Test-Cost Modeling and Optimal Test-Flow Selection of 3D-Stacked ICs*

Mukesh Agrawal, *Student Member, IEEE,* and Krishnendu Chakrabarty, *Fellow, IEEE*

***Abstract*—Three-dimensional (3D) integration is an attractive technology platform for next-generation ICs. Despite the benefits offered by 3D integration, test cost remains a major concern, and analysis and tools are needed to understand test flows and minimize test cost. We propose a generic cost model to account for various test costs involved in 3D integration and present a formal representation of the solution space to minimize the overall cost. We present an algorithm based on A*—a best-first search technique—to obtain an optimal solution. An approximation algorithm with provable bounds on optimality is proposed to further reduce the search space. In contrast to prior work, which is based on explicit enumeration of test flows, we adopt a formal optimization approach, which allows us to select an effective test flow by systematically exploring an exponentially large number of candidate test flows. Experimental results highlight the effectiveness of the proposed method. Adopting a formal approach to solving the cost-minimization problem provides useful insights that cannot be derived via selective enumeration of a smaller number of candidate test flows.**

I. Introduction

Three-dimensional (3D) stacking involves the integration of multiple silicon dies in a vertical stack using short through-silicon vias (TSVs) [2]. Compared to traditional core-integration technologies, 3D stacking offers several benefits, such as reduced wire length, reduction in interconnect delays and power consumption, and higher interconnect bandwidth with improved performance. 3D-stacked memory chips are already in production [3], [4] and the semiconductor industry is headed towards further exploitation of the benefits provided by 3D integration in a variety of product lines, such as 3D NOC [5], 3D memory-on-processor [6], and 3D FPGA [7]. The emergence of 3D logic-logic stacks has also been predicted for the near future [8]. Motivated by advances in design and technology, researchers have started investigating test and design-for-testability techniques for 3D ICs [9]–[12].

Test cost has emerged as a potential showstopper in the adoption of 3D integration. The choice of test flow, i.e., what tests are used and when they are applied during 3D integration ("what to test", "when to test") affects test cost. 3D stacking involves many possible test insertions. Due to multiple yield and test cost parameters corresponding to different dies and tests, such as for pre-bond, post-bond, and partial stack, an exponentially large number of test flows must be evaluated. Therefore, analysis methods and tools are needed for test-cost optimization and automated test-flow selection. A comprehensive cost model is also needed to quantify the difference between the various test flows and to guide the selection process.

Several papers have been published recently on various aspects of test-cost modeling and optimization for 3D ICs [13]–[19]. These papers have primarily explored the test-cost modeling part of the problem, and the test-flow selection problem has largely remained ignored. A small number of selected candidate test flows were explicitly enumerated and the best among them was reported as an 'optimal' test flow. A systematic and exhaustive exploration of all possible test flows is clearly needed to achieve the best trade-off between cost and yield.

In this report, we address test-cost optimization for 3D ICs by developing a cost model that takes into account various test costs at each step of the stacking process. The model is generic and flexible in that it provides placeholders for different test costs that are typically incurred during 3D integration. The proposed model can be adapted for wafer-to-wafer (W2W), die-to-wafer (D2W), and die-to-die (D2D) stacking. We formulate the optimization problem as a search problem and describe a method based on the A* search algorithm [20] to provide an optimal solution—the optimality is with respect to the cost of manufacturing and testing 3D-stacked ICs per good package. The formal objective function is stated in Section III. Although there is no straightforward theoretical bound on the runtime of this search algorithm, A* has been shown in practice to be efficient for various search problems [21]–[23]. We further propose an approximation step that effectively reduces the state space of the search problem and provides a solution within provable bounds to an optimal solution. Results are presented to highlight the impact of various parameters on test cost and test-flow selection.

The major contributions of this report include:

- A generic cost model to incorporate different kinds of test costs involved in 3D integration.
- A formulation of cost-optimization as a search problem.
- A method based on the A*-search algorithm to find an optimal solution.
- An approximation strategy in addition to the A*-based method to reduce the state space.
- Rational insights into the relationship between yield, test cost, and the selection of various pre-bond and stack tests on the basis of experimental results.

*A preliminary version of this report was presented at the IEEE VLSI Test Symposium (VTS), 2013 [1]. This research was supported in part by the National Science Foundation under grant no. CCF-1017391, the Semiconductor Research Corporation under contract no. 2118, a grant from Intel Corporation, and a gift from Cisco Systems through the Silicon Valley Community Foundation.

M. Agrawal and K. Chakrabarty are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA. E-mail: mukesh.agrawal@duke.edu, krish@ee.duke.edu.

The rest of the report is organized as follows. Section II details the motivation for this work and discusses related prior work. The notation used in the report and the cost model are presented in Section III. Section IV presents two models to further simplify the cost model. Section V formulates test-cost optimization and test-flow selection as a search problem, and a solution based on A*-search algorithm is presented in Section VI. Results are shown in Section VII and conclusions are drawn in Section VIII.

II. Motivation and Prior Work

For today's 2D ICs, tests can be applied at two stages: (i) at the wafer level (wafer sort); (ii) after the chip is packaged. Depending on the yield, wafer sort can be a significant cost saver by alleviating the need for packaging defective dies.

In the manufacturing of a 3D-stacked, tests can be applied at multiple stages—individual wafers or dies can be tested prior to bonding (pre-bond test) and testing can be carried out again after partial stacks are created (mid-bond test). A post-bond stack test can be applied to the complete stack with all the dies. During testing of a stack, dies at different layers in the stack can be tested (or re-tested), or their testing can be omitted.

Fig. 1 sketches the typical integration of three dies to form a stack. This process involves incremental stacking of one die at a time. During testing of a stack, tests targeting faults in either of the dies present in the stack can be applied. Hence, in addition to pre-bond testing, a die can be tested at multiple stages of stacking as well. Moreover, there can be multiple types of tests, having different fault coverage and test application costs. If a test with lower cost is selected, that can reduce test cost upfront, but the lower fault coverage can result in the stacking of defective dies, thereby lowering the overall stack yield, and consequently increasing the overall cost of stack creation. Therefore, test-flow selection involves selection of the test insertions (also referred to as test moments in the literature [13]), and tests that provide the best trade-off between cost and quality. For this simple example, there exist $2^{3+3+2} = 256$ different combination of test insertions: tests for Die 1 and Die 2 can be applied at all the three stages and there are two stages in which tests for Die 3 can be applied. The package test, being the last step in the integration flow, is always applied, hence it is not considered for the purpose of optimization. For each selection of insertions, there are different ways in which tests can be selected out of a given test set. It can be easily shown (as discussed in the appendix) that the total number of possible test flows is $(N+1)^{\frac{l^2+3l-2}{2}} = O(N^{l^2})$, where l is the total number of dies in a stack and N is the number of tests available per test insertion.

The optimization problem rapidly becomes intractable with the addition of dies, and because of the added complexity associated with selection of tests. If we consider inter-die interconnect tests, even more test flows are possible. Our goal is to minimize the total cost per "good" package that includes test cost and the part of manufacturing cost that is affected by yield and test escapes. A package is called good if it passes the package test. Since the application of the package test is the last step of the integration flow that we consider, we assume that the package is always tested with highest-quality of tests available to test the package.

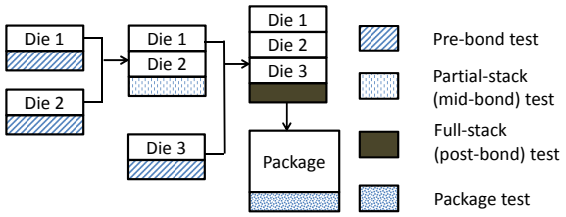

Fig. 1. 3D integration of a stack and the associated test insertions.

Prior work include attempts to hand-pick a test flow or select a test flow based on explicit enumeration of a few candidate test flows. A tool was developed to estimate overall cost of a 3D test flow in [16]. A limited number of 3D and 2.5D test flows were studied in [13] and [17], respectively. The final cost of creating 3D-stacked ICs is not only a function of manufacturing cost of wafers, test cost, logistics cost, and packaging cost, but it is also a function of the fault coverage of selected tests and selected test insertions. The work listed above lacked an analytical framework to estimate the final cost of creating 3D-stacked ICs, and the dependence of the final cost on the choice of test insertions was not adequately explained. The above work also lacks an analysis for the scenario where a die can be tested multiple times (at different test insertions), but with test sets that are different and involve different coverage and cost. In addition, the yield loss due to defects introduced in already bonded dies during stacking steps was not modeled.

These limitations were first discussed in [1], and more general models (i.e., less restrictive models) were considered in [19]. An expression for final test escape rate was provided in [19], but it implicitly assumed that all test insertions were selected, i.e., the expression is correct only if every test insertion is actually selected. Therefore, the impact on the test escape rate, and hence final cost, cannot be analyzed if some test insertions are omitted. The impact of mid-bond testing (testing at intermediate stages of integration) and logistics on the cost of test flows was reported in [18]. A model to predict the impact of test-flows on product quality in terms of defective parts per million (DPPM) was examined in [19].

We therefore conclude that prior work only analyzed a limited number of flows and it does not provide any means for systematically exploring (e.g., through implicit enumeration) the solution space of all possible test flows and reporting the test flow that minimizes the overall cost. While the model in [1] overcame these limitations to some extent, the cost function used for the optimization problem does not account for the number of fault-free packages resulting from the selected test flow, and hence the impact of a test flow on overall product yield was not considered. In addition, the heuristic proposed in [1] does not provide optimal results.

III. Cost Model

In this section, we formally define the cost model that we use for the test-flow selection problem. We highlight the

notation for various parameters and decision variables that constitute the optimization problem in Table I and Table II, respectively. The table entries include key parameters such as the pre-bond and stack tests that are available, test costs, yield, packaging cost, decision variables, etc. The definitions of these parameters and variables are also provided when they are introduced. The reader is referred to the notation tables for more details.

TABLE I
TABLE DESCRIBING THE GIVEN PARAMETERS USED IN THE COST MODEL.

