Efficient Exact Spare Allocation via Boolean Satisfiability

Fang Yu[2], Chung-Hung Tsai[2], Yao-Wen Huang[12], Hung-Yau Lin[1], D. T. Lee[2], Sy-Yen Kuo[1]

[1]Department of Electrical Engineering,
National Taiwan University,
Taipei 106, Taiwan.
sykuo@cc.ee.ntu.edu.tw

[2]Institute of Information Science,
Academia Sinica,
Taipei 115, Taiwan
{ yuf,chtsai, ywhuang, dtlee}@iis.sinica.edu.tw

Abstract

Fabricating large memory and processor arrays is subject to physical failures resulting in yield degradation. The strategy of incorporating spare rows and columns to obtain reasonable production yields was first proposed in the 1970s, and continues to serve as an important role in recent VLSI developments. Since the spare allocation problem (SAP) is NP-complete but requires solving during fabrication, an efficient exact spare allocation algorithm has great value. Here we propose a new Boolean encoding of SAP and a new SAT-based exact algorithm SATRepair. We used a realistic fault distribution model to compare SATRepair's performances against BDDRepair and those of algorithms found in the literature. We suggest that a) our novel Boolean encoding of SAP allows for the development of efficient exact SAP algorithms, and b) our SAT-based algorithm outperforms previous algorithms, especially for large problems.

1. Introduction

Repairable arrays continue to serve as an important component in recent VLSI developments, and not only because fabrication imperfections grow proportionally with chip size and density. As today's photolithography techniques approach physical limits in terms of density growth and device miniaturization, future fabrication will depend on such new technologies as Next Generation Lithography (NGL) techniques and Chemically Assembled Electronic Nanotechnology (CAEN). NGL and CAEN share one major disadvantage: the likelihood of having significantly higher defect densities; an expected defect density of as high as 10 percent requires efficient fault repair algorithms [32]. In addition, in 2001 the International Technology Roadmap for Semiconductors (ITRS) [1] reported that SoCs have moved from logic-dominant to memory-dominant chips (see Figure 1), and will embed memories of increasing sizes. This trend, which was confirmed by the ITRS 2003 report [2], implies that SoC yields are dominated by memory yields; hence, SAP algorithm efficiency directly affects today's SoC yields. This holds true not only for SoCs, but also for current state-of-the-art processors in general. For example, in the Intel Itanium 2 Processor, the 9MB third-level on-die cache contains over 90 percent of all the 592M transistors on the 432mm^2 die [37]. Therefore, as feature sizes shrink to single digit nanometer dimensions and memory takes up a major area share of SoCs and advanced processors, defect tolerance is becoming increasingly important.

A common treatment since the 1970s has been to use redundancy to replace faulty modules (see for example Schuster [35]). This strategy is most suitable for uniformly structured chips such as memory arrays, in which rows and columns of spare cells can be employed to replace faulty ones that occur during fabrication. The goal of the *spare allocation* (sometimes referred to as *memory reconfiguration*) *problem (SAP)* is to find an assignment of spare rows and columns to faulty rows and columns such that all faulty cells

are eliminated. Kuo and Fuchs showed that SAP is NP-Complete [25]. Since a) redundancy was quickly incorporated into the design and fabrication of memory chips (see [8] [16] [23] [13] [14] [38] for early chips with spares), and b) memories began to comprise a major portion of processors (e.g., 44% of all UltraSparc's transistors in 1997 [26]) and therefore became yield drivers [40] (see for example reports on DEC Alpha [3] and UltraSparc [26] in 1997), consistent efforts have been devoted to developing heuristic (approximation) algorithms or efficient exact algorithms for SAP.

To solve SAP, Kuo and Fuchs proposed a branch-and-bound (B&B) algorithm that searches the most promising partial solutions first. The algorithm is exact (i.e., it always finds a solution if one exists) and has been shown to outperform most of its predecessors. Therefore, subsequent efforts have focused on reducing B&B's search space (e.g., Lombardi and Huang's *Faulty-Line Covering* [27] and Hemmady and Reddy's *Quick-Terminate* [22]) or on developing faster exhaustive search algorithms with reduced search spaces (e.g., Hasan and Liu's *Critical Sets* [21], Hemmady and Reddy's *Ternary Repair Algorithmn (TAR)* [22], and Libeskind-Hadas and Liu's *Excess-k Critical Sets* [19]).

