Random 3-SAT and BDDs: The Plot Thickens Further

Alfonso San Miguel Aguirre[1]* and Moshe Y. Vardi[2]**

[1] Dept. of Computer Science
Instituto Tecnologico Autonomo de Mexico
Rio Hondo 1, 01000 Mexico City, Mexico
[2] Department of Computer Science
Rice University
6100 S. Main St MS 132
Houston TX 77005-1892,USA

Abstract. This paper contains an experimental study of the impact of the construction strategy of reduced, ordered binary decision diagrams (ROBDDs) on the average-case computational complexity of random 3-SAT, using the CUDD package. We study the variation of median running times for a large collection of random 3-SAT problems as a function of the density as well as the order (number of variables) of the instances. We used ROBDD-based pure SAT-solving algorithms, which we obtained by an aggressive application of existential quantification, augmented by several heuristic optimizations. Our main finding is that our algorithms display an "*easy-hard-less-hard*" pattern that is quite similar to that observed earlier for search-based solvers. When we start with low-density instances and then increase the density, we go from a region of polynomial running time, to a region of exponential running time, where the exponent first increases and then decreases as a function of the density. The locations of both transitions, from polynomial to exponential and from increasing to decreasing exponent, are algorithm dependent. In particular, the running time peak is quite independent from the crossover density of 4.26 (where the probability of satisfiability declines precipitously); it occurs at density 3.8 for one algorithm and at density 2.3 for for another, demonstrating that the correlation between the crossover density and computational hardness is algorithm dependent.

1 Introduction

The last decade has seen an intense focus on the complexity of randomly generated combinatorial problems. This interest was stimulated by the discovery of a fascinating connection between the *density* of combinatorial problems and their computational complexity, see [11,31]. A problem that has received a lot of attention in this area is the *3-satisfiability problem* (3-SAT), which is a paradigmatic combinatorial problem,

* Part of this work was done while this author was on sabbatical at Rice University, funded in part by CONACyT grant 145502.
** Work partially supported by NSF grants IIS-9908435, IIS-9978135, CCR-9988322, and EIA-0086264, and by a grant from the Intel Corporation.

and also important for its own sake. An instance of 3-SAT consists of a conjunction of clauses, each one a disjunction of three literals. The goal is to find a truth assignment that satisfies all clauses. The density of a 3-SAT instance is the ratio of the number of clauses to the number of Boolean variables (we refer to the latter number as the *order* of the instance). Clearly, a low density suggests that the instance is under-constrained, and therefore is likely to be satisfiable, while a high density suggests that the instance is over-constrained and is unlikely to be satisfiable. Experimental research [15,31] has shown that for ratio below (roughly) 4.26, the probability of satisfiability goes to 1 as the order increases, while for ratio above 4.26 the probability goes to 0. At 4.26, the probability of satisfiability is 0.5. We call this density the *crossover density*. Formally establishing the crossover density is known to be quite difficult, and is the subject of continuing research, cf. [18,17,1].

The experiments in [15,31], which applied algorithms based on the so-called *Davis-Logemann-Loveland method* (abbr., DLL method) (a depth-first search with unit propagation [16]), also show that the density of a 3-SAT instance is intimately related to its computational complexity. Intuitively, it seems that under-constrained instances are easy to solve, as a satisfying assignment can be found fast, and over-constrained instances are also easy to solve, as all branches of the search terminate quickly. Indeed, the data displayed in [15,31] show how the running time increases with increasing density until the crossover density and then declines with increasing density, with a marked running-time peak essentially at the crossover density. What we see at the crossover density is in essence a *phase transition*, viz., a marked qualitative change in the structural properties of the problem. This pattern of behavior with a running-time peak at the crossover density is called the *easy-hard-easy* pattern and is the subject of extensive research, cf. [30].

In [13] it was pointed out that this picture is quite simplistic for various reasons. First, it is not clear where the boundaries between the "easy", "hard", and "easy" regions are. Second, the terms "easy" and "hard" do not carry any rigorous meaning. The computational complexity of a problem is typically studied on an infinite collection of instances, and is specified as a function of problem size or order. The easy-hard-easy pattern, however, is observed when the order is fixed while the density varies, but once the order is fixed, there are only finitely many possible instances. For that reason, theoretical analyses of the random 3-SAT problem focus on collections of fixed-density instances, rather than on collections of fixed-order instances.[1] Third, in the context of a concrete application, e.g., bounded model checking [4], it is typically the order that tends to grow while the density stays fixed, for example, as we search for longer and longer counterexamples in bounded model checking. Thus, the easy-hard-easy pattern tells us little about the complexity of 3-SAT in such settings. Until recently, however, there was little experimental work that studies how the running time of a SAT solver varies as a function of the order for fixed-density instances. Finally, the experiments reported in [31,15] are focused solely on DLL-based algorithms. While these are indeed the most popular algorithms for the satisfiability problem, one cannot jump to conclu-

[1] For example, it is known that in the high-density region, above density 5.2, the DLL method is provably exponential [12]; see also [3].

sions about the inherent and practical complexity of random 3-SAT based solely on experiments using these algorithms.

