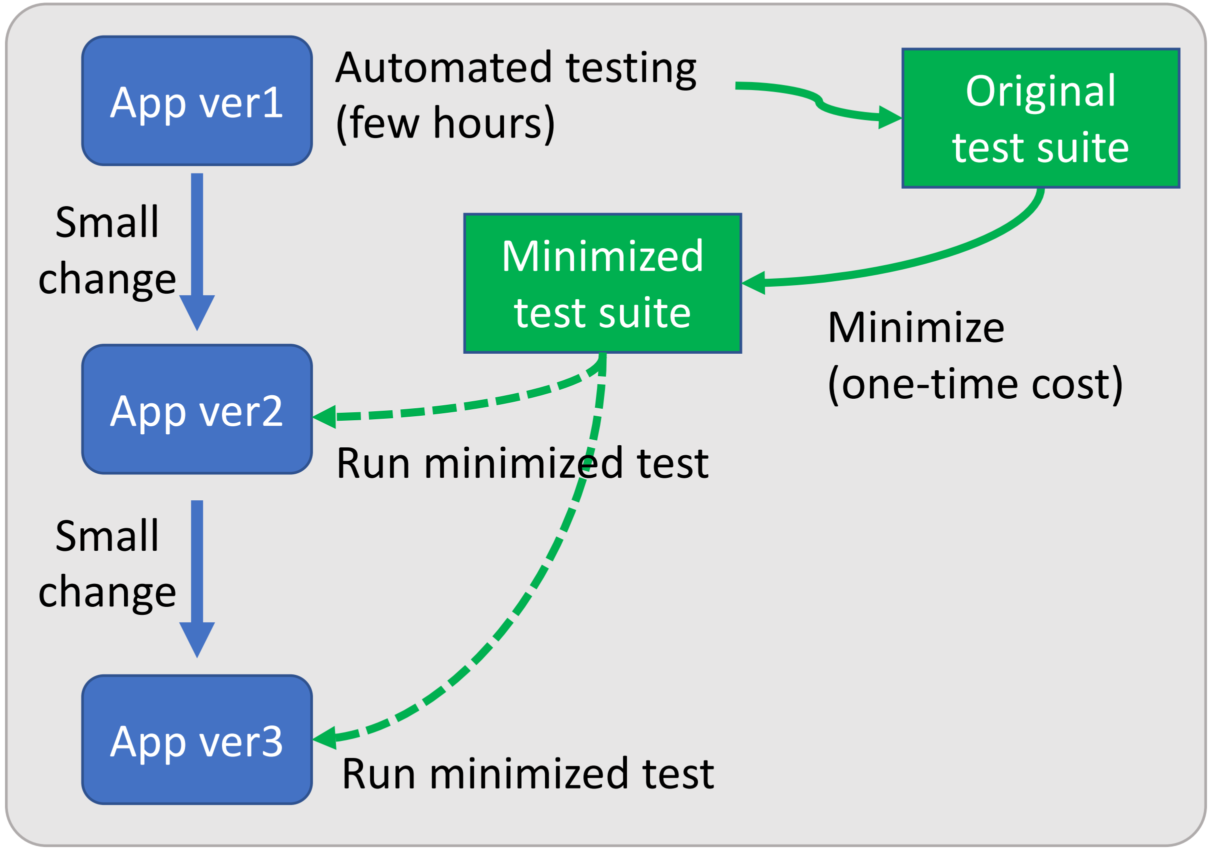
Why minimize?

- Without Minimization



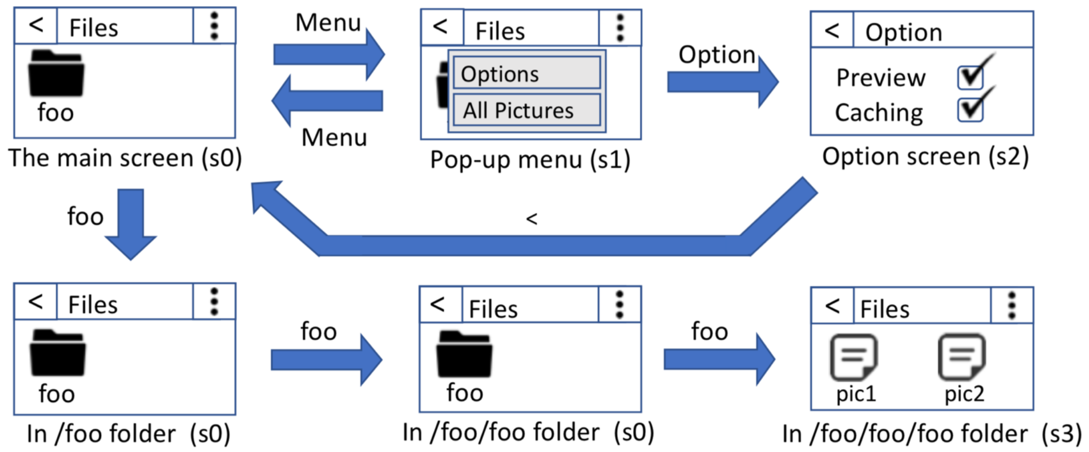
Repetitively pay a high cost

- With Minimization



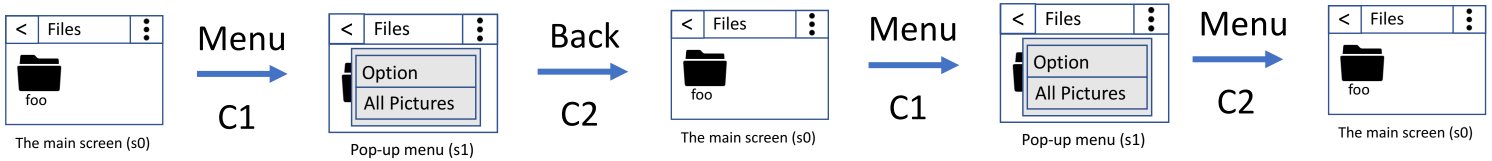
One-time high cost + cheaper repetition cost

