APPENDIX

A BINDIFF SIMILARITY SCORE

BinDiff's similarity score of two binary code, ranging from 0.0 to 1.0 (a higher score indicates more similar), is a weighted sum of the following five metrics:

(1) 35%: the proportion of matched control flow graph edges;
(2) 25%: the proportion of matched basic blocks;
(3) 20%: the difference in call graph topological order;
(4) 10%: the proportion of matched functions;
(5) 10%: the proportion of matched instructions;

The similarity score for a function-pair comparison has similar factors (1, 2, 3, and 5), and the major difference is that CFG topological order takes up 50% weight.

B METAHEURISTIC SEARCH: GENETIC ALGORITHM

As illustrated in Figure 8, genetic algorithm (GA) encodes compiler optimization flags as a chromosome-like data structure, and then it applies crossover and mutation operations to these structures to simulate the evolution process under natural selection pressure. After the initial population and the selection of fitness function, pairs of parents go through the crossover to reproduce new individuals. These individuals will further mutate their genes to produce diversified populations. In addition to the traditional procedure, we add a constraint solver to verify the correctness of an optimization sequence and define a new fitness function to evaluate how individuals adapt to our needs. Following this style, this generational evolution is repeated until one of the termination criteria is met.

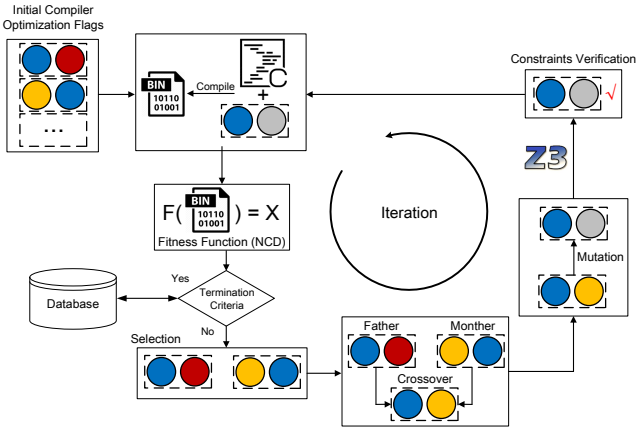

Figure 8: BinTuner's genetic algorithm iteration: the automatic selection of compiler optimization options towards maximizing the effect of binary code difference.

C CORRELATION BETWEEN NCD AND BINDIFF

We calculated Pearson correlation values between NCD scores and BinDiff difference values for two relatively small SPEC benchmark programs: 462.libquantum & 429.mcf. The reason for selecting these two programs is we can terminate iterative compilation within a reasonable time. In statistics, when the Pearson correlation coefficient is greater than 0.4, it means there is a correlation between the two groups of data. A significant positive correlation happens when the coefficient is greater than 0.6. Figure 9 shows that the experimental results have about 70% significant positive correlations between NCD scores and BinDiff difference values. Taking NCD as fitness function, BinTuner completes two experiments in *42 minutes*; while BinTuner takes as much as *15.6 hours* to terminate if we use BinDiff difference score as fitness function.

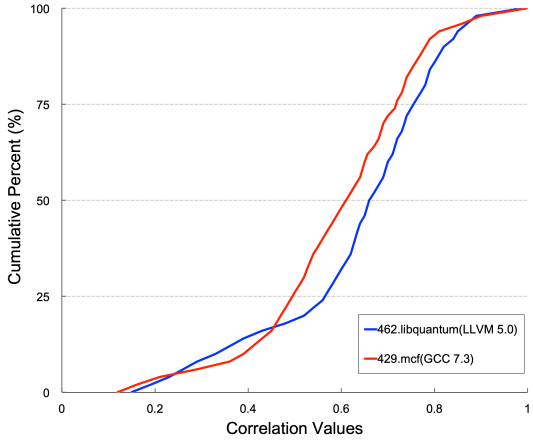

Figure 9: Pearson correlation data between NCD scores and BinDiff difference values versus cumulative percent. The plot shows significant positive correlations for most cases.

D CROSS COMPARISON

For the significant cases shown in Figure 5, Table 4 and Table 5 present the cross comparison results among BinTuner and -Ox optimization levels. The last column shows the sum of cross comparison results for each compilation setting, and BinTuner's results are unquestionably the most significant ones.