Recently, Fernau and Niedermeir [15] have defined SAP as a *Constraint Bipartite Vertex Cover (CBVC)* problem. Using parameterized theory [12], they proposed a CBVC algorithm having running time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$ for an array with k_1 spare rows, k_2 spare columns, and an n sum of rows and columns. Chen and Kanj [9] later developed a parameterized algorithm that runs in $O(1.26^{k_1+k_2} + (k_1 + k_2)n)$.

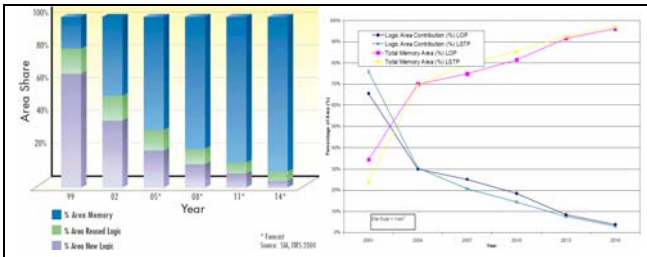

Figure 1. ITRS 2000 and 2001 forecast that memory will occupy more than 90 percent of SoC area by 2016. (Source: ITRS 2000 and 2001)

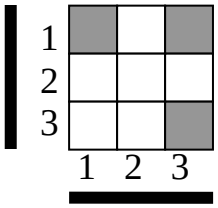

Figure 2. A 3 by 3 array. S=({(1,1), (1,3), (3,3)}, 1, 1)

Rather than propose a new B&B enhancement or parameterized algorithm, in this paper we will propose a novel algorithm that formulates SAP as an instance of the Boolean Satisfiability Problem (SAT) [10] [17]. Since SAT is theoretically a *core* of a large family of intractable NP-Complete problems, methods for solving SAT play important roles in the development of computing theory. Furthermore, since practical problems from such areas as AI planning, circuit testing, and software verification (among others) can be formulated as SAT instances [42], there is a strong motivation to study SAT and to develop practically efficient SAT algorithms [18]. By formulating SAP as an SAT problem, we can benefit from existing SAT algorithms. The two major difficulties in deriving an efficient formulation are a) preventing explicitly enumerating all problem combinations and b) restraining the formula size to a minimal. Here we address both difficulties and propose a new Boolean SAP encoding based on dynamic programming.

The two most common approaches to solving Boolean formulas are by using BDDs (Binary Decision Diagrams, which can be used to represent Boolean functions and to support efficient Boolean operations) [7] or by using SAT solvers (i.e., those SAT solvers based on the Davis-Logemann-Loveland algorithm [11]). Lin et al. recently [28] introduced a BDD-based SAP algorithm *BDDRepair*, which outperformed the B&B algorithm incorporating

enhancements. This attractive strategy has two drawbacks: a) its performance relies largely on variable ordering, and selecting an optimal ordering is considered an NP-Complete problem; b) BDD size expands exponentially with problem size and therefore makes it difficult to handle complex problems.

On the other hand, contemporary SAT solvers can handle hundreds of thousands of variables without being challenged by the variable ordering problem; however, they are often inefficient in terms of proving a problem's unsatisfiability. The purpose of spare incorporation is to improve yield, or more precisely, throughput. Therefore in practice, fabrication processes require that SAP solutions be found within very limited time and those chips whose solutions cannot be derived within time limit are simply discarded. This makes a SAT-based algorithm ideal for practical application, since SAT solvers are efficient in proving a problem's satisfiability (i.e., finding a solution when one exists), especially when many solutions exist. In IC fabrication, it is the manufacturing engineers' job to monitor yield and make sure enough spares are incorporated so that the majority of defective chips are repairable. While it may not be possible to construct a corresponding BDD for a large repairable array with many faulty cells and many spares, a SAT solver can quickly identify a solution. The disadvantage of SAT solvers is that they are slow for SAP instances having few or no solutions, since they have to exhaustively consider all possible values for branching variables; however, such a situation is unlikely to occur in practice.