What is test case? Example

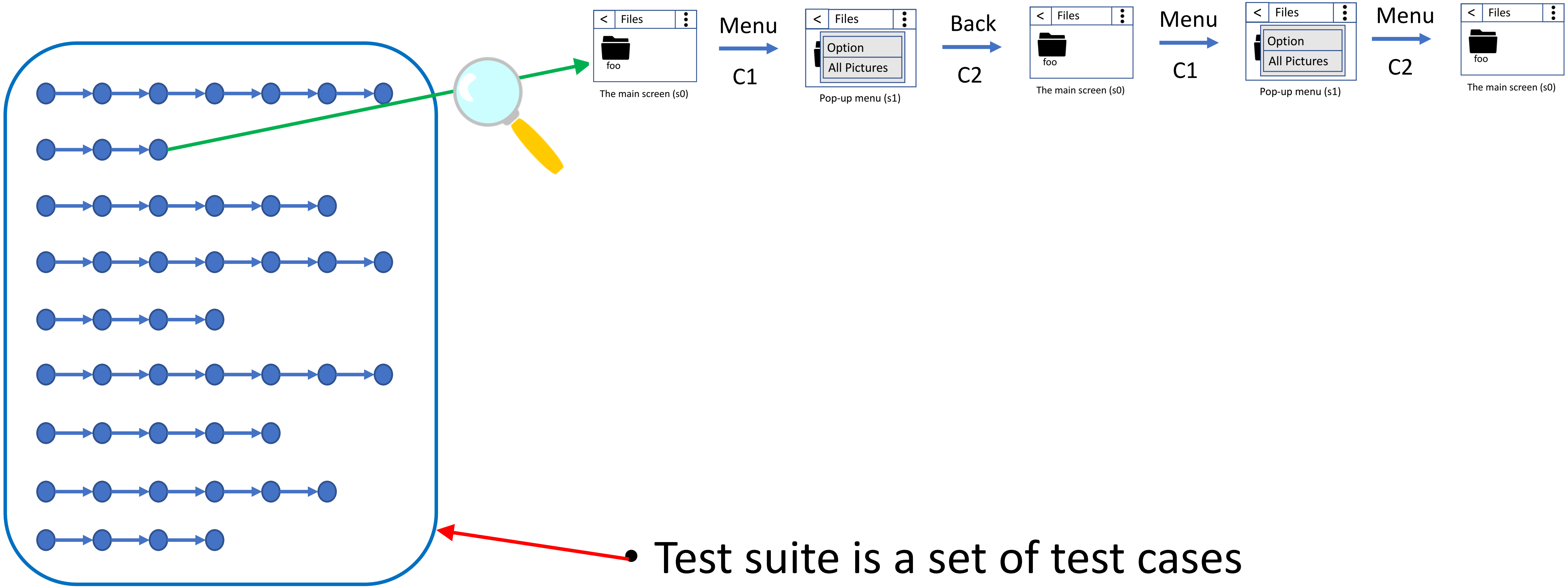


A partial model of a file browser app

A test case

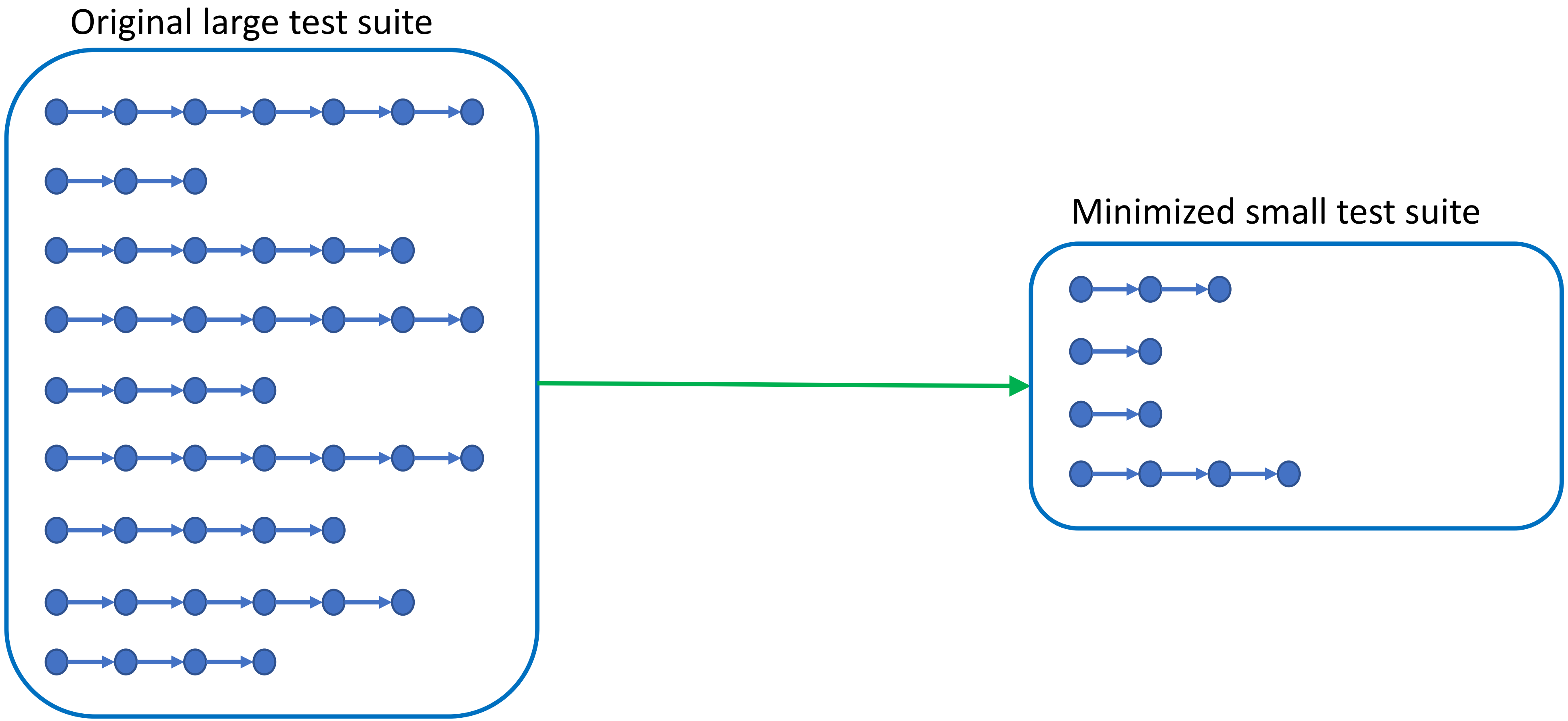


Test suite and test cases



- Test suite is a set of test cases

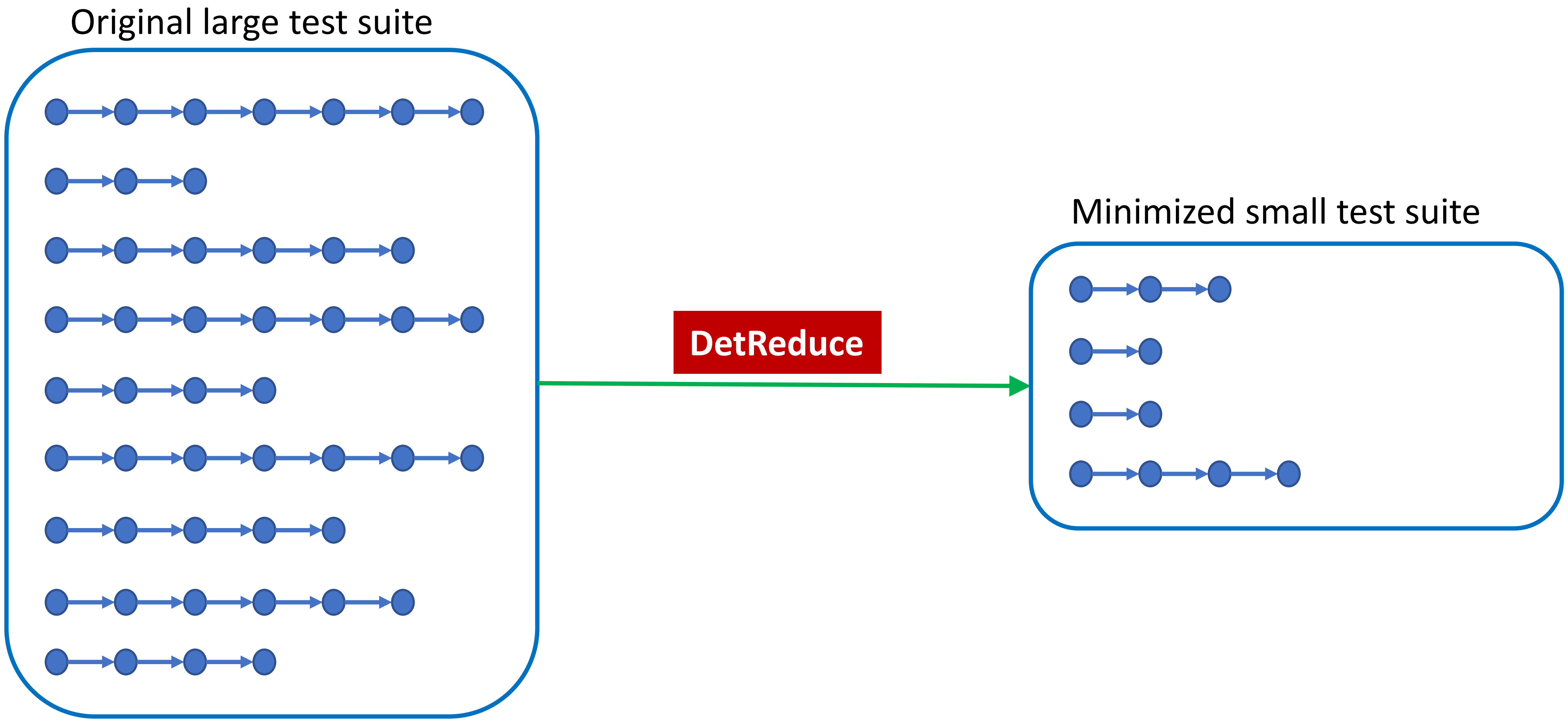
Our goal



Original large test suite

Minimized small test suite

Our goal



Original large test suite
DetReduce
Minimized small test suite

How to Minimize? Existing work

- Optimal reduction => NP hard
- Delta debugging based minimization [Clapp et al.]
  - Creates a lot of intermediate test cases
  - Expensive to test feasibility of each each intermediate test case
