On the Optimality of *K* Longest Path Generation Algorithm Under Memory Constraints

Jie Jiang[1], Matthias Sauer[2], Alexander Czutro[2], Bernd Becker[2], Ilia Polian[1]
[1]Department of Informatics and Mathematics, University of Passau, Innstr. 43, D-94032 Passau, Germany
[2]Department of Computer Science, Albert-Ludwigs-University, Georges-Koehler-Allee 51, D-79110 Freiburg, Germany
{jie.jiang | ilia.polian}@uni-passau.de {sauerm | aczutro | becker}@informatik.uni-freiburg.de

Abstract—Adequate coverage of small-delay defects in circuits affected by statistical process variations requires identification and sensitization of multiple paths through potential defect sites. Existing *K* longest path generation (KLPG) algorithms use a data structure called *path store* to prune the search space by restricting the number of sub-paths considered at the same time. While this restriction speeds up the KLPG process, the algorithms lose their optimality and do not guarantee that the *K* longest sensitizable paths are indeed found. We investigate, for the first time, the effects of missing some of the longest paths on the defect coverage. We systematically quantify how setting different limits on the path-store size affects the numbers and relative lengths of identified paths, as well as the run-times of the algorithm. We also introduce a new optimal KLPG algorithm that works iteratively and pinpointedly addresses defect locations for which the path-store size limit has been exceeded in previous iterations. We compare this algorithm with a naïve KLPG approach that achieves optimality by setting the path-store size limit to a very large value. Extensive experiments are reported for 45nm-technology data.

***Index Terms*—Parameter variations, small-delay testing, *K* longest path generation**

I. INTRODUCTION

Delays in nanoscale circuits are heavily affected by statistical process variations [1][2][8][14][21][22][23]. This has a fundamental impact on the testability of small-delay defects (SDDs). A given SDD may lead to violating timing conditions (and therefore be a critical defect) in one manufactured instance of a circuit, but may not impact the functionality of a different instance of the same circuit. To reliably detect a critical SDD, different excitation and propagation conditions must be fulfilled for each circuit instance. Hence, standard SDD test methods [16][18] do not always provide a good defect coverage under process variations.

The following three basic classes of approaches have been suggested to deal with this problem. Approaches from the first class are based on constructing a model of process variations and integrating it into the test generation framework [10][11][13][24]. For example, a space of process parameters is defined in [10]; for a given SDD and a test pattern, the process-parameter combinations for which the SDD is detected are identified. Test generation is continued until a sufficient portion of the parameter space has been covered. The drawback of such approaches is the difficulty to obtain a technology-aware model of process variations that is accurate as well as sufficiently compact for integration into a test-generation flow. Specific challenges are the unclear validity of assumptions on process variations in a given technology; the high effort to perform a statistically significant number of either electrical simulations or actual measurements; the large number of process parameters which affect many locations of the circuit; and the complex interaction of process variations and actual defect mechanisms (such as resistive opens [6]).

The second class of approaches enriches the models by incorporating actual delays measured on manufactured instances of the circuit [12]. For example, tunable delay elements called Clock Vernier Devices were employed in the clock signal distribution network of Intel's Montecito processor to manipulate the delays of specific paths. Then, the delay of a path can be determined by tuning it until it fails [15]. Obviously, such approaches require the presence of an on-chip infrastructure and can only be run after first silicon is available.

The third class of approaches covers an SDD by propagating the defect through a number of different paths [17][19][20]. For each SDD-location from a given set (for instance, outputs of all the logic gates in a circuit), a number *K* of longest sensitizable paths are identified (where the length of the path is determined using nominal gate delays). For each found path, a test pair detecting the defect by sensitizing this path is generated. This *K* Longest Path Generation (KLPG) is done in order to account for the fact that the lengths of the paths vary in different manufactured instances of a circuit. Detection of a critical SDD in a manufactured circuit instance is guaranteed as long as one of the *K* paths through its location selected by KLPG is indeed the longest sensitizable path in the regarded circuit instance. KLPG does not require explicit modeling of process variations and can be assumed to be more computationally feasible than the variation-aware test generation mentioned before.[1] A related problem is the search for *K* globally longest sensitizable paths in a circuit [5][9][14].

KLPG algorithms maintain a number of partial paths which are possible sub-paths of one of the *K* longest sensitizable paths. These sub-paths are continuously extended until they either become a complete sensitized input-to-output path, or until they are proven to be unsensitizable. A number of ad-

[S] This work has been supported by the German National Science Foundation (DFG) under grant numbers BE 1176-15/2, PO 1220-2/2 and GRK 1103.

