Scalable Statistical Bug Isolation

- Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, Michael I. Jordan

Tao Xia

Oct 12, 2006

Algorithm

- A debugging algorithm to locate the cause of a failure (bug)
- Identifies the most important bug with most failures
- Suggest the area of the bug, but not the exact location

Random Sampling

- Goal
  - Keep performance overhead low
  - Limit storage and transmission costs
- Sampling rate: 1/100
- Each sample is independent from each other

Instrumentation Site

- Any collection of statements within a program
- Three instrumentation schema for C:
  - – Branches (true/false)
  - – Returns (<0, <=0, >0, >=0, =0, !=0)
  - – Scalar-pairs (assignment) (<, <=, >, >=, =, !=)

Definitions

- Failure

$$Failure(P) = \frac{F(P)}{S(P)+F(P)}$$

- Context

$$Context(P) = \frac{F(P\ \text{observed})}{S(P\ \text{observed})+F(P\ \text{observed})}$$

- Increase

$$Increase(P) \equiv Failure(P) - Context(P)$$

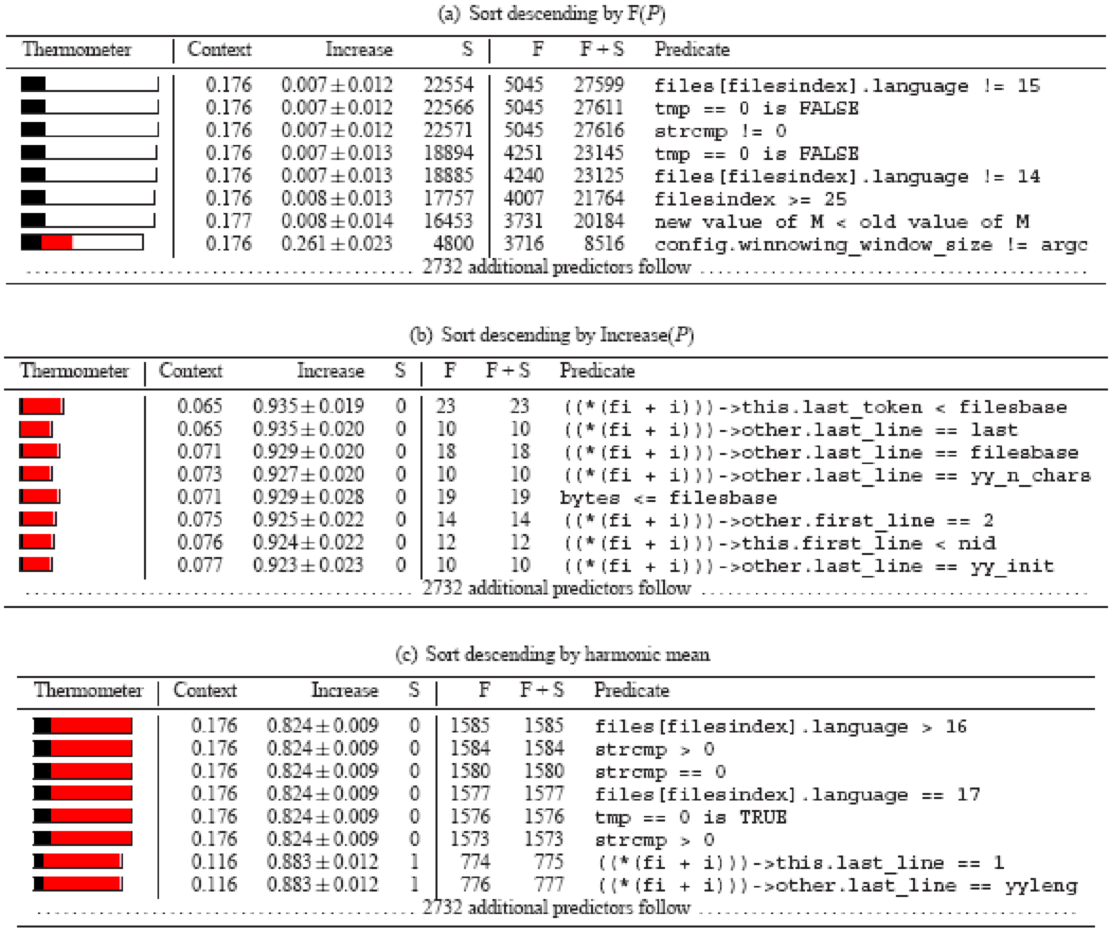
  - Predicate P is a bug predictor
  - F(P): number of failure runs in which P is obesrved to be true
  - S(P): number of success runs in which P is obesrved to be true