Table 4: LLVM 5.0 & 462.libquantum

	vs. O0	vs. O1	vs. O2	vs. O3	vs. BinTuner	Sum
O0		0.43	0.46	0.47	0.75	2.11
O1	0.43		0.17	0.21	0.73	1.54
O2	0.46	0.17		0.05	0.7	1.38
O3	0.47	0.21	0.05		0.71	1.44
BinTuner	0.75	0.73	0.7	0.71		2.89

Table 5: GCC 7.3 & Coreutils

	vs. O0	vs. O1	vs. Os	vs. O2	vs. O3	vs. BinTuner	Sum
O0		0.41	0.59	0.45	0.49	0.72	2.66
O1	0.41		0.49	0.22	0.29	0.71	2.12
Os	0.59	0.49		0.46	0.53	0.72	2.79
O2	0.45	0.22	0.46		0.15	0.73	2.01
O3	0.49	0.29	0.53	0.15		0.72	2.18
BinTuner	0.72	0.71	0.72	0.73	0.72		3.6

Table 6: List of top-venue papers related to binary diffing in the last decade. All of them evaluate the binary code or bytecode produced by default compiler optimization settings. The fifth column summarizes the code representations that are used to measure similarity. "BB" and "CFG" are short for basic block and control flow graph, respectively. ● in the last column denotes this paper compares the proposed method with the industry-standard binary diffing tool: BinDiff [57, 58]; ◐ indicates this paper relies on BinDiff's output; and ○ means this paper mentions BinDiff as related work.

#	Authors	Top Ven.	Topic	Representation	B.D.
1	Ding et al.	S&P'19	Learning-based binary code clone search against compiler-level transformation.	Function	○
2	Zuo et al.	NDSS'19	Cross-architecture basic-block similarity comparison using neural network.	Basic block	
3	Gao et al.	FSE'18	Learning-based vulnerability search via function semantic emulation.	Function	
4	Liu et al.	ASE'18	Binary code similarity detection with deep neural network.	Function	●
5	David et al.	ASPLOS'18	Scalable vulnerable procedure search in stripped firmware images.	BB slices	●
6	Caliskan-Islam et al.	NDSS'18	De-anonymize the authors of binary code by detecting their unique coding style.	Function	
7	Xu et al.	CCS'17	Cross-platform binary diffing via neural network-based graph embedding.	BB & CFG	○
8	Wang et al.	ASE'17	Identify similar functions in binary code using in-memory fuzzing.	Function	●
9	Kargén et al.	ASE'17	Scalable instruction trace alignment of binary code.	Trace	○
10	Ming et al.	Security'17	Binary diffing via equivalence checking of API call sliced segments.	API call slices	●
11	Xu et al.	ICSE'17	A patch analysis framework to learn security patch/vulnerability patterns.	BB & CFG	◐
12	David et al.	PLDI'17	Binary procedure similarity detection via re-optimizing basic block strands.	BB slices	●
13	Xu et al.	S&P'17	Known cryptographic function detection via bit-precise symbolic loop mapping.	Loop	●
14	Chandramohan et al.	FSE'16	Scalable binary function matching via selective inlining.	BB tracelets	●
15	Su et al.	FSE'16	Code relatives detection via efficiently matching instruction dependency graph.	Instruction graph	
16	Kalra et al.	FSE'16	Detect application-affecting changes across two library binary versions.	Function	●
17	Feng et al.	CCS'16	Scalable bug search in firmware images with CFG's numeric feature vector.	BB & CFG	○