[1] This is similar to n-detection, i.e., testing the same stuck-at fault by multiple different patterns, in order to cover not modeled defects such as shorts or opens [6]

978-3-9810801-8-6/DATE12/©2012 EDAA

vanced speed-up techniques for KLPG have been shown to scale this basic algorithm to large circuits [5][9][17]. Most of these techniques have no impact on the *optimality* of the algorithm, i.e., the algorithm yields K longest sensitizable paths if that many different paths exist. A KLPG algorithm is not optimal if it may miss one or several of the K longest sensitizable paths and either finds a shorter path or no path at all. The optimality of KLPG implementations in [5][17] is affected by imposing an upper limit on the number of partial paths maintained at any point of time. The corresponding data structure is called a *path store* in [17] (and corresponds to Ψ in [5]). It contains partial paths for which it is not yet definitely known whether they can be extended to one of the K longest sensitizable paths. Imposing a limit on the path store size leads to a *path-store overflow*, which implies that some of the partial paths cannot be maintained. If some of the actual K longest paths are extensions of such an excluded partial path, they will not be found, thus leading to a sub-optimal solution.

In [17], a path-store size limit of 3,000 is used for the experiments. Different limits varying between 10 and 100 are used in [5] (these low values might be an explanation for the very low run-times reported in that paper). This prompts a question whether the impact of path-store overflows on the quality of the obtained solution is significant. This is the subject of investigation in this paper.

Using an implementation of a KLPG algorithm along the lines of [17], we systematically evaluate the impact of path-store overflows on the quality of the found paths. We identify SDDs for which not all partial paths fitted into the path store and compare the paths determined by the KLPG algorithm with the optimal solution, i.e., with actual K longest sensitizable paths. For different values of the path-store size limit π_{max}, we report the number of SDDs for which less sensitizable paths were found due to the overflows. For SDDs with an equal number of found paths, we compare the lengths of the paths. Using a path of a lower length for testing implies a sub-optimal coverage of certain defect sizes. In particular, we calculate the number of SDDs for which the overflows lead to substantially shorter paths.

Since the aim of imposing a path-store size limit is run-time efficiency, we also study the scalability of the algorithm as a function of π_{max}. Moreover, we investigate the cost of generating a provably optimal solution. One trivial way to obtain such a solution is to use a very high value of π_{max}, or a dynamic data structure with no imposed limit. We propose an alternative iterative algorithm *Opt-KLPG*. We investigate the run-time behavior of *Opt-KLPG* and compare it to the original algorithm. We are not aware of a previous study that systematically investigates the optimality of KLPG.

II. K LONGEST PATH GENERATION ALGORITHM

A. Overview of the method

The KLPG algorithm was implemented along the lines of [17] (a flowchart is found in Figure 1). As in [17], partial paths are grown from the circuit's inputs to the outputs. They are required to pass through the SDD location. To grow a path, the side-inputs of the corresponding gates are assigned appropriate logic values and their implications are calculated. Whenever an assignment results in a local conflict, the partial path is dropped. For each partial path under consideration, a value called *max esperance*, corresponding to the maximal length of an extension of the partial path, is calculated and continuously updated. When K complete sensitizable paths are available, all partial paths with a max esperance less than the minimal length of these K paths are dropped. This is done because these paths will definitely be shorter than the K already known paths when fully extended.

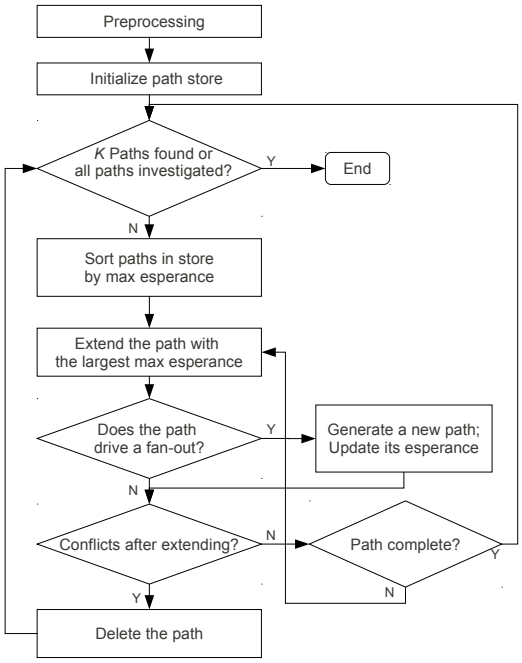

Figure 1. Flowchart of the KLPG algorithm from Section II.