The goal of the research reported in [13] was to determine how the average-case complexity of random 3-SAT, understood as a function of the order for fixed density instances, depends on the density for a variety of SAT solvers. Is there a phase transition in which the complexity shifts from polynomial to exponential? Is such a transition dependent or independent of the solver? To explore these questions, Coarfa et el. [13] set out to obtain a good coverage of an initial quadrangle of the two-dimensional $d \times n$ quadrant, where d is the density and n is the order, exploring the range $0 \leq d \leq 15$ using three different SAT solvers, embodying different underlying algorithms: GRASP, which is based on the DLL method [27], the CPLEX MIP Solver, which is a commercial optimizer for integer-programming problems, and CUDD[2], which implements functions to manipulate Reduced Ordered Binary Decision Diagrams (ROBDDs), providing an efficient representation for Boolean functions [7].[3]

The findings in [13] show that for GRASP and CPLEX the easy-hard-easy pattern is better described as an *easy-hard-less-hard* pattern, where, as is the standard usage in computational complexity theory, "easy" means *polynomial time* and "hard" means *exponential time*. When we start with low-density instances and then increase the density, we go from a region of polynomial running time to a region of exponential running time, where the exponent first increases and then decreases as a function of the density. Thus, one observes at least *two* phase transitions as the density is increased: a transition at about density 3.8 from polynomial to exponential running time and a transition at about density 4.26 (the crossover density) from an increasing exponent to a decreasing exponent.[4] The region between 3.8 and 4.26 is also characterized by the prevalence of very hard instances, the so called "heavy-tail phenomenon", cf. [23,28,30].

A very different picture emerged in [13] for CUDD (described in Section 3). Here the algorithm is exponential (in both time and space) for densities between 0.5 and 15. There is, however, no running-time peak near the crossover density and no heavy-tail phenomenon was observed. A peak, however, is observed in the size of the final ROBDDs constructed by the algorithm at about density 2, indicating a phase transition at about this density. At a very low density (0.1), a polynomial (cubic) behavior is observed, which suggests that another phase transition is "lurking" between densities 0.1 and 0.5. Thus, unlike earlier predictions (cf. [26]), phase-transition phenomena related to random 3-SAT are not solver independent.

Our interest in studying ROBDD-based algorithms is motivated by the fact that ROBDDs have proven to be very effective in the context of hardware verification [9,25] and they are very different from standard search-based SAT solving methods. Uribe and Stickel [35] compared ROBDDs with the DLL method for SAT solving, concluding that the methods are incomparable, and that ROBDDs dominate the DLL method on many examples. Recent work by Groote and Zantema formally proved the incomparability of

[2] `http://bessie.colorado.edu/~fabio/CUDD`

[3] We use ROBDDs to represent Boolean functions. This is different than the usage in [10] of (zero-suppressed) ROBDDs to represent compactly sets of clauses.

[4] The polynomial to exponential phase transition, preceding the crossover point, was discovered independently by Cocco and Monasson [14].

ROBDDs and resolution (which is the proof system underlying the DLL method) [22]. The comparison in [13] between GRASP and CPLEX, on one hand, and CUDD, on the other hand, is, however, somewhat unenlightening. Unlike GRASP and CPLEX, CUDD does not search for a single satisfying truth assignment. Rather, it constructs a compact symbolic representation of the set of *all* satisfying truth assignments and then checks whether this set is nonempty. (Note, however, that for extremely sparse formula, the ROBDD-based algorithm is polynomial in spite of the fact that we have exponentially many satisfying truth assignments, due to the compactness of the representation.) In this paper we study the behavior of *pure* ROBDD-based SAT solvers. A pure SAT solver has to simply decide for a given propositional formula whether or not it is satisfiable; unlike search-based SAT solvers, it need not return a satisfying truth assignment.[5] The key step in constructing an ROBDD-based pure SAT solver is an aggressive application of existential quantification. (We describe the algorithm later on.) Once we have the basic algorithm, we can apply several heuristic optimizations, resulting in rather dramatic improvement in running time.

Our aim, however, is not to directly compare the performance of the different algorithms in order to see which one has the "best" performance, but rather to understand their behavior in the $d \times n$ quadrant in order to make qualitative observations on how the complexity of random 3-SAT is viewed from different algorithmic perspectives. It is important to note that the algorithms we used do not explicitly refer to the density of the input instances. Thus, a qualitative change in the behavior of the algorithm, as a result of changing the density, indicates a genuine structural change in the SAT instances from the perspective of the algorithm.

Our main finding is that the optimized ROBDD-based pure SAT-solving algorithms display *easy-hard-less-hard* pattern that is quite similar to that observed for GRASP and CPLEX in [13]. When we start with low-density instances and then increase the density, we go from a region of polynomial running time, to a region of exponential running time, where the exponent first increases and then decreases as a function of the density. Thus, one again observes at least *two* phase transitions as the density is increased: a transition from polynomial to exponential running time, accompanied by a heavy-tail phenomenon, and a transition from an increasing exponent to a decreasing exponent. Surprisingly, however, the location of both phase transitions is algorithm dependent. Unlike what has been observed so far in numerous papers, the transition from increasing to decreasing exponent, which corresponds to the running-time peak as one increase the density for a fixed order, does not occur at the crossover density of density 4.26. For one algorithm this transition occurs at density 3.8 and for the other at density 2.3.

Our findings provide further experimental evidence for the following two hypotheses. First, the running-time peak can change with the choice of solver not only in a minor way, as noted in [28], but in quite a major way, moving quite dramatically from the crossover density. This demonstrates that the correlation between the crossover density and computational hardness is algorithm-dependent, challenging the widely-held belief that the "hard problems" are always located at the crossover density [11]. Second, as

[5] Note, however, that by successively assigning truth values to the variables we can use a pure SAT solver to find a satisfying truth assignment, increasing the running time only by a linear multiplicative factor. This means that SAT enjoyes self-reducibility [2].

observed in [13], the density-order quadrant contains several phase transitions; in fact, the region between density 0 and density 4.26 seems to be rife with phase transitions, which are also solver dependent. In essence, each solver provides us with a different tool with which to study the complexity of random 3-SAT. This is analogous to astronomers observing the sky using telescopes that operate at different wave lengths. While our results are purely empirical, as the lack of success with formally proving a sharp threshold at the crossover density indicates (cf. [18,17,1]), providing rigorous proof for our qualitative observations may be a rather difficult task.

