

Optimal Interconnect ATPG Under a Ground-Bounce Constraint

Henk D.L. Hollmann

Erik Jan Marinissen

Bart Vermeulen

Philips Research Laboratories

Prof. Holstlaan 4

5656 AA Eindhoven, The Netherlands

{Henk.D.L.Hollmann, Erik.Jan.Marinissen, Bart.Vermeulen}@philips.com

Abstract

In order to prevent ground bounce, Automatic Test Pattern Generation (ATPG) algorithms for wire interconnects have recently been extended with the capability to restrict the maximum Hamming distance between any two consecutive test patterns to a user-defined integer, referred to as *Simultaneously-Switching Outputs Limit* (SSOL). The conventional approach to meet this SSOL constraint is to insert additional test patterns between consecutive test patterns if their Hamming distance is too large; this approach often leads to many more test patterns than strictly necessary. This paper presents an algorithm that generates, for a user-defined number of interconnect wires, a minimal set of test patterns that respect a user-defined SSOL constraint. Experimental results show that, in comparison to the conventional approach, our algorithm leads to a significant reduction in the test pattern count and corresponding test application time. For example, for problem instances with 5000, 6000, 7000, and 8000 wires, the algorithm reduces the corresponding test application time on average with 45%.

1 Introduction

We consider the situation in which a number of interconnect wires (*nets*) between two or more digital components need to be tested. In order to test these nets, combinations of digital test stimuli are applied to the net inputs, and the responses are observed at the net outputs and compared to expected responses. We assume full controllability over the inputs of these nets, i.e., at the outputs of the components. Likewise, we assume full observability over the outputs of these nets, i.e., at the inputs of the components. The typical problem setting we have in mind is the testing of board-level interconnect wires between IEEE 1149.1-compliant Boundary Scan ICs [1] on a Printed Circuit Board (PCB). However, most of the theory described in this paper is also applicable to other situations of testing interconnect wiring, such as with interconnect wires between modules within one IC, a multi-conductor cable, etc.

Throughout this paper, we use the following terms for the test stimuli.

- *Test Pattern*. A test pattern is one set of test stimuli that are simultaneously applied in parallel on all nets of the interconnect network. In Figure 1, test patterns are the columns in the set of stimuli. Others [2, 3] refer to test patterns as *Parallel Test Vectors* (PTVs).
- *Code Word*. A code word is the list of test stimuli that are subsequently applied on one individual net. In Figure 1, code words are the rows in the set of stimuli. Others [2, 3] refer to code words as *Sequential Test Vectors* (STVs).

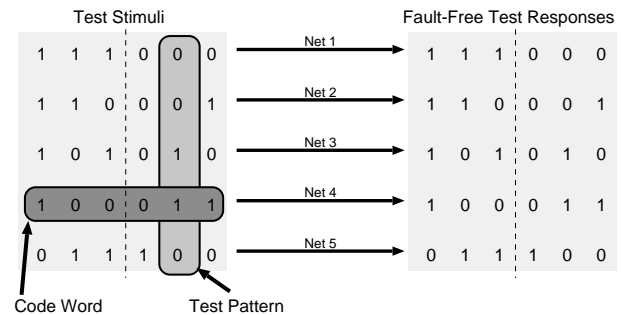


Figure 1: True/Complement Test for five nets.

In the literature, there is a multitude of ATPG algorithms for wire interconnects [3, 4, 5, 6, 7]. Commonly used test generation algorithms are based on the Counting Sequence Algorithm by Kautz [4]. It uses simple binary counting to generate distinct code words. The True/Complement Test Algorithm by Wagner [6] doubles the amount of test patterns, by generating code words that consist of the concatenation of the original Counting Sequence code words and their inverted values. True/Complement Tests guarantee detection of single-net opens and multiple-net shorts, while being free from *aliasing* [2].

For k nets, the Counting Sequence Algorithm requires $\lceil \log_2 k \rceil$ test patterns. The True/Complement Test Algorithm doubles that, and hence requires $2 \cdot \lceil \log_2 k \rceil$ test patterns. Figure 1 shows an example True/Complement test set

with $k = 5$ and $2 \cdot \lceil \log_2 5 \rceil = 6$ test patterns. The code words are 111000, 110001, 101010, 100011, and 011100, and they can be arbitrarily assigned to the five nets.

Recently, preventing incorrect Boundary Scan Test operation due to *ground bounce* has become a new constraint for test generation algorithms. Ground bounce refers to the phenomenon of shifting ground and power voltage levels, for example between the IC-internal ground and power levels and those at the board [8, 9, 10]. The negative effects of ground bounce can be prevented by introducing an upper limit on the Hamming distance between consecutive test patterns [8]. In this paper, we refer to this upper limit as the *Simultaneously-Switching Outputs Limit* (SSOL), denoted by a user-defined integer s .

Many vendors of automatic test pattern generation (ATPG) tools for board-level interconnects have added this SSOL constraint into their products [8, 11]. The conventional way of resolving SSOL violations is by inserting one or more additional test patterns between two consecutive test patterns that have an SSOL violation. Per inserted test pattern, s remaining SSOL violations can be resolved. Hence, between distinct consecutive test patterns p_1 and p_2 , we need to insert $\lceil d_H(p_1, p_2)/s \rceil - 1$ additional test patterns in order to resolve their SSOL violations (where $d_H(p_1, p_2)$ represents the Hamming distance of test patterns p_1 and p_2). Figure 2(a) shows an example test pattern set, generated by the True/Complement Test Algorithm for $k = 5$.

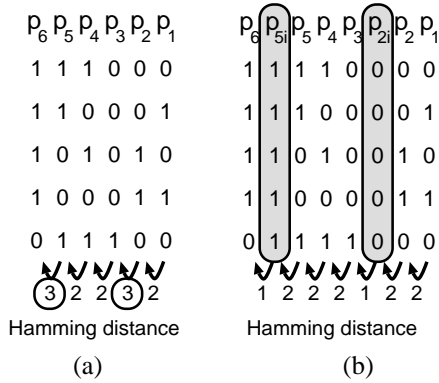


Figure 2: True/Complement test set for $k = 5$ (a) with violations against $s = 2$, and (b) the violations resolved by inserting two additional test patterns p_{2i} and p_{5i} .