As soon as a complete path of sufficient length and without local conflicts has been calculated, it is sensitized. In [17], the FAN algorithm is used for this purpose. In our implementation, a Boolean-Satisfiability (SAT) instance is constructed and the sensitization is performed using the TIGUAN engine [7]. Unlike [5], no incremental SAT solving is used.

B. Pre-processing

Before path generation starts, all gates in the circuit are topologically sorted in decreasing order of their levels. The max esperance is calculated for all gates in the sorted order (where max esperance of a gate is an upper bound of maximal length of a partial path from the gate to an output). The recursive max esperance calculation works as follows: if a gate drives the primary output and no other gates, the max esperance of the gate equals its maximal (port-to-port) gate delay. Otherwise the gate's max esperance is the sum of its maximal gate delay and the maximum of max esperances of gates driven by its output (which have been calculated already).

C. Path generation

For each SDD in the fault list, the path store is progressively filled with the partial paths that start at a primary input in the SDD's input cone, since only these paths can be potentially extended to the SDD location. The max esperance of a partial

path is calculated for each considered path, by adding the actual partial-path length and the max esperance of the last gate on the partial path. The partial path with the largest max esperance is selected and extended by one gate. In case of a fanout to multiple gates, a copy of the partial path is stored in the path store unless its size limit π_{max} is exceeded. Then, the partial path is extended by the gate with the largest max esperance. For example, if partial path *a–b–c* is considered and *c* has three successors *d*, *e* and *f*, a copy of the partial path *a–b–c* is stored in the path store and the partial path is extended such as to have the highest max esperance, e.g., *a–b–c–d*. The copy of partial path *a–b–c* kept in the path store contains only two successors (*e* and *f*), as successor *d* is being evaluated already. Moreover, the max esperance of the copy of *a–b–c* in the path store is computed taking into consideration successors *e* and *f* only.

After path extension, the *sensitizability* of the extended partial path, i.e., its ability to propagate a signal transition from the primary input to its current last gate, is checked. We implemented sensitizability conditions for robust, non-robust and functional sensitization [4]. To guarantee the path's sensitization, the side inputs of the newly added gate are assigned logic values according to the selected sensitizability conditions. The value assignments can imply values on other signals in the circuit; if these signals are already set to a different value, a *conflict* is detected and the extending fails. We apply the "direct implication" technique [17] to identify conflicts. Not all possible conflicts are guaranteed to be detected by this technique.

If the extension of a partial path fails, that partial path is dropped as it cannot be extended to a sensitizable path. Then, the partial path with the largest max esperance is taken out of the path store. The algorithm attempts to extend the next path with the largest max esperance of the successor gates. Partial paths that have been considered previously are excluded from consideration. For example, if *a–b–c–d* could not be sensitized, partial path *a–b–c* may be selected and extended by the remaining successor (e.g., *e*) with the highest max esperance.

D. Path sensitization

A completely extended path (going from a primary input through the SDD location to a primary output) has no conflicts identified by the direct implication technique, but such a path is still not guaranteed to be sensitizable. In order to check the sensitizability of a complete path, we derive a *conditional multiple stuck-at* (CMS@) fault [7] from the generated path. A CMS@ fault consists of a list of fault activation conditions and a list of stuck-at-like fault effects. In our case, fault activation conditions are given by the values assigned to the side inputs of the gates on the path, and the only fault effect is the erroneous value on the SDD location. We use the SAT-based ATPG tool TIGUAN [7] to find a test that detects the constructed CMS@ fault. If a test pattern is found, it sensitizes the path. If TIGUAN reports that the CMS@ fault is untestable, the path is not sensitizable.

The KLPG algorithm terminates when either *K* sensitizable paths have been found or when the path store becomes empty.

III. THE OPTIMAL KLPG ALGORITHM

The algorithm *Opt-KLPG* is outlined in Figure 2. It starts by running KLPG with a user-defined path-store size limit π_{max}. The path store is empty at the beginning of the first iteration of *Opt-KLPG*. In contrast to the algorithm in the previous section, partial paths which are not included in the path store due to overflows (*overflow paths*) are kept for subsequent iterations. Only the paths in the path store are considered by the KLPG algorithm. The first iteration of *Opt-KLPG* yields a (potentially sub-optimal) set of sensitizable paths for each SDD.

If an SDD is affected by an overflow, subsequent iterations of *Opt-KLPG* invoke further KLPG runs (using the same π_{max}-value). In these iterations, the path store is initialized with overflow paths generated in previous iterations. If the number of such partial paths exceeds π_{max}, the ones with the largest max esperance are selected. This means that some of the paths that have not been selected may be re-generated during KLPG.