SAP algorithm evaluations are best performed by testing them with actual fault distributions obtained from an IC manufacturing line. However, such data are not easily obtainable because many commercial secrets can be uncovered by analyzing them. Therefore, SAP algorithm evaluation is usually conducted by generating simulated fault distributions and then applying the algorithm to find a repair solution. This implies that a precise fault model is critical for correct evaluation. Lin et al. used a Poisson fault model in their previous evaluation of BDDRepair [28]—that is, they assumed a uniform and random distribution of defects so that each cell is statically independent. However, as early as 1964, Murphy [34] argued that this model does not correctly capture realistic fault distributions. For the present project we adopted the enhanced center-satellite model proposed by Blough and Pelc [5] (see Meyer and Pradhan [30] for the original). In Section 4, we explain our reasons for selecting this model and present the results of our experiment comparing BDD- and SAT-based algorithms using 14 test sets representing different array configurations (e.g., size and spare rows/columns), with each test set containing 100 fault distributions generated randomly by Blough and Pelc's model.

We believe this paper makes five contributions to the current literature on this topic. First, we formulate SAP into a Boolean Satisfiability problem, and our novel Boolean encoding allows for the development of more efficient BDD- and SAT-based algorithms. Second, we observe that the nature of SAP makes it ideal for SAT application. Third, we introduce a novel SAT-based algorithm named *SATRepair*. Fourth, a precise fault model is critical for correct evaluation of SAP algorithms, and we proposed a realistic fault model. Fifth, we report the results of experiments comparing SATRepair against BDDRepair and those of algorithms found in the literature, and suggest that SATRepair is more efficient in practice.

2. SAP and BOOLEAN encoding

Here we present a novel Boolean encoding of SAP that is solvable using either BDDs or SAT solvers. We first give our formal definition of SAP.

SAP Definition: A SAP instance S is a tuple (*D, m, n*), where *m* and *n* are two integers and *D* is a set of defects with each defect denoted as (i, j). i indicates the row and j indicates the

column that contain the defect Given a set *D* of defects, $R_D = \{i \mid \exists(i, j) \in D\}$ denotes the set of defect rows and $C_D = \{j \mid \exists(i, j) \in D\}$ denotes the set of defect columns. We say a SAP instance $S = (D, m, n)$ is solvable if there exists a solution pair *(R,C)* that satisfies the following conditions: (1) $\forall(i, j) \in D, i \in R$ or $j \in C$, and (2) $|R| \le m$ and $|C| \le n$.

Our proposed Boolean encoding of a SAP instance *S* uses defective rows (e.g.,r_i) and columns (e.g., c_j) as our formula's literals. Our two objectives in this formula design are (1) S is solvable if the Boolean formula is satisfied and (2) the pair *(R,C)* is one of *S*'s solutions, where R={i|α(r_i)=1}, C={j|α(c_j)=1}, and $\alpha \in A : R_D \cup C_D \mapsto \{0,1\}$ is a truth assignment of the Boolean formula. The encoding is composed of two Boolean formulas: the *defect function (DF)* and the *constraint function (CFR or CFC)*. *DF* encodes the *lines* (rows or columns) containing defects, while *CFR/CFC* encodes the constraints of all combinations of defective rows/columns that spare lines can repair.

2.1 The Defect Function

The defect function *DF* encodes the lines containing defects and apparently the following defect function *DF* encodes all faulty lines of a SAP instance. A defect function *DF* is a function mapping a defect distribution *D* to a Boolean formula.

$$DF(D) = \prod_{(i,j)\in D} r_i \vee c_j \ .$$

2.2 The Constraint Function

To limit the solution set to the fixed number of spare lines, we apply the constraint function, which maps a set of defective lines and a maximum spare line number (an integer) to a Boolean formula such that the formula encodes all combinations of defect lines that spare lines can repair. Formally, given a set *L* of defective lines and an integer *m*, a naïve solution is to enumerate all possible combinations by choosing the repaired lines. For example, in Figure 2-a, there are 2 faulty rows (i.e., $R_D = \{1,3\}$) but only one spare row (i.e., m=1) could be used. Therefore, we have two options: a) replace either row 1 or 3, or b) replace no row. Combining these terms yields the formula $\overline{r_1}\,\overline{r_3} \vee \overline{r_1} r_3 \vee r_1 \overline{r_3}$. A similar idea was recently proposed by Lin et al. [28].