Symbol	Meaning
	Parameters related to pre-bond testing
l	Total number of dies in the stack
$\mathcal{D}_i$	i^{th} die in the stack
DC_i	Manufacturing cost of each instance of $\mathcal{D}_i$
n	Number of instances of $\mathcal{D}_1$ manufactured.
λ_i	Yield for die $\mathcal{D}_i$.
B_i	Number of pre-bond tests of $\mathcal{D}_i$, $\forall i$.
PB_{ij}	j^{th} pre-bond test for $\mathcal{D}_i$, where $1 \leq j \leq B_i$.
b_{ij}	Cost associated with pre-bond test PB_{ij}.
$fc(\tau)$	A function that returns fault coverage of a selected test τ. If the cost-optimization tool does not select a test, the function returns zero.
	Parameters related to stack testing
$\mathcal{S}_k$	A stack of dies $\mathcal{D}_1$, $\mathcal{D}_2$, ... $\mathcal{D}_k$, $2 \leq k \leq l$. The stack $\mathcal{S}_k$ is partial for $k < l$.
SC_k	Manufacturing cost of each instance of $\mathcal{S}_k$
ω_{ik}	The component of bond yield of stack $\mathcal{S}_k$ related to defects induced in die $\mathcal{D}_i$ during the creation of $\mathcal{S}_k$. The overall bond yield for stack $\mathcal{S}_k$ is given by $\prod_{i=1}^{k} \omega_{ik}$.
N_i	Number of different tests that can be applied to $\mathcal{D}_i$ during stack testing.
$\mathcal{T}_{ij}$	j^{th} stack test for $\mathcal{D}_i$, where $1 \leq j \leq N_i$.
t_{ij}	Cost of applying stack test $\mathcal{T}_{ij}$ (this remains the same regardless of the moment at which $\mathcal{T}_{ij}$ is applied).
a_{ijk}	An entry in a binary 3-D matrix that indicates whether $\mathcal{T}_{ij}$ can be applied during stack testing of $\mathcal{S}_k$. Not all tests are applicable at every stage.
PC	Cost of packaging and testing a full stack
	Parameters related to interconnect testing
$\mathcal{I}_i$	Interconnects between dies $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, $1 \leq i < l$
ic_i	Cost of testing interconnect $\mathcal{I}_i$.
ρ_i	Interconnect yield of $\mathcal{I}_i$.

TABLE II
TABLE DESCRIBING THE UNKNOWN VARIABLES USED IN THE COST MODEL.

Symbol	Meaning
m_k	Number of instances of $\mathcal{S}_k$ manufactured.
m'_k	Number of instances of $\mathcal{S}_k$ that have been detected to be fault-free after stack testing.
x_{ij}	A binary variable that is 1 if pre-bond test PB_{ij} of $\mathcal{D}_i$ is carried out, otherwise it is zero.
y_{ijk}	A binary variable that is 1 if $\mathcal{D}_i$ is tested by applying $\mathcal{T}_{ij}$ during stack testing of $\mathcal{S}_k$. It can be assigned 1 only if $a_{i,j,k} = 1$.
z_{ik}	A binary variable indicating whether $\mathcal{I}_i$ is tested during testing of $\mathcal{S}_k$. The test time for interconnects is small compared to test time for dies [12], so we merge the testing of $\mathcal{I}_i$ with the stack test that first uses $\mathcal{I}_i$ for transporting test data; hence z_{ik} depends on y_{ijk}.

For our discussion, we refer to a "test set", i.e., a set of test patterns, by "test". A collection of tests is referred to as a "set of tests".

Without loss of generality, we use the Williams and Brown model [24] for calculating the number of instances of a die that are detected to be fault free after applying a test of a certain fault coverage. The model postulates the following relationship between defect level DL, fault coverage fc, and yield λ: $DL = 1 - (\lambda)^{1-fc}$. If n is the total number of chips manufactured using a process having production yield λ, the number of non-faulty chips is $n \cdot \lambda$. If a test having fault coverage fc is applied, the number of chips that pass the test is given by $\frac{n \cdot \lambda}{1-(1-\lambda^{1-fc})} = n \cdot \lambda^{fc}$. Note that if a perfect test was applied, the count of non-faulty chips would have been $n\lambda$. This means that $n\lambda^{1-fc}$ instances have escaped the test, and can potentially be detected on application of a better test later during the 3D integration process. Other models can be easily used in place of the Williams and Brown model in the proposed optimization framework—the proposed framework is general and not specific to any particular yield model.

We next quantify the total cost of manufacturing and pre-bond testing of the dies. The cost of manufacturing and testing n instances of $\mathcal{D}_1$ is $n \cdot (DC_1 + \sum_{j=1}^{B_1} x_{1j} \cdot b_{1j})$, where DC_i is the cost of manufacturing each instance of die $\mathcal{D}_i$, b_{ij} refers to the cost associated with the j^{th} pre-bond test out of a total of B_i pre-bond tests available to $\mathcal{D}_i$, and x_{ij} is a binary decision variable that indicates the selection of j^{th} pre-bond test of $\mathcal{D}_i$. The term $\sum_{j=1}^{B_1} x_{1j} \cdot b_{1j}$ refers to the cost of the pre-bond test that is applied. Clearly, each die needs to be tested by at most one pre-bond test. In other words,

$$\sum_{j=1}^{B_i} x_{ij} \leq 1, \forall i.$$

If for die $\mathcal{D}_i$, α_i is the pre-bond test selected from the set of B_i pre-bond tests, the number of instances of $\mathcal{D}_1$ that are determined to be fault free is $n \cdot \lambda_1^{fc(\alpha_1)}$. Note that $fc(\alpha_i)$ is the fault coverage of the selected pre-bond test for $\mathcal{D}_i$. If none of the pre-bond tests for $\mathcal{D}_1$ is selected, then the function 'f' returns a zero (see Table I) to give $n \cdot \lambda_1^{fc(\alpha_1)} = n$, where λ_i is manufacturing yield of $\mathcal{D}_i$.

Note that we need to manufacture that many instances of $\mathcal{D}_2$ as there are instances of $\mathcal{D}_1$ available for stacking. If a pre-bond test is applied on $\mathcal{D}_2$, we must manufacture $\frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}}$ instances of $\mathcal{D}_2$. Therefore, the total cost of manufacturing and testing $\mathcal{D}_2$ is

$$\frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}} \cdot (DC_2 + \sum_{j=1}^{B_2} x_{2j} \cdot b_{2j}).$$

The stacking of k dies results in the creation of stack $\mathcal{S}_k$. We use the notation m_k to denote the number of instances of $\mathcal{S}_k$ created, and m'_k to denote the number of instances of $\mathcal{S}_k$ that are determined to be fault-free after $\mathcal{S}_k$ has been tested. For creating m_k instances of $\mathcal{S}_k$, m'_{k-1} instances of $\mathcal{S}_{k-1}$ are available for stacking with $\mathcal{D}_k$. The total cost of manufacturing $\mathcal{D}_k$ is then given by

$$\frac{m_k}{\lambda_k^{fc(\alpha_k)}} \cdot (DC_k + \sum_{j=1}^{B_k} x_{kj} \cdot b_{kj}).$$

Note that $m_k = m'_{k-1}$. The total cost of manufacturing and running pre-bond tests for all dies (C_1) is then given by the following equation.

$$\begin{aligned} C_1 = n \cdot (DC_1 + \sum_{j=1}^{B_1} x_{1j} \cdot b_{1j}) \\ + \frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}} \cdot (DC_2 + \sum_{j=1}^{B_2} x_{2j} \cdot b_{2j}) \\ + \sum_{i=3}^{l} \frac{m_i}{\lambda_i^{fc(\alpha_i)}} \cdot (DC_i + \sum_{j=1}^{B_i} x_{ij} \cdot b_{ij}) \end{aligned} \quad (1)$$

The total cost of stacking l dies ($C2$) is given by: $C_2 = \sum_{k=2}^{l} m_k \cdot SC_k$, where the number of stacks of k dies (m_k) is determined by the number of dies and partial stacks available after defect screening, and SC_k is the cost of stacking operation to create an instance of $\mathcal{S}_k$.

The cost of testing $\mathcal{I}_i$, interconnect layer between the dies $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, during the testing of S_k is $m_k \cdot z_{ik} \cdot ic_i$, where ic_i is the cost of testing $\mathcal{I}_i$, and z_{ik} is a decision binary variable indicating whether $\mathcal{I}_i$ is tested during testing of S_k. The total cost of testing all the interconnect layers (C_3) is given by:

$$C_3 = \sum_{i=1}^{l-1} ic_i \cdot (\sum_{k=i+1}^{l} m_k \cdot z_{ik}) \tag{2}$$

The cost of testing interconnect is much less compared to the cost incurred on testing dies [12]; therefore, to avoid unnecessary complexity in the optimization problem, we restrict an interconnect layer be tested no more than once. Hence, the constraint $\sum_{k=i+1}^{l} z_{ik} = 1$ is imposed on the variables z_{ik} for $i < l$.

The total cost of stack testing (C_4) can now be stated as:

$$C_4 = \sum_{k=2}^{l} m_k (\sum_{i=1}^{k} \sum_{j=1}^{N_i} a_{ijk} y_{ijk} . t_{ij}), \tag{3}$$

where t_{ij} is the cost associated with test $\mathcal{T}_{ij}$ or the j^{th} stack test available for $\mathcal{D}_i$, y_{ijk} is a decision variable indicating the selection of $\mathcal{T}_{ij}$, a_{ijk} is a binary parameter that indicates whether $\mathcal{T}_{ij}$ can be applied during the testing of stack $\mathcal{S}_k$, and N_i is the total number of stack tests available for $\mathcal{D}_i$. Since we do not apply more than one stack test for a die at any given test stage, the following constraint on the variables y_{ijk} is applied: $\sum_{j=1}^{N_i} a_{ijk} y_{ijk} \leq 1$.

Finally, we account for the cost associated with packaging and final test, which is given by the equation $C_5 = m_l^{'} \cdot PC$, where PC is the cost of packaging and application of package test.

The total cost for manufacturing and testing the 3D IC is $CT = C_1 + C_2 + C_3 + C_4 + C_5$. If the total number of packages that are deemed fault free after applying the package test is P, our objective is to minimize $\frac{CT}{P}$ by assigning appropriate values to x_{ij} and y_{ijk}. The variables z_{ik} depend on y_{ijk}, as explained in Table II.