As soon as *K* sensitizable paths have been found, all partial paths with max esperance below the length of the shortest of the *K* paths are dropped immediately. This procedure is iterated until all overflow partial paths have been inserted into the path store and investigated. Note that the solution quality yielded by *Opt-KLPG* is not affected by the choice of π_{max} as the algorithm is guaranteed to always find the optimal solution.

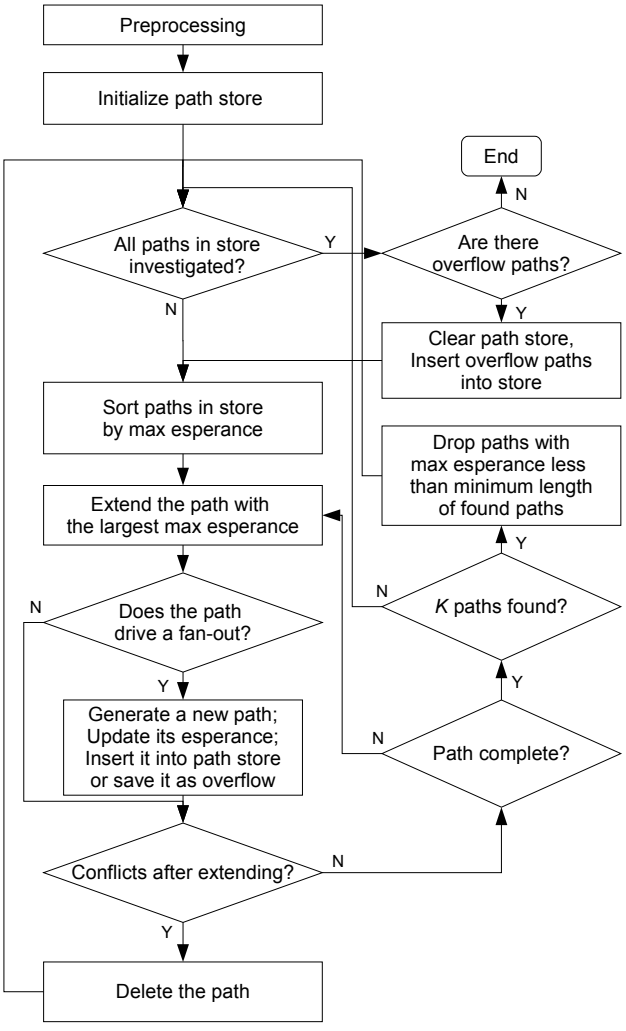

Figure 2. Flowchart of the *Opt-KLPG* algorithm from Section III.