2 Experimental Setup

Our experimental setup is identical to that of [15,31,13]. We generate dn clauses, each by picking three distinct variables at random and choosing their polarity uniformly. For each studied point in the $d \times n$ quadrant we generate at least 100 random instances and apply a solver. Our experiments were run on Sun Ultra 1 machines, with a 167MHZ UltraSPARC processor and 256MB RAM. The CUDD package has been used through the GLU C–interface [34], a set of low-level utilities to access BDD packages. It is well known that the size of the ROBDD for a given function depends on the variable order chosen for that function. We have used automatic dynamic reordering during the tests with the default method for automatic reordering of CUDD (except in Section 6, where we used a certain fixed order).

As in [31], we chose to focus on median running time rather than mean running time. The difficulty of completing the runs on very hard instances makes it less practical to measure the mean. Furthermore, the median and the mean are typically quite close to each other, except for the regions that display heavy-tail phenomena, where the median and the mean diverge dramatically [20,30,13]. It would be interesting to analyze our data at percentiles other than the 50th percentile (the median) (cf. [30]), though a meaningful analysis for high percentiles would require many more sample points than we have in our experiments.

For the statistical analysis and plotting of data, we used MATLAB [6], which is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. The MATLAB functions we used for statistical analysis were:

- *polyfit*, for computing the best fit to a set of data using polynomial regression, and
- *corrcoef*, for computing r^2, the square of correlation (r^2 is the fraction of the variance of one variable that is explained by regression on the other variable).

For all the results reported in this paper, r^2 exceeded 0.98. This establishes high confidence in the validity of the fit of the curve to the data points.

3 Random 3-SAT and CUDD

In this section we review the results of [13] regarding Random 3-SAT and CUDD. CUDD [32] is a package that provides functions for the manipulation of Boolean func-

[6] `http://www.mathworks.com`

tions, based on the reduced, ordered, binary decision diagram (ROBDD) representation [7]. A binary decision diagram (BDD) is a rooted directed acyclic graph that has only two terminal nodes labeled 0 and 1. Every non-terminal node is labeled with a Boolean variable and has two outgoing edges labeled 0 and 1. An ordered binary decision diagram (OBDD) is a BDD with the constraint that the input variables are ordered and every path in the OBDD visits the variables in ascending order. An ROBDD is an OBDD where every node represents a distinct logic function. The *support set* of an ROBDD is the set of variables labeling its internal nodes.

CUDD constructs a compact representation of the set of satisfying truth assignments. The input formula φ is a conjunction $c_1 \wedge \ldots \wedge c_m$ of 3-clauses, where $m = dn$. Our algorithm constructs an ROBDD A_i for each clause c_i. (Note that A_i has to represent only the seven satisfying truth assignments of c_i.) An ROBDD for the set of satisfying truth assignment is then constructed incrementally; B_1 is A_1, while B_{i+1} is the result of $\textsc{apply}(B_i, A_i, \wedge)$, where $\textsc{apply}(A, B, \circ)$ is the result of applying a Boolean operator $\circ$ to two ROBDDs A and B. Finally, the resulting ROBDD B_m is compared against the predefined constant 0 (the empty ROBDD) in order to find if an instance is (un)satisfiable. We call this the BDD algorithm.

The goal of the experiments was to evaluate CUDD's performance on an initial quadrangle of the $d \times n$ quadrant. Densities 0.1, 0.5, and 1 to 15 were explored in [13]. In Figure 1 the median running time is shown on a logarithmic (base 2) scale. Note the absence of a peak; the running-time curve flattens roughly at density 2. The explanation for the lack of running-time peak is that the running time of ROBDD-based algorithms is determined mostly by the size of the manipulated ROBDDs. Our algorithm involves $m = dn$ conjunction operations between the possibly large ROBDD B_i and the small ROBDD A_i. Thus, the running time of our algorithm is determined by the largest intermediate ROBDD B_i constructed. As shown in [13], the peak in ROBDD size is attained after processing about $2n$ clauses, which explains the flattening of the running-time plot at density 2, and suggests that a phase transition in terms of ROBDD size occurs at about this density.

The median running time was analyzed as a function of the order for fixed-density instances. At densities 0.5 and above, the median running time of CUDD is exponential in the order, i.e., it behaves as $2^{\alpha n}$. In contrast, at density 0.1 the running time is cubic. This is explained by the fact that ROBDDs can represent very large sets quite compactly, which is why the method is quite effective for very low-densities instances, where the number of satisfying truth assignments is very large. Unlike what is observed for search-based algorithms, the BDD algorithms does not exhibit a heavy-tail phenomenon.

4 Existential Quantification of Variables

CUDD enables us to apply existential quantification to an ROBDD B:

$$(\exists x)B = \textsc{apply}(B|_{x \leftarrow 1}, B|_{x \leftarrow 0}, \vee),$$

where $B|_{x \leftarrow c}$ restricts B to truth assignments that assign the value c to the variable x. Note that quantifying x existentially eliminates it from the support set of B. We now see how we can take advantage of existential quantification.