Discarded Data

- if Failure(P) = 0, then P has no predict power
- If Increase(P) <=0, then P has no predict power
- Redundancy elimination
  - Importance

$$Importance(P) = \frac{2}{\frac{1}{Increase(P)} + \frac{1}{log(F(P))/log(NumF)}}$$

Predicates stats with out redundancv elimination



(a) Sort descending by F(P)

Thermometer	Context	Increase	S	F	F + S	Predicate
	0.176	0.007 ± 0.012	22554	5045	27599	`files[filesindex].language != 15`
	0.176	0.007 ± 0.012	22566	5045	27611	`tmp == 0 is FALSE`
	0.176	0.007 ± 0.012	22571	5045	27616	`strcmp != 0`
	0.176	0.007 ± 0.013	18894	4251	23145	`tmp == 0 is FALSE`
	0.176	0.007 ± 0.013	18885	4240	23125	`files[filesindex].language != 14`
	0.176	0.008 ± 0.013	17757	4007	21764	`filesindex >= 25`
	0.177	0.008 ± 0.014	16453	3731	20184	`new value of M < old value of M`
	0.176	0.261 ± 0.023	4800	3716	8516	`config.winnowing_window_size != argc`

. . . . 2732 additional predictors follow . . . .

(b) Sort descending by Increase(P)

Thermometer	Context	Increase	S	F	F + S	Predicate
	0.065	0.935 ± 0.019	0	23	23	`((*(fi + i)))->this.last_token < filesbase`
	0.065	0.935 ± 0.020	0	10	10	`((*(fi + i)))->other.last_line == last`
	0.071	0.929 ± 0.020	0	18	18	`((*(fi + i)))->other.last_line == filesbase`
	0.073	0.927 ± 0.020	0	10	10	`((*(fi + i)))->other.last_line == yy_n_chars`
	0.071	0.929 ± 0.028	0	19	19	`bytes <= filesbase`
	0.075	0.925 ± 0.022	0	14	14	`((*(fi + i)))->other.first_line == 2`
	0.076	0.924 ± 0.022	0	12	12	`((*(fi + i)))->this.first_line < nid`
	0.077	0.923 ± 0.023	0	10	10	`((*(fi + i)))->other.last_line == yy_init`

. . . . 2732 additional predictors follow . . . .

(c) Sort descending by harmonic mean

Thermometer	Context	Increase	S	F	F + S	Predicate
	0.176	0.824 ± 0.009	0	1585	1585	`files[filesindex].language > 16`
	0.176	0.824 ± 0.009	0	1584	1584	`strcmp > 0`
	0.176	0.824 ± 0.009	0	1580	1580	`strcmp == 0`
	0.176	0.824 ± 0.009	0	1577	1577	`files[filesindex].language == 17`
	0.176	0.824 ± 0.009	0	1576	1576	`tmp == 0 is TRUE`
	0.176	0.824 ± 0.009	0	1573	1573	`strcmp > 0`
	0.116	0.883 ± 0.012	1	774	775	`((*(fi + i)))->this.last_line == 1`
	0.116	0.883 ± 0.012	1	776	777	`((*(fi + i)))->other.last_line == yyleng`

. . . . 2732 additional predictors follow . . . .