In our proposed encoding, we choose lines which are not allowed to be repaired rather than the repaired lines. That is, given a line set *L* and an integer *m*, we choose |*L*|-*m* lines and construct our formula using the conjunction of the negation of these chosen lines. Our constraint function for Figure 2-a is therefore $\overline{r_1} \vee \overline{r_3}$. Compared to Lin et al.'s encoding [28], ours has at least two advantages: a) since we do not enumerate all possible lines used, we reduce the number of combinations from

$\sum_{k=0 \text{ to m}} \frac{n!}{k!(n-k)!}$ to $\frac{n!}{m!(n-m)!}$, where *n* is the size of the set *L*,

and b) we reduce the formula length of each combination from *n* to *n-m* since our formula is the conjunction of the negation of the chosen lines (compared to the conjunction of the used lines and the conjunction of the negation of the unused ones). Even with these advantages, enumerating all combinations of large *L* and *m* usually makes satisfiability solving infeasible (see results in Section 4). Therefore, we further introduce the function *CFR()* and *CFC()* for which we apply dynamic programming for efficient formula construction. We assign to each pair (*L*,*m*) a unique hash key referring to the literal representing the Boolean formula returned by *CFR(L,m) or CFC(L,m)*. Adopting dynamic programming allows for preventing formula reconstruction and

yields two major benefits: a) reduction of computation cost and b) reduction of the number of literals (reduction of unnecessary literals).

CFR(L,m) {	*CFC(L,m)* {
if m<=0, return *false*;	**if** m<=0, return *false*;
else if(|*L*|>m)	**else if**(|*L*|>m)
select some *i*∈L;	select some *i*∈L;
return $\left(\overline{r_i} \wedge CFR(L \setminus i, m-1)\right) \vee CFR(L \setminus i, m)$;	return $\left(\overline{c_i} \wedge CFC(L \setminus i, m-1)\right) \vee CFC(L \setminus i, m)$;
else if (|L|==m) return $\prod_{i \in L} \overline{r_i}$;	**else if** (|L|==m) return $\prod_{i \in L} \overline{c_i}$;
else return *true*; }	**else** return *true*; }

Finally, for any SAP instance *(D,m,n)*, we construct a formula $F = DF(D) \wedge CFR(R_D, m) \wedge CFC(C_D, n)$ and solve the SAP problem by solving *F*'s satisfiability.

3. Solving the Boolean Formulas

Although here we design a SAT-based approach, we note that our proposed SAP Boolean encoding allows the use of BDD-based approaches as well. Given a proposition formula, the Boolean Satisfiability problem (SAT) is to determine whether there exists a variable assignment such that the formula evaluates to true. Lin et al.'s BDDRepair [28] essentially solves the Boolean formulas by yielding *all* satisfying assignments. However, BDDs for large problems are difficult to construct and good variable orderings are also difficult to derive. Array repair only requires a single satisfying assignment, therefore an obvious alternative would be to locate that assignment in an *n*-dimensional Boolean variable space. Many researchers are working to develop efficient versions of this kind of search-based algorithms (commonly referred to as *SAT solvers*), since a large family of intractable and practical problems (e.g., NP-Complete, AI planning, circuit testing, FPGA routing, VLSI CAD, and software verification) can be modeled as SAT instances [18] [42].

SAT solvers typically take a propositional formula as input and generate a solution as output (i.e., a variable assignment) if one exists. To improve solver efficiency, a propositional formula is usually presented in a product of *Sum* form, usually called a *Conjunctive Normal Form (CNF)*. A CNF formula is a logical *and* (denoted by ∧) of one or more *clauses*, with each clause being a logical *or* (denoted by ∨) of one or more *literals*. A literal is either the positive or the negative occurrence of a variable. Satisfying a CNF formula requires that each clause be satisfied individually. A variable assignment that satisfies a formula (i.e., allows the formula to evaluate to true) is called a *truth assignment* of the formula. A variable assignment is not a formula's truth assignment if all literals within a certain clause (called a *conflicting clause*) of the formula evaluate to false according to the assignment. At each node in the search tree, modern SAT solvers—usually based on the Davis-Logemann-Loveland (DPLL) algorithm [11]—uses current conflicts to prune subsequent search space. If a current variable assignment satisfies the formula, a solution is found. If a current variable assignment induces conflicts, SAT solvers will backtrack on the assignment (e.g., performing the conflict-driven analysis). If a current variable assignment fails to either satisfy a formula or induce conflicts, solvers will select a new branching variable based on their individual heuristics and repeat the process until a) a solution is found or b) they have exhausted the search space, implying that no solutions exist.