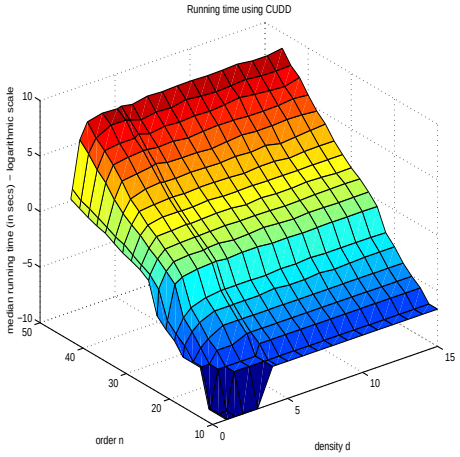

Fig. 1. BDD – 3-D Plot of median running time

The satisfiability problem is to determine whether a given formula $c_1 \wedge \ldots \wedge c_m$ is satisfiable. In other words, the problem is to determine whether the existential formula $(\exists x_1) \ldots (\exists x_n)(c_1 \wedge \ldots \wedge c_m)$ is true. Since checking whether the final ROBDD B_m is equal to 0 can be done by CUDD in constant time, it makes little sense, however, to apply existential quantification to B_m. Suppose, however, that a variable x_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the existential formula can be rewritten as

$$(\exists x_1) \ldots (\exists x_{j-1})(\exists x_{j+1}) \ldots (\exists x_n)((\exists x_j)(c_1 \wedge \ldots \wedge c_i) \wedge (c_{i+1} \wedge \ldots \wedge c_m)).$$

This means that after constructing the ROBDD B_i, we can existentially quantify x_j before conjuncting B_i with $A_{i+1}, \ldots, A_m$.

This suggests the following modification of our algorithm: after constructing the ROBDD B_i, quantify existentially variables that do not occur in the clauses $c_{i+1}, \ldots, c_m$. In this case we say that the variable x has been *quantified out*. The computational advantage of quantifying out stems from the fact that reducing the size of the support set of an ROBDD typically (though not necessarily) results in a reduction of its size; that is, the size of $(\exists x)B$ is typically smaller than that of B. This method is call the *early quantification method*, and proposed first in the context of symbolic model checking [8]. Early quantification was applied to SAT solving in [21] (under the name of *hiding functions*) and tried on random 3-SAT instances, but without a systematic study of the complexity of random 3-SAT. Our implementation adds the slight improvement of stopping the construction as soon as we construct a B_i that is equal to 0; this is called *early termination*. We will call this algorithm, i.e., early quantification with early termination, BDD(Q).

Figure 2 (left) shows the median running time of BDD(Q) on a logarithmic (base 2) scale. The median running time has decreased with respect to the BDD algorithm. At order 46, for densities less than or equal to two we got an order of magnitude improve-

ment (10X) in running time. For greater densities, the improvement is only between 5% to 15%.

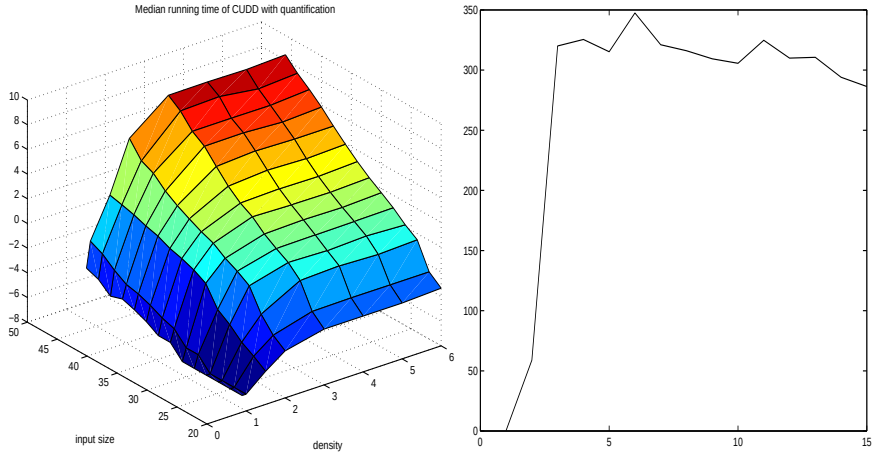

Fig. 2. BDD(Q) – (left) 3-D Plot of median running time, and (right) median running time as a function of the density for order 46

The overall shape of the running-time surface is somewhat similar to that observed in Section 3; the running time increases with density and then seems to flatten. The flattening, however, occurs at about density 4, rather than density 2. Note that once we have processed $i = 4.3n$ clauses, the conjunction $c_1 \wedge \ldots \wedge c_i$ is with very high probability unsatisfiable, which means that B_i is with high probability equal to 0. Thus, BDD(Q) typically terminates by the time $5n$ clauses have been processed, which explains the flattening of the run-time surface for densities over 5. In Figure 2 (right) median running times are shown as a function of the density, for order 46.

An interesting difference between the BDD and BDD(Q) algorithms is that the transition from polynomial to exponential has shifted to the right. Our results indicate a quadratic-time behavior at density 0.5—see Figure 3 (left)—while at densities 1 and above the median running time is exponential in the order, see Figure 3 (right) for median running times for instances of density 1, on a logarithmic (base 2) scale. It should also be noted that BDD(Q) also does not exhibit a heavy-tail phenomenon.

5 Reordering the clauses

BDD(Q) processes the clauses of the input formula in a linear fashion. Since the main point of early quantification is to quantify variables out as early as possible, reordering the clauses may enable us to do more aggressive early quantification. That is, instead of processing the clauses in the order $c_1, \ldots, c_m$, we can apply a permutation π and process the clauses in the order $c_{\pi(1)}, \ldots, c_{\pi(m)}$. The permutation π should be chosen