- **Black**: Context(P) **Red**: Increase(P) **Pink**: Confidence Interval **White**: S(P)

Experiment

- Five case studies
- About 32000 random inputs

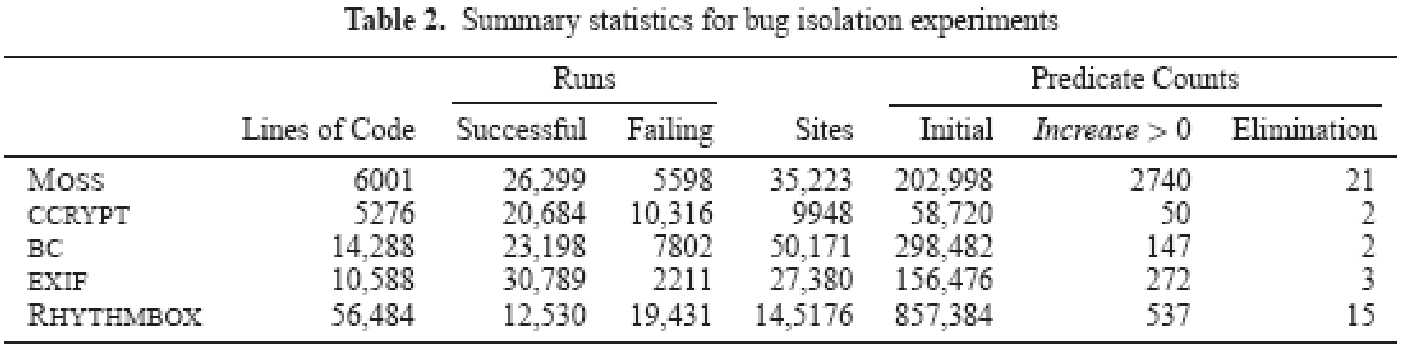
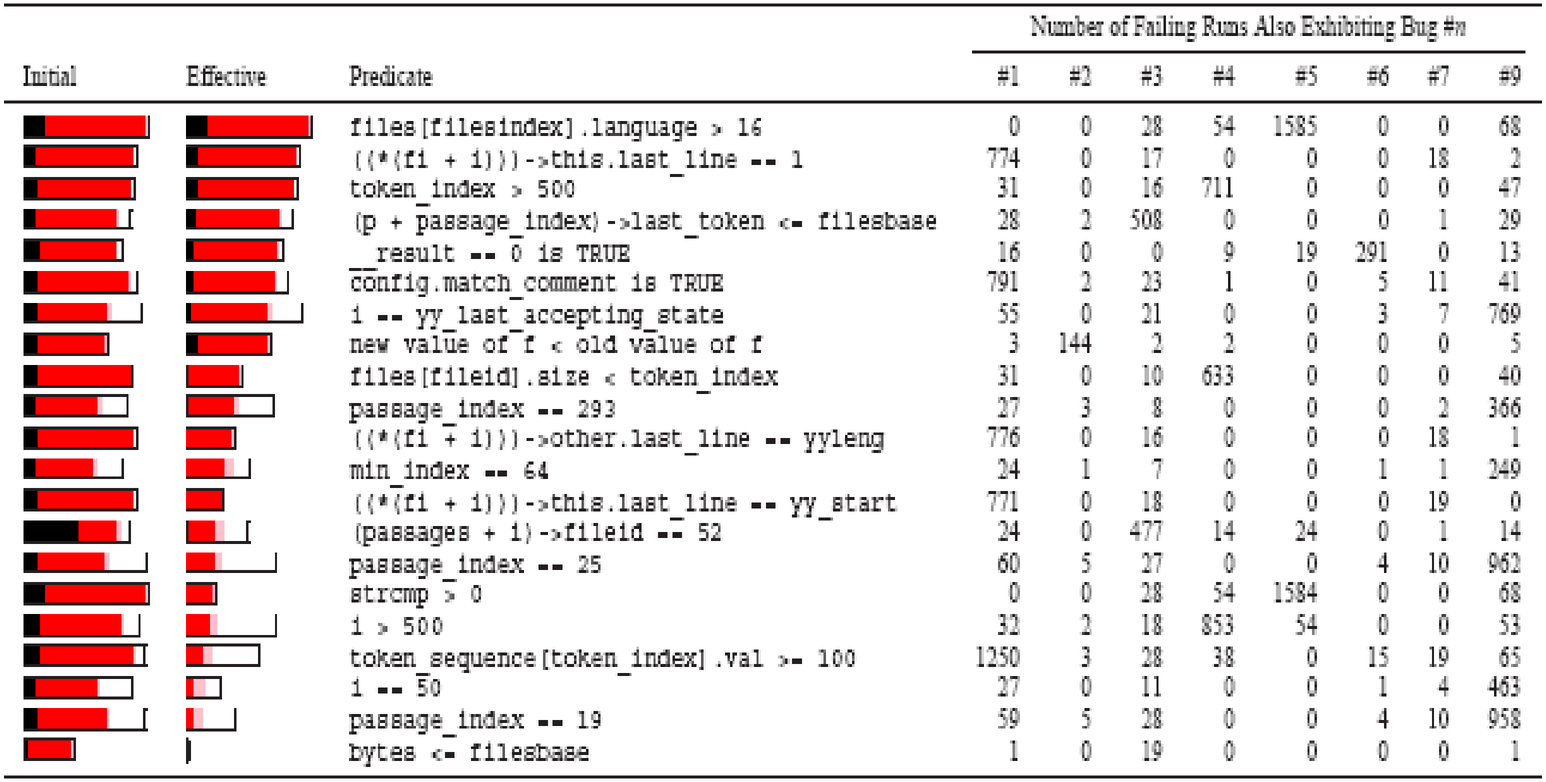