The process of pruning subsequent search space (known as deduction) is a key part of DPLL-based SAT solvers. Deduction derives a set of necessary variable assignments that can be deduced from the existing variables assignments, usually by iteratively applying the unit

clause and pure literal rules [41]. This process (sometimes referred to as Boolean Constraint Propagation or BCP) accounts for the major portion of a solver's running time. The zChaff solver [33] (which incorporates a novel BCP algorithm called 2-literal) outperforms many SAT solvers. As part of this project, we leverage zChaff in SATRepair to solve SAP forluma satisfiability. We reserve our Boolean state variables (i.e., defective lines) using a bit-vector, and dynamically construct the hash table of *CFR()* such that each hash key of *(L,m)* indicating an internal variable exactly equals to the evaluation of *CFR(L,m)*. (*CFC()* is treated similarly). This can be achieved via dynamically calling the circuit construction of zChaff. Finally, the translation of formulas into CNF is trivial since we have built a circuit representation.

4. Experiment Design and Results

Most SAP algorithms are evaluated using simulated fault distributions, implying that precise fault models are critical to correct evaluations, especially for *probabilistic analyses* of SAP algorithms.

4.1 Fault Model

Since SAP is NP-Complete, exact algorithms rely on exhaustive searches with exponential running times. However, it is possible that the worst cases occur very infrequently, and therefore some algorithms exist that are efficient for most cases [4]. Shi and Fuchs [36] and Blough [4] used this observation to propose separate algorithms and used *probabilistic analyses* to determine their average-case running times according to certain defect distribution models. Murphy [34] discovered as early as 1964 that the model tends to underestimate yields for larger die sizes, and therefore proposed a compound Poisson model. It later became clear that lower predicted yields resulted from the fact that faults occur in clusters rather than independently in different chip regions [24] [39]. Clustered faults have a higher chance of being repaired compared to those that are uniformly distributed.

Multiple proposals have been offered to describe the physical basis and thereby appropriate mathematical modeling for defect distributions [31]. Stapper [39] proposed the quadrat-based model that Blough [4] used to prove the average-case running time of his *ClusterReconfig* heuristic algorithm. The model was simple but too restrictive, and its assumption of non-overlapping defects was unlikely in practice [5]. Blough [5] therefore used Meyer and Pradhan's center-satellite model [30] to perform a probabilistic analysis of his *ClusterReconfigNew* heuristic algorithm. In a later study of the *QRCF* heuristic algorithm [6], Blough argued that while it is well suited to defect cluster modeling, the large number of parameters associated with the center-satellite model makes parameter estimation difficult, and therefore used again the quadrat-based model to evaluate QRCF.

We used the center-satellite model described by Blough in [5] for our test case generation, since although it involves complex parameters, it can more accurately reflect realistic fault distributions. Other proposed center-satellite model variations and defect models are beyond the scope of this paper; interested readers are referred to a recent survey by Meyer and Park [31].

4.2 Experiment Design and Results

Lin et al. [28] showed that BDDRepair is significantly faster than Kou and Fuchs's [22] original B&B algorithm with numerous enhancements. In this project we compared SATRepair with BDDRepair as well as with B&B and Hadas and Liu [24]'s *excess-k* algorithm, which has been proved to outperform all predecessors. We compared three

versions of SATRepair. *SATRepair_1* implements our proposed Boolean encoding and SAT algorithm, but without the dynamic programming mechanism we proposed in Section 2.2. Instead of using the new Boolean encoding we proposed in Section 2, *SATRepair_2* uses Lin et al.'s [28] original Boolean encoding, but solves the formulas using our SAT-based algorithm instead of Lin et al.'s BDD-based algorithm. Although it does not use our proposed encoding, it incorporates our proposed dynamic programming mechanism. *SatRepair_3* enhances SATRepair_1 with dynamic programming.

Set	n	SR=SC	m_n	p_1	p_2	Faulty cells
1	1024	32	5	0.000009	0.8	189
2	1024	32	5	0.000013	0.8	270
3	1024	32	5	0.000017	0.8	355
4	1024	32	9	0.000003	0.8	204
5	1024	32	9	0.000007	0.8	472
6	1024	32	9	0.000009	0.8	608
7	1024	32	15	0.000002	0.8	372
8	1024	32	15	0.000004	0.8	759
9	1024	32	15	0.000007	0.8	1309
10	1024	36	15	0.000004	0.8	752
11	1024	36	9	0.000007	0.7	412
12	2048	64	7	0.000004	0.7	576
13	2048	64	9	0.000003	0.5	511
14	4096	128	7	0.000002	0.7	1149

n: array size SR/SC: spare row/column number
m_n, p_1, p_2: parameters of Blough's model [5]
Faulty cells: Average number of faulty cells

Figure 3 The fourteen test sets (100 test cases each) used to evaluate SATRepair.