  - Few hours to minimize a test case with 500 transitions
  - 10,000 transitions will take months
- Problem: Creates intermediate test cases by removing transitions
  - Expensive to check feasibility of each test case
- Our goal: develop a technique that can run within a day

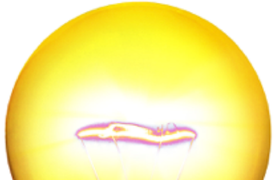
DetReduce: Idea

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



DetReduce: Idea 1

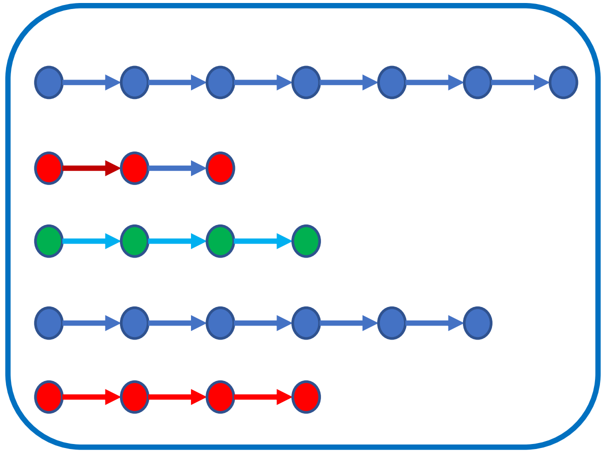
- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces

Original set of test cases



DetReduce: Idea 1

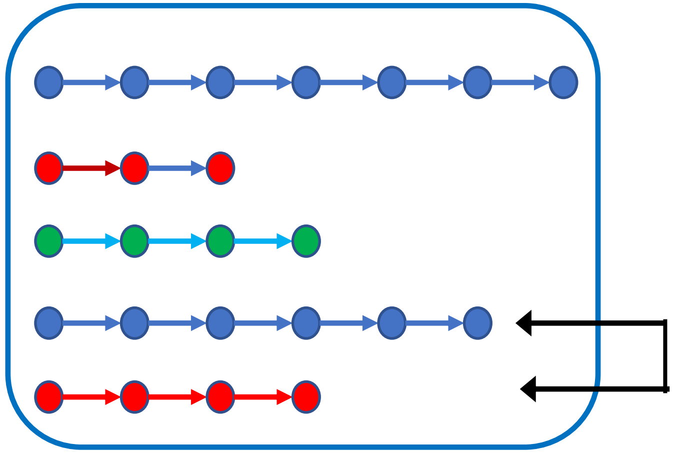
- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces

Original set of test cases



Redundant traces

Same color = Same test coverage

Only need to keep one of them

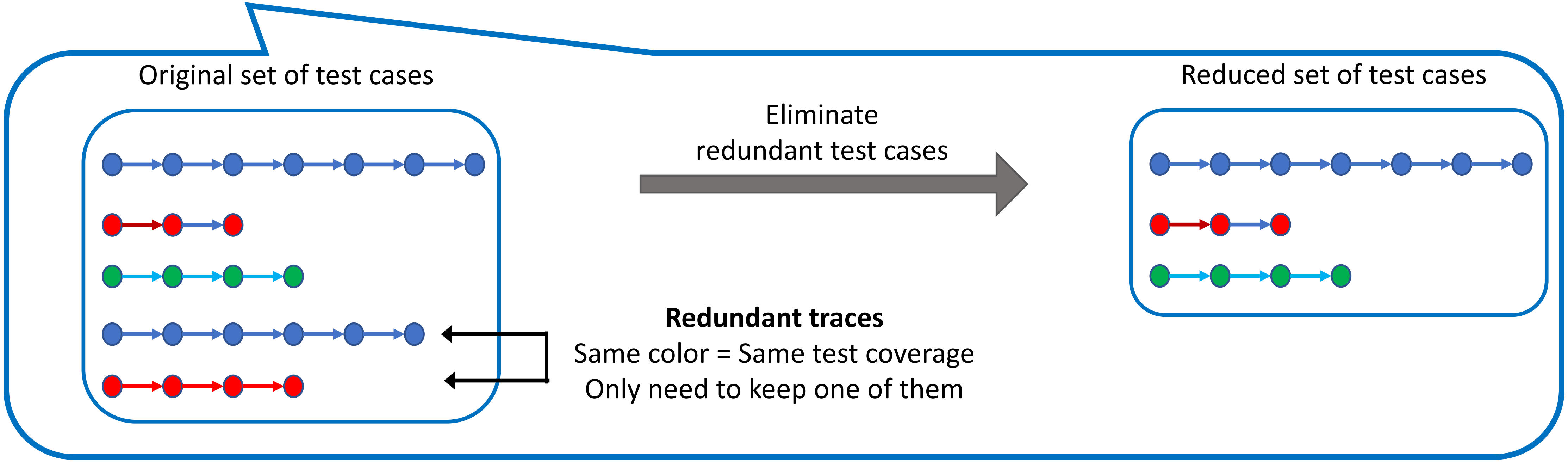
DetReduce: Idea 1

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Original set of test cases

Eliminate redundant test cases

Reduced set of test cases

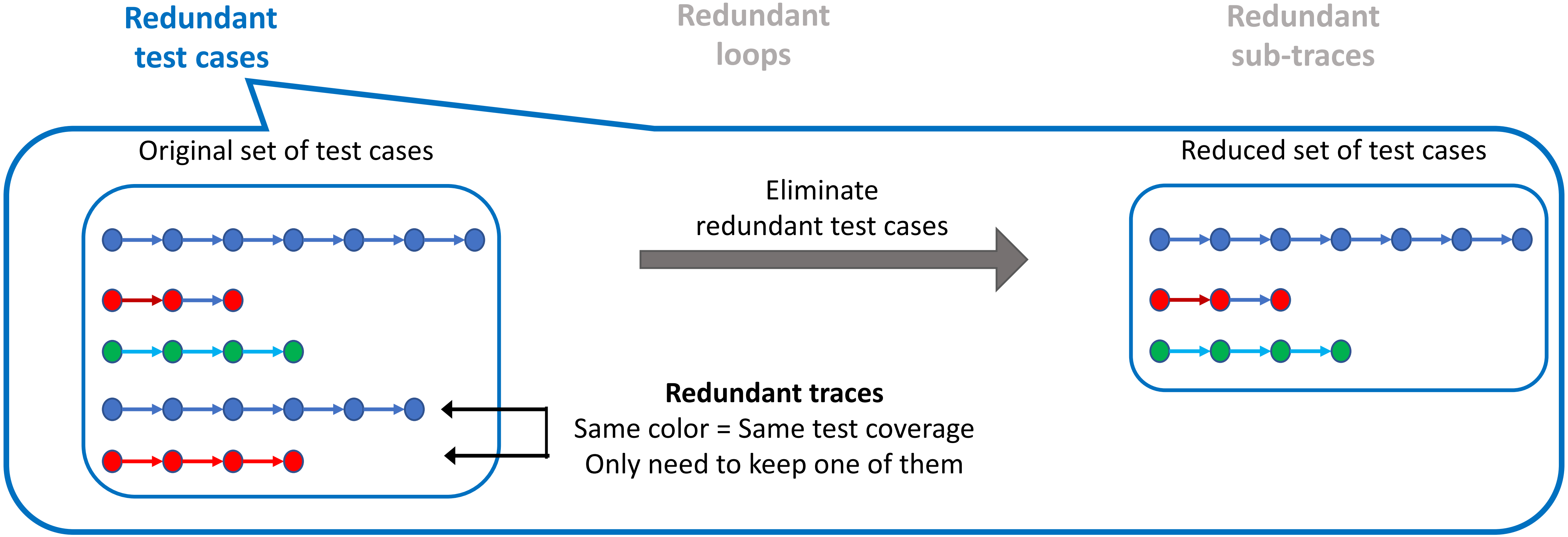
Redundant traces

Same color = Same test coverage

Only need to keep one of them

DetReduce: Idea 1

- Key observation: 3-types of common redundancies


Redundant test cases
Redundant loops
Redundant sub-traces
Original set of test cases
Eliminate redundant test cases
Reduced set of test cases

Redundant traces

Same color = Same test coverage

Only need to keep one of them

Removing redundant traces = Minimal vertex cover problem (NP-hard)

Solved using a greedy selection algorithm (no feasibility check is necessary)

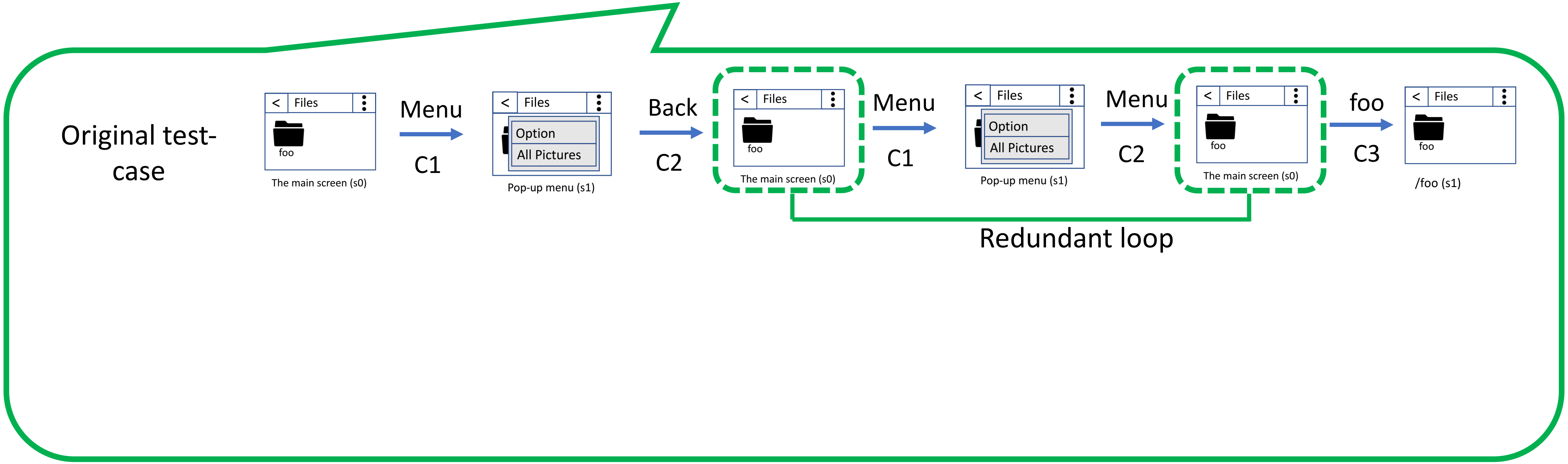
DetReduce: Idea 2

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Loop = sub-trace starts and ends with the same screen

A loop is redundant if it can be removed without affecting the coverage of the trace

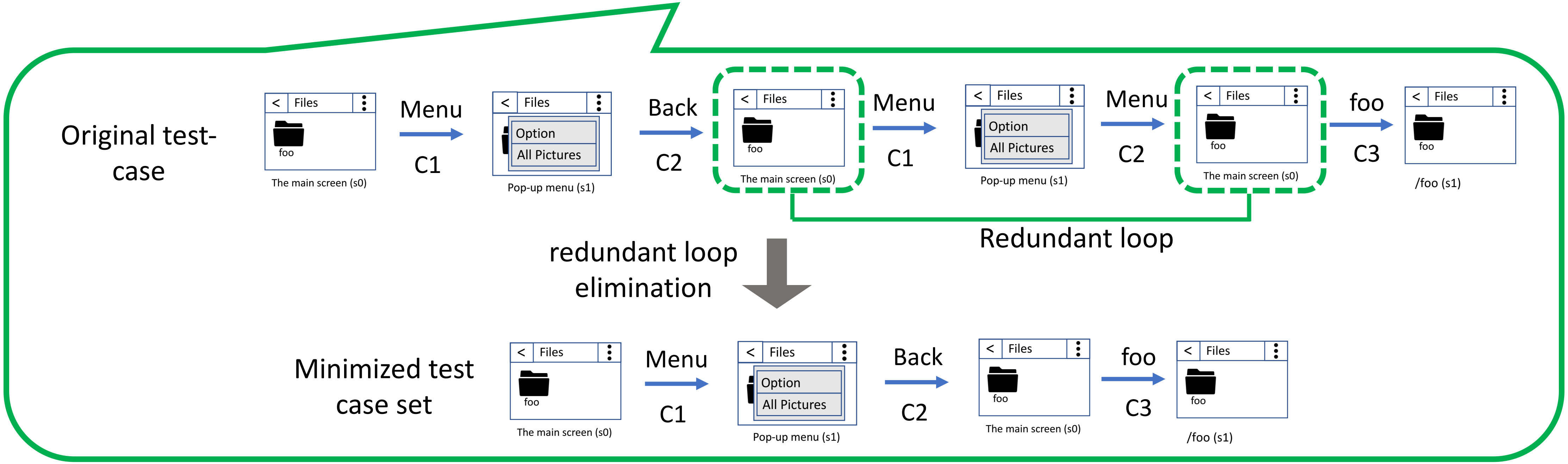
DetReduce: Idea 2

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Remove redundant loops using an exhaustive search (# of loops per test case is small)
Learns infeasible prefixes to reduce search space.