The Hamming distances between consecutive test patterns are depicted as weights of the arrows at the bottom of the figure. For $s = 2$, there are two SSOL violations, one between patterns p_2 and p_3 , and one between patterns p_5 and p_6 . These violations are resolved in Figure 2(b) by inserting two additional test patterns, p_{2i} and p_{5i} , leading to a total of eight test patterns. The conventional approach is valid, but often leads to many more test patterns than strictly neces-

sary.

This paper presents an algorithm that generates, for a user-defined number of nets, a minimal set of test patterns that respects a user-defined SSOL constraint. The sequel of this paper is organized as follows. Section 2 describes prior work in ground-bounce-constrained interconnect ATPG and defines the problem addressed in this paper. In Section 3, we prove that SSOL-constrained test sets are only small if they consist of code words containing few transitions. This result motivates the definition in Section 4 of an ATPG algorithm for a given number of nets and given code word length. Then, in Section 5, we show that this algorithm, when applied with a specially-computed code word length, is guaranteed to produce a minimal set of test patterns for the given number of nets and SSOL constraint. Section 6 compares our algorithm with the conventional method and another recently published approach. It shows that we obtain, for problem instances ranging from 5,000 to 8,000 interconnect wires, on average 45% reduction in the number of test patterns and corresponding test application time. Section 7 concludes this paper.

2 Prior Work and Problem Definition

In [7, 12], Marinissen et al. show that the conventional approach of simply inserting additional test patterns to overcome SSOL violations can lead to an unnecessarily large test set. They identified two degrees of freedom, that can be exploited in order to reduce the number of additional test patterns that need to be inserted, while maintaining all detection and diagnostic properties of the original test set.

The first degree is named *code word subset selection*; in case the number of nets is not an exact power of two (i.e., $k = 2^b - r$ with $b \in \mathbb{N}^+$ and $0 < r < 2^{b-1}$), we can freely choose which k code words will be used out of the set of 2^b code words. The number of subsets to be considered is $\binom{2^b}{2^b - r}$. This number can be very large, and hence [7] proposed two heuristic algorithms for efficient selection of a code word set, viz. based on either *transition counts* or *difference counts* per code word.

The second degree of freedom is that test patterns can be freely re-ordered. The problem of re-ordering the set of test patterns such that the Hamming distance of consecutive test patterns is minimized is similar to the well-known \mathcal{NP} -hard *Traveling Salesman Problem* (TSP) [13]. In [7], a greedy TSP heuristic was proposed to solve this problem.

Marinissen et al. [7] provide equations for theoretical upper and lower bounds for the minimal number of test patterns for k nets under SSOL constraint s . The difference between these upper and lower bound is the room for optimization, and is largely determined by (1) the SSOL value s and (2) r , the difference between the number of nets k

and the next higher power of two. Experimental results in [7] show that exploiting these two degrees of freedom for $k \in \{5000, 6000, 7000, 8000\}$ results for *specific values* of s in a reduction of the test pattern count of 45%, when compared to the conventional approach of simply inserting additional test patterns. Especially exploiting the first degree of freedom by means of the transition count heuristic algorithm turns out to be effective. However, for many values of s , the degrees of freedom do not provide any room for optimization. The average reduction of the test pattern count for $k \in \{5000, 6000, 7000, 8000\}$ and s varying from 5% to 50% of k is respectively only 17%, 12%, 9%, and 2%.

The problem addressed in this paper can formally be stated as follows.

Problem [Minimal SSOL-Constrained Test]

Given k interconnect nets and an SSOL value s with $s \in \mathbb{N}^+$. Find a test for k nets, such that (1) the test consists of k distinct code words, (2) each pair of consecutive test patterns p_1 and p_2 have a Hamming distance $d_H(p_1, p_2) \leq s$, and (3) the number of test patterns is minimized.

Note that the requirement for k distinct code words guarantees that the test produced provides the same fault coverage as the Counting Sequence Algorithm by Kautz [4], viz. coverage of all multiple-net shorts modeled as wired-AND or wired-OR. The problem definition does *not* specify that the code words need to be generated by counting, and hence opens the possibility to generate tests in other ways.

Also note that it is easy to extend our approach and corresponding algorithm with coverage beyond wired-AND or wired-OR shorts. Single-net opens modeled as stuck-at faults can be covered by excluding all-zero and all-one code words. Aliasing can be prevented by adding a complement to the test.

3 Low-Transition-Count Code Words

In this section, we analyse the relationship between the binary contents of a given test set and the amount of test patterns that need to be inserted to resolve any violations of the maximum Hamming distance constraint for consecutive test patterns. We show that the amount of additional test patterns that need to be inserted is strongly related to the number of transitions in the code words. Our main conclusion is that in order to obtain a test set containing few test patterns after possible insertion of additional test patterns to satisfy the SSOL constraint, we necessarily have to choose code words containing few transitions.

Consider a fixed test, represented by a binary $k \times b$ matrix C , where the rows correspond to the k code words and the columns to the b test patterns. We denote the k code words by c_1, \dots, c_k . Let $C_{i,j}$ refer to bit j in code word c_i , for

$1 \leq i \leq k$ and $1 \leq j \leq b$. We denote the test patterns by p_1, \dots, p_b , where $p_j = (C_{1,j}, \dots, C_{k,j})$ is the j th column of C .

As transitions and transition-counts are important in our analysis, we often refer to an alternative notation of test C , viz. the one in which a test is represented by a binary $k \times (b - 1)$ transition matrix T and k initial code word values $C_{i,1}$. Here, the relation between the matrices C and T is defined by

$$T_{i,j} = \begin{cases} 1, & \text{if } C_{i,j} \neq C_{i,j+1}; \\ 0, & \text{otherwise.} \end{cases}$$

The two alternative test set representations are illustrated in Figure 3.

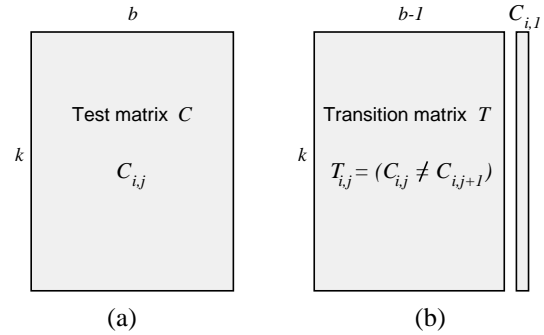


Figure 3: A test can be encoded by (a) a binary $k \times b$ test matrix C , or (b) a binary $k \times (b - 1)$ transition matrix T and a set of k initial code word values $C_{i,1}$.