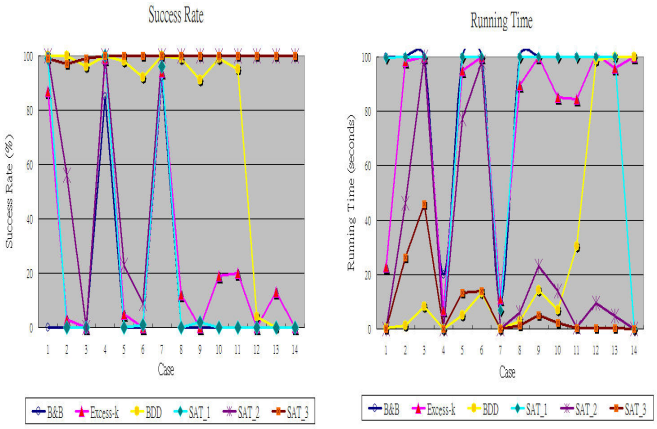

Figure 4. Plot of average running times and success rates for *BDDRepair* and *SATRepair*.

As shown in Figure 3, we used Blough's [5] model to randomly generate 1400 test cases (14 sets of 100 cases each). The test sets contained different combinations of array size, spare amount, and defect density. In the figure they are ordered by problem size—the higher the test set number, the more complex the problem.

We ran B&B, Excess-k, BDDRepair and SATRepair (all three modes) against these 1400 test cases and recorded their running times. We conducted our experiments with an Intel Xeon processor (2.40GHz with 512 KB cache) with 1 gigabyte of memory running Red Hat Linux release 8.0 (Psyche) with kernel v2.4.18-14 (2 gigabytes of swap). We compiled our programs using gcc 3.3.4 with the optimization features turned on. In practice, a major concern in the chip reconfiguration process is *throughput* [20], defined as the number of good chips produced per unit time. Chips are discarded if their reconfiguration times are too long compared to their manufacturing times [29]. Accordingly, we aborted the algorithm and marked it as *failed* if an algorithm failed to reach completion within 100 seconds. Average running times and success rates for all compared algorithms are plotted in Figures 4. We counted the running times of failed cases as 100 seconds. The figures show that in terms of running time, BDDRepair outperformed B&B and Excess-k for all test sets, while B&B and SATRepair_1 exhibited the worst performance. SATRepair_2 outperformed BDDRepair starting from test set 11 when the array size reached 1024. In fact, BDDRepair exhibited a sharp increase in running time from test set 10 to 11, and failed for almost all cases in test sets 12 to 14. However, BDDRepair exhibited a distinct advantage over SATRepair_2 for the first 7 test sets, and a slight advantage for test sets 7 to 10. SATRepair_3 demonstrated notable improvements over SATRepair_2—compared with BDDRepair, its performance was slightly worse from sets 1 to 4, comparable or better from sets 4 to 7, slightly better from 7 to 10, and remarkably better from 11 to 14.

5. Conclusion

In this paper we argued that the nature of SAP makes it ideal for SAT application and proposed a new Boolean SAP encoding that allows for more efficient development of both BDD- and SAT-based SAP algorithms. Based on the new encoding, we developed a novel SAT-based SAP algorithm named *SATRepair*. A precise fault model is critical for the correct evaluation of SAP algorithms. We used that Blough's model to randomly generate 1400 test cases representing different combinations of array size, spare amount, and defect density. Our results showed that BDDRepair slightly outperformed SATRepair_3 for very small problems (test sets 1 to 4), but that it failed to meet our 100-second time constraint as problem size increased. We also observed that its average running time increased at a much faster rate than that of SATRepair. These results support our assumptions that BDD size expands exponentially with problem size and that SAT solvers, which are designed to handle hundreds of thousands of variables, are capable of quickly solving SAPs with many solutions.

We therefore suggest the following for BDDRepair and SATRepair.