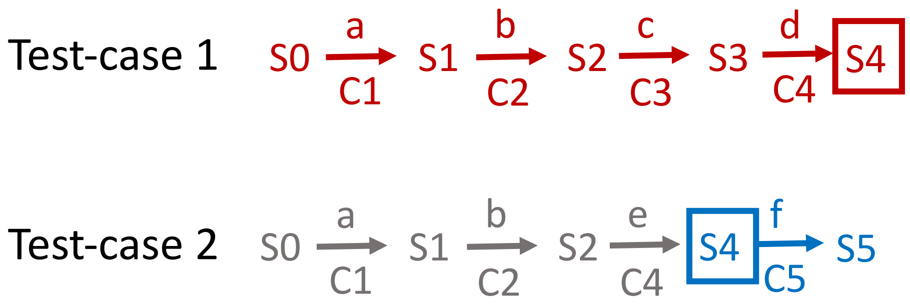
DetReduce: Idea 3

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Test-case 1 $S0 \xrightarrow[C1]{a} S1 \xrightarrow[C2]{b} S2 \xrightarrow[C3]{c} S3 \xrightarrow[C4]{d} \boxed{S4}$

Test-case 2 $S0 \xrightarrow[C1]{a} S1 \xrightarrow[C2]{b} S2 \xrightarrow[C4]{e} \boxed{S4} \xrightarrow[C5]{f} S5$

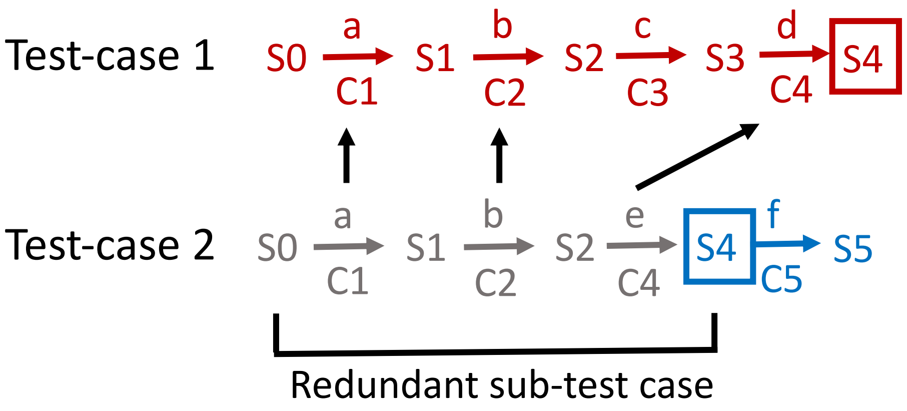
DetReduce: Idea 3

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Test-case 1 S0 →a/C1 S1 →b/C2 S2 →c/C3 S3 →d/C4 S4

Test-case 2 S0 →a/C1 S1 →b/C2 S2 →e/C4 S4 →f/C5 S5

Redundant sub-test case

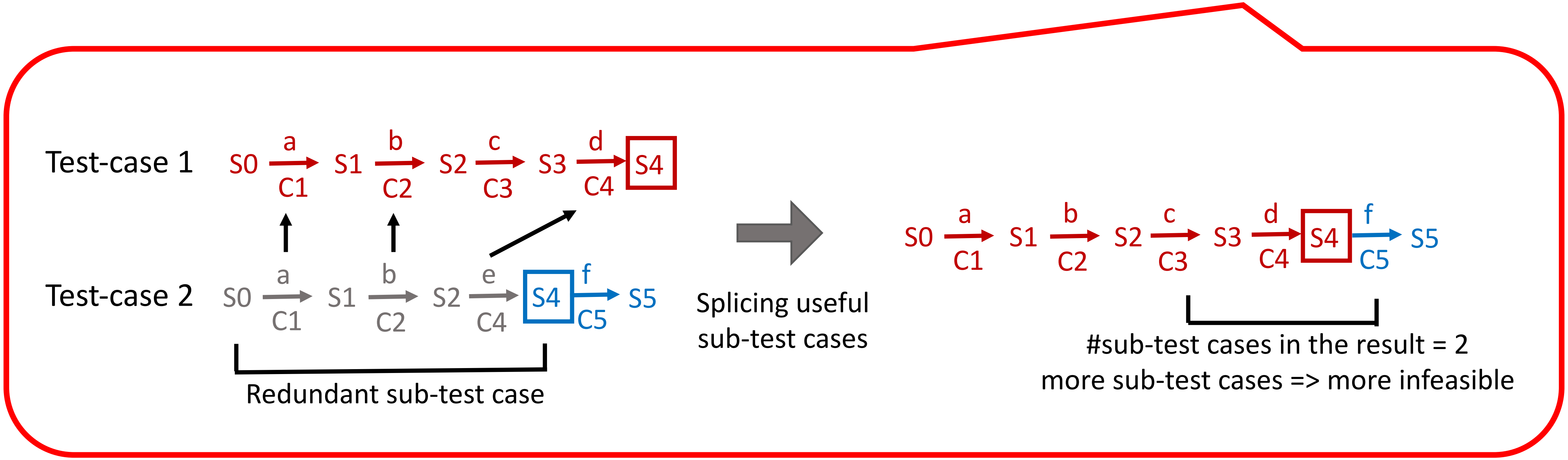
DetReduce: Idea 3

- Key observation: 3-types of common redundancies

Redundant test cases

Redundant loops

Redundant sub-traces



Test-case 1 $S0 \xrightarrow[C1]{a} S1 \xrightarrow[C2]{b} S2 \xrightarrow[C3]{c} S3 \xrightarrow[C4]{d} \boxed{S4}$

Test-case 2 $S0 \xrightarrow[C1]{a} S1 \xrightarrow[C2]{b} S2 \xrightarrow[C4]{e} \boxed{S4} \xrightarrow[C5]{f} S5$

Redundant sub-test case

Splicing useful sub-test cases

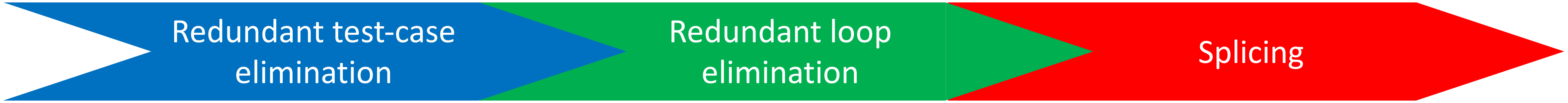
$S0 \xrightarrow[C1]{a} S1 \xrightarrow[C2]{b} S2 \xrightarrow[C3]{c} S3 \xrightarrow[C4]{d} \boxed{S4} \xrightarrow[C5]{f} S5$