TABLE I. EXPERIMENTAL RESULTS FOR FUNCTIONAL SENSITIZATION

Circuit (gate cnt)	π_{max}	**KLPG**				***Opt-KLPG***					**Comparison**		
		#Over-flows	Path length	CPU time [s] Total	CPU time [s] Excl SAT	#Over-flows	Average # re-iterations	Path length	CPU time [s] Total	CPU time [s] Excl SAT	Δ path length [%]	Gates with less paths	Gates with short paths
c1908	10	143619	75569	36	4	332249	42.25	86185	346	26	14.05	15	671
(795)	50	215713	82819	76	8	280743	7.56	86185	337	22	4.06	1	352
	100	125803	85717	68	8	163067	2.46	86185	309	21	0.55	0	118
	500	13356	86155	82	13	29950	0.08	86185	306	42	0.03	0	4
	1000	14932	86181	122	30	23696	0.03	86185	343	77	0.00	0	2
	1500	16278	86181	177	56	20164	0.02	86185	386	122	0.00	0	2
	3000	12902	86181	375	186	12902	0.01	86185	526	252	0.00	0	2
	30000	0	86185	815	510	0	0.00	86185	774	490	0.00	0	0
c5315	10	117690	201540	60	7	170043	8.02	215813	147	9	7.08	8	1306
(2228)	50	68354	215615	94	8	74667	0.97	215813	126	8	0.09	0	96
	100	33548	215770	111	9	35998	0.26	215813	131	8	0.02	0	21
	500	1402	215812	132	11	1402	0.01	215813	134	11	0.00	0	1
	1000	0	215813	134	11	0	0.00	215813	134	11	0.00	0	0
	1500	0	215813	134	12	0	0.00	215813	134	11	0.00	0	0
	3000	0	215813	134	11	0	0.00	215813	133	11	0.00	0	0
	30000	0	215813	134	12	0	0.00	215813	135	12	0.00	0	0
c7552	10	210254	249957	141	17	258557	9.19	260443	324	22	4.20	25	1858
(2952)	50	102766	259465	211	17	110967	1.06	260443	277	20	0.38	0	382
	100	49843	260108	237	20	52150	0.30	260443	268	20	0.13	0	125
	500	509	260443	265	23	509	0.00	260443	264	22	0.00	0	0
	1000	0	260443	264	22	0	0.00	260443	265	22	0.00	0	0
	1500	0	260443	264	23	0	0.00	260443	265	22	0.00	0	0
	3000	0	260443	269	23	0	0.00	260443	264	23	0.00	0	0
	30000	0	260443	264	23	0	0.00	260443	270	23	0.00	0	0
b14	10	214621	97653	127	15	1417923	21.03	126305	5688	336	29.34	19	931
(6763)	50	475719	113633	367	19	1360685	4.09	126305	5792	147	11.15	2	662
	100	573137	121862	490	23	1201388	1.84	126305	5255	114	3.65	0	550
	500	485778	125788	1745	48	761236	0.26	126305	5214	131	0.41	0	90
	1000	378266	126103	2614	108	560882	0.11	126305	5559	246	0.16	0	38
	1500	309834	126169	3084	175	414736	0.05	126305	5378	320	0.11	0	32
	3000	221434	126252	3825	360	267347	0.02	126305	4984	511	0.04	0	14
	30000	0	126305	6959	2717	0	0	126305	7928	3566	0	0	0
b15	10	185349	50203	138	58	879949	9.9	69250	5268	944	37.94	61	700
(8931)	50	231739	61128	270	78	823864	1.88	69250	5249	903	13.29	15	530
	100	250348	62495	398	97	799232	0.93	69250	5306	941	10.81	15	484
	500	252861	69093	1357	341	373079	0.09	69250	4406	850	0.23	0	51
	1000	179548	69188	2239	578	250773	0.03	69250	4727	1002	0.09	0	20
	1500	150366	69219	2948	803	198749	0.02	69250	5003	1215	0.04	0	12
	3000	78511	69223	4279	1404	97662	0.01	69250	5872	1910	0.04	0	10
	30000	0	69250	7151	3242	0	0	69250	8545	4244	0	0	0
p35	10	293796	7395	542	40	360685	1.83	8223	1218	55	11.19	0	852
(19964)	50	270197	7813	589	40	342143	0.36	8223	1329	51	5.25	0	219
	100	263782	7869	701	39	359731	0.19	8223	2178	63	4.49	0	160
	500	242401	7988	1095	48	368222	0.04	8223	2959	95	2.94	0	116
	1000	268786	8091	1749	69	285228	0.02	8223	2644	92	1.62	0	87
	1500	216978	8150	1451	74	226846	0.01	8223	1710	92	0.90	0	51
	3000	159280	8196	1594	116	160445	0.00	8223	1887	150	0.33	0	40
	30000	0	8223	506	57	0	0.00	8223	489	55	0.00	0	0
p45	10	80796	3824	142	58	609588	2.68	4569	8048	647	19.47	17	699
(22877)	50	122172	4376	261	68	480945	0.43	4569	7715	265	4.41	0	342
	100	115619	4494	470	82	425049	0.19	4569	6717	240	1.66	0	169
	500	111723	4555	1105	123	326295	0.03	4569	9029	549	0.30	0	50
	1000	100109	4562	1580	216	267548	0.01	4569	7105	947	0.15	0	32
	1500	102996	4563	1703	312	239735	0.01	4569	6102	1153	0.12	0	24
	3000	98656	4568	2898	849	195398	0.00	4569	7082	2276	0.03	0	12
	30000	7513	4569	20185	15750	7513	0.00	4569	20211	15756	0.00	0	1
p78	10	166206	6342	240	101	689810	1.20	7149	7139	418	12.72	8	901
(57608)	50	177257	7039	391	112	501469	0.18	7149	5451	198	1.56	0	722
	100	99069	7147	335	108	373759	0.07	7149	5483	174	0.03	0	96
	500	135763	7148	703	125	321940	0.01	7149	5240	245	0.01	0	5
	1000	150340	7148	1168	180	299284	0.01	7149	7062	413	0.01	0	6
	1500	151226	7149	1344	208	283626	0.00	7149	5099	498	0.00	0	2
	3000	148676	7149	2257	461	243633	0.00	7149	5545	927	0.00	0	2
	30000	0	7149	10590	5859	0	0.00	7149	10560	5840	0.00	0	0

IV. EXPERIMENTAL RESULTS

We applied our implementation of KLPG and *Opt-KLPG* to ISCAS-85 circuits (which are small but are known to have a large number of reconvergencies and are therefore challenging for path-selection algorithms), combinational cores of ITC-99 circuits, and industrial circuits provided by NXP. We synthesized the original circuits using a publicly available 45nm cell library [3] in order to use realistic pin-to-pin gate delays. We omit results for circuit c6288, a multiplier with a large number of long non-sensitizable paths. In [17], a specific technique called *Smart-PERT* was developed in order to handle c6288; our implementation does not include this technique.