For $j = 1, \dots, b - 1$, let $d_j = d_H(p_j, p_{j+1})$ denote the Hamming distance between consecutive test patterns p_j and p_{j+1} , i.e., the number of code words where p_j and p_{j+1} differ. For simplicity, we assume that consecutive test patterns are distinct, that is, we assume that $d_j > 0$ for all j . (Our analysis can easily be adapted to the case where some of the distances d_j are zero.) Given this assumption, if test set C is used in combination with an SSOL constraint s , then $\lceil d_j/s \rceil - 1$ test patterns have to be inserted between p_j and p_{j+1} . Hence, after insertion the total number of test patterns P_s required under SSOL constraint s based on the given test set satisfies

$$P_s = b + \sum_{j=1}^{b-1} \left(\left\lceil \frac{d_j}{s} \right\rceil - 1 \right) = 1 + \sum_{j=1}^{b-1} \left\lceil \frac{d_j}{s} \right\rceil. \quad (1)$$

Let τ_i , for $i = 1, \dots, k$, denote the number of transitions in code word c_i , i.e., $\tau_i = \#\{j \in \{1, \dots, b - 1\} \mid C_{i,j} \neq C_{i,j+1}\}$.

Our aim in this section is to derive lower and upper bounds for P_s in terms of the total number of transitions $\tau_{\text{tot}} = \sum_{i=1}^k \tau_i$ in test C . Our analysis is based on the following simple fact. Define the total distance d_{tot} of the test set by $d_{\text{tot}} = \sum_{j=1}^{b-1} d_j$.

Proposition 1: The total distance d_{tot} is equal to the total number of transitions τ_{tot} , i.e., $d_{\text{tot}} = \tau_{\text{tot}}$.

Proof: Observe that the number of transitions in a row of C is given by the number of ones in the corresponding row of T , while the Hamming distance between consecutive test patterns is given by the number of ones in the relevant column of T . So τ_{tot} and d_{tot} both count the total number of ones in T ; one does so by rows and the other by columns. \square

We use Proposition 1 to derive upper and lower bounds for P_s that depend only on τ_{tot} .

Theorem 2: Let a test set consisting of code words of length b contain a total of τ_{tot} transitions. Then the number P_s of test patterns after insertions to satisfy an SSOL constraint s satisfies

$$1 + \left\lceil \frac{\tau_{\text{tot}}}{s} \right\rceil \leq P_s \leq b + \left\lfloor \frac{\tau_{\text{tot}}}{s} \right\rfloor. \quad (2)$$

Proof: By Equation (1), we have that $P_s = 1 + \sum_{j=1}^{b-1} \lceil d_j/s \rceil$. So, using Proposition 1, we have on the one hand that $P_s \geq 1 + \sum_{j=1}^{b-1} d_j/s = 1 + \tau_{\text{tot}}/s$; on the other hand, $P_s < 1 + \sum_{j=1}^{b-1} (d_j/s + 1) = b + \tau_{\text{tot}}/s$. Since P_s is integer, the theorem follows. \square

From this theorem, we conclude that for fixed code word length b , the total number of test patterns after insertion under an SSOL constraint s can be small only by using a collection of code words for which the total number of transitions is small. This conclusion is in line with the observation in [7] that, of the various approaches considered, code word subset selection by means of the Transition-Count heuristic was the most effective.

4 ATPG for Fixed Code Word Length

The conventional approach to generate a ground-bounce-constrained test set for k nets is to create k code words of length $b = b_{\text{min}} = \lceil \log_2 k \rceil$ based on straightforward counting. All violations of the constraint that Hamming distances between consecutive test patterns should be less than or equal to SSOL s are resolved by means of inserting additional test patterns. In [7, 12], Marinissen et al. showed how additional optimization procedures performed before insertion can reduce the Hamming distance between consecutive test patterns, and hence reduce the number of additional test patterns that need to be inserted.

In this section, we explain how to choose almost optimally k code words of any fixed length b , in order to minimize under a given SSOL constraint s the number of patterns that need to be inserted and overall to minimize the total number of test patterns required. Note that in order to have at least

k code words available, we must have that $2^b \geq k$. So, we assume that $b \geq b_{\text{min}} = \lceil \log_2 k \rceil$.

The analysis in Section 3 suggests to choose the k code words in such a way that the total number of transitions in these code words is minimized. To push that idea to the limit, we propose the following code word selection algorithm. Choose all words with 0 transitions, then all words with 1 transition, up to all words with $t-1$ transitions, and finally some additional words with t transitions. Here t is such that the number k of required code words is larger than the number of words of length b with at most $t-1$ transitions, but not larger than the number of words of length b with at most t transitions.

Note that the total number of transitions in a test set chosen in this way does not depend on the particular choice of code words, and is by construction the minimal number of transitions in any set of k code words of length b . We denote this number by $\tau_{\text{tot}}(b)$.

In order to analyse this algorithm and to be able to compute $\tau_{\text{tot}}(b)$, we need the following result.

Proposition 3: The number of words of length b with i transitions is equal to $2^{\binom{b-1}{i}}$.

Proof: The number of words of length $b-1$ with precisely i ones is equal to $\binom{b-1}{i}$. From the correspondence between the $k \times b$ code word matrix C and the $k \times (b-1)$ transition matrix T as indicated in Figure 3, we see that each of these words corresponds to precisely two code words with i transitions. \square

We can now formalize our method as follows. Let k_i be the number of code words with i transitions that are included in our test set. In order to minimize the total number of transitions $\sum_{i=0}^t i k_i$ in our test set, our code word selection method chooses as code words all words containing at most $t-1$ transitions, and some of the words containing t transitions. So $k_i = 2^{\binom{b-1}{i}}$ for $0 \leq i \leq t-1$ and t is such that

$$k = \sum_{i=0}^t k_i = \sum_{i=0}^{t-1} 2^{\binom{b-1}{i}} + k_t, \text{ with } 0 < k_t \leq 2^{\binom{b-1}{t}}. \quad (3)$$

This code word selection method is illustrated in Figure 4. For both test set representations, the figure shows by means of gray shading which code words are generated.

