Experiments on random 2-SoftSAT

V.E.P. Heinink, M.J. Seckington, F.S.D. van der Werf
{*v.e.p.heinink, m.j.seckington, f.s.d.vanderwerf*}*@student.tudelft.nl*

August 7, 2006

Abstract

In this paper we present results of some performance experiments on the 2-SoftSAT problem. 2-SoftSAT is a mixture of 2-SAT and 2-MaxSAT. An interesting aspect of this domain is that the complexity of the first is in P while the second is in NP. We tested the performance of two state-of-the-art solvers, SoftSAT-D (Argelich et al.) and PM-SAT(Fu et al.), at various densities. We modified the local search module of the SoftSAT-D solver such that it is more suitable on the experimented domain. Our results show that SoftSAT-D performs better than PM-SAT on the various interesting transition densities. We observe that, for both methods, the required time to solve 2-SoftSAT instances heavily increases when the number of solutions for the 2-SAT part of the formula drops.

1 Introduction

The Boolean Satisfiability problem (SAT) is a NP-complete decision problem. An instance of SAT is a propositional