Table I summarizes the results for $K = 5$ (i.e., five longest sensitizable paths are generated for each gate output in the circuit) and functional sensitization conditions. For each circuit, KLPG and *Opt-KLPG* were run for a number of different path-store size limits π_{max} between 10 and 30,000. We used fault lists consisting of all gates for ISCAS circuits and 1,000 randomly selected gates for ITC-99 and NXP circuits. Column 1 shows the circuit name with its gate count in parentheses. Column 2 and 3 contain π_{max} and the number of overflows encountered by KLPG. In column 4, the sum of lengths of the (up to) five paths identified by KLPG is reported. If the number of overflows is 0, then KLPG's results are optimal and the length in column 4 is maximal; otherwise it can be sub-maximal. Note that the lengths are real numbers rounded to the nearest integer. For instance, KLPG fails to find the optimal solution for p45 with $\pi_{max} = 30{,}000$, but the reported length of 4659 corresponds to the optimal length due to rounding artifacts. The run-times of KLPG can be found in columns 5 and 6. We report the total run-times as well as the times excluding the calls of the SAT-ATPG engine (columns "Excl SAT"), which describe the effort spent in actual search for paths.

From the KLPG results, it can be seen that very small values of π_{max} typically result in a large number of overflows (we will discuss the impact of the overflows on test quality related metrics such as the lengths of the found paths later). The minimal value of π_{max} which results in no overflows varies among the circuits (and does not always grow with circuit size). The number of overflows does not always decrease for larger path store size limits. The reason is that a larger path store may contain a partial path which in turn results in generation of more partial paths, while a smaller path store may simply not contain this path. The run-time difference of our implementation and [17] can be partially attributed to the lack of a number of advanced speed-up techniques such as forward-trimming or Smart-PERT in our implementation. Our aim is to study the impact of π_{max} on the results and not to outperform earlier implementations. A further reason is the amount of time spent for the path sensitization. In contrast to the FAN algorithm which can be seamlessly integrated into the KLPG framework in [17], each path sensitization requires a lengthy process in our case: A path is mapped onto a CMS@ fault for which an ATPG-instance is constructed, which in turn is mapped onto a SAT-instance and solved. Even though the employed SAT-ATPG engine can perform these tasks in a fraction of a second, but the large number of paths to be evaluated results in a large number of SAT-ATPG instances and therefore high run-time requirements. In general, run times rise with growing π_{max}.

Columns 7 through 11 provide information on the optimal algorithm *Opt-KLPG*. Column 7 contains the number of overflows encountered during the complete run of all the iterations. This number is always higher than the number of overflows in column 3, as KLPG constitutes the first iteration of *Opt-KLPG*. Column 8 contains the number of required "re-iterations", that is, iterations after the initial KLPG run. The value reported is the average over all the SDD locations (gate outputs). This number is always 0 if there were no overflows in the initial KLPG run. In column 9, the sum of the lengths of the optimal paths can be found (this value is independent from π_{max}). Columns 10 and 11 contain the run-times. As for KLPG, the run times tend to be larger for a larger π_{max}, yet the trend is less pronounced. This is because a larger π_{max} implies less re-iteration of *Opt-KLPG*. While *Opt-KLPG* requires more time than KLPG for the same π_{max}, it always yields the optimal solution. For instance, circuit c1908 requires $\pi_{max} = 30{,}000$ in order to avoid overflows altogether; this necessitates a total run time of 815 CPU-seconds. However, *Opt-KLPG* achieves the same result for $\pi_{max} = 500$ using 306 CPU-seconds. Furthermore, even running *Opt-KLPG* with $\pi_{max} = 10$ results in a large number of re-iterations per gate, but the total run time of 346 CPU-seconds is still less than for KLPG with $\pi_{max} = 30{,}000$.