1. BDDRepair is better suited to smaller problems.
2. BDDRepair is better suited to fabrication processes that yield higher percentages of irreparable arrays. However, this implies an uneconomical and unprofitable manufacturing process that is unlikely to exist in practice.
3. SATRepair is inefficient for irreparable arrays, but is otherwise more efficient than BDDRepair.
4. SATRepair can handle problems with sizes that are too large for BDDRepair.

As mentioned, condition 2 is unlikely to occur in practice. Chip designers use various fault models and yield analysis algorithms to ensure that they incorporate enough spares so as to make most chips repairable. Since this implies solutions will exist in the vast majority of cases, SATRepair has an advantage over BDDRepair. For condition 3, SATRepair can be aborted before 100 seconds, since the guiding goal is to improve the throughput of a manufacturing process. Doing so will yield a higher average running time for SATRepair. Finally, since there is currently a strong movement toward developing efficient SAT algorithms for practical applications [18], SATRepair will benefit from future SAT solver developments.

REFERENCES

[1] *International Technology Roadmap for Semiconductors*, 2001.

[2] *International Technology Roadmap for Semiconductors*, 2003.

[3] D. K. Bhavsar, J. H. Edmondson. "Alpha 21164 Testability Strategy," IEEE Design and Test, 14(1):25-33, 1997.

[4] D. M. Blough, "On the Reconfiguration of Memory Arrays Containing Clustered Faults," *IEEE Int'l Fault-Tolerant Computing Symp. (FTCS) Dig. Papers*, pp.444-451, Jun. 1991.

[5] D. M. Blough and A. Pelc, "A clustered failure model for the memory array reconfiguration problem," *IEEE Trans. Computers*, May 1993, pp. 518-528.

[6] D. M. Blough, "Performance evaluation of a reconfiguration algorithm for memory arrays containing clustered faults," *IEEE Trans. Reliability*, June 1996, pp. 274-284.

[7] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, Aug. 1986, pp. 677-691.

[8] R. P. Cenker, D. G. Clemons, W.P. Huber, J. P. Petrizzi, F. J. Procyk, and G. M. Trout, "A Fault-Tolerant 64K Dynamic Random-Access Memory," *IEEE Trans. Electron Devices*, vol. ED-26, no. 6, pp. 853-860, June 1979.

[9] J. Chen and I. A. Kanj, "Constrained Minimum Vertex Cover in Bipartite Graphs: Complexity and Parameterized Algorithms," *J. Computer System Sciences*, 67:833-847, 2003.

[10] S. A. Cook, "The Complexity of Theorem-Proving Procedures," *Proceedings of the Third ACM Symposium on Theory of Computing*, pp. 151-158, 1971.