For $\tau_{\text{tot}}(b)$ we now obtain the expression

$$\tau_{\text{tot}}(b) = \sum_{i=0}^t i k_i = \sum_{i=0}^{t-1} i \cdot 2^{\binom{b-1}{i}} + t \cdot k_t \quad (4)$$

Next we use this code word selection method to derive

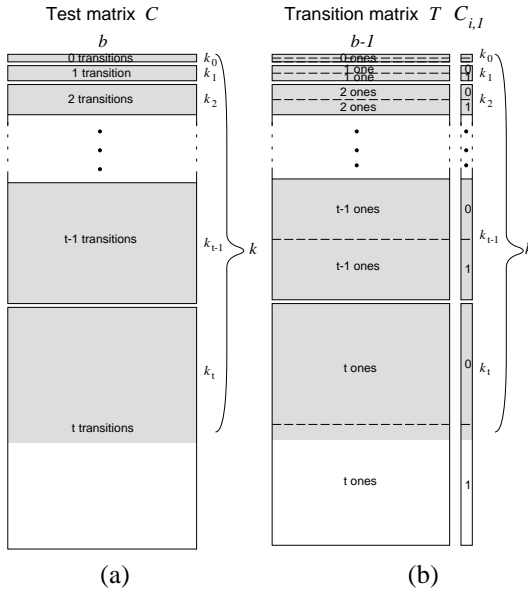


Figure 4: The code word selection method.

bounds on the minimal number of test patterns $p(k, s, b)$ required in a test set for k nets, using code words of length b , and satisfying an SSOL constraint s . Since we showed above that there exists a test set (in fact, many test sets) using code words of length b for which $\tau_{\text{tot}} = \tau_{\text{tot}}(b)$ and since obviously no such test can have a smaller value of τ_{tot} , we conclude from Theorem 2 that

$$1 + \left\lceil \frac{\tau_{\text{tot}}(b)}{s} \right\rceil \leq p(k, s, b) \leq b + \left\lceil \frac{\tau_{\text{tot}}(b)}{s} \right\rceil \quad (5)$$

Note that any algorithm using this code word selection method and some specific method to choose the k_t additional code words achieves a total number of test patterns after insertion that lies within the bounds in Equation (5). In particular, this holds for the Transition-Count code word subset selection heuristic algorithm in [7] which is based on this method. In this paper, we improve the upper bound in Equation (5) by proposing an algorithm that has a better worst-case behavior. (And as we show later, although the algorithm is not optimal for every b , it is optimal where it counts.) In fact, we will show the following result.

Theorem 4: Let k and b be fixed, and let $\tau_{\text{tot}}(b)$ be defined as in Equation (4). Then

$$\max \left(b, 1 + \left\lceil \frac{\tau_{\text{tot}}(b)}{s} \right\rceil \right) \leq p(k, s, b) \leq 1 + (b-1) \left\lceil \frac{\tau_{\text{tot}}(b)}{(b-1)s} \right\rceil.$$

The lower bound in this theorem combines the result in Equation (5) with the obvious fact that $p(k, s, b) \geq b$. In the remainder of this section we describe a method based on our code word selection algorithm which produces at most

$1 + (b-1) \left\lceil \frac{\tau_{\text{tot}}(b)}{(b-1)s} \right\rceil$ test patterns satisfying SSOL s , thus establishing the upper bound.

For our algorithm, the *total distance* $d_{\text{tot}} = \sum_{j=1}^{b-1} d_j$ will always be the same (namely $\tau_{\text{tot}}(b)$), but the values of individual distances d_j depend on the choice of the code words. Our algorithm fixes the code words with less than t transitions. The only remaining freedom is the selection of k_t code words with t transitions.

Further in Proposition 5, we prove that the part of the distances d_j on code words with at most $t-1$ transitions is in fact constant, independent of j . Therefore the k_t additional code words we still have to choose determine the resulting distances d_j and consequently determine the resulting number of test patterns after insertion, as seen in Equation (1). Subsequently we explain how to choose the remaining k_t code words such that the d_j values remain about equal.

First, we write $d_j = \sum_{\ell=0}^t d_{j,\ell}$, where $d_{j,\ell}$ is the part of the Hamming distance between p_j and p_{j+1} on code words containing ℓ transitions, that is

$$d_{j,\ell} = \#\{i \in \{1, \dots, k\} \mid C_{i,j} \neq C_{i,j+1} \text{ and } \tau_i = \ell\}.$$

Proposition 5: The part of the distances d_j on code words with $\ell \geq 1$ transitions and beginning with a fixed value is equal to $\binom{b-2}{\ell-1}$. So we have that $d_{j,0} = 0$ and $d_{j,\ell} = 2 \binom{b-2}{\ell-1}$ for $\ell = 1, \dots, t-1$, independent of $j \in \{1, \dots, b-1\}$.

Proof: There are only two code words with 0 transitions, viz. the all-zero and the all-one code word, and hence $d_{j,0} = 0$. The number of words in T of length $b-1$ with ℓ ones that contain a 1 in a given position $j \in \{1, \dots, b-1\}$ is $\binom{b-2}{\ell-1}$. Since each such word in T corresponds to two code words in C with ℓ transitions (one beginning with a 0, the other with a 1) that each contribute one to $d_{j,\ell}$, the result follows. \square

As a consequence, we have that

$$d_j = \sum_{\ell=0}^t d_{j,\ell} = \sum_{\ell=0}^{t-1} 2 \binom{b-2}{\ell-1} + d_{j,t}, \quad (6)$$

Next, consider the choice of the k_t code words containing t transitions. In case $k_t \geq \binom{b-1}{t}$, we choose in addition all words with t transitions and starting with a zero. Note that due to Proposition 5, the part of the distances d_j due to the code words chosen up to now is constant, independent of j . Now, let k_t^* be defined by

$$k_t^* = \begin{cases} k_t, & \text{if } 0 \leq k_t < \binom{b-1}{t}; \\ k_t - \binom{b-1}{t}, & \text{otherwise.} \end{cases} \quad (7)$$

For later use, we note that $0 \leq k_t^* \leq \binom{b-1}{t}$. Then, we still have to choose k_t^* additional code words with t transitions

(all starting with 0 if $k_t < \binom{b-1}{t}$ and with 1 if $k_t \geq \binom{b-1}{t}$), or, equivalently, k_t^* words of length $b-1$ containing t ones (representing the transitions in the code words). For convenience, we identify words of length $b-1$ containing t ones with subsets of $\{1, \dots, b-1\}$ of size t . Then, in terms of sets, we still have to choose k_t^* distinct subsets of size t from $\{1, \dots, b-1\}$. Note that the (variable) part of the distance d_j on positions corresponding to these k_t^* words becomes, in the terms of sets, the frequency f_j of the occurrence of the element j in these k_t^* sets. We need the following definition.