The last three columns of Table I evaluate the coverage problems due to sub-optimal path selection by the KLPG algorithm. Column 12 shows the difference between the total length of the paths found by *Opt-KLPG* (column 9) and KLPG (column 4). This difference is rather large for very small values of π_{max}; it tends to be within 1% for $\pi_{max} \geq 100$. This is because many SDDs are not affected by overflows or because the identified paths are not much shorter than the optimal ones. The second-to-last column contains the number of SDD-locations (gates) through which KLPG found less paths than are in existence. This is a severe situation with a significant impact on testability under process variations. It rarely occurs for $\pi_{max} > 10$. The final column shows the number of gates for which the total length of paths identified by KLPG was shorter than the optimum. This corresponds to a coverage impact for the SDD associated with this particular gate. In some of the manufactured instances of the circuit, such SDDs will not be adequately tested. A significant number of gates turn out to be vulnerable to this problem even for large values of π_{max}. Figure 3 shows, in graph form, the number of such gates in circuit p35, accompanied by run times for KLPG and *Opt-KLPG* (which is not affected by this problem).

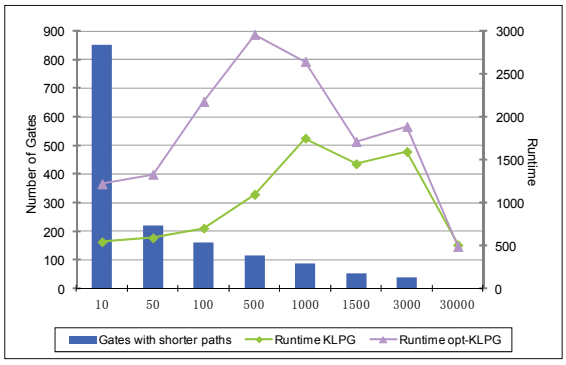

Figure 3. Experimental results for circuit p35. Run times are shown by lines; number of gates are shown by bars.

TABLE II. COMPARISON OF KLPG AND *OPT-KLPG* FOR DIFFERENT MODEL ASSUMPTIONS, CIRCUIT C7552

π_{max}	Non-robust sensitization			Unit delay, functional sens.			Unit delay, non-robust sens.		
	Δ path length [%]	Gates w. less paths	Gates w.short paths	Δ path length [%]	Gates w. less paths	Gates w.short paths	Δ path length [%]	Gates w. less paths	Gates w.short paths
10	3.95	19	1603	4.22	32	1638	4.60	33	1755
50	0.24	0	197	0.40	0	407	0.37	0	354
100	0.10	0	81	0.12	0	118	0.12	0	106
300	0.00	0	2	0.00	0	7	0.00	0	2
500	0.00	0	0	0.00	0	0	0.00	0	0
1000	0.00	0	0	0.00	0	0	0.00	0	0
1500	0.00	0	0	0.00	0	0	0.00	0	0
3000	0.00	0	0	0.00	0	0	0.00	0	0
30000	0.00	0	0	0.00	0	0	0.00	0	0

TABLE III. COMPARISON OF KLPG AND *OPT-KLPG* FOR DIFFERENT MODEL ASSUMPTIONS, INDUSTRIAL CIRCUIT P45

π_{max}	Non-robust sensitization			Unit delay, functional sens.			Unit delay, non-robust sens.		
	Δ path length [%]	Gates w. less paths	Gates w.short paths	Δ path length [%]	Gates w. less paths	Gates w.short paths	Δ path length [%]	Gates w. less paths	Gates w.short paths
10	22.17	25	698	17.12	19	673	21.02	27	676
50	5.59	1	358	4.36	0	310	6.11	2	338
100	2.01	1	180	2.08	0	156	2.79	1	178
300	0.51	1	62	0.67	0	57	0.76	0	58
500	0.33	1	44	0.47	0	47	0.50	0	44
1000	0.13	0	23	0.27	0	32	0.25	0	28
1500	0.08	0	14	0.21	0	27	0.12	0	19
3000	0.01	0	5	0.08	0	13	0.02	0	5
30000	0.00	0	0	0.00	0	0	0.00	0	0

Tables II and III contain the comparison between KLPG and Opt-KLPG (as in the final three columns of Table I) under non-robust sensitization conditions (as opposed to functional sensitization in Table I) for two alternative delay models: gate delays extracted from the technology library and unit-delay model which assumes that each gate has a delay of 1. The influence of the model assumptions tends to be limited. We made similar observation for further benchmarks which are not reported here. It can be seen that path store size affects the optimality of the solution for different assumptions.

V. CONCLUSIONS

K longest path generation is an attractive technique to test small-delay defects under process variations without having to explicitly model the variation mechanism. We studied, for the first time, the optimality of the KLPG algorithm in a systematic way, and discovered that the path-store size limit π_{max} has a significant impact on the results. In practice, conservative setting of this limit to values beyond 1,000 does yield optimal results for most (but not all) of the considered circuits. In contrast, setting π_{max} to very small values, as suggested by some recent publications, appears to have grave impact on the test quality. We suggested a provably optimal algorithm *Opt-KLPG*, which works iteratively and outperforms conventional KLPG with a large path store. Overall, it seems that KLPG optimality should be paid attention to when high test quality is required for circuits affected by process variations.