[11] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, pp. 394-397, 1962.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[13] S. Eaton, D. Wooten, W. Slemmer, J. Brady, "A 100ns 64K Dynamic RAM Using Redundancy Techniques," *IEEE Int'l Conf. Solid-State Circuits (ISSCC) Dig. Tech. Papers*, vol. XXIV, Feb. 1981, pp. 84 – 85.
[14] R. C. Evans, "Testing Repairable RAMs and Mostly Good Memories," *Proc. Int'l Test Conf. (ITC)*, 1981, pp. 49-55.
[15] H. Fernau and R. Niedermeier, "An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover," J. Algorithms, 38(2):374-410, 2001.
[16] B. F. Fitzgerald and E. P. Thoma, "Circuit Implementation of Fusible Redundant Addresses of RAMs for Productivity Enhancement," *IBM J. Res. Develop.*, vol. 24, pp. 291-298, 1980.
[17] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
[18] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, "Algorithms for the Satisfiability (SAT) Problem: A Survey," in DIMACS Series in Discrete Mathematics and Theoretical Computer Science: American Mathematical Society, 1997.
[19] R. L. Hadas and C. L. Liu, "Fast Search Algorithms for Reconfiguration Problems," *Proc. IEEE Int'l Wkshp. Defect Fault Tolerance VLSI Systems*, Nov. 1991, pp. 260-273.
[20] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, "Increased throughput for the testing and repair of RAMs with redundancy," *IEEE Trans. Computers*, Feb. 1991, pp. 154-166.
[21] N. Hasan and C. L. Liu, "Minimum fault coverage in reconfigurable arrays," *IEEE Int'l Fault-Tolerant Computing Symp. (FTCS) Dig. Papers*, June 1988, pp. 348-353.
[22] V. G. Hemmady and S. M. Reddy, "On the Repair of Redundant RAM's," *Proc. 26th Design Automat. Conf.*, 1989, pp. 710-713.
[23] K. Kokkonen, P. Sharp, R. Albers, J. Dishaw, F. Louie, R. Smith, "Redundancy Techniques for Fast Static RAMs," *IEEE Int'l Conf. Solid-State Circuits (ISSCC) Dig. Tech. Papers*, vol. XXIV, Feb. 1981, pp. 80 – 81.
[24] I. Koren and Z. Koren, "Defect Tolerance in the VLSI Circuits: Techniques and Yield Analysis," *Proc. IEEE*, 86(9):1819-1836, Sep. 1998.
[25] S. Y. Kuo, W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design & Test*, Feb 1987, pp. 24-31.
[26] M. E. Levitt. "Designing UltraSparc for testability," *IEEE Design and Test*, 14(1):10-17, 1997.
[27] F. Lombardi and W. Huang, "Approaches For The Repair of VLSI/WSI RRAMs by Row/column Deletion," *IEEE Int'l Fault-Tolerant Computing Symp. (FTCS) Dig. Papers*, 1988, pp. 342-347.
[28] H. Y. Lin, F. M. Yeh, I. Y. Chen, and S. Y. Kuo. "An Efficient Perfect Algorithm for Memory Repair Problems," *Proc. 19th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT)*, Oct. 2004, pp. 306-313.
[29] C. P. Low and H. W. Leong, "A new class of efficient algorithms for reconfiguration of memory arrays," *IEEE Trans. Computers*, May 1996, pp. 614-618.
[30] F. J. Meyer and D. K. Pradhan, "Modeling Defect Spatial Distribution," *IEEE Trans. Comput.*, vol. 39, pp. 538-546, Apr. 1989.
[31] F. J. Meyer and N. Park, "Predicting Defect-Tolerant Yield in the Embedded Core Context," *IEEE Transactions on Computers*, 52(11):1470-1479, Nov. 2003.
[32] M. Mishra and S. C. Goldstein, "Defect Tolerance at the End of the Roadmap," In *Proc. Int'l Test Conf. (ITC) 2003*, pp. 1201-1210, NC, USA, Sep. 2003.
[33] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, S. "Chaff: Engineering an Efficient SAT Solver." *Proc. 38th Design Automation Conf.*, session 33.1, New Orleans, LA, 2001
[34] B. T. Murphy, "Cost-Size Optima of Monolithic Integrated Circuits," *Proc. IEEE*, vol. 52, Dec. 1964, pp. 1537-1545.
[35] S. E. Schuster. "Multiple word/bit line redundancy for semiconductor memories," *IEEE J. Solid-State Circuits*, 13(5):698-703, 1978.
[36] W. Shi and W. K Fuchs, "Probabilistic analysis and algorithms for reconfiguration of memory arrays," *IEEE Trans. Computer-Aided Design*, Sep. 1992, pp. 1153-1160.
[37] J. Shoemaker, M. Haque, M. Huang, K. Truong, M. Karim, S. Chiu, G. Leong, K. Desai, R. Goe, S. Kulkarni, A. Rao, D. Hannoun, S. Rusu "A 0.13µm Triple-Vt 9MB Third Level On-Die Cache for the Itanium® 2 Processor," *IEEE Int'l Conf. Solid-State Circuits (ISSCC) Dig. Tech. Papers*, San Francisco, USA, Feb. 2004, pp. 496-497.
[38] R. T. Smith, J. D., Chlipala, J. F. M., Bindels, R. G. Nelson, F. H. Fischer, T. F. Mantz. "Laser Programmable Redundancy and Yield Improvement in a 64k DRAM," *IEEE J. Solid-State Circuits*, Vol. SC16, No. 5, pp. 506-514, 1981.
[39] C. H. Stapper, "On Yield, Fault Distributions, and Clustering of Particles," *IBM J. Research and Development*, 30(3):326-338, 1986.
[40] L. Youngs and S. Paramanandam. "Mapping and Repairing Embedded Memory Defects," *IEEE Design and Test*, 14(1):18-24, 1997.
[41] R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," *Proc. National Conf. Artificial Intelligence*, 1988, pp. 155-160.
[42] L. Zhang and S. Malik, "The Quest for Efficient Boolean Satisfiability Solvers", Proc. 14th Conf. Computer Aided Verification (CAV2002), Copenhagen, Denmark, Jul. 2002.