If we assume that defects are induced only on top two dies during the stacking process, and only these dies are considered for testing at this stage, as in [13], then we can use constraint $\omega_{ik} = 1$ for $i \leq k-2$, where ω_{ik} is the component of bond yield for stack $\mathcal{S}_k$ that models the defects introduced in $\mathcal{D}_i$ during the creation of $\mathcal{S}_k$.

Given the manufacturing cost of dies, the tests available for every test insertion along with corresponding test cost and fault coverage, stacking cost, bond-yield components for every die, and the package cost, the goal of the optimization problem is to minimize the total cost incurred in manufacturing and testing per fault-free (shipped to customer) package.

A die can be tested multiple times with different tests having different fault coverage. If two tests, say τ_1 and τ_2, are derived using the same fault model, i.e., target the same set of defects, then their effective fault coverage, $EFC(\tau_1, \tau_2)$, can be computed by merging the corresponding list of faults that they detect, and reporting the fault coverage obtained from the new list. Note that while merging the individual fault lists, duplicate entries are removed to avoid counting a detected fault multiple times. The effective fault coverage, or EFC, can be computed for more than two tests in a similar way. This method requires availability of fault lists for every available test. Approximate models that circumvent this requirement are presented in Section IV.

A single fault model is often not sufficient to cover all types of defects; therefore, we must account for multiple fault models in the proposed cost model. It is well known that the relationship between different fault models is difficult to quantify. Questions such as, does high coverage for one fault model ensures high coverage for another fault model, have yet to be satisfactorily answered. Recent work [25] argues that a near-100% single-stuck-at (SSA) fault coverage does not ensure high coverage of transition faults by constructing a circuit, which the author refers to as an "extreme" example, and demonstrating that 0% transition fault coverage is achievable with a close-to-100% SSA fault coverage for that circuit. He further proves, however, that 100% SSA fault coverage leads to 100% transition fault coverage. Delving into these research topics is beyond the scope of this work, and we make simplifying assumptions for our calculation. Assuming that "extreme" circuits are not often found in real-life examples, we may write the aggregate fault coverage of two tests, say τ_1 and τ_2, that are for different fault models, as $AFC(\tau_1, \tau_2) = \max\{fc(\tau_1), fc(\tau_2)\}$. If multiple fault models were not present, we would have just used the fault coverage of the given test for the purpose of calculation; therefore, using a simple expression is justifiable. In Section IV, we do not specify fault lists of individual tests, and only numeric value of fault coverage of each test is provided as an input; therefore, a test engineer, instead of using the simplistic equation above, can provide a value that is deemed to be appropriate to the cost optimization tool. This is particularly useful in the following case. If it is required from the tool to select a combination of tests targeting multiple fault models, then the combination has to be provided as one of the inputs to the optimization tool. Note that the aggregated fault coverage, or AFC of two tests for the same fault model is same as their effective fault coverage, or their EFC.

For computing the aggregate fault coverage, or AFC, of more than two tests, the set of given tests is first partitioned on the basis of fault models that they target. Then EFC is computed for every partition, and the maximum EFC among them is reported as the AFC of the given set of tests. It can be seen that EFC or AFC for a given set of tests cannot exceed 100%.

We augment the notation of the aggregate fault coverage as $AFC_{i,j,k}$, where $i \leq j \leq k$, to denote the aggregate fault coverage of all tests that are applied on $\mathcal{D}_i$ between mid-bond testing of $\mathcal{S}_i$ and $\mathcal{S}_k$, both inclusive. We also use the symbol $AFC_{i,1,k}$ to denote the aggregate fault coverage of all tests applied to $\mathcal{D}_i$ between test insertions of its pre-bond test and the corresponding mid-bond test of $\mathcal{S}_k$. Figure 2 illustrates the notation using an example of a 4-die stack when $\mathcal{D}_3$ (Die 3)

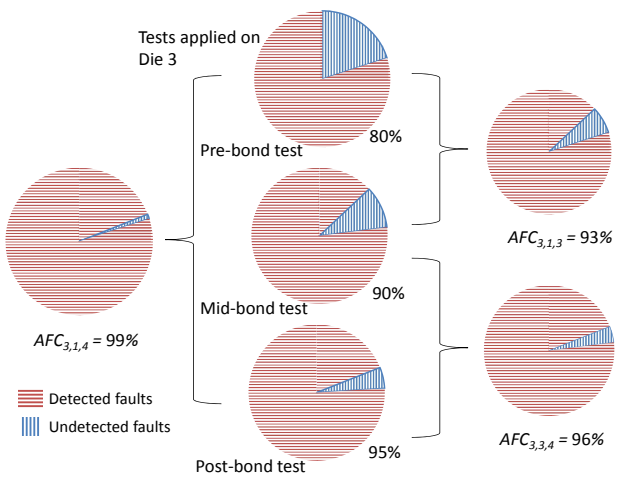

Fig. 2. An example illustrating the usage of the symbol $AFC_{i,j,k}$ for a 4-die stack. Multiple tests applied to Die 3 are for the same fault model and the relative positioning of slices in pie charts also reflects common faults that these tests detect.

is tested at all test insertions.

We next show how to compute the number of non-faulty stacks after applying mid-bond tests. The William and Brown model is repeatedly applied to estimate the number of instances after multiple tests are applied to a stack. The following assumptions are made for our subsequent discussion.

Assumption 1 The occurrence of defects due to different manufacturing steps are independent to each other.

Assumption 2 We assume that failure of dies in a stack during mid-bond testing and post-bond testing are uncorrelated. Since dies in a heterogeneous stack are likely to be dissimilar, the above independence assumption can be justified in practice.

If tests are only applied to $\mathcal{D}_1$, then the number of stacks remaining after mid-bond test of $\mathcal{S}_k$ is applied is given by

$$m'_k = n \cdot \lambda_1^{AFC_{1,1,k}} \cdot \prod_{j=2}^{k} \omega_{1,j}^{AFC_{1,j,k}}$$

Note that every yield parameter captures a different manufacturing step, the above expression is obtained by the assumption of independence of occurrence of defects to these manufacturing steps.

If tests are applied on all dies, then we can rewrite the above equation as

$$m'_k = n \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}} \cdot \prod_{j=i,j\neq 1}^{k} \omega_{i,j}^{AFC_{i,j,k}})$$

The yield components from all dies are multiplied together due to the assumption of the independence of die failures to tests.

Note that the instances of $\mathcal{D}_i$, for $i \geq 2$, are added to the stack after pre-bond tests are applied to them. If pre-bond tests are applied to dies before stacking, then the defects that have already been screened during pre-bond test will not cause further yield loss. After accounting the elimination of faulty dies during their respective pre-bond tests, the above equation can then be corrected to

$$m'_k = \frac{n \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}} \cdot \prod_{j=i,j\neq 1}^{k} \omega_{i,j}^{AFC_{i,j,k}})}{\prod_{i=2}^{k} \lambda_i^{fc(\alpha_i)}}$$

$$\implies m'_k = n\lambda_1^{fc(\alpha_1)} \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}-fc(\alpha_i)} \cdot \prod_{j=i,j\neq 1}^{k} \omega_{i,j}^{AFC_{i,j,k}}) \quad (4)$$

An example on the computation of m'_k is presented for a two-die stack in the appendix.

If we further assume that the fault coverage of each interconnect test is 100%, the expression for m'_k can be extended to

$$m'_k = n\lambda_1^{fc(\alpha_1)} \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}-fc(\alpha_i)} \cdot \prod_{j=i,j\neq 1}^{k} \omega_{i,j}^{AFC_{i,j,k}}) \prod_{j=i+1}^{k} \rho_i^{z_{ij}}$$

The above equation accounts for the interconnect yield ρ_i of the interconnect layer $\mathcal{I}_i$.

As stated in Section I, the cost model described above can be adopted for W2W, D2D, and D2W integration. For W2W integration, if the pre-bond tests are skipped, the x_{ij} variables can be constrained to zero. Some mid-bond tests can be skipped (and the decision variables fixed appropriately) due to constraints on test access. For D2W stacking, the cost model can consider pre-bond tests as decision variables, but certain mid-bond tests may be omitted and corresponding decision variables fixed. Note that D2D stacking offers the most flexibility in terms of test flows, hence it leads to the largest number of decision variables.

IV. Approximation to the cost model

Test engineers may not have access to fault lists of tests that are available for testing dies, thus making it impossible to estimate the cost associated with a test flow, or to find an optimal test flow using the cost model presented in Section III. In this section, we present two models that can be used for estimating cost incurred by a test flow without requiring the fault list.

A. Model I

We simplify the calculation of the effective fault coverage (EFC) of two or more tests that target faults from the same fault model. The EFC of a set of tests cannot be less than the maximum fault coverage of individual tests, and at best, it can be 100%. Therefore, we make a pessimistic assumption that the EFC of a set of tests is equal to the maximum fault coverage of all tests in the test set. Note that the assumption is true only in the case when the test having the maximum fault coverage is the superset of every other element tests.

Figure 3 shows the fraction of components passing a test as the fault coverage of the test varies from 80% to 100% (according to the William and Brown model). This is shown for different yields values ranging from 0.75 to 0.9. Note that the fraction does not vary much as the value of fault coverage

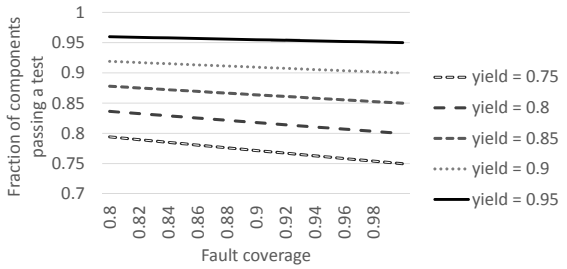

Fig. 3. Variation of fraction of components passing a test on fault coverage according to the William and Brown model.

changes, especially for high yield values. If we report EFC of a set of tests as 80%, and the component being tested has a yield of 90%, we are inaccurate only by a margin of $0.9^{0.8} - 0.9^{1.0} = 1.9\%$ in the worst case, because EFC can be at best 100% when EFC is computed using fault lists. With better yields, the difference further reduces. Moreover, as the manufacturing processes matures, yield values will be on the higher side. Therefore, assumption in this model is practical for the purpose of this calculation.