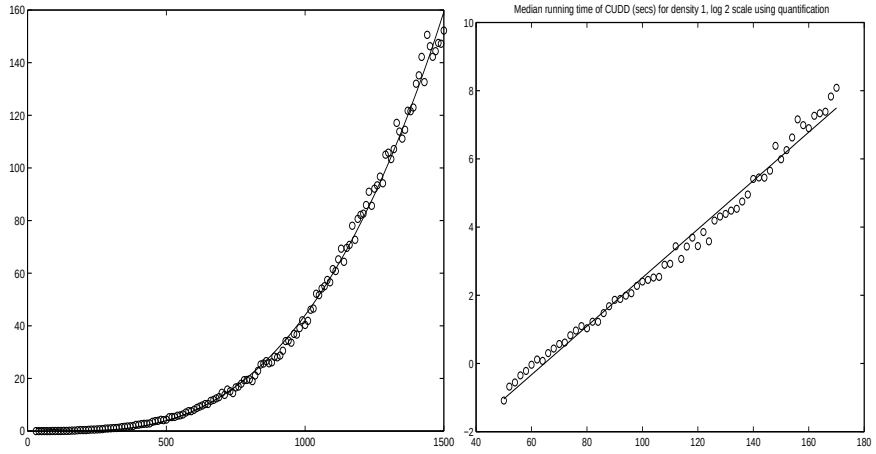

Fig. 3. BDD(Q) – (left) median running time for density 0.5 as a function of the order of the instances; a quadratic function fits these points better than an exponential function, and (right) median running time for density 1 (log scale)

so as to minimize the number of variables in the support sets of the intermediates ROBDDs. This observation was first made in the context of symbolic model checking, cf. [8,19,24,5]. Unfortunately, finding an optimal permutation π is by itself a difficult optimization problem, motivating a *greedy* approach: searching at each step for the clause that would result in the maximum number of variables to be quantified out.

Our proposed algorithm searches for a clause with the maximum number of variables with only one occurrence in the remaining clauses. If more than one clause is a possible candidate then a second criterion is applied; from the candidate clauses, the algorithm looks for one that shares least variables with the remaining clauses. (This is as opposed to [19], where the algorithm looks for a candidate that shares most variables with the remaining clauses. We have tried this latter heuristic, and the results are not as good as using our heuristic.) The rationale of our heuristic is trying to quantify out variables as soon as possible. We will call this algorithm BDD(Q,R).

Figure 4 shows median running time using our algorithm. The median running time has decreased quite dramatically with respect to the BDD algorithm. The improvements are most dramatic at low and high densities. For example, for order 46, for density 1 we get a 30X improvement (i.e., the running time of BDD(Q,R) is about 0.03 times that of that of BDD) and for densities 9 and above we get a 100X improvement, while for density 4 we get a 6X improvement. Most interestingly, the shape of the running-time surface is now similar to the shape of the running-time surface for search-based algorithms (GRASP and CPLEX) in [13].

Unlike what we saw in [13], where the running-time peak roughly occurs at the crossover density, running-time peak for BDD(Q,R) seems to occur at about density 3.8. In Figure 5, we plot the median running time in the “hard” zone, for 40 and 46 variables, respectively, with 1000 experiments per point. It is interesting to note that

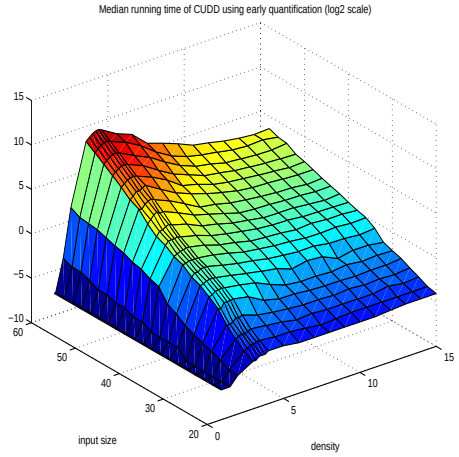

Fig. 4. BDD(Q,R) – 3-D Plot of median running time

density 3.8 is where the transition from polynomial to exponential running time for search-based solvers was observed in [13].

Another interesting development is a further shift to the right of the transition from polynomial to exponential median running time. At density 1 our data indicate a quadratic running time. See Figure 6 (left) for median running times for instances of density 1, with 200 instances per point. For densities 1.5 and above the running time is exponential. See Figure 6 (right) for median running times for instances of density 1.5 on a logarithmic (base 2) scale. Thus, the transition occurs between densities 1 and 1.5. Recall that, in contrast, the transition for the BDD algorithm occurs between densities 0.1 and 0.5, while for BDD(Q) it occurs between densities 0.5 and 1. Thus, the improvement in the algorithm is not merely quantitative, it is also qualitative, as it expands the region in which the algorithm is feasible.

As with GRASP and CPLEX [13], the transition from polynomial to exponential behavior of BDD(Q,R) is accompanied by a "heavy-tail phenomenon", which is a prevalence of *outliers*, i.e., instances on which the actual running time is at least an order of magnitude (10X) larger than the median running time, as well as a divergence of the mean and the median. See Figure 7, where we plot the mean to median ratio and the proportion of outliers as a function of the density. Thus, in spite of the incomparability of search-based solvers and ROBDD-based solvers [35,22], we see a significant similarity between the qualitative results in [13] and here. For both GRASP, CPLEX, and BDD(Q,R). For low densities, the algorithms are polynomial. As the density increases, we see a transition from polynomial to exponential behavior, accompanied by a heavy-tail phenomenon. As the density increases further, the exponent first increases and then decreases. BDD(Q,R) differs, however, in the location of the running-time peak, which is roughly at the crossover density for GRASP and CPLEX, and markedly to its left for BDD(Q,R).

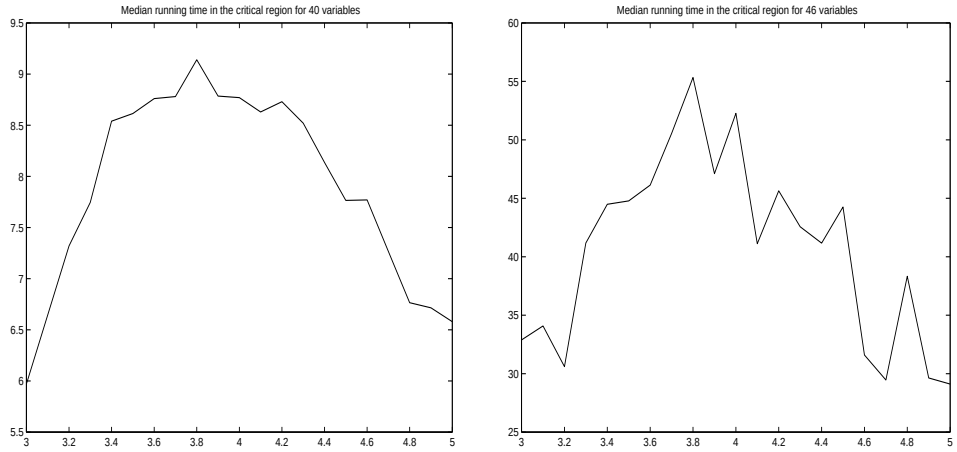

Fig. 5. BDD(Q,R) – median running time in the hard region, for order 40 (left) and 46 (right)

A further improvement of early quantification and reordering was proposed in the context of symbolic model checking in [29]. In this approach, the clauses are not processed one at a time, but several clauses are first *clustered* together without being processed. Once the *size* (number of clauses) of a cluster C attains a pre-established bound, then we first apply conjunction to all the ROBDDs of the clauses in the C to obtain an ROBDD B_C and we then combine B_C with the ROBDD B_i (which corresponds to all the clauses processed earlier) and apply early quantification. Obviously, setting higher limits in the cluster size leads to fewer clusters, but a larger cluster C results in a larger OBDD B_C. To quote [29]: "as the size of the clusters is raised, the number of iterations is reduced, while the BDD sizes of the formula increase. In the beginning, the reduction in the number of iterations offsets the increase in BDD sizes. Hence initially, runtime is reduced as the cluster size increases. But later, the BDD computation time starts to dominate and the running time increases".

We implemented clustering on top of BDD(Q,R) (that is, we order the clauses as in BDD(Q,R) before clustering). We will call this algorithm BDD(Q,R,C). Experimentation showed that the best results are obtained when cluster size is set to the "magic number" 20. We found out that BDD(Q,R,C) performs badly at low densities, but yields an improvement of 10%-30% for densities above 3. The qualitative behavior of BDD(Q,R,C) is, however, quite similar to that of BDD(Q,R): we observe a transition from polynomial to exponential, accompanied by a heavy-tail phenomenon, between densities 1.0 and 1.5, and the exponent then rises and declines, peaking at about density 3.8.

6 Variable Ordering

The previous ROBDD-based methods focused on the processing of the input clauses, while at the same time letting CUDD handle the critical issue of variable ordering (in-

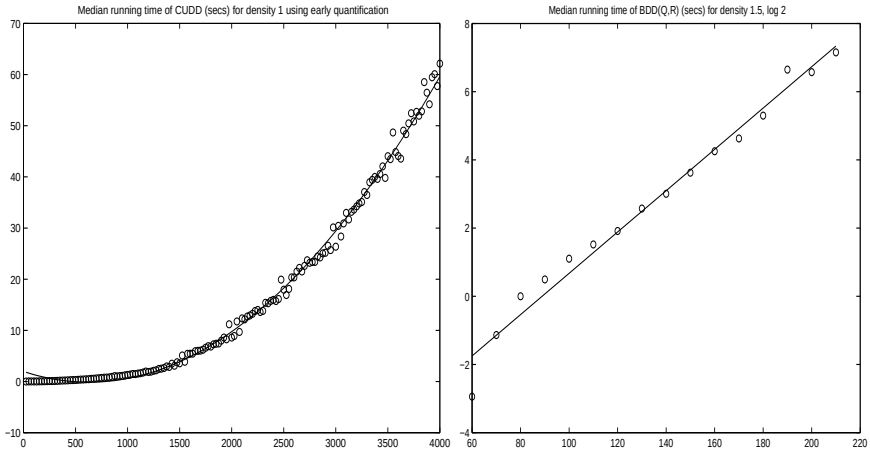

Fig. 6. BDD(Q,R) – (left) median running time for density 1 as a function of the order of the instances; a quadratic function fits these points better than an exponential function, and (right) median running time for density 1.5 (log scale)

cluding dynamic reordering). Inspired by work of Bouquet [6], we studied an ROBDD-based algorithm using variable ordering based on a graph representation of the input formula. As we shall see, by using knowledge about the structure of the input formula, we can obtain dramatic improvement in running time.

The graph associated with a CNF formula $\varphi = \bigwedge_i c_i$ is $G_\varphi = (V, E)$, where V is the set of variables in φ and an edge $\{x_i, x_j\}$ is in E if there exists a clause c_k such that x_i and x_j occur in c_k. To extract variable order from G_φ. Bouquet uses the "maximum cardinality search" (MCS) of [33]. Let n be the number of vertices of G_φ. MCS numbers the vertices from 1 to n in the following way: As the next vertex to number, select the vertex adjacent to the largest number of previously numbered vertices, breaking ties arbitrarily. It is this variable ordering that we now provide to CUDD (turning off dynamic reordering).

Bouquet then uses the variable order to cluster the clauses. Let the *rank* of a clause $c = \{l_1, l_2, l_3\}$ be $rank(c) = \max(order(x_1), order(x_2), order(x_3))$, where x_i is the variable of the literal l_i. The clusters are the equivalent classes of the relation $\sim$ defined by: $c \sim c'$ iff $rank(c) = rank(c')$. For each cluster $C_j = \{c_{j_1}, \ldots, c_{j_k}\}$, we then construct an ROBDD A_{C_j} by applying conjunction to the ROBDDs $A_{j_1}, \ldots, A_{j_k}$. The rank of a cluster is the rank of its clauses (by definition, all the clauses in a cluster have the same rank).

In [6], the final ROBDD is constructed by applying conjunction to the ROBDDs A_{C_j}'s of the clusters. We have combined Bouquet's method with the method of early quantification. We process the clusters in ascending rank order and quantify variables out as early as possible. We observed that early quantification plays an important role in the low densities, where satisfying truth assignments abound. We denote the combined method by BDD(B,Q,C). For densities 2 or above, BDD(B,Q,C) is significantly faster than BDD(Q,R,C). At order 46 we saw improvement between 5X and 10X (for

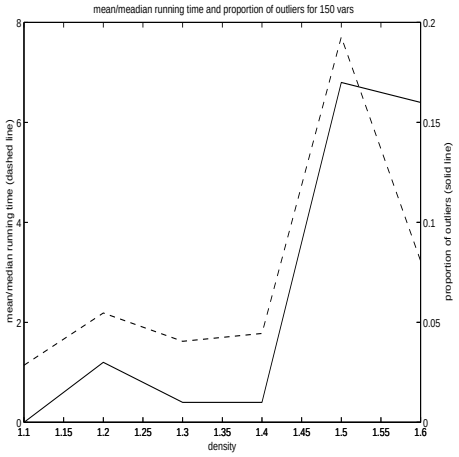

Fig. 7. BDD(Q,R) – ratio of mean to median running time and proportion of outliers

lower densities BDD(B,Q,C) is about 30% slower). More interestingly, the shape of the running-time surface is quite different for BDD(B,Q,C). Figure 8 shows the median running time of BDD(B,Q,C) on a logarithmic (base 2) scale. As we can see, the interesting region has moved to the left. The running-time peak now seems to occur at about density 2.3. Figure 8 shows median running times for order 60.

We again see a transition from polynomial to exponential behavior before the running-time peak, between densities 0.2 and 1 (to the left of the analogous transition for BDD(Q,R)). For very low densities (0.2 or below) our data indicate a cubic running time. See Figure 9 (left) for median running times for instances of density 0.2. For densities 1 and above the median running time of BDD(B,Q,C) is exponential (but see remark below). See Figure 9 (right) for median running time for instances of density 1 on a logarithmic (base 2) scale. The transition from polynomial to exponential behavior is again accompanied by a heavy-tail phenomenon. The pattern of that phenomenon is significantly more complex than that observed for BDD(Q,R) and we have not yet been able to characterize it.

Remark: Note that the running time decreased quite dramatically with increasing densities above 2.3. Is it possible that at high enough density we see again polynomial behavior? Our data is inconclusive. For example, at density 20 our data fit cubic and exponential curves almost equally well. This issue requires further investigation.

7 Discussion

In this paper we studied the complexity of random 3-SAT experimentally using ROBBD-based pure SAT solvers. Our main finding is that these solvers display *easy-hard-less-hard* pattern that is quite similar to that observed for search-based solvers in [13]. When we start with low-density instances and then increase the density, we go from a region

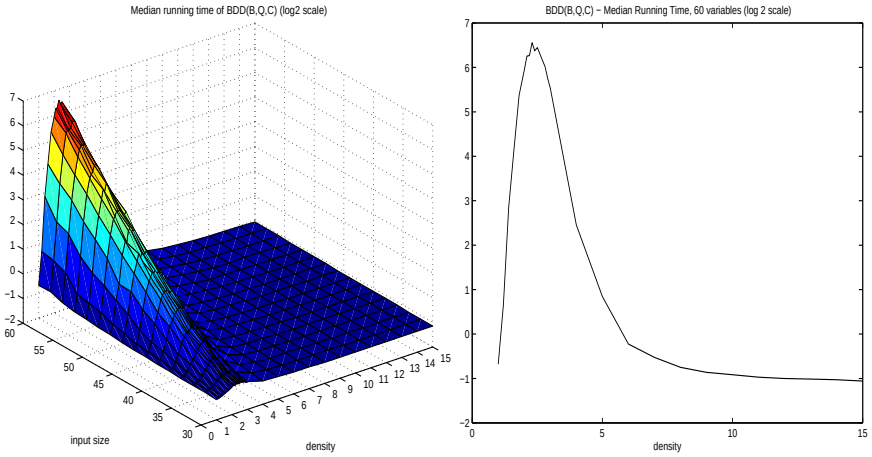

Fig. 8. BDD(B,Q,C) – (left) 3-D Plot of median running time, and (right) median running times as a function of the density for order 60

of polynomial running time, to a region of exponential running time, where the exponent first increases and then decreases as a function of the density. The location of both transitions, from polynomial to exponential and from increasing to decreasing exponent, are algorithm dependent. In particular, the running time peak is quite independent than the crossover density, challenging the widely-held belief that the "hard problems" are always located near the crossover density [11].

These findings should be contrasted with those of [13], which revealed a marked difference between solvers like GRASP and CPLEX, which are search based and display interesting similarities in the shapes of the median running time surface despite their different underlying algorithmic techniques, and ROBDD-based solvers, like CUDD, which are based on compactly representing all satisfying truth assignments. By developing here ROBDD-based *pure* SAT solvers, we showed that certain qualitative features of the complexity of random 3-SAT do seem to be algorithm independent. Explaining these common features is a challenging research problem.

References

1. D. Achlioptas. Setting two variables at a time yields a new lower bound for random 3-SAT. In *Proc. 32th ACM Symp. on Theory of Computing*, pages 28–37, 2000.
2. J. Balcazar. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.
3. P. Beame, R. M. Karp, T. Pitassi, and M. E. Saks. On the complexity of unsatisfiability proofs for random k-CNF formulas. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 561–571, 1998.
4. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. 36th Conf. on Design Automation*, pages 317–320, 1999.

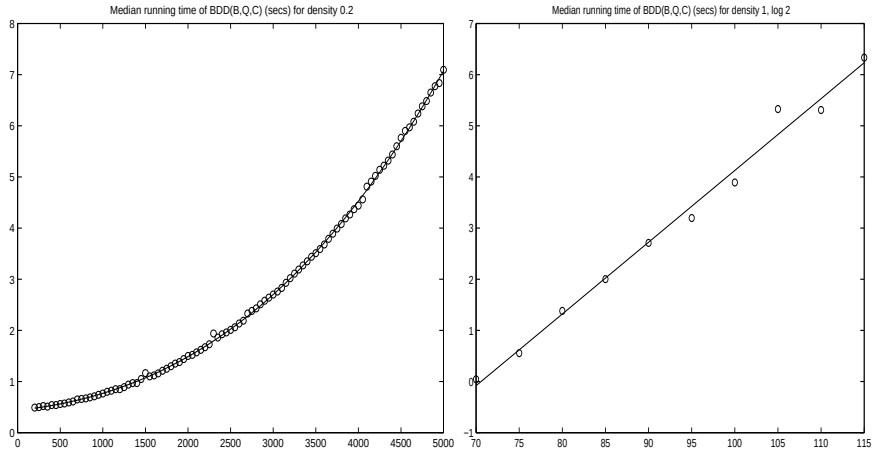

Fig. 9. BDD(B,Q,C) – (left) median running time for density 0.2 as a function of the order of the instances; a quadratic function fits these points better than an exponential function, and (right) median running time for density 1 (log scale)

5. M. Block, C. Gröpl, H. Preuß, H. L. Proömel, and A. Srivastav. Efficient ordering of state variables and transition relation partitions in symbolic model checking. Technical report, Institute of Informatics, Humboldt University of Berlin, 1997.
6. F. Bouquet. *Gestion de la dynamicité et énumération d'implicants premiers: une approche fondée sur les Diagrammes de Décision Binaire*. PhD thesis, Université de Provence, France, 1999.
7. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, 35(8):677–691, 1986.
8. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proc. IFIP TC10/WG 10.5 Int'l Conf. on Very Large Scale Integration, Edinburgh, Scotland (VLSI'91)*, pages 49–58, 1991.
9. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
10. P. Chatalic and L. Simon. The old Davis-Putnam procedure meets ZBDDs. In D. McAllester, editor, *17th Int'l Conf. on Automated Deduction (CADE'17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 449–454, 2000.
11. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. 12th Int'l Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 331–337, 1991.
12. V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. of the ACM*, 35(4):759–768, 1988.
13. C. Coarfa, D.D. Demopolous, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. In R. Dechter, editor, *Proc. Principles and Practice of Constraint Programming (CP'2000)*, Lecture Notes in Computer Science 1894, pages 143–159, 2000.
14. S. Cocco and R. Monasson. Trajectories in phase diagrams, growth processes and computational complexity: how search algorithms solve the 3-Satisfiability problem. *Phys. Rev. Lett.*, 86:1654–1657, 2001.
15. J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1-2):31–57, 1996.

16. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.
17. O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 126–127, 2000.
18. E. Friedgut. Necessary and sufficient conditions for sharp threshold of graph properties and the k-SAT problem. *J. Amer. Math. Soc.*, 12:1017–1054, 1999.
19. D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Proc. 6th Int'l Conf. on Computer Aided Verification (CAV '94)*, pages 299–310, 1994.
20. I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1-2):335–345, 1994.
21. J. F. Groote. Hiding propositional constants in BDDs. *Formal Methods in System Design*, 8:91–96, 1996.
22. J.F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. Technical report, Department of Computer Science, Utrecht University, 2000. Technical Report UU-CS-2000-14.
23. T. Hogg and C. P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69(1-2):359–377, 1994.
24. R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *Proc. 1996 Int'l Conf. on Computer Design*, pages 12–19, 1996.
25. S. Jha, Y. Lu, M. Minea, and E.M. Clarke. Equivalence checking using abstract BDDs. In *Proc. Int'l Conf. on Computer Design (ICCD'97)*, pages 332–337, 1997.
26. T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. In *Working Notes of AAAI 1993 Spring Symposium: AI and NP-Hard Problems*, pages 112–118, 1993.
27. J. P. Marques Silva and K. A. Sakallah. GRASP–A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, 48(5):506–521, 1999.
28. D. G. Mitchell and H. J. Levesque. Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, 81(1-2):111–125, 1996.
29. R. K. Ranjan, A. A. Aziz, R. K. Brayton, B. Plessier, and C. Pixley. Efficient formal design verification: Data structure + algorithms. Technical report, University of California at Berkeley, 1994. Tech. Rep. UCB/ERL M94/100.
30. B. Selman and S. Kirkpatrick. Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence*, 81(1-2):273–295, 1996.
31. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.
32. F. Somenzi. CUDD: CU Decision Diagram package. release 2.3.0., 1998. Dept. of Electrical and Computer Engineering. University of Colorado at Boulder.
33. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to tests chordiality of graphs, tests acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. on Computing*, 13(3):566–579, 1984.
34. The VIS Group. VIS: A system for verification and synthesis. In *Proc. 8th Int'l Conf. on Computer Aided Verification (CAV '96)*, pages 428–432, 1996. LNCS 1102. Ed. by R. Alur and T. Henziger.
35. T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *First Int'l Conf. on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 34–49, Munich, September 1994. Springer-Verlag.