#sub-test cases in the result = 2
more sub-test cases => more infeasible

Optimal splicing = TSP problem (NP-hard)
Solved using a greedy search with **a bound on # of sub-test cases per resulting trace (N=3)**

DetReduce: Summary

- Combination of 3 heuristics
  - Successive reduction steps
  - Applies cheaper reductions first



Granularity	Optimizes a test-suite by removing test-cases	Optimizes each test-case by removing loops	Optimizes a test-suite by splicing sub-test cases
Feasibility check	Not required	Required	Required
Cost	Almost free (greedy + no feasibility check)	Linear in # of test cases	O(#sub-test cases ^ bound)

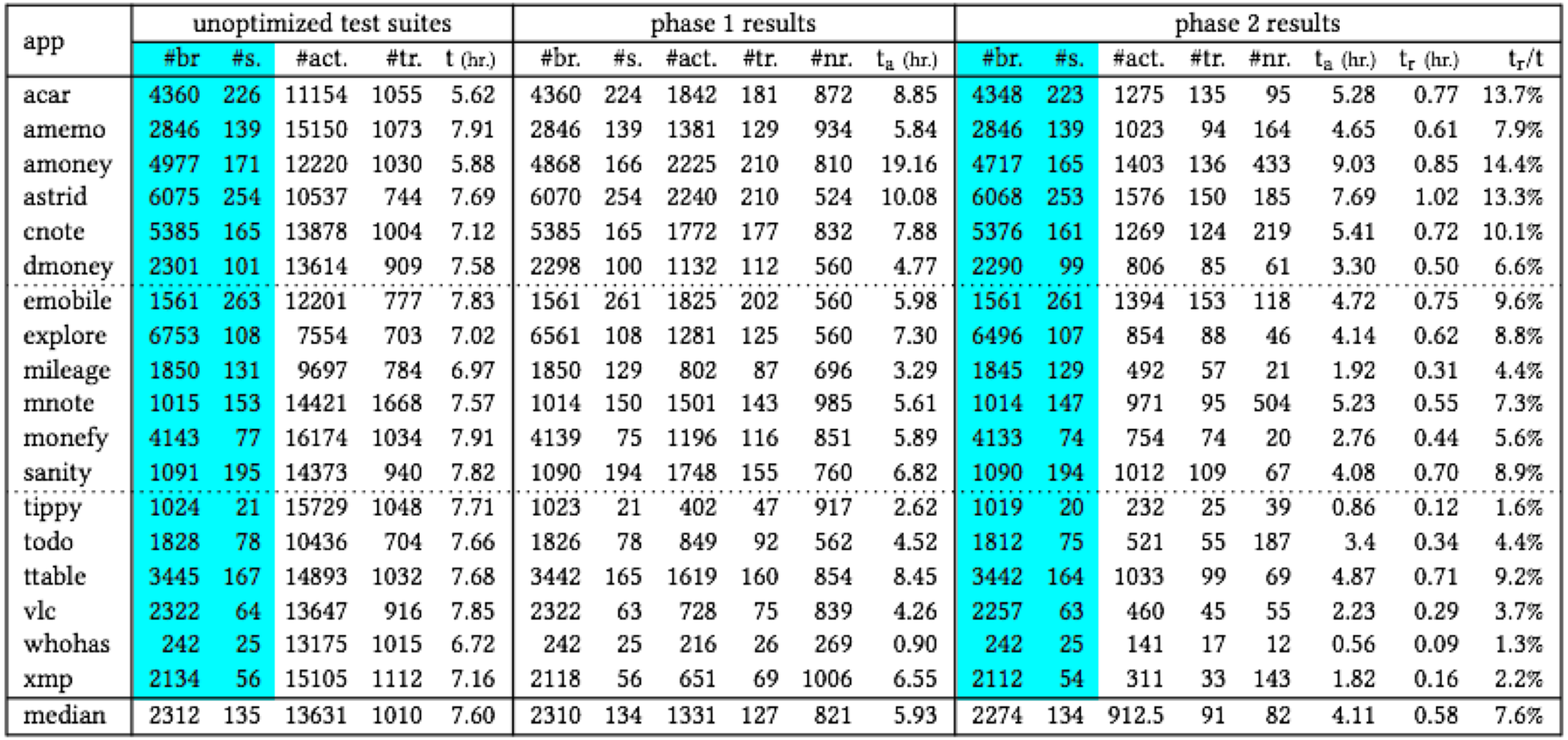
Evaluation

- Implementation
  - **Front-end**: instruments an app to get runtime info
  - **Back-end**: guides testing using runtime info

https://github.com/wtchoi/swifthand2

- Setup
  - 18 Android Apps
  - on real devices
  - Run **SwiftHand**/**Random** for 8 hours
    => Remove non-deterministic test cases
    => Run **DetReduce**