With this model, $AFC_{i,j,k}$ is reduced to the fault coverage of the test with maximum fault coverage among the tests that are applied on $\mathcal{D}_i$ between the mid-bond test insertions of $\mathcal{S}_j$ and $\mathcal{S}_k$, and $AFC_{i,1,k}$ is reduced to the maximum fault coverage of any test that is applied to $\mathcal{D}_i$ during or before the mid-bond testing of $\mathcal{S}_k$.

B. Model II

This model is equivalent to the model presented in [1]. The defects introduced by a manufacturing step is "forgotten" once a test is applied to screen those defects. In other words, the test escapes after applying a test are not considered faulty subsequently. In effect, $AFC_{i,j,k}$ is reduced to the fault coverage of the first test that is applied on $\mathcal{D}_i$ between the mid-bond test insertions of $\mathcal{S}_j$ and $\mathcal{S}_k$, both inclusive, and $AFC_{i,1,k}$ is reduced to the fault coverage of the first test that is applied to $\mathcal{D}_i$ during or before the mid-bond testing of $\mathcal{S}_k$.

The proposed approximations to the cost model is illustrated through an example of a two-die stack in the appendix. Model I is closer to reality, and all results presented in this work are based on Model I.

V. Problem Formulation

The optimization problem that we solve is complex, and it involves many trade-offs. We formulate the problem as a typical search problem that is defined by states, actions, a goal, and a performance measure or a cost function. For finding a solution, a problem solver starts in the initial state, jumps to successive states based on actions taken while in previous states, and terminates when it achieves the goal with the optimum value of the cost function. The state space is the set of all states reachable from the initial state.

Let us define every component of the corresponding search problem using a simple example. In the initial state, no dies have been manufactured, and the actions to be considered in this state are selection of a pre-bond test for $\mathcal{D}_1$ or apply no test at all. Depending on which action, or test, is chosen, the problem solver reaches another state, where n instances of $\mathcal{D}_1$ are manufactured and tested using the pre-bond test selected in the parent state. Every child state of the initial state differs in the pre-bond test that is applied to it. In a newly added state, the problem solver has to chose from the available set of pre-bond tests for $\mathcal{D}_2$, following which new states corresponding to mid-bond test insertions of $\mathcal{S}_2$ are added to the state space explored so far. After test insertions of $\mathcal{S}_2$ are explored, actions for selecting a pre-bond test of $\mathcal{D}_3$ becomes available. New states corresponding to the pre-bond test of $\mathcal{D}_3$ are added, after which actions related to post-bond test insertions of $\mathcal{D}_3$ are considered. This process of choosing actions and addition of new states are carried out until we explore the goal states corresponding to packaging and applying the package test. Every set of new states that we add corresponds to a test insertion. The state space can be represented by a graph in which the nodes are states and the arcs are actions, or choices in our problem.

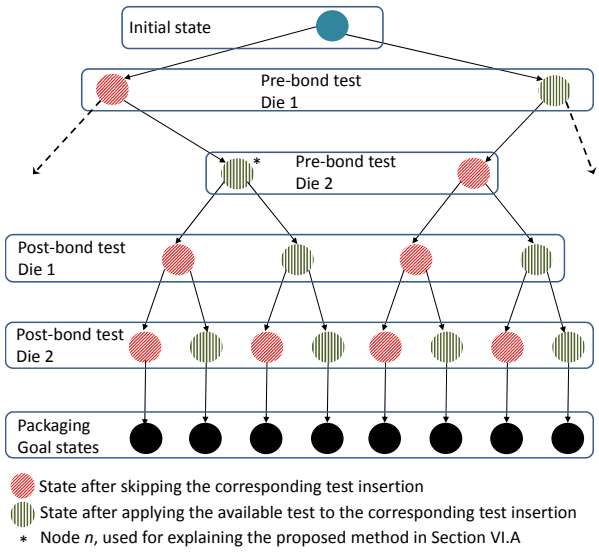

Fig. 4. A partial state space for an example of two-die stack.

Figure 4 shows the state space for an example of a two-die stack. We have four levels of states after the initial state, each corresponding to a test insertion. In this example, we assume that only one test is available for each die at each test insertion; therefore, a branching factor of two is seen at every level of states—one branch for discarding the corresponding test insertion, and the other for selecting the only available test for the test insertion. During a mid (post)-bond test stage after stacking, a decision regarding testing the constituent dies of the partial (full) stack is made sequentially starting from the bottom-most die. For example, in Figure 4, choices for *Die* 1 are considered before choices for *Die* 2 in the path to a goal node during the post-bond test stage. This provides a systematic method of exploring the state space for reaching the goal. The only choice possible in the states corresponding to the last test insertion is to package the full stack and apply the package test; therefore, there is no branching at that test insertion to reach the last level of states that are goal states. Due to lack of branching in the penultimate level, we can

readily merge the last two levels, but we show them separately here for the purpose of clarity.

It can be seen that every state in the last level is a goal state, or a goal node; therefore, a path from the initial state to any of these states is a valid solution candidate, i.e., a valid test flow. Moreover, there exists a unique path from the initial state to a goal state because the state-space graph is a tree. We require from the problem solver to return a path that has the minimum *path cost*, or the minimum cost per good package. In a traditional search problem, path cost of a path is the sum of the individual cost of each arc constituting the path; however, our cost function also accounts for the total good packages, which may be different for two goal nodes, thereby making costs of internal arcs undefined until the full path is made known. The dependence of arc costs on full paths makes our search problem different, and presumably more difficult.

VI. Solution based on A* search algorithm

A* search is a popular form of best-first search [26]. It evaluates a node n using a function $f(n)$ that is defined as the sum of $g(n)$, the actual cost of reaching node n from the start node, and a heuristic function $h(n)$, an estimate of the cost to reach a goal node from n. Since $h(n)$ is an estimated cost of the cheapest path from n to the goal, the function $f(n)$ can be interpreted as an estimate of the cheapest solution through n. For a goal node, $f(n)$ is the actual value of the cost function.

The function $h(n)$ is usually computed through a computationally inexpensive heuristic. For a minimization problem, $h(n)$ is an underestimate of the actual cost of reaching the goal using any path starting from n, i.e., $h(n)$ *never overestimates* the cost to reach the goal. If nodes do not repeat during traversal of the search graph, it can be shown that this property of $h(n)$ makes A* optimal.

A basic version of A* is outlined in Figure 5. The algorithm maintains a priority queue of active nodes or the states seen so far. The node with the minimum value of $f(n)$ is at the top of the queue. It also maintains a map, which we call *parent*, to reconstruct the solution once a goal node is found. The node that is dequeued from the priority queue is explored and all its neighbors are placed in the priority queue. Since our search graph is a tree, a node once explored (or dequeued) cannot repeat, and hence cannot be queued back to the priority queue. Therefore, $g(n)$ of an explored node is never updated, as is the case with running A* on graphs that are not trees. Note that inside the for loop, a goal node may be enqueued to the priority queue. We may eventually have multiple goal nodes present in the priority queue simultaneously, but the algorithm terminates as soon as the first goal node is dequeued.

It remains to be shown that a suboptimal goal node (say G) is never dequeued. Let the cost of an optimal solution be C^*. Since $h(n)$ of every goal node is equal to zero by definition, $f(G) = g(G) + h(G) = g(G) > C^*$. Suppose n is a node on the path to an optimal goal node and it is present in the priority queue (such a node always exists if a solution exists). Because $h(n)$ never overestimates the true cost a path from n, $f(n) = g(n) + h(n) \leq C^*$. Therefore, $f(n) \leq C^* < f(G)$. It implies that the suboptimal node G cannot be explored, and A* must return an optimal solution.

```
basic_a_star(start_node)
 1: priority_q ← a priority queue with f(n) as key
 2: Enqueue start_node to the priority_queue
 3: parent(start_node) ← null
 4: while priority_q is not empty do
 5:     n ← peek priority_q
 6:     if n is a goal node then
 7:         Report the path from start_node to n using parent
 8:         break
 9:     end if
10:     for each neighbor n′ of n do
11:         parent(n′) ← n; compute f(n′)
12:         enqueue n′ to priority_q
13:     end for
14: end while
```

Fig. 5. A basic version of the A* algorithm.

A. Computation of $h(n)$

The heuristic function $h(n)$ must never overestimate the cost of any path from n to a goal node; therefore, $h(n)$ in this work is computed by minimizing the cost at each step of the stacking process. We do the following for computing $h(n)$:

- We set the test cost at those test insertions to zero for which no decision has been made so far for reaching n. For example, for node n in the state space in Figure 4, no decision has been taken for post-bond test of *Die* 1 and *Die* 2; therefore, it is assumed that no money is spent on those test insertions.
- Since the total cost incurred on manufacturing die $\mathcal{D}_i$ ($i > 0$) is dependent on the number of instances of partial stack $\mathcal{S}_{i-1}$ available after testing (m'_{i-1}), m'_{i-1} is underestimated by assuming that highest-quality tests are applied at those test insertions for which no decision has been made so far for reaching n. For example, for node n in Figure 4, m'_2 is underestimated by assuming that all defective components are removed by applying appropriate tests for the unexplored states below n.

The above assumptions also ensure that we underestimate the amount spent on stack creation, a condition that must be ensured to guarantee optimality.

B. Adaptation of A* to the cost-minimization problem

If $g(n)$ is the actual cost incurred in reaching node n, and $h(n)$ is computed as outlined in the previous subsection, the function $f(n) = g(n) + h(n)$ can be used for finding a test flow that minimizes the total cost incurred during the stacking process. We are interested in minimizing the cost per good package; therefore, we redefine $f(n)$ as $f(n) = \frac{g(n)+h(n)}{q(n)}$, where $q(n)$ refers to an estimate of the quantity, or the number of good packages finally created after the package testing is applied. Since $q(n)$ is the denominator, we must overestimate $q(n)$. The basis for the overestimation is given by the following lemma.