Definition 1: A vector $f = (f_1, \dots, f_n)$ is called *balanced* if there is an integer e such that $f_i \in \{e, e+1\}$ for all $i = 1, \dots, n$. \square

We now ensure that the remaining k_t^* code words are chosen in such a way that the part of the distances d_j on these k_t^* code words is balanced. That is, in terms of sets, we ensure that the frequencies f_j are balanced. Our algorithm is based on the following result.

Theorem 6: Let $1 \leq t \leq n$. Suppose we are given a balanced integer vector $f = (f_1, \dots, f_n)$ with weight $f_1 + \dots + f_n = mt$ such that $0 \leq m \leq \binom{n}{t}$. Then there is a collection $S = \{S_1, \dots, S_m\}$ of m distinct subsets of size t of $\{1, \dots, n\}$ such that each i occurs in f_i of the sets S_j .

It seems highly unlikely that either the above result or our method of proving it is new, but at the time of writing we were not aware of any suitable reference. In Appendix A, we prove this result *constructively*, that is, we describe an algorithm to actually construct these sets S_1, \dots, S_m . We use this algorithm to choose $m = k_t^*$ code words, by choosing m t -subsets from $\{1, \dots, b-1\}$ such that the frequencies f_1, \dots, f_{b-1} are balanced.

Summarizing, our algorithm $\text{BALANCED}(k, s, b)$ does the following. First, choose numbers t and k_t according to Equation (3). Next, determine the number $m = k_t^*$ from Equation (7). Now, we take as code words all words of length b with at most $t-1$ transitions, and if $m \neq k_t$ also all words of length b with t transitions and beginning with a 1. Finally, determine another m words of length b with t transitions and beginning with a 0 (if $m < k_t$) or a 1 (if $m \geq k_t$) by using the algorithm in Appendix A. Here, if $mt = q(b-1) + r$, we choose $f_i = q+1$ for r indices i and $f_i = q$ for the remaining $b-1-r$ indices i . This algorithm then ensures that the distances d_j are balanced.

Now if $\tau_{\text{tot}}(b)$ is the number determined as in Equation (4), then the average distance d_{av} is equal to $\tau_{\text{tot}}(b)/(b-1)$; hence $d_j \leq \lceil \tau_{\text{tot}}(b)/(b-1) \rceil$ and therefore

$$\left\lceil \frac{d_j}{s} \right\rceil \leq \left\lceil \frac{\tau_{\text{tot}}(b)}{(b-1)s} \right\rceil.$$

We conclude that this algorithm produces a test set such that,

after insertions to satisfy an SSOL constraint s , the total number of patterns is (at most) $1 + (b-1) \lceil \frac{\tau_{\text{tot}}(b)}{(b-1)s} \rceil$, thus proving the upper bound in Theorem 4.

5 ATPG for Variable Code Word Length

In the previous section we already remarked that in order to have at least k code words available, the code word length b has to be chosen such that $b \geq b_{\min} := \lceil \log_2 k \rceil$. However, if we choose b equal to this minimum b_{\min} , then necessarily a lot of code words contain many transitions. Now we make the following crucial observation. According to Proposition 3, the number of available code words of length b with i transitions is given by $2 \binom{b-1}{i}$. Hence the number of available code words with i transitions *increases* when the value of b increases. So we conclude that by choosing k code words containing the minimum number of transitions as in our code word selection method in Section 4, the resulting total number of transitions $\tau_{\text{tot}}(b)$, and hence certainly the quantity $\tau_{\text{tot}}(b)/(b-1)$ figuring in Theorem 4, decreases with increasing b .

Proposition 7: The function $\tau_{\text{tot}}(b)$ is a non-increasing function of b for $b \geq b_{\min}$, that is, if $b_{\min} \leq b < b'$, then $\tau_{\text{tot}}(b) \geq \tau_{\text{tot}}(b')$.

This observation suggests that

$$p(k, s) = \min_{b \geq b_{\min}} p(k, s, b),$$

the minimal number of test patterns required in any test set for k nets and satisfying an SSOL constraint s , is realised for a value b that is often much larger than b_{\min} .

To indicate the behavior of the bounds in Theorem 4 for varying b , we have plotted in Figure 5 for $k = 6000$ and $s = 600$ both the upper bound

$$U(b) = 1 + (b-1) \left\lceil \frac{\tau_{\text{tot}}(b)}{(b-1)s} \right\rceil$$

and one of the lower bounds

$$L(b) = 1 + \left\lceil \frac{\tau_{\text{tot}}(b)}{s} \right\rceil,$$

for $b \geq b_{\min}$.

It is intuitively obvious that the best choice for b is close to the smallest value b^* of b for which the average distance $\tau_{\text{tot}}(b)/(b-1)$ is at most s . Note that for this value b^* , we have that $U(b^*) = b^*$. This idea is vindicated by Figure 5. In Appendix B, we give a rough estimate of the value of b^* and hence for corresponding total number of test patterns $p(k, s, b^*) = b^*$, given k and s .

It turns out that the optimal value of b , that is, the value of b that minimizes $p(k, s, b)$, is in fact always *equal* to the

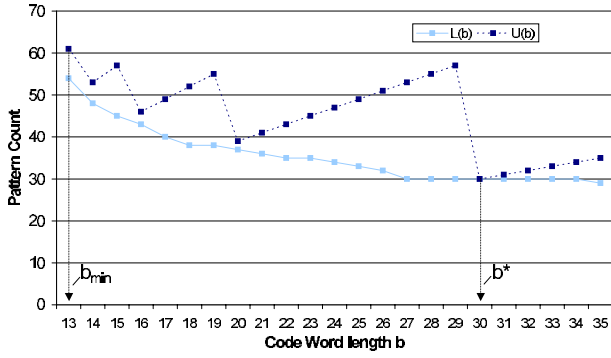


Figure 5: Upper and lower bounds for $p(6000, 600, b)$, for $b \geq b_{\min}$.

number b^* as defined above, that is, we have the following result.

Theorem 8: Let k and s be positive integers, with $k \geq 2$. the function $\tau_{\text{tot}}(b)$ as in Equation (4). Let $b_{\min} = \lceil \log_2 k \rceil$, and define the number b^* to be the smallest integer $b \geq b_{\min}$ for which $\tau_{\text{tot}}(b) \leq (b-1)s$. Then the minimum number $p(k, s)$ of test patterns required in any test set for k nets and satisfying an SSOL constraint s satisfies $p(k, s) = b^*$.