Evaluation: Retained Coverage

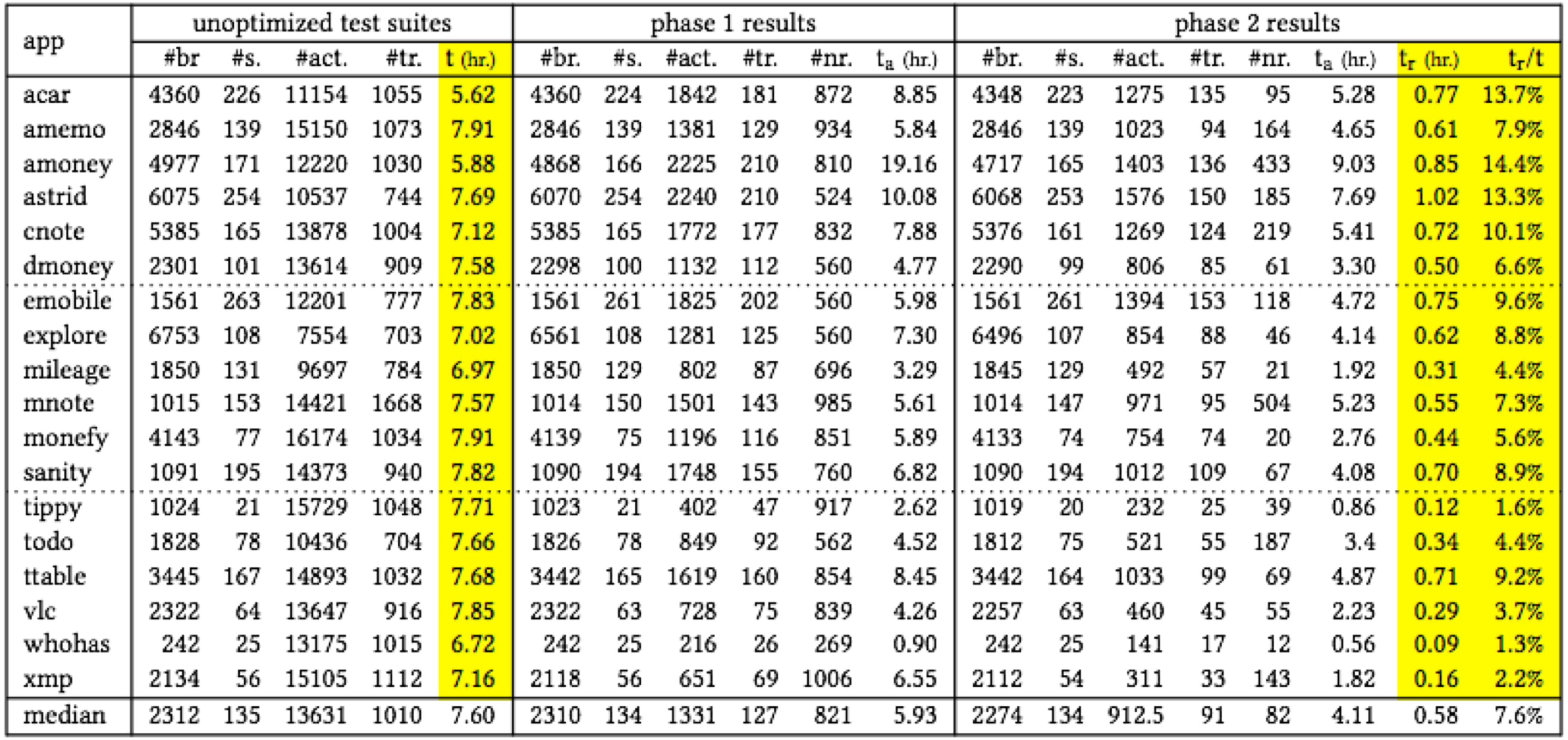


app	unoptimized test suites					phase 1 results						phase 2 results							
	#br	#s.	#act.	#tr.	t (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	t_r (hr.)	t_r/t
acar	4360	226	11154	1055	5.62	4360	224	1842	181	872	8.85	4348	223	1275	135	95	5.28	0.77	13.7%
amemo	2846	139	15150	1073	7.91	2846	139	1381	129	934	5.84	2846	139	1023	94	164	4.65	0.61	7.9%
amoney	4977	171	12220	1030	5.88	4868	166	2225	210	810	19.16	4717	165	1403	136	433	9.03	0.85	14.4%
astrid	6075	254	10537	744	7.69	6070	254	2240	210	524	10.08	6068	253	1576	150	185	7.69	1.02	13.3%
cnote	5385	165	13878	1004	7.12	5385	165	1772	177	832	7.88	5376	161	1269	124	219	5.41	0.72	10.1%
dmoney	2301	101	13614	909	7.58	2298	100	1132	112	560	4.77	2290	99	806	85	61	3.30	0.50	6.6%
emobile	1561	263	12201	777	7.83	1561	261	1825	202	560	5.98	1561	261	1394	153	118	4.72	0.75	9.6%
explore	6753	108	7554	703	7.02	6561	108	1281	125	560	7.30	6496	107	854	88	46	4.14	0.62	8.8%
mileage	1850	131	9697	784	6.97	1850	129	802	87	696	3.29	1845	129	492	57	21	1.92	0.31	4.4%
mnote	1015	153	14421	1668	7.57	1014	150	1501	143	985	5.61	1014	147	971	95	504	5.23	0.55	7.3%
monefy	4143	77	16174	1034	7.91	4139	75	1196	116	851	5.89	4133	74	754	74	20	2.76	0.44	5.6%
sanity	1091	195	14373	940	7.82	1090	194	1748	155	760	6.82	1090	194	1012	109	67	4.08	0.70	8.9%
tippy	1024	21	15729	1048	7.71	1023	21	402	47	917	2.62	1019	20	232	25	39	0.86	0.12	1.6%
todo	1828	78	10436	704	7.66	1826	78	849	92	562	4.52	1812	75	521	55	187	3.4	0.34	4.4%
ttable	3445	167	14893	1032	7.68	3442	165	1619	160	854	8.45	3442	164	1033	99	69	4.87	0.71	9.2%
vlc	2322	64	13647	916	7.85	2322	63	728	75	839	4.26	2257	63	460	45	55	2.23	0.29	3.7%
whohas	242	25	13175	1015	6.72	242	25	216	26	269	0.90	242	25	141	17	12	0.56	0.09	1.3%
xmp	2134	56	15105	1112	7.16	2118	56	651	69	1006	6.55	2112	54	311	33	143	1.82	0.16	2.2%
median	2312	135	13631	1010	7.60	2310	134	1331	127	821	5.93	2274	134	912.5	91	82	4.11	0.58	7.6%

Table 2. Test reduction result using DetReduce

- Retains 99.8% of branches / 98.3% of screens.
- Inherent non-determinism of apps => prevents 100% coverage