Lemma 1. *The number of instances of good packages obtained after applying the package test is* $n\lambda_1 \prod_{i=2}^{l} \lambda_i^{1-fc(i)} \prod_{j=2}^{l} \prod_{i=1}^{i<=j} \omega_{ij}$, *where* $fc(i)$ *is the fault coverage of the pre-bond test applied on* $\mathcal{D}_i$.

Proof. As assumed earlier, highest-quality tests are applied during the package test that eliminates every defective package

```
modified_a_star(start_node, Δ)
priority_q ← a priority queue with f(n) as key
Enqueue start_node to the priority_queue
parent(start_node) ← null
current_solution ← ∞
while priority_q is not empty do
    n ← peek priority_q
    if n is a goal node then
        Report the path from start_node to n using parent
        break
    end if
    for each neighbor n′ of n do
        parent(n′) ← n; compute f(n′)
        if n′ is goal node then
            if f(n′) < current_solution then
                current_solution ← f(n′)
            end if
        else if f(n′) ≥ (1 − Δ) · current_solution then
            continue
        end if
        enqueue n′ to priority_q
    end for
end while
```

Fig. 6. A $(1-\Delta)^{-1}$-approximation algorithm.

from the lot of packaged ICs. Therefore, we can set $AFC_{i,j,k} = 1$ and $AFC_{i,1,k} = 1$ in Equation 4 for $k = l$ to obtain the number of packages as $n\lambda_1^{fc(\alpha_1)} \cdot \prod_{i=1}^{k}(\lambda_i^{1-fc(\alpha_i)} \cdot \prod_{j=i,j\neq 1}^{k} \omega_{i,j})$. After rearranging factors, we will obtain the expression for the total number of good packages as given in the lemma. □

We overestimate $q(n)$ on the basis of the above lemma. As can be seen from the above expression, the total number of good packages differ for different flows only based on whether pre-bond tests of dies have been applied. Since $\lambda_i^{1-fc(\alpha_i)} < 1$ and $0 \leq fc(\alpha_i) < 1$, $fc(\alpha_i)$ can be set to 1, to overestimate the total number of good packages, or $q(n)$. We just assume that a pre-bond test (with 100% fault coverage) is applied to those dies for which no decision has yet been made for reaching node n in the state space.

C. $(1-\Delta)^{-1}$-approximation step

On running the basic A* algorithm on several instances of our problem, we found that we were unnecessarily exploring nodes that are not goal nodes, and have values of $f(n)$ that are reasonably close to the optimum cost C^*. To better control exploration of the search space, and thereby reduce the runtime of the algorithm, we add a parameter Δ, where $0 \leq \Delta < 1$, and propose a modification to the basic A* algorithm. Figure 6 shows the modified version of the A* algorithm.

The variable *current_solution* is the minimum of all costs of goal nodes seen so far during the search process. It is updated with $f(n')$ on finding a goal node n' if $f(n')$ is lower than the current value of *current_solution*. A node n' is not added to the queue if n' is not a goal node and $f(n') \geq (1-\Delta) \cdot$ *current_solution*. This prevents exploration of n' in future. The following theorem establishes an approximation bound on the result derived from the approximation step.

Theorem 1. *The algorithm shown in Figure 6 guarantees a solution that is provably* $(1-\Delta)^{-1}$ *times* C^* *(optimum cost) in the worst case, where* $0 \leq \Delta < 1$.

TABLE III
INPUT PARAMETERS FOR AN EXAMPLE OF 2-DIE STACK.

Die	Manufacturing cost	Test cost (100% coverage)	Stacking cost	Packaging cost
$\mathcal{D}_1$	\$1.80	\$0.35	\$0.40	\$3.50
$\mathcal{D}_2$	\$2.20	\$0.20		

PROOF. If an optimal goal node G^* ($f(G^*) = C^*$) is present in the priority queue, then from the proof of the optimality of A*, it can be readily concluded that G^* is always explored, and an optimal solution is returned. A sub-optimal solution is only returned in the case when a non-goal node n' lies in the path to reach G^* and n' is not added to the priority queue because $f(n') \geq (1-\Delta) \cdot f(G)$ for a sub-optimal goal node G present in the priority queue with the least f-value among all goal nodes present in the priority queue. If the cost returned after termination of the algorithm is C, then the following relationship holds.

$$f(n') \leq C^* < C \leq f(G) \leq \frac{f(n')}{1-\Delta} \leq \frac{C^*}{1-\Delta}$$
$$\implies C^* < C \leq (1-\Delta)^{-1} C^* \qquad \square$$

With Δ set to zero, optimality of the solution is not affected. On setting $\Delta = 0.5$, the approximation factor becomes two, and the solver provides a solution with the value of the cost function no worse than twice its optimum value. As Δ increases, we can also see a significant reduction in the number of states explored by the problem solver, thereby reducing the run time.

VII. RESULTS

We assessed the proposed cost model and test-flow selection method using a series of experiments. The objectives of our experiments are twofold:

- Find underlying relationships between different test parameters on an optimal test flow, and to provide insights into why certain choices were made by the test-flow selection tool.
- Show that the A*-based method is practical, and quantify the extent to which it outperforms the exhaustive enumeration method in terms of CPU time.

First, for a small example of two dies, we compare the two objective functions—total cost per good package and total cost [1]. Next for a bigger example of four dies, we make important observations about selection of tests and test insertions. We compare the methods based on A* and the approximation algorithm with another exact method based on trivial exhaustive enumeration.

Using the wafer-cost estimator from [27] for a standard wafer with 300mm diameter and an edge clearance of 3mm, and applying the "Gross Die per Wafer formulas" from [28] for such wafers, the cost of manufacturing per die was estimated to be \$1.77 in [19]. On adding the cost of manufacturing TSVs, the cost of manufacturing per die increases to \$1.92 [19]. The cost of manufacturing TSV is 60% of the cost of stacking operation [29], thus making the stacking cost \$0.25 per stack. The cost of pre-bond tests were assumed to be \$0.50 for 99.6%

TABLE IV
SHOWING 16 POSSIBLE TEST FLOWS FROM THE EXAMPLE OF 2-DIE STACK AND THE CORRESPONDING SELECTION OF TEST INSERTIONS.

Test insertions	Test flows															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathcal{D}_1$, PrB									✓	✓	✓	✓	✓	✓	✓	✓
$\mathcal{D}_2$, PrB					✓	✓	✓	✓					✓	✓	✓	✓
$\mathcal{D}_1$, PoB			✓	✓			✓	✓			✓	✓			✓	✓
$\mathcal{D}_2$, PoB		✓		✓		✓		✓		✓		✓		✓		✓

PrB: pre-bond test, PoB: post-bond test

fault coverage in [19]. We have assumed similar values for the various parameters in our test cases.

For the first experiment, a two-die stack is considered. Table III provides details of the input parameters used in the example. Die yields of 0.9 and stack-yield components for both the dies are assumed to be 0.95.

There are $2^4 = 16$ test flows possible in this simple example. Table IV shows the selection of test insertions in these test flows. The test flows, numbered 1 to 16, correspond to the goal nodes in the solution tree. They are numbered in the same order in which they are found in the solution tree, if the tree is explored in a depth-first style. Therefore, the first test flow does not select a test insertion and the last test flow selects every test insertion.

Figure 7 compares the objective functions of "cost per good package" with "total cost". For each test flow, data points are normalized with respect to the optimum value obtained from the corresponding objective functions. While the former metric selected only the pre-bond test insertions as the optimum flow, the latter metric selected the pre-bond test for $\mathcal{D}_1$ and the post-bond test of $\mathcal{D}_2$. It can also be seen that while the total cost declines sharply from the eighth test flow to the ninth test flow, profit margin is negatively affected per good stack. Using total cost as the objective function offsets the profit margin per good stack by 2% for this example, but for large examples, the profit margin may be significantly reduced with the objective function used in [1].

There are certain test flows that lead to contrasting values of the objective functions, as shown in Figure 7. These are precisely those test flows that select pre-bond test insertion of $\mathcal{D}_2$. While substantial amount is spent in pre-bond testing of $\mathcal{D}_2$, it helps in lowering the cost per good package. Note that the objective function of "total cost" is not a "bad" choice for some test flows, and even those test flows that are good with respect to the objective function of "cost per good package" require comparatively more money to be spent. When we have a limited supply of resources for manufacturing and testing of 3D-Stacked ICs, our approach can be adapted to report an optimal test flow under an additional constraint on the total amount that can spent. The A*-based method can discard nodes with the value of $g(n) + h(n)$ exceeding the available resources.

Next we consider a larger example consisting of four dies. Table V lists the input parameters used in the example. For each test insertion, we have three tests in this example. The cost and the associated fault coverage are listed in the table. The cost of creating stack is assumed to be 40 cents per stack and the cost of applying package test is set at $3.50 per package. The stack yield component for the top two dies

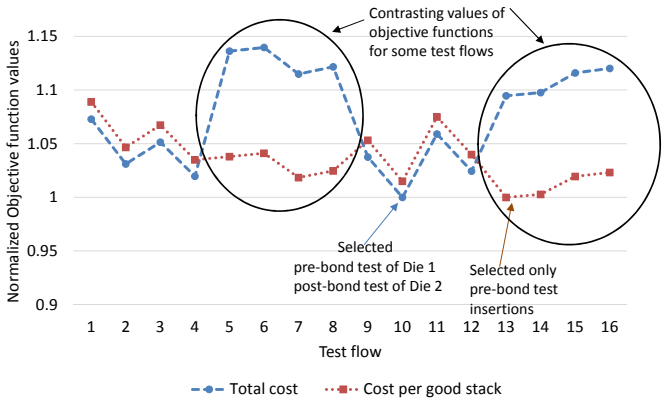

Fig. 7. Comparing our objective function with the objection function in [1] for a two-die stack.

for a stacking operation is assumed to be 0.95, and for the remaining underlying dies, the stack-yield component is set at 0.99.

If only one test with 100% coverage is assumed to be available, the total number of test flows that are possible for this example is 8192. Table VI shows the best eight and the worst eight test flows with respect to the objective function used in this work. The cost-effective test flows have a high fraction of test insertions selected, and there are some 'bad' test insertions that were only selected in the worst test flows. Since the stack-yield components for dies $\mathcal{D}_1$ and $\mathcal{D}_2$ for the final stacking step is very high (0.99 in this example), the corresponding post-bond test insertions are not selected in the best test flows. Figure 8 shows the cost of manufacturing and testing per good package for these test flows. The scale of the Y-axis in the figure is not linear and is adjusted to better illustrate the difference in cost-function values of different test flows. The profit margin reduces by 35% on choosing the worst test flow over the best test flow.