REFERENCES

[1] ITRS, "International Technology Roadmap for Semiconductors," 2009 Edition, http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[2] DATE Workshop on Process Variability, "New Techniques for the Design and Test of Nanoscale Electronics", Proc. Design, Automation and Test in Europe, 2009.

[3] Nangate 45nm Open Cell Library, http://www.nangate.com.

[4] M.L. Bushnell and V.D. Agrawal. *Essentials of Electronic Testing.* Kluwer Academic Publishers, 2000.

[5] J. Chung, J. Xiong, V. Zolotov and J.A. Abraham, "Testability Driven Statistical Path Selection," in IEEE Design Automation Conf., pp. 417–422, June 2011.

[6] A. Czutro, N. Houarche, P. Engelke, I. Polian, M. Comte, M. Renovell, and B. Becker. A Simulator of Small-delay Faults Caused by Resistive-open Defects. Proc. IEEE European Test Symp., pages 113–118, Verbania, I, 2008.

[7] A. Czutro, I. Polian, M. Lewis, P. Engelke, S.M. Reddy and B. Becker, "Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability Analysis," Int'l Jour. of Parallel Programming, vol. 38, pp. 185–202, June 2010.

[8] D. Gizopoulos (Ed.), "Advances in Electronic Testing: Challenges and Methodologies," Series: Frontiers in Electronic Testing, Vol. 27, Springer, 2006.

[9] Z. He, T. Lv, H. Li, and X. Li, "An Efficient Algorithm for Finding a Universal Set of Testable Long Paths," IEEE Asian Test Symp., 2010, pp.319–324.

[10] F. Hopsch et al, "Variation-Aware Fault Modeling," in Asian Test Symp., 2011, pp. 87–93.

[11] U. Ingelsson, B.M. Al-Hashimi, S. Khursheed, S.M. Reddy, and P. Harrod. "Process variation-aware test for resistive bridges," IEEE Trans. on CAD, Vol. 28, No. 8, Aug. 2009, pp. 1269–1274.

[12] K. Killpack, C. Kashyap, and E. Chiprot, "Silicon speedpath measurement and feedback into EDA flows," Prod. ACM Design Atomation Conf., 2007, pp. 390–395.

[13] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng, "Modeling, testing, and analysis for delay defects and noise effects in deep submicron devices," IEEE Trans. on CAD, Vol. 22, No. 6, June 2003, pp. 756–769.

[14] J.-J. Liou, A. Krstic, L.-C. Wang, and K.-T. Cheng, "False-path-aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation," Design Automation Conf., 2002, pp. 566–569.

[15] S. Naffziger et al., "The implementation of a 2-core, multi-threaded Itanium family processor," IEEE Jour. Solid-State Circuits, Vol. 41, No. 1, January 2006, pp. 197–208.

[16] A. K. Pramanick and S. M. Reddy, "On the detection of delay faults," Proc. IEEE Int'l Test Conf., 1988, pp. 845–856.

[17] W. Qiu and D.M.H. Walker, "An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit," in Int'l Test Conf., pp. 592–601, 2003.

[18] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama, and S. Kajihara. "Invisible Delay Quality – SDQM Model Lights up What Could not be Seen," Proc. Int'l Test Conf. 2005, Paper 47.1.

[19] M. Sauer, J. Jiang, A. Czutro, I. Polian and B. Becker, "Efficient SAT-based Search for Longest Sensitisable Paths," in IEEE Asian Test Symp., 2011.

[20] Y. Shao, S.M. Reddy, I. Pomeranz, and S. Kajihara, "On Selecting Testable Paths in Scan Designs," Jour. Electronic Testing, vol. 19, no. 4, pp. 447–456, 2003.

[21] A. Srivastava, D. Sylvester, and D. Blaauw, "Statistical Analysis and Optimization for VLSI: Timing and Power," Springer, New York, NY, USA, 2005

[22] D. Sylvester, K. Agarwal, and S. Shaha, "Variability in Nanometer CMOS: Impact, Analysis, and Minimization," Integration, the VLSI Journal, Vol. 41, No. 3, May 2008, pp. 319–339.

[23] C. Visweswariah, "Fear, uncertainty and statistics," Proc. Int'l Symp. on Physical Design, 2007, p. 169.

[24] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor. "Interconnect-aware and Layout-oriented Test Pattern Selection for Small-delay Defects," Proc. Int'l Test Conf. 2008, Paper 28.3.