Proof: First we show that $p(k, s) \geq b^*$. Suppose that $b = p(k, s)$ and that C is a $k \times b$ matrix with distinct rows satisfying the SSOL constraint, that is, for $j = 1, \dots, b-1$, the distance d_j between the d_j -th and the d_{j+1} -th column satisfies $d_j \leq s$. SSOL constraint. Then, using Proposition 1, we have that τ_{tot} , the total number of transitions in the matrix C , satisfies $\tau_{\text{tot}} = d_{\text{tot}} \leq (b-1)s$. Since obviously $\tau_{\text{tot}}(b) \leq \tau_{\text{tot}}$, we conclude that $\tau_{\text{tot}}(b) \leq (b-1)s$, and hence that $p(k, s) = b \geq b^*$.

Next, we use our algorithm $\text{BALANCED}(k, s, b^*)$ to show that $p(k, s) \leq b^*$. Suppose that $\tau_{\text{tot}}(b) \leq (b-1)s$. Again by Proposition 1, we see that a $k \times b$ matrix containing $\tau_{\text{tot}}(b)$ transitions has an average distance d_{av} satisfying $d_{\text{av}} \leq s$. The algorithm $\text{BALANCED}(k, s, b)$ delivers a matrix where the distances d_j satisfy $d_j \in \{\lfloor d_{\text{av}} \rfloor, \lceil d_{\text{av}} \rceil\}$, hence with $d_j \leq s$ for all j . So whenever $\tau_{\text{tot}}(b) \leq (b-1)s$, we have that $p(k, s) \leq b$. In particular, we have that $p(k, s) \leq b^*$. \square

As a result of this theorem, to generate a test set for k nets that satisfies an SSOL constraint s and contains a minimum number of test patterns, proceed as follows. First, determine the number b^* as defined in Theorem 8. Then apply the algorithm $\text{BALANCED}(k, s, b^*)$ as described in Section 4 to generate a test set with b^* patterns where consecutive patterns have a distance of at most s .

6 Experimental Results

This section presents experimental results for our new SSOL-constrained ATPG algorithm and compares those

with other methods. We use True/Complement Tests [6] as the basis for our comparison. This is motivated by three reasons.

- True/Complement Tests provide complete fault detection for the most-commonly used fault models (multiple-net wired-AND and wired-OR shorts and single-net stuck-at opens) and prevent aliasing.
- True/Complement Tests are the basis of many professional (commercially available) ATPG tools for wire interconnects.
- Previous publications to which we can compare ourselves are also based on True/Complement Tests.

We compare three methods.

1. Conventional Method.

This method generates a True/Complement Test by means of conventional counting, followed by complementing. Subsequently, the SSOL constraint is met by conventional insertion of additional test patterns. This method is the basis of many professional (commercially available) tools. In [7], it is referred to as “T/C+I”.

2. Best Method of Marinissen et al. [7].

This method generates a True/Complement Test by means of conventional counting, followed by complementing. Subsequently, transition-count-based code word subset selection and test pattern re-ordering are applied. Remaining SSOL violations are resolved by means of conventional insertion of additional test patterns. This method is the one that performed best from the various methods described in [7], and in that paper is referred to as “T/C+T+R+I”.

3. ATPG for Variable Code Word Length.

This method generates a “True” test set as described in Section 5 of this paper. This test is guaranteed to be free of SSOL violations and hence does not require insertion of additional test patterns. Subsequently, the “Complement” part is generated by adding the complementary test set, which, again, is free of SSOL violations. At the border between “True” and “Complement” test patterns, usually a few additional test patterns need to be inserted.

The left-hand side of Figure 6 shows experimental results for a problem instances with $k \in \{5000, 6000, 7000, 8000\}$ nets. For SSOL value s varying between 5% and 50% of k , Figure 6 plots the test pattern count $p(k, s)$ for all three methods. The right-hand side of Figure 6 shows the improvements in test pattern count of the T/C+T+R+I method and our new ATPG for Variable Code Word Length relative to the base-line conventional method T/C+I.

Figure 6 shows that, for all SSOL values s , the conventional method yields the highest number of test patterns.