A defective die, if it remains untested before stacking, can make the entire stack defective. Moreover, if the die yield is low, it becomes increasingly important to screen the die for defects prior to bonding. Therefore, it is only natural to think that if we vary the die yield of a single die from low to high values after fixing every other yield parameters to known values, we will obtain a threshold value of the die yield, below which a pre-bond test is always selected. For the example of four dies, first we assume that only one test of 100% fault coverage is available for each die at the corresponding test insertions. For finding the threshold die-yield value for a die, we sweep the yield for that die from small to large values, and select that value after which we see a change in the pre-bond test selection. In this experiment, we set yield of all other dies at 0.99 (a very high value). Furthermore, we also varied the number of available tests, from one to three, for each die and measured the threshold value of die yield for pre-bond test selection. First we add a test with 95% fault coverage and then the test with 90% fault coverage is added to the pool of available tests.

Table VII shows the threshold values of die-yield for different dies. We can draw the following inferences from this experiment.

TABLE V
INPUT PARAMETERS FOR AN EXAMPLE OF 4-DIE STACK.

Die	Manufacturing cost	Test cost		
		100% coverage	95% coverage	90% coverage
$\mathcal{D}_1$	\$1.80	\$0.35	\$0.18	\$0.09
$\mathcal{D}_2$	\$2.20	\$0.20	\$0.10	\$0.05
$\mathcal{D}_3$	\$2.30	\$0.30	\$0.15	\$0.08
$\mathcal{D}_4$	\$1.90	\$0.25	\$0.13	\$0.07

TABLE VI
THE BEST EIGHT AND THE WORST EIGHT TEST FLOWS FOR THE EXAMPLE OF THE 4-DIE STACK.

Test insertions	Best test flows								Worst test flows							
	A	B	C	D	E	F	G	H	S	T	U	V	W	X	Y	Z
PrB: $\mathcal{D}_1$		✓		✓	✓		✓					✓				
PrB: $\mathcal{D}_2$	✓	✓	✓	✓	✓	✓	✓									
MiB: $\mathcal{S}_2$, $\mathcal{D}_1$	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓				
MiB: $\mathcal{S}_2$, $\mathcal{D}_2$				✓			✓									
PrB: $\mathcal{D}_3$	✓	✓	✓	✓	✓	✓	✓	✓								
MiB: $\mathcal{S}_3$, $\mathcal{D}_1$										✓		✓	✓			
MiB: $\mathcal{S}_3$, $\mathcal{D}_2$	✓	✓	✓	✓	✓	✓	✓	✓								
MiB: $\mathcal{S}_3$, $\mathcal{D}_3$						✓		✓								
PrB: $\mathcal{D}_4$	✓	✓	✓	✓	✓	✓	✓	✓								
PoB: $\mathcal{S}_4$, $\mathcal{D}_1$									✓	✓		✓	✓		✓	
PoB: $\mathcal{S}_4$, $\mathcal{D}_2$																
PoB: $\mathcal{S}_4$, $\mathcal{D}_3$	✓	✓		✓			✓				✓					
PoB: $\mathcal{S}_4$, $\mathcal{D}_4$									✓					✓		

PrB: pre-bond test, MiB: mid-bond test, PoB: post-bond test
$\mathcal{S}_k$, $\mathcal{D}_i$: test applied on $\mathcal{D}_i$ during stack testing of $\mathcal{S}_k$

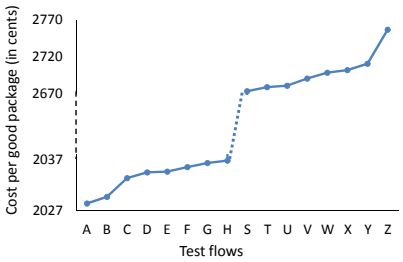

Fig. 8. Cost per good package for the test flows in Table VI

TABLE VII
DIE-YIELD THRESHOLD FOR SELECTION OF A PRE-BOND TEST FOR VARYING NUMBER OF AVAILABLE TESTS.

Die	Die-yield threshold to select a pre-bond test		
	1 test	2 tests	3 tests
$\mathcal{D}_1$	0.88	0.93	0.96
$\mathcal{D}_2$	0.94	0.96	0.98
$\mathcal{D}_3$	0.96	0.97	0.98
$\mathcal{D}_4$	0.98	0.98	1.0

- Pre-bond test selection is dependent on the value of die yield. Based on the interplay of given parameter values, each die has a threshold value of die yield below which a pre-bond test is always selected.
- From Table VIII, it is clear that as more pre-bond tests are available, the benefit of selecting a pre-bond test increases. Cheaper pre-bond tests are now selected even for those values of die yield for which pre-bond tests were skipped earlier.
- The tendency to select pre-bond test increases consistently with the stack size. For example, we see higher threshold die-yield value for dies that are higher up the stack. This can be explained by observing that a defective die, if bonded to a stack that has already been tested to be non-faulty, will make the entire stack faulty; hence it is better to discard the defective die before it enters the stack.

TABLE VIII
SELECTION OF TEST INSERTION AND TESTS FOR $\mathcal{D}_2$ ON VARYING DIE YIELD.

#Tests	Test insertion	Range of die-yield values		
		100% coverage	95% coverage	90% coverage
1	pre-bond	0.6 – 0.94	Not available	Not available
	mid-bond of $\mathcal{S}_2$	0.95 – 0.97		
	mid-bond of $\mathcal{S}_3$	0.6 – 0.99		
	mid-bond of $\mathcal{S}_4$	—		
2	pre-bond	—	0.6 – 0.96	
	mid-bond of $\mathcal{S}_2$	0.6 – 0.79	0.80 – 0.99	
	mid-bond of $\mathcal{S}_3$	—	0.6 – 0.99	
	mid-bond of $\mathcal{S}_4$	—	—	
3	pre-bond	0.6 – 0.67	—	0.68 – 0.97
	mid-bond of $\mathcal{S}_2$	0.68 – 0.80	0.81 – 0.92	0.93 – 0.99
	mid-bond of $\mathcal{S}_3$	—	—	0.6 – 0.99
	mid-bond of $\mathcal{S}_4$	—	—	—

TABLE IX
SELECTION OF TEST INSERTION AND TESTS FOR $\mathcal{D}_1$ ON VARYING TEST COST.

#Tests	Test insertion	Range of test cost		
		100% coverage	95% coverage	90% coverage
1	pre-bond	\$0.10–\$0.34	Not available	Not available
	mid-bond of $\mathcal{S}_2$	\$0.10–\$1.32		
	mid-bond of $\mathcal{S}_3$	—		
	mid-bond of $\mathcal{S}_4$	—		
2	pre-bond	—	\$0.05–\$0.32	
	mid-bond of $\mathcal{S}_2$	\$0.10-\$0.13	\$0.07-\$0.75	
	mid-bond of $\mathcal{S}_3$	—	—	
	mid-bond of $\mathcal{S}_4$	\$0.10-\$0.11	—	
3	pre-bond	—	—	\$0.025–\$0.305
	mid-bond of $\mathcal{S}_2$	\$0.10–\$0.13	\$0.07–\$0.135	\$0.07-\$0.375
	mid-bond of $\mathcal{S}_3$	—	—	\$0.025
	mid-bond of $\mathcal{S}_4$	—	—	\$0.025-\$0.055

If a cheaper test is available, it is chosen to reduce the test cost (depending on the fault coverage of the test). But the decrease in test quality has to be compensated. For the running example, we observed that if a pre-bond test of lower fault coverage is chosen, a post-bond test of higher fault coverage is almost always added in the selected test flow. For example, when just one test of 100% fault coverage is available to $\mathcal{D}_2$, the pre-bond test is selected until the yield reaches its threshold value of 94% (see Table VIII). Starting from 95% onwards, we observed that the pre-bond test is skipped and a mid-bond test for $\mathcal{D}_2$ during stack testing of $\mathcal{S}_2$ is applied later, which is also skipped after die yield exceeds 97%. Another mid-bond test for $\mathcal{D}_2$ during stack testing of $\mathcal{S}_3$ is always selected for all values of the die yield.

Next an extra test for $\mathcal{D}_2$ is added. The fault coverage of the added test is 95%, and it is half the cost of the test with 100% fault coverage. We see that the added test is now selected as pre-bond test. On adding a test, the threshold value of die-yield to select a pre-bond test increases to 96%. On top of that, an additional mid-bond test of 100% fault coverage is also selected for $\mathcal{D}_2$ during testing of stack $\mathcal{S}_2$. For yield values greater than 80%, the test selection changes to the test with 95% coverage for this test insertion. The test with lower fault coverage is also chosen for the test insertion of $\mathcal{D}_3$ during mid-bond test of $\mathcal{S}_3$.

We draw two conclusions here.

- As more tests are available, more test insertions are chosen—typically tests with different coverage values are chosen to compensate for the decrease in quality and to minimize cost at the same time.

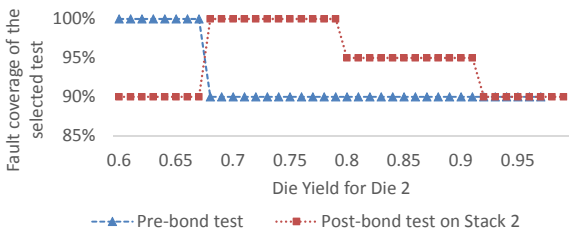

Fig. 9. Comparing test selection at two test insertions of $\mathcal{D}_2$.

- As die yield increases, we see benefit in selecting, at earlier test insertions, tests of relatively lower quality.