Evaluation: Test Running Time Reduction

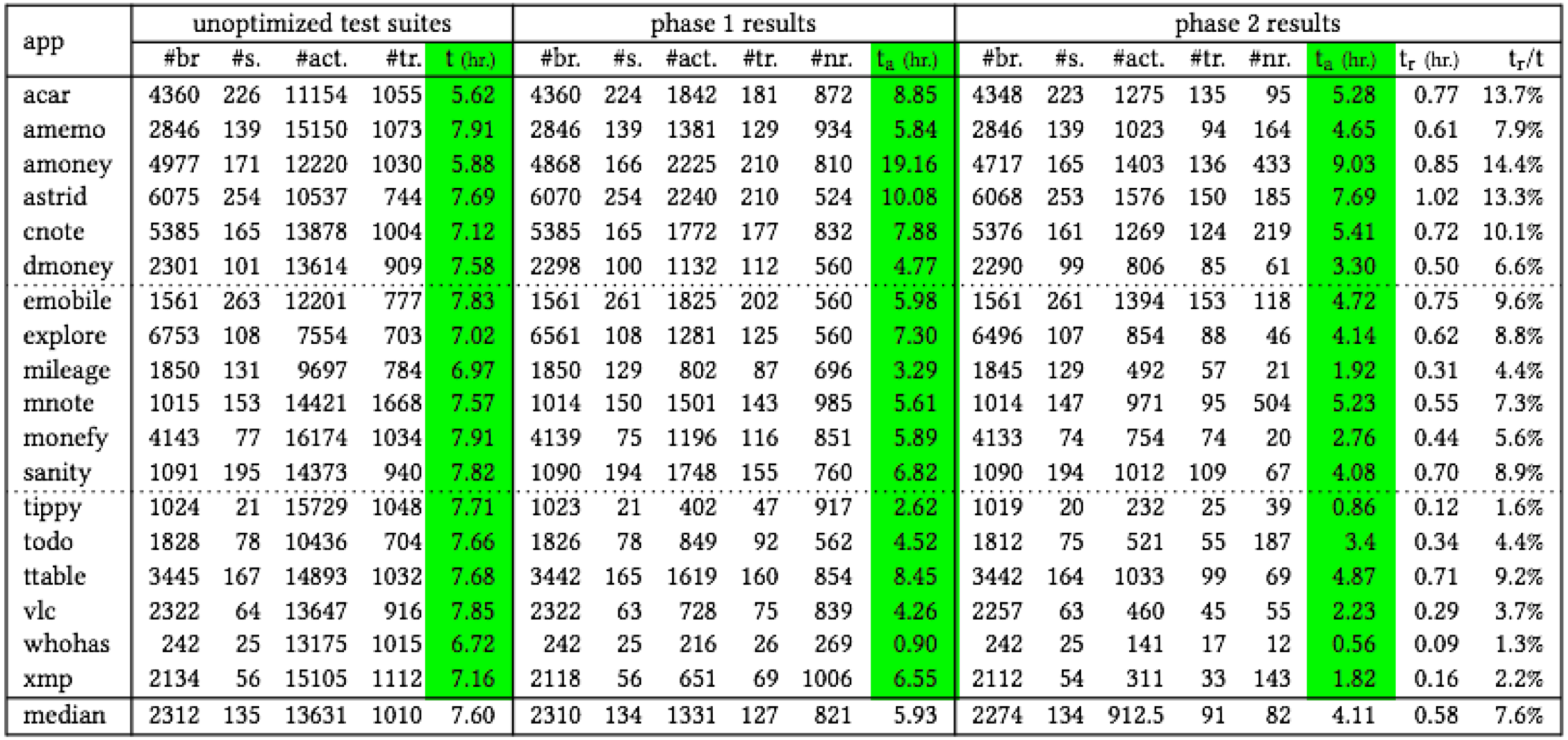


app	unoptimized test suites					phase 1 results						phase 2 results							
	#br	#s.	#act.	#tr.	t (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	t_r (hr.)	t_r/t
acar	4360	226	11154	1055	5.62	4360	224	1842	181	872	8.85	4348	223	1275	135	95	5.28	0.77	13.7%
amemo	2846	139	15150	1073	7.91	2846	139	1381	129	934	5.84	2846	139	1023	94	164	4.65	0.61	7.9%
amoney	4977	171	12220	1030	5.88	4868	166	2225	210	810	19.16	4717	165	1403	136	433	9.03	0.85	14.4%
astrid	6075	254	10537	744	7.69	6070	254	2240	210	524	10.08	6068	253	1576	150	185	7.69	1.02	13.3%
cnote	5385	165	13878	1004	7.12	5385	165	1772	177	832	7.88	5376	161	1269	124	219	5.41	0.72	10.1%
dmoney	2301	101	13614	909	7.58	2298	100	1132	112	560	4.77	2290	99	806	85	61	3.30	0.50	6.6%
emobile	1561	263	12201	777	7.83	1561	261	1825	202	560	5.98	1561	261	1394	153	118	4.72	0.75	9.6%
explore	6753	108	7554	703	7.02	6561	108	1281	125	560	7.30	6496	107	854	88	46	4.14	0.62	8.8%
mileage	1850	131	9697	784	6.97	1850	129	802	87	696	3.29	1845	129	492	57	21	1.92	0.31	4.4%
mnote	1015	153	14421	1668	7.57	1014	150	1501	143	985	5.61	1014	147	971	95	504	5.23	0.55	7.3%
monefy	4143	77	16174	1034	7.91	4139	75	1196	116	851	5.89	4133	74	754	74	20	2.76	0.44	5.6%
sanity	1091	195	14373	940	7.82	1090	194	1748	155	760	6.82	1090	194	1012	109	67	4.08	0.70	8.9%
tippy	1024	21	15729	1048	7.71	1023	21	402	47	917	2.62	1019	20	232	25	39	0.86	0.12	1.6%
todo	1828	78	10436	704	7.66	1826	78	849	92	562	4.52	1812	75	521	55	187	3.4	0.34	4.4%
ttable	3445	167	14893	1032	7.68	3442	165	1619	160	854	8.45	3442	164	1033	99	69	4.87	0.71	9.2%
vlc	2322	64	13647	916	7.85	2322	63	728	75	839	4.26	2257	63	460	45	55	2.23	0.29	3.7%
whohas	242	25	13175	1015	6.72	242	25	216	26	269	0.90	242	25	141	17	12	0.56	0.09	1.3%
xmp	2134	56	15105	1112	7.16	2118	56	651	69	1006	6.55	2112	54	311	33	143	1.82	0.16	2.2%
median	2312	135	13631	1010	7.60	2310	134	1331	127	821	5.93	2274	134	912.5	91	82	4.11	0.58	7.6%

Table 2. Test reduction result using DetReduce

- Running time is reduced by factor of 13.2x (on average)

Evaluation: Minimization Cost



app	unoptimized test suites					phase 1 results						phase 2 results							
	#br	#s.	#act.	#tr.	t (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	#br.	#s.	#act.	#tr.	#nr.	t_a (hr.)	t_r (hr.)	t_r/t
acar	4360	226	11154	1055	5.62	4360	224	1842	181	872	8.85	4348	223	1275	135	95	5.28	0.77	13.7%
amemo	2846	139	15150	1073	7.91	2846	139	1381	129	934	5.84	2846	139	1023	94	164	4.65	0.61	7.9%
amoney	4977	171	12220	1030	5.88	4868	166	2225	210	810	19.16	4717	165	1403	136	433	9.03	0.85	14.4%
astrid	6075	254	10537	744	7.69	6070	254	2240	210	524	10.08	6068	253	1576	150	185	7.69	1.02	13.3%
cnote	5385	165	13878	1004	7.12	5385	165	1772	177	832	7.88	5376	161	1269	124	219	5.41	0.72	10.1%
dmoney	2301	101	13614	909	7.58	2298	100	1132	112	560	4.77	2290	99	806	85	61	3.30	0.50	6.6%
emobile	1561	263	12201	777	7.83	1561	261	1825	202	560	5.98	1561	261	1394	153	118	4.72	0.75	9.6%
explore	6753	108	7554	703	7.02	6561	108	1281	125	560	7.30	6496	107	854	88	46	4.14	0.62	8.8%
mileage	1850	131	9697	784	6.97	1850	129	802	87	696	3.29	1845	129	492	57	21	1.92	0.31	4.4%
mnote	1015	153	14421	1668	7.57	1014	150	1501	143	985	5.61	1014	147	971	95	504	5.23	0.55	7.3%
monefy	4143	77	16174	1034	7.91	4139	75	1196	116	851	5.89	4133	74	754	74	20	2.76	0.44	5.6%
sanity	1091	195	14373	940	7.82	1090	194	1748	155	760	6.82	1090	194	1012	109	67	4.08	0.70	8.9%
tippy	1024	21	15729	1048	7.71	1023	21	402	47	917	2.62	1019	20	232	25	39	0.86	0.12	1.6%
todo	1828	78	10436	704	7.66	1826	78	849	92	562	4.52	1812	75	521	55	187	3.4	0.34	4.4%
ttable	3445	167	14893	1032	7.68	3442	165	1619	160	854	8.45	3442	164	1033	99	69	4.87	0.71	9.2%
vlc	2322	64	13647	916	7.85	2322	63	728	75	839	4.26	2257	63	460	45	55	2.23	0.29	3.7%
whohas	242	25	13175	1015	6.72	242	25	216	26	269	0.90	242	25	141	17	12	0.56	0.09	1.3%
xmp	2134	56	15105	1112	7.16	2118	56	651	69	1006	6.55	2112	54	311	33	143	1.82	0.16	2.2%
median	2312	135	13631	1010	7.60	2310	134	1331	127	821	5.93	2274	134	912.5	91	82	4.11	0.58	7.6%

Table 2. Test reduction result using DetReduce

- Minimization time < 6x of the input test suite's running time

Summary

- Minimizing an automatically generated test suite is challenging
  - NP hard problem => need heuristic
  - Feasibility check => each reduction attempt is expensive

- Automatically generated GUI test suites can be minimized
  - Problem specific heuristic is key to scalability
  - **DetReduce** focuses on 3-common types of redundancies in GUI test suites

https://github.com/wtchoi/swifthand2