Table 2. Summary statistics for bug isolation experiments

		Runs			Predicate Counts		
	Lines of Code	Successful	Failing	Sites	Initial	*Increase* > 0	Elimination
Moss	6001	26,299	5598	35,223	202,998	2740	21
CCRYPT	5276	20,684	10,316	9948	58,720	50	2
BC	14,288	23,198	7802	50,171	298,482	147	2
EXIF	10,588	30,789	2211	27,380	156,476	272	3
Rhythmbox	56,484	12,530	19,431	14,5176	857,384	537	15

Experiment Results

Table 3. Moss failure predictors using nonuniform sampling



			Number of Failing Runs Also Exhibiting Bug #n							
Initial	Effective	Predicate	#1	#2	#3	#4	#5	#6	#7	#9
		files[filesindex].language > 16	0	0	28	54	1585	0	0	68
		((*(fi + i)))->this.last_line == 1	774	0	17	0	0	0	18	2
		token_index > 500	31	0	16	711	0	0	0	47
		(p + passage_index)->last_token <= filesbase	28	2	508	0	0	0	1	29
		__result == 0 is TRUE	16	0	0	9	19	291	0	13
		config.match_comment is TRUE	791	2	23	1	0	5	11	41
		i == yy_last_accepting_state	55	0	21	0	0	3	7	769
		new value of f < old value of f	3	144	2	2	0	0	0	5
		files[fileid].size < token_index	31	0	10	633	0	0	0	40
		passage_index == 293	27	3	8	0	0	0	2	366
		((*(fi + i)))->other.last_line == yyleng	776	0	16	0	0	0	18	1
		min_index == 64	24	1	7	0	0	1	1	249
		((*(fi + i)))->this.last_line == yy_start	771	0	18	0	0	0	19	0
		(passages + i)->fileid == 52	24	0	477	14	24	0	1	14
		passage_index == 25	60	5	27	0	0	4	10	962
		strcmp > 0	0	0	28	54	1584	0	0	68
		i > 500	32	2	18	853	54	0	0	53
		token_sequence[token_index].val >= 100	1250	3	28	38	0	15	19	65
		i == 50	27	0	11	0	0	1	4	463
		passage_index == 19	59	5	28	0	0	4	10	958
		bytes <= filesbase	1	0	19	0	0	0	0	1

- **Black**: Context(P) **Red**: Increase(P) **Pink**: Confidence Interval **White**: S(P)

Feedback

- Positive:
  - look at bugs from a different perspective
  - Attack bugs from the statistic experimentation

- Negative
  - didn’t show how accurate the algorithm is (precision and recall)
  - How useful is it?