Next we add one more test of 90% coverage with its test cost being half that of the test with 95% coverage. In addition to Table VIII, we compare test selection at the two test insertions of $\mathcal{D}_2$, namely, pre-bond test and mid-bond test of $\mathcal{S}_2$ in Figure 9. As can be seen from the figure, the tests selected for the two test insertions are always complimentary to each other. If an expensive test is chosen for the pre-bond test insertion, a cheaper test is chosen as a mid-bond test, and vice versa. After 67% of die yield value, the pre-bond test of 90% coverage (cheapest test) is chosen, and at the mid-bond test insertion, as the die yield increases, the quality of the chosen test decreases. In this example with three tests, $\mathcal{D}_2$ is always tested during mid-bond test of $\mathcal{S}_3$, and the fault coverage of the selected test is 90%.

Next we examine the effect of varying the test cost on selection of tests and test insertions for $\mathcal{D}_1$ in Table IX. We varied the test cost of the die—from a small fraction to a large fraction of its manufacturing cost ($1.80). The test cost of the test with 100% coverage is varied from $0.10 to $1.50 for $\mathcal{D}_1$. The ratio between the test cost of different test is kept the same as before for each case, i.e., the cost of the test with fault coverage of 95% is varied between $0.05 to $0.75 and that of the test with 90% fault coverage is varied between $0.025 to $0.375. The die yield of every die is set to 0.9. We see that the cheapest pre-bond test is always selected except when the test cost becomes expensive and the cost incurred on applying a pre-bond test is no more commensurate with the benefit in applying the test(see Table IX). Increasing the number of available test increases the number of selected test insertion, but with an increase in the test cost, even those test insertions are skipped that were selected earlier for tests with lower costs.

Figure 10 shows the effect of varying the die yield on total cost (in cents) per good package on the primary axis and on the secondary axis, we show two quantities—total good packages and total cost—that are normalized with respect to values of respective quantities when the value of die yield is set to 0.6. As expected, as the die yield increases (or the manufacturing processes mature), the cost per good package declines monotonically, indicating an increase in profitability. The number of fault-free packages increases until the yield for each die becomes 0.94, after which it decreases. A closer look on the fraction of test insertions selected for testing reveals that optimal test flow changes after this point; pre-bond test of $\mathcal{D}_1$ is skipped and a post-bond test for the die is chosen instead (not shown in the figure). Figure 11 shows the number

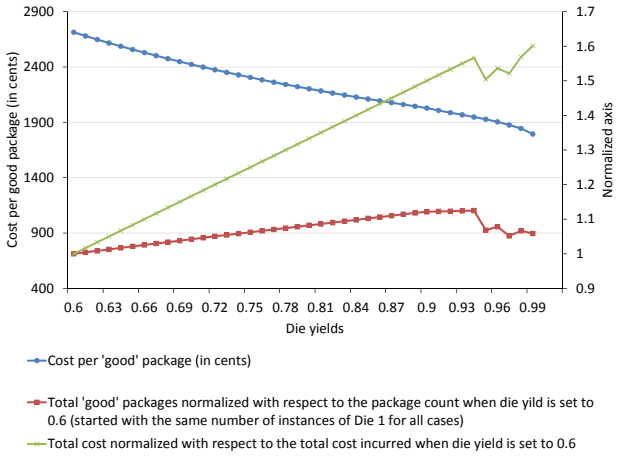

Fig. 10. Effect of varying die yield of all dies simultaneously on cost per good package, total number of good packages, and overall cost.

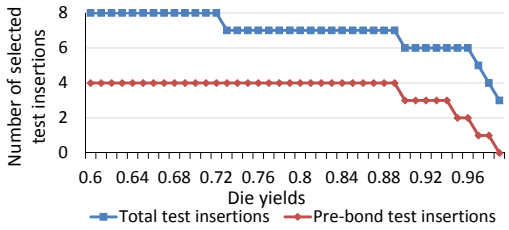

Fig. 11. Effect of varying die yield of all dies simultaneously on test-flow selection.

of test insertions selected in an optimal test flow for the same experiment. We found that as the die yields vary, the selection of test insertions does not change rapidly, except for very high yields. A minor deviation in the actual parameter values is not expected to significantly affect the selection of test flow.

We next compare the proposed methods with a method that exhaustively enumerates all goal nodes (solution candidates). The test cost for a die varies with the fault coverage of the tests being applied to it. We use the number of test patterns as a surrogate measure of test cost [30]. The fault coverage of the available tests are set between 75% to 100%, and the ratio between the test costs of multiple tests is set to match the ratio between the test patterns needed for achieving the fault coverage of the corresponding tests. In addition, we use cost versus fault coverage data provided in [19] as an input parameter for one of the dies in our experiment.

Table X compares the number of nodes explored by different methods. The total number of nodes in the search tree is given by the expression $\sum_{i=0}^{Z}(N+1)^i$, where N is the number of tests available at each test insertion and Z is the number of total test insertions. It can be easily shown that $Z = \frac{l^2+3\times l-2}{2}$, where l is the number of dies in the 3D stack (refer to the appendix). The table also reports the number of nodes explored by the $(1-\Delta)^{-1}$ approximation strategy with the value of Δ set to 0.05. We see that the proposed method explores significantly fewer nodes than the total nodes in the search tree and the approximation scheme further reduces this number. While the result obtained by A*-based method is provably optimal by construction, we always obtained either optimal or near-optimal solutions using the approximation strategy. Even when Δ is 0.05, which can

TABLE X
NUMBER OF NODES EXPLORED USING VARIOUS METHODS.

#Tests per test insertion	two-die stack			three-die stack			four-die stack		
	Exhaustive enumeration	A*-based method	$(1-\Delta)^{-1}$ approx. strategy	Exhaustive enumeration	A*-based method	$(1-\Delta)^{-1}$ approx. strategy	Exhaustive enumeration	A*-based method	$(1-\Delta)^{-1}$ approx. strategy
1	21	12	6	511	75	51	16383	710	585
2	121	11	9	9841	157	120	2391484	7020	5673
3	341	28	15	87381	915	386	8.95×10^7	73063	50452
4	781	52	17	488281	1510	936	1.53×10^9	268029	191848
5	1556	61	25	2015539	3944	2195	1.57×10^{10}	1358892	941427
6	2801	110	36	6725601	11467	5374	1.13×10^{11}	3543762	2960101

TABLE XI
CPU TIMES FOR VARIOUS METHODS FOR THE EXAMPLE OF A FOUR-DIE STACK.

#Tests per test insertion	CPU times		
	Exhaustive enumeration	A*-based method	$(1-\Delta)^{-1}$ approx. strategy
1	< 1 s	< 1 s	< 1 s
2	2 s	< 1 s	< 1 s
3	73 s	< 2	1 s
4	20 m 27 s	6 s	5 s
5	3 h 50 m	41 s	27 s
6	28 h 9 m	7 m 10 s	3 m 2 s

report a solution with a value of the objective function that is $\frac{1}{1-0.05} \approx 1.053$ times its optimum value in the worst case, we found the approximation ratio to be less than 1.001 in every reported case. Note that the approximation strategy starts discarding a "seen" node from subsequent exploration only after a goal node is found. Until then, the A* method is executed. Since the A* method is already guiding the search towards a "good" solution, the first solution that appears on the frontier of "seen" nodes is close to optimal, and a near-optimal solution is subsequently reported.

All results were obtained on a 32-core Intel(R) Xeon(R) machine with a processor speed of 2.60 GHz, 64 GB memory and a cache size of 20480 KB. Implementation was done using the Java programming language and 16 GB of memory was allocated to the Java virtual machine for all runs. CPU times for the example of the four-die stack is shown in Table XI. The run time increases exponentially with increase in the number of tests per test insertion. While the baseline method requires negligible space to the number of test insertions, the memory requirement of the proposed methods increases linearly with the number of states explored, which is true for every A*-based approach. We also ran an experiment on five dies with three tests per test insertion. While the method based on exhaustive enumeration took 72 hours (3 days), the A* based method and the proposed approximation method took 40 minutes and 28 minutes, respectively. For a realistic number of dies per stack and a realistic number of tests per test insertion, we significantly outperform the baseline method with respect to the CPU time.

Thus we note that exhaustive enumeration is impractical for realistic scenarios. Note also that different manufacturing flows lead to different manufacturing cost and die yield, which affect the optimal test flow. In order to evaluate manufacturing flows along with the associated test flows, the test-flow selection tool must be to be invoked repeatedly. For example, depending on the type of vias used, e.g., via-first or via-last, the silicon footprint of a die changes [31], thereby resulting in different manufacturing cost and die yield values. As a result, multiple invocation of the test-flow selection tool may be required to assess the economic viability of these manufacturing flows. The enormous CPU time taken by trivial exhaustive enumeration-based search makes its usage impractical for multiple runs. Since exhaustive enumeration is computationally so expensive, it is likely to require repeated invocation of the test-flow selection even for a fixed manufacturing flow. This can happen because not all options can be considered in a single run of test-flow selection due to the exponential rate at which complexity grows with the number of options. With the proposed intelligent search technique, run time complexity is limited to less than an hour and all options can be considered in one run.

VIII. CONCLUSION

We have studied the test-flow selection problem for achieving cost minimization, and proposed a generic and flexible cost model to account for various test costs incurred during 3D integration. We have formulated the optimization problem as a search problem and proposed a method based on the A*-search algorithm to find an optimal solution. A $(1-\Delta)^{-1}$ approximation step has also been presented that further reduces the run time of the algorithm. Solutions to the test-flow selection problem depends on the problem instance. Because of the interplay of given parameter values, optimal choices of tests and test insertions are dependent on problem instances; therefore, a generic set of rules cannot be specified for minimizing cost. Nevertheless, experimental results have helped us to rationally explain the choices made by the problem solver for obtaining an optimal solution. The cost-optimization method has provided us interesting insights into relationships between yield, test cost, and the selection of various pre-bond and stack tests.

REFERENCES

[1] M. Agrawal and K. Chakrabarty, "Test-Cost Optimization and Test-Flow Selection for 3D-Stacked ICs," in *VProc. VTS*, 2013.

[2] K. Banerjee *et al.*, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proc. IEEE*, vol. 89, no. 5, pp. 602–633, May 2001.

[3] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," *IEEE J. Solid-State Circuits*, vol. 45, pp. 111–119, Jan. 2010.

[4] M. Kawano *et al.*, "A 3D Packaging Technology for 4 Gbit Stacked DRAM with 3 Gbps Data Transfer," in *Proc. IEDM*, 2006, pp. 1–4.