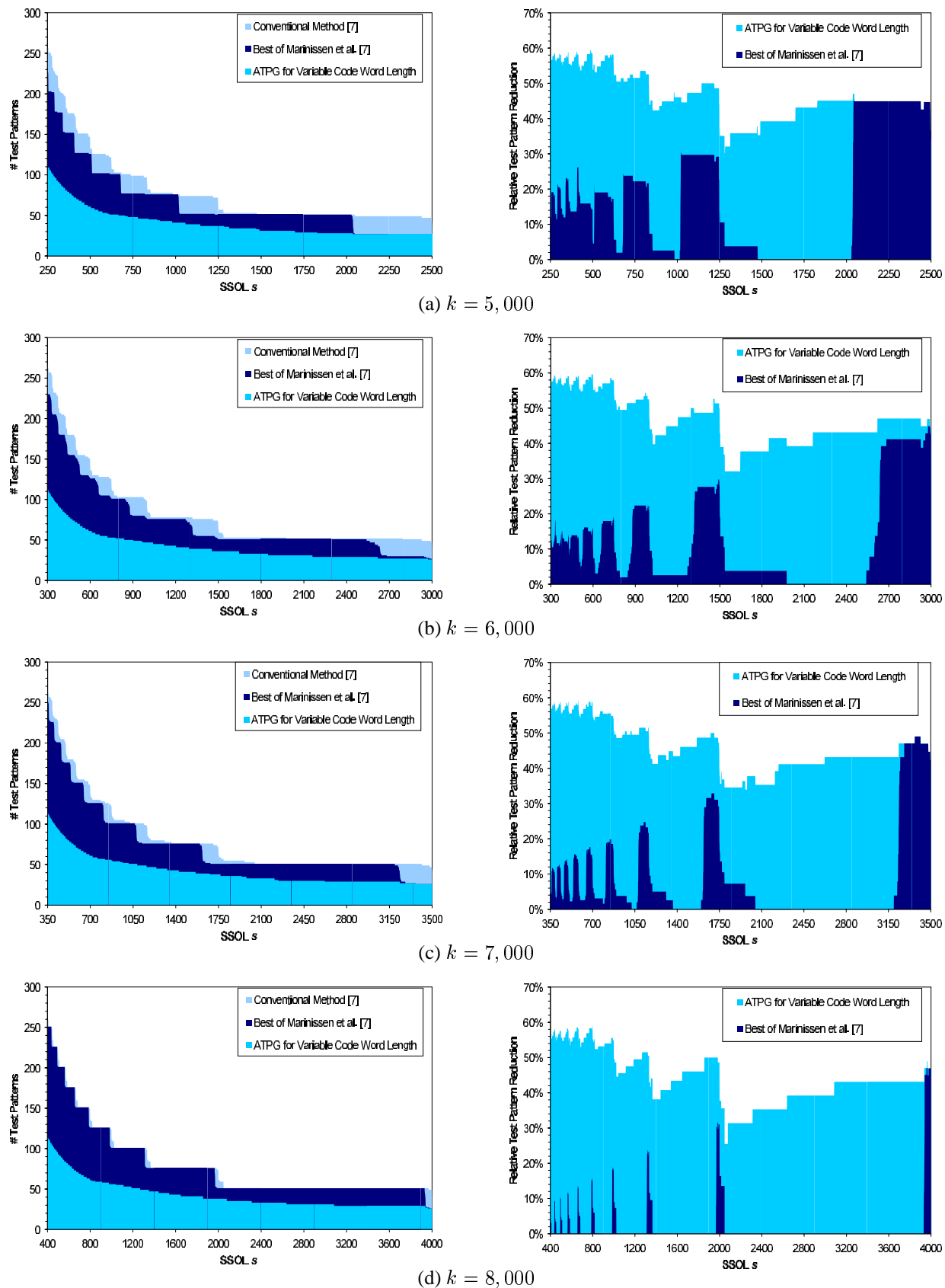


Figure 6: Test pattern count $p(k, s)$ and relative test pattern count reduction for $k \in \{5000, 6000, 7000, 8000\}$ nets and SSOL s varying from 5% to 50% of k .

For some values of k and s , the T/C+T+R+I method of [7] provides substantial improvement over the conventional method; however, there are also many values of k and s where T/C+T+R+I does not yield any improvement at all. Our new ATPG for Variable Code Word Lengths clearly outperforms the other approaches; for $k = 5000, 6000, 7000, 8000$, the average relative improvement over the conventional method is respectively 46%, 46%, 45%, and 44%. Our new ATPG algorithm also consistently improves on the conventional method for all values of k and s .

7 Conclusion

In this paper, we have presented an ATPG algorithm that generates, for a user-defined number of nets k , a minimal set of test patterns $p(k, s)$ that respect a user-defined SSOL constraint s .

The number of test patterns after insertion is strongly related to the number of transitions in the code words. Hence, in order to obtain a valid test set with a small number of test patterns, we have to select code words with few transitions. Based on the above observation, we have presented an algorithm that, for given k , s , and b , chooses as code words all words of length b with at most $t-1$ transitions and subsequently carefully selects some words of length b with t transitions, such that we obtain a total of exactly k code words, with balanced Hamming distance s between consecutive test patterns. The resulting test set consists of the test patterns thus obtained, plus additional test patterns inserted to resolve remaining SSOL violations. Using the first algorithm as a subroutine, we have presented a second algorithm that, for given k and s , finds a value $b = b^*$ such that if the subroutine is invoked for this b^* , we can guarantee that no additional test patterns need to be inserted to meet the SSOL constraint; moreover, we have shown that this second algorithm generates provably minimal test sets.

Using the commonly-used True/Complement Test algorithm as an example, we have presented experimental results for $k \in \{5000, 6000, 7000, 8000\}$ and s varying from 5% to 50% of k . The experiments show that our algorithm yields on average 45% reduction in the number of test patterns when compared to the conventional method.

Acknowledgements

We thank Ben Bennetts of Bennetts Associates for drawing our attention to this problem at the European Test Workshop 2002. We also thank our colleagues Sebastian Egner, Jack van Lint, Ludo Tolhuizen, and Evgeny Verbitskiy for fruitful discussions on mathematical aspects of this problem.

A Proof of Theorem 6

For a vector $f = (f_1, \dots, f_n)$, we define its weight $w(f)$ and its average coefficient $av(f)$ as

$$w(f) = \sum_{i=1}^n f_i, \quad av(f) = w(f)/n.$$

We need the following simple result on balanced integer vectors. **Lemma 1:** Let the integer vector $f \in \mathbf{Z}^n$ be balanced, and let M_- and M_+ be integers. Then we have $M_- \leq f_i \leq M_+$ for all $i = 1, \dots, n$ if and only if $M_- \leq av(f) \leq M_+$. \square

Proof: If $M_- \leq f_i \leq M_+$ for all $i = 1, \dots, n$, then obviously also $M_- \leq av(f) \leq M_+$. The converse follows immediately from the observation that if f is balanced, then $f_i \in \{\lfloor av(f) \rfloor, \lceil av(f) \rceil\}$ for all $i = 1, \dots, n$. \square

In what follows, we prove Theorem 3 by describing a recursive procedure

set_list solution(int n , int t , int m , int_vector f)

that, given integers n, t, m with $1 \leq t \leq n$ and $0 \leq m \leq \binom{n}{t}$ and a balanced integer vector $f = (f_1, \dots, f_n)$ with weight $w(f) = mt$, delivers a (reversely lexicographically ordered) list of sets $\mathcal{S} = \langle S_1, \dots, S_m \rangle$ of m distinct subsets of $\{1, \dots, n\}$, each of size t , such that each $i \in \{1, \dots, n\}$ occurs in precisely f_i members of \mathcal{S} .

For later use, we first derive two bounds on the components f_i of f . First note that by our assumptions on m and f , we have that

$$0 \leq av(f) = mt/n \leq \frac{t}{n} \binom{n}{t} = \binom{n-1}{t-1},$$

hence using Lemma 1, we conclude that

$$0 \leq f_i \leq \binom{n-1}{t-1} \quad (8)$$

for all $i = 1, \dots, n$. Similarly, consider the vector $f^* = (m - f_1, \dots, m - f_n)$. Note that f^* is again balanced, and $av(f^*) = (nm - w(f))/n = (n-t)m/n$. So by our assumption on m , we find that

$$0 \leq av(f^*) = (n-t)m/n \leq \frac{n-t}{n} \binom{n}{t} = \binom{n-1}{t},$$

so using Lemma 1, we conclude that

$$0 \leq m - f_i \leq \binom{n-1}{t} \quad (9)$$

for all $i = 1, \dots, n$.