18	Ding et al.	KDD'16	Large-scale assembly function clone search using MapReduce subgraph search.	Function	○
19	David et al.	PLDI'16	Compare similar binary procedures with small comparable basic block strands.	BB slices	●
20	Eschweiler et al.	NDSS'16	Cross-architecture search for similar binary functions using bipartite matching.	BB & CFG	●
21	Graziano et al.	Security'15	Compare malware code submitted to sandboxes to study malware developments.	BB & CFG	◐
22	Pewny et al.	S&P'15	Cross-architecture bug search in binary code via basic block sampling.	BB & CFG	○
23	Dalla Preda et al.	POPL'15	A formal semantic model for mixed syntactic/semantic binary similarity analysis.	Basic block	◐
24	Luo et al.	FSE'14	Software plagiarism detection by matching equivalent basic block subsequence.	Basic block	●
25	Egele et al.	Security'14	Search similar functions among binary code with dynamic similarity testing.	Function	●
26	David et al.	PLDI'14	Binary function search by comparing fixed length of basic block tracelets.	BB tracelets	○
27	Sharma et al.	OOPSLA'13	Verify the equivalence of two loops in binary code for loop optimizations.	Loop	
28	Schuster et al.	CCS'13	Compare control flow traces of various inputs to identify backdoor regions.	Trace	○
29	Jang et al.	Security'13	Recover the evolutionary relationship among a set of binary programs.	BB& Behavior	○
30	Hu et al.	ATC'13	Scalable malware clustering with instruction n-grams after generic unpacking.	Instruction n-grams	
31	Calvet et al.	CCS'12	Cryptographic function detection in binary code with I/O parameter sampling.	Loop	
32	Zhang et al.	ISSTA'12	Algorithm plagiarism detection with dynamic value-based approaches.	Runtime invariants	
33	Jang et al.	CCS'11	Large-scale malware similarity detection using feature hashing.	N-gram & Behavior	○
34	Chaki et al.	KDD'11	Binary code provenance-similarity detection with supervised learning.	Function	○
35	Jhi et al.	ICSE'11	Software plagiarism detection by matching critical runtime invariant values.	Runtime invariants	
36	Rosenblum et al.	ISSTA'11	Recover the provenance of source language and compiler from binary code.	Function	○
37	Comparetti et al.	S&P'10	Identify similar malicious functionality that is not active at run time.	BB & CFG	○
38	Fredrikson et al.	S&P'10	Synthesize behavior-based malware specifications to match new malware.	Syscall graph	
39	Wang et al.	CCS'09	Software plagiarism detection via system call dependence graph comparison.	Syscall graph	
40	Hu et al.	CCS'09	Find similar malware variants via fast function-call graph comparison.	Call graph	○
41	Bayer et al.	NDSS'09	Scalable malware clustering with behavioral profiles.	Behavioral profile	○
42	Sæbjørnsen et al.	ISSTA'09	Binary clone detection by matching normalized instruction sequences.	Instruction n-grams	○
43	Brumley et al.	S&P'08	Automatic 1-day exploit generation via patch difference analysis.	BB & CFG	◐