[5] B. Feero and P. Pande, "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 32–45, Jan. 2009.

[6] G. H. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro*, vol. 27, no. 3, pp. 31–48, May-June 2007.

[7] C. Ababei, P. Maidee, and K. Bazargan, "Exploring Potential Benefits of 3D FPGA Integration." in *Field Programmable Logic and Application*, vol. 3203. Springer, 2004, pp. 874–880.

[8] B. Black *et al.*, "3D Processing Technology and its Impact on iA32 Microprocessors," in *Proc. ICCD*, 2004, pp. 316–318.

[9] B. Noia and K. Chakrabarty, "Pre-Bond Probing of TSVs in 3D Stacked ICs," in *Proc. ITC*, 2011.

[10] B. Noia *et al.*, "Test-Architecture Optimization and Test Scheduling for TSV-Based 3-D Stacked ICs," *IEEE Trans. CAD*, vol. 30, Nov. 2011.

[11] H. Lee and K. Chakrabarty, "Test Challenges for 3D Integrated Circuits," *IEEE Design & Test of Computers*, vol. 26, pp. 26–35, Sept.-Oct. 2009.

[12] E. J. Marinissen and Y. Zorian, "Testing 3D Chips Containing Through-Silicon Vias," in *Proc. ITC*, 2009.

[13] M. Taouil *et al.*, "Test Cost Analysis for 3D Die-to-Wafer Stacking," in *Proc. ATS*, 2010, pp. 435–441.

[14] Y. Chen *et al.*, "Cost-Effective Integration of Three-dimensional (3D) ICs Emphasizing Testing Cost Analysis," in *Proc. ICCAD*, 2010, pp. 471–476.

[15] Y.-W. Chou *et al.*, "Cost Modeling and Analysis for Interposer-Based Three-Dimensional IC," in *Proc. VTS*, 2012, pp. 108–113.

[16] M. Taouil, S. Hamdioui, E. Marinissen, and S. Bhawmik, "3D-COSTAR: A Cost Model For 3D Stacked ICs," in *International Workshop on Testing Three-Dimensional Stacked Integrated Circuits*, 2012.

[17] M. Taouil, S. Hamdioui, E. J. Marinissen, and S. Bhawmik, "Using 3D-COSTAR for 2.5 D Test Cost Optimization," in *3D Systems Integration Conference (3DIC)*, 2013, pp. 1–8.

[18] ——, "Impact of Mid-Bond Testing in 3D Stacked ICs," in *Intl. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2013, pp. 178–183.

[19] M. Taouil, S. Hamdioui, and E. J. Marinissen, "Quality versus Cost Analysis for 3D Stacked ICs," in *Proc. VTS*, 2014.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[21] D. Klein and C. D. Manning, "A Parsing: Fast Exact Viterbi Parse Selection," in *Proc. Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003, pp. 40–47.

[22] J. Liu and Y. Liu, "Application of A* Algorithm in Traffic Navigational System," in *Intl. Symp. Information Engineering and Electronic Commerce (IEEC)*, July 2010.

[23] A. Autere, "A* as an Optimal Resource Allocation Policy in Path Finding Problems," in *Proc. Florida Artificial Intelligence Research Society Conference*, 2001, pp. 139–144.

[24] T. Williams and N. Brown, "Defect Level as a Function of Fault Coverage," *IEEE Trans. Computers*, vol. C-30, pp. 987–988, Dec. 1981.

[25] J. Schat, "On the Relationship between Stuck-at Fault Coverage and Transition Fault Coverage," in *Proc. DATE*, 2009, pp. 1218–1221.

[26] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009.

[27] "Sematech Wafer Cost Comparison Calculator." [Online]. Available: http://ismi.sematech.org/modeling/agreements/wafercalc.htm

[28] D. De Vries, "Investigation of Gross Die per Wafer Formulas," *Tran. Semiconductor Manufacturing*, vol. 18, no. 1, pp. 136–139, Feb 2005.

[29] D. Velenis *et al.*, "Impact of 3D Design Choices on Manufacturing Cost," in *Intl. Conf. 3D System Integration*, Sept. 2009.

[30] S. Edbom and E. Larsson, "An Integrated Technique for Test Vector Selection and Test Scheduling under Test Time Constraint," in *Asian Test Symposium (ATS)*, Nov. 2004, pp. 254–257.

[31] D. H. Kim, K. Athikulwongse, and S.-K. Lim, "A Study of Through-Silicon-Via Impact on the 3D-Stacked IC Layout," in *Intl. Conf. Computer-Aided Design*, Nov 2009, pp. 674–680.

Appendix

Proof for the complexity of the selection of test flow

Given a stack of l dies, we show that the total number of ways in which test insertions can be selected is $O(2^{l^2})$. Corresponding to each pre-bond stage, there is one test insertion each. Since k^{th} die can potentially be tested at each subsequent stacking stages, this accounts for an additional $l-k+1$ test insertions for the k^{th} die ($l-1$ insertions for $k=1$). Therefore, the total number of possible test insertions is given by the expression:

$$l + \sum_{k=2}^{l}(l-k+1) + l - 1 = 2l - 1 + \frac{l(l-1)}{2} = \frac{l^2+3l-2}{2}.$$

A test insertion can be selected or discarded independently of the selection (or omission) of other insertions, hence the number of ways in which test insertions can be selected is $2^{\frac{l^2+3l-2}{2}} = O(2^{l^2})$.

If the number of available tests per test insertion is N, then there are $N+1$ choices at each test insertion. Therefore, the number of possible test flows is $O((N+1)^{l^2}) = O(N^{l^2})$. In general, the number of possible test flows is the product of the available number of choices at each test insertion.

An Example for a two-die stack

The cost model is illustrated through an example of a two-die stack in Table XII. Three tests (all derived from same fault model) with fault coverage of 100%, 95% and 90%, respectively, are assumed to be available for testing die $\mathcal{D}_1$ at both of its test insertions. The effective fault coverage (EFC) of tests with 90% and 95% of fault coverage is assumed to be 97%. Since the tests are for the same fault model, AFC of the tests is same as their EFC. For testing $\mathcal{D}_2$, only one test with fault coverage of 100% is available. Every possible test flow is enumerated, and an expression for the number of stacks available after applying post-bond test of stack $\mathcal{S}_2$ (m_2') for each test flow is shown in Table XII. This is the number of stacks that are packaged and further tested using the package test. The number of packages available after applying the package test is also provided against each test flow.

If $\mathcal{D}_1$ is tested using tests with fault coverage of 90% and 95% at different test insertions (see test flows with indices 39, 40, 47, 48, 53, 54, 61 and 62 in Table XII), in the column corresponding to the cost model presented in Section III, we see that that λ_1 is raised to the power of the aggregate fault coverage of the two tests because the defect introduced during the manufacturing of die is tested twice. The power of $\omega_{1,2}$, on the other hand, is always the fault coverage of the test applied during the post-bond test because the defects introduced during stacking is tested only once.

For Model I discussed in Section IV.A, the aggregate fault coverage is rounded to the maximum fault coverage of the applied tests, and for Model II, test escapes from earlier test insertions were ignored in calculation of m_2'. The difference between these models can be clearly seen for the test flows with indices from 51 through 64.

If $\mathcal{D}_2$ is tested using its only available pre-bond test, then λ_2 does not appear in the expression for m_2' because manufacturing defects for the die are already screened out by applying the pre-bond test. If another pre-bond test with fault coverage fc was applied to $\mathcal{D}_2$, we would have seen a factor of λ_2^{1-fc} in the expression.

The number of packages remaining after applying package test is $n\lambda_1\omega_{1,2}\omega_{2,2}$ for the flows when a pre-bond test is applied to $\mathcal{D}_2$, otherwise the package count reduces to $n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$. If a pre-bond test with fault coverage fc was applied to $\mathcal{D}_2$, then the number of packages available would have been $n\lambda_1\omega_{1,2}\lambda_2^{1-fc}\omega_{2,2}$.

TABLE XII
ILLUSTRATION OF THE COST MODEL THROUGH AN EXAMPLE OF A TWO-DIE STACK.

Test-flow index	Fault coverage of the selected test (%)				Expression for m_2'			Total package count
	Pre-bond		Post-bond		Cost model from Section III	Models from Section IV		
	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_1$	$\mathcal{D}_2$		Model I	Model II	
1	—	—	—	—	n	n	n	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
2	—	—	—	100	$n\lambda_2\omega_{2,2}$	$n\lambda_2\omega_{2,2}$	$n\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
3	—	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
4	—	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
5	—	—	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
6	—	—	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
7	—	—	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
8	—	—	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
9	—	100	—	—	n	n	n	$n\lambda_1\omega_{1,2}\omega_{2,2}$
10	—	100	—	100	$n\omega_{2,2}$	$n\omega_{2,2}$	$n\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
11	—	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
12	—	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
13	—	100	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
14	—	100	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
15	—	100	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
16	—	100	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
17	100	—	—	—	$n\lambda_1$	$n\lambda_1$	$n\lambda_1$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
18	100	—	—	100	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
19	100	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
20	100	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
21	100	—	95	—	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
22	100	—	95	100	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
23	100	—	90	—	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
24	100	—	90	100	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
25	100	100	—	—	$n\lambda_1$	$n\lambda_1$	$n\lambda_1$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
26	100	100	—	100	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
27	100	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
28	100	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
29	100	100	95	—	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
30	100	100	95	100	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
31	100	100	90	—	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
32	100	100	90	100	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
33	95	—	—	—	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
34	95	—	—	100	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
35	95	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.95}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
36	95	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
37	95	—	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
38	95	—	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
39	95	—	90	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
40	95	—	90	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
41	95	100	—	—	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
42	95	100	—	100	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
43	95	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.95}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
44	95	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
45	95	100	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
46	95	100	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
47	95	100	90	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
48	95	100	90	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
49	90	—	—	—	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
50	90	—	—	100	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
51	90	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.9}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
52	90	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
53	90	—	95	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
54	90	—	95	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
55	90	—	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
56	90	—	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
57	90	100	—	—	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
58	90	100	—	100	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
59	90	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.9}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
60	90	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
61	90	100	95	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
62	90	100	95	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
63	90	100	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
64	90	100	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$