First, we handle the cases where $t = 1$ or $t = n$. By (8), in these cases we have that $f_i \in \{0, 1\}$ for all i . So if $t = 1$,

we can take $\mathcal{S} = \langle \{i\} \mid f_i = 1 \rangle$; similarly, if $t = n$, then we have that $m = 0$ or $m = 1$, so we can take $\mathcal{S} = \emptyset$ (if $m = 0$) or $\mathcal{S} = \{1, \dots, n\}$ (if $m = 1$).

Next, we handle the cases where $1 < t < n$ by recursion. The idea behind our method is to think of the list \mathcal{S} that is to be generated as a union $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$, where \mathcal{S}_0 consists of the $m - f_n$ sets in \mathcal{S} that do not contain n and \mathcal{S}_1 consists of the f_n sets in \mathcal{S} that contain n . Now \mathcal{S}_0 and the list $\mathcal{S}'_1 = \langle S \setminus \{n\} \mid S \in \mathcal{S}_1 \rangle$ can be considered as a collection of subsets of $\{1, \dots, n - 1\}$ of size t or size $t - 1$, respectively. So to generate the list \mathcal{S} recursively, we need to write the vector $f' = (f_1, \dots, f_{n-1})$ as a sum of two balanced vectors g (for \mathcal{S}_0) and h (for \mathcal{S}'_1) with $w(g) = t(m - f_n)$ and $w(h) = (t - 1)f_n$; note that by (8) and (9), both $m - f_n$ and f_n already satisfy the required bounds. Once we have found such vectors g and h , we obtain the desired list \mathcal{S} from recursive calls $\text{solution}(n - 1, t, m - f_n, g)$ and $\text{solution}(n - 1, t - 1, f_n, h)$.

To construct the vector h , write $w(h) = (t - 1)f_n = q(n - 1) + r$ with $0 \leq r < n - 1$. We construct h to have r components equal to $q + 1$ and $n - 1 - r$ components equal to q , so that h is balanced and has the desired weight. Moreover, in order to ensure that $g = f' - h$ is also balanced, we choose the set of r components of h where h is maximal to be a subset of the positions where f is maximal, if the number of these latter positions is at least r , or to include all the latter positions, if their number is less than r . It is not difficult to see that this construction guarantees that g indeed is also balanced. Since $w(g) = w(f' - h) = w(f') - w(h) = (mt - f_n) - (t - 1)f_n = t(m - f_n)$, the vector g automatically has the desired weight.

B An estimate of b^*

Let the number k of codewords and the SSOL value s be fixed. As we saw earlier, for each b the value of t is chosen such that

$$k \approx 2^{\binom{b-1}{t}}, \quad (10)$$

so that by Stirling's approximation [14]

$$\log_2 k \approx 1 + (b - 1)h(t/(b - 1)), \quad (11)$$

where

$$h(x) = -x \log_2 x - (1 - x) \log_2(1 - x).$$

Also, for the total distance we have that

$$d_{\text{tot}} = \tau_{\text{tot}} \approx tk. \quad (12)$$

So we have that $d_{\text{tot}} \approx (b - 1)s$ if

$$t/(b - 1) \approx s/k. \quad (13)$$

Combining Equations (11) and (12), we obtain that

$$P^* = b^* \approx 1 + (-1 + \log_2 k)/h(s/k). \quad (14)$$

As it turns out, this estimate for b^* is somewhat too small, but always at least about 80 % of the true value.

References

- [1] IEEE Computer Society. *IEEE Standard Test Access Port and Boundary-Scan Architecture - IEEE Std. 1149.1-2001*. IEEE, New York, NY, USA, July 2001.
- [2] Najmi Jarwala and Chi W. Jau. A New Framework for Analyzing Test Generation and Diagnosis Algorithm for Wiring Interconnects. In *Proceedings IEEE International Test Conference (ITC)*, pages 63–70, October 1989.
- [3] José T. de Sousa and Peter Y.K. Cheung. *Boundary-Scan Interconnect Diagnosis*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.
- [4] William H. Kautz. Testing of Faults in Wiring Interconnects. *IEEE Transactions on Computers*, Vol. C-23(No. 4):358–363, April 1974.
- [5] P. Goel and M.T. McMahon. Electronic Chip-In-Place Test. In *Proceedings IEEE International Test Conference (ITC)*, pages 83–90, October 1982.
- [6] Paul T. Wagner. Interconnect Testing with Boundary Scan. In *Proceedings IEEE International Test Conference (ITC)*, pages 52–57, October 1987.
- [7] Erik Jan Marinissen, Bart Vermeulen, Henk Hollmann, and Ben Bennetts. Minimizing Pattern Count for Interconnect Tests Under A Ground-Bounce Constraint. *IEEE Design & Test of Computers*, 20(2), March/April 2003.
- [8] Hans Peter Richter and Norbert Münch. Boundary-Scan Test Triumphs Over Ground-Bounce. *Test & Measurement World Europe*, 5(4):9–15, August/September 1997.
- [9] Amitava Majumdar, Michio Komoda, and Tim Ayres. Ground Bounce Considerations in DC Parametric Test Generation using Boundary Scan. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 86–91, April 1998.
- [10] Jeffery C. Phillips. *Boundary-Scan Technology, Justification, and Test Implementation for Designers*. White paper by Agilent Technologies, Inc. http://we.home.agilent.com/upload/cmc_upload/ept_phillips.pdf.
- [11] David E. Rolince. *Leveraging XML and Web Technologies for Boundary Scan Test Debug*. White paper by Teradyne, Inc. http://www.teradyne.com/prods/cbt/products/library/vict/XML_boundary.pdf.
- [12] Erik Jan Marinissen, Ben Bennetts, and Bart Vermeulen. Incorporating A Ground-Bounce Preventing Constraint into Wiring Interconnect Test Pattern Generation Algorithms. In *Digest IEEE Intl. Board Test Workshop (BTW)*, Baltimore, MD, USA, October 2002. (available at <http://www.dft.co.uk/BTW02/btw02-1-3.pdf>).
- [13] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study. In Emile H.L. Aarts and Jan-Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons Ltd., Chichester, England, 1997.
- [14] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions*. Dover Publications, Inc., New York, NY, USA, 1972.