Table 7: BinDiff's detailed metrics under LLVM's optimization settings. The three numbers in each tuple mean the ratio of matched basic blocks, the ratio of matched CFG edges, and the ratio of matched non-library functions. We take the larger value between two sides as the denominator. The last two columns list BinTuner's search iterations and total running time (hour). "C.u." and "O.S." are short for Coreutils and OpenSSL, respectively.

Prog.	O1 vs. O0	O2 vs. O0	O3 vs. O0	Top vs. O0	# Iter.	Hr.
400	(29K/51K, 24K/75K, 1.9K/1.9K)	(27K/51K, 22K/79K, 1.7K/1.9K)	(26K/53K, 21K/84K,1.7K/1.9K)	(23K/63K, 18K/98K, 1.2K/1.9K)	687	6.7
401	(145/2K, 86/2.9K, 42/131)	(180/2.6K, 97/4K, 44/100)	(206/2.7k, 112/4.1k, 43/100)	(234/4K,140/6.6K, 49/101)	519	0.5
429	(309/410, 208/666, 50/52)	(312/525, 209/666, 50/52)	(313/525, 210/666, 50/52)	(329/560, 233/804, 50/58)	482	0.4
445	(18K/28K, 15K/38K, 2.7K/2.7K)	(16K/28K, 12K/38K, 2.5K/2.7K)	(16K/29K, 12K/43K, 2.5K/2.7K)	(13K/54K, 10K/84K, 2K/2.7K)	577	3.4
456	(6.5K/10K, 5K/14K, 609/612)	(5.8K/10K, 4.4K/15K, 541/609)	(6.5K/10K, 5K/14K, 609/612)	(1.6K/8.9K, 1.4K/13K, 145/612)	436	1.1
458	(2.5K/5.6K, 1.9K/8.1K, 185/190)	(2.3K/5.6K, 1.8K/8.1K, 171/185)	(2.5K/5.6K, 1.9K/8.1K, 185/190)	(1.7K/8.4K, 1.2K/14K, 88/185)	347	0.5
462	(878/1.2K, 673/1.5K, 154/155)	(838/1.2K, 661/1.6K, 135/154)	(853/1.3K, 661/1.7K, 135/154)	(232/1.2K, 194/1.5K, 42/154)	363	0.3
464	(9.4K/19K, 6.7K/25K, 643/643)	(8.1K/19K, 5.8K/25K, 571/643)	(7.2K/19K, 5.1K/26K, 571/643)	(4.8K/19K, 3.3K/25K, 285/643)	457	3.2
473	(904/1.4K, 600/1.7K, 181/192)	(879/1.4K, 597/1.9K, 118/192)	(856/1.4K, 583/2K, 118/192)	(351/1.4K, 962/1.7K, 62/192)	371	0.3
483	(67K/94K, 38K/91K, 28K/30K)	(45K/103K, 28K/137K, 12K/30K)	(44K/107K, 27K/143K, 12K/30K)	(57K/120K, 31K/119K, 10K/30K)	584	22.7
600	(48K/109K, 36K/167K, 3.1K/3.1K)	(45K/118K, 33K/183K, 2.4K/3.1K)	(44K/124K, 32K/195K, 2.4K/3.1K)	(44K/149K, 31K/230K, 1.6K/3.1K)	549	13.8
605	(468/835, 326/1K, 68/70)	(517/1.3K, 344/2K, 66/68)	(481/1.4K, 323/2.1K, 66/68)	(141/973, 88/1.4K, 38/68)	316	0.3
620	(28K/48K, 16K/51K, 11K/12K)	(21K/65K, 13K/87K, 6.3K/12K)	(21K/68K, 13K/94K, 6.2K/12K)	(20K/62K, 13K/83K, 6.2K/12.2K)	585	28.7
623	(74K/104K, 43K/104K, 29K/31K)	(49K/125K, 31K/171K, 12.8K/31K)	(48K/129K, 39K/178K,12.7K/31K)	(48K/125K, 30K/171K, 12.6K/31K)	415	48.5
625	(386/531, 308/667, 48/49)	(385/609, 285/839, 48/49)	(379/615, 292/848, 48/49)	(204/616, 126/876, 41/48)	405	0.3
631	(1.8K/3.8K, 1.4K/5.3K, 164/167)	(1.58K/3.8K, 1.21K/5.3K, 146/167)	(1.57K/3.8K, 1.19K/5.3K, 146/167)	(1.5K/4.3K, 1.1K/7K, 142/167)	442	0.4
641	(2.9K/5.8K, 1.8K/6.9K, 805/829)	(2.3K/6K, 1.5K/6.9K, 459/829)	(2.3K/6.7K, 1.4K/9.8K, 457/829)	(815/5.8K, 533/6.9K, 188/829)	481	2.0
648	(1.8K/2.9K, 1.5K/4.6K, 73/74)	(551/2.9K, 402/4.6K, 62/73)	(518/2.9K, 377/4.6K, 63/73)	(516/2.9K, 378/4.6K, 61/73)	351	0.6
657	(3.4K/5.6K, 2.6K/7.4K, 588/590)	(2.6K/5.8K, 2K/8.5K, 413/588)	(2.7K/6.4K, 2K/9.4K, 414/588)	(1.1K/6K, 759/9.2K, 170/588)	279	0.6
C.u.	(23K/44K, 16.7K/60K, 2.84K/2.85K)	(15K/44K, 10K/65K, 1.5K/2.85K)	(14.7K/44K, 9.7K/72K, 1.48K/2.85K)	(14K/44K, 11K/60K, 1.4K/2.85K)	527	4.4
O.S.	(10K/17K, 6.6K/22K, 2.1K/2.7K)	(8.9K/17K, 5.7K/22K, 1.7K/2.7K)	(8.8K/17K, 5.6K/22K, 1.68K/2.7K)	(8.6K/17K, 5.3K/22K, 1.66K/2.7K)	593	4.9

Table 8: BinDiff's detailed metrics under GCC's optimization settings.

Prog.	Os vs. O0	O2 vs. O0	O3 vs. O0	Top vs. O0	# Iter.	Hr.
400	(30K/49K, 28K/74K, 1.85K/2K)	(28K/49K, 23K/78K, 1.85K/2K)	(27.8K/64K, 22.2K/99K, 1.78K/2K)	(27K/81K, 21K/119K, 1.7K/2K)	635	12.4
403	(67K/171K, 59K/272K, 4.7K/5.7K)	(62K/172K, 48K/276K, 4.7K/5.7K)	(63K/219K, 48K/356K, 4.5K/5.7K)	(62K/205K, 49K/325K, 4.1K/5.7K)	561	16.6
429	(351/476, 305/608, 52/53)	(339/476, 239/608, 52/53)	(346/569, 232/812, 52/53)	(376/1.56K, 241/2.17K, 53/124)	891	0.4
445	(18K/28.9K, 15.7K/39.5K, 2.6K/2.74K)	(17K/28.9K, 13K/39.5K, 2.59K/2.74K)	(17K/38K, 12.4K/56K, 2.53K/2.74K)	(18K/56K, 13K/80K, 2.5K/2.74K)	491	6.1
456	(6.9K/10K, 7.1K/14K, 572/612)	(6.1K/11K, 4.7K/16K, 568/612)	(6K/13K, 4.5K/19K, 548/612)	(5.3K/11.8K, 4.1K/17K, 508/612)	849	2.7
458	(2.8K/5K, 2.6K/7.5K, 185/189)	(2.6K/5K, 2K/7.5K, 179/189)	(2.4K/5K, 2K/11K, 175/189)	(2.6K/8K, 2.3K/12.8K, 166/189)	588	0.9
462	(608/5K, 391/7.5K, 124/189)	(574/5K, 312/7.5K, 119/189)	(582/5K, 330/7.5K, 114/189)	(561/5K, 431/7.6K, 106/189)	937	0.6
471	(7.7K/13K, 5.9K/15K, 1.9K/2.68K)	(7K/14.7K, 4.5K/17.8K, 1.9K/2.68K)	(7K/12K, 4.3K/24K, 1.88K/26.8K)	(7.4K/21.5K, 4.9K/28K, 1.8K/2.68K)	937	8.1
473	(962/1.4K, 995/1.77K, 123/187)	(890/1.4K, 673/1.77K, 125/187)	(877/1.5K, 665/2.1K,121/187)	(1K/5.9K, 811/8.6K, 159/191)	510	0.7
483	(51K/96.8K, 39.6K/107K, 13K/29K)	(47K/138K, 30K/185K, 12.2K/29K)	(46K/179K, 29K/239K,12.1K/29K)	(46K/178K, 33K/249K, 12K/29K)	573	31.3
600	(49K/105K, 43K/154K, 2.76K/5.2K)	(46K/108K, 36K/165K, 2.66K/5.2K)	(46K/108K, 36K/165K, 2.66K/5.2K)	(47K/137K, 36K/202K, 2.48K/5.2K)	946	23.8
605	(520/757, 452/987, 68/71)	(471/778, 340/1K, 66/71)	(473/1.4K, 316/2.2K, 66/71)	(558/2.9K, 377/4.2K, 71/137)	1,881	1.6
620	(25K/46K, 19K/50K, 6.3K/11.5K)	(23K/60K, 15K/77K, 6.2K/11.5K)	(24K/87K, 15K/119K, 6.2K/11.5K)	(23.8K/88K, 15.7K/120K, 6.1K/11.5K)	932	31.2
623	(73.8K/148K, 58.9K/176K, 16K/30K)	(61.8K/294K, 40K/423K, 14.1K/30K)	(60K/328K, 37.9K/476K, 14K/30K)	(55K/366K, 31K/515K, 13.5K/30K)	473	70.9
625	(410/540, 391/648, 47/50)	(370/540, 286/695, 49/50)	(374/540, 278/710, 49/50)	(364/1.2K, 273/1.68K, 50/93)	1,684	0.5
631	(1.86K/3.6K, 1.6K/5.3K, 147/162)	(1.6K/3.7K, 1.24K/50K, 148/162)	(1.75K/4.9K, 1.3K/7.6K, 149/162)	(1.8K/5.8K, 1.39K/8.6K, 146/162)	1,790	2.6
641	(2.1K/4.9K, 1.6K/4.7K, 477/1.5K)	(1.9K/4.9K, 1.3K/9.8K, 426/1.5K)	(1.95K/11K, 1.2K/16K, 416/1.5K)	(2K/16.4K, 1.2K/23.6K, 435/1.5K)	469	3.2
648	(1.9K/3.5K, 1.6K/4.9K, 48/53)	(1.76K/3.5K, 1.36K/4.9K, 48/53)	(1.5K/3.5K, 1.2K/3.7K, 47/53)	(1.5K/8.1K, 1.1K/12K, 47/97)	1,677	3.7
657	(3.2K/5.6K, 2.98K/7.4K, 451/589)	(2.7K/5.8K, 2K/8.5K, 413/588)	(2.69K/6.4K, 2K/9.4K, 414/588)	(3K/11K, 2.1K/15.4K, 429/623)	611	1.8
C.u.	(19.5K/40K, 16.8K/56.2K, 1.8K/2.67K)	(23K/40K, 25.6K/59.5K, 1.69K/2.67K)	(21.8K/44.5K, 24.6K/66.8K, 1.5K/2.67K)	(18.5K/79.4K, 12.9K/114K, 1.4K/2.67K)	841	7.0
O.S.	(10K/15.8K, 7.5K/20K, 1.72K/2.71K)	(9.7K/15.8K, 6.7K/10K, 1.70K/2.71K)	(9.5K/15.8K, 6.5K/21.6K, 1.69K/2.71K)	(9.5K/19.7K, 6.4K/27.1K, 1.68K/2.74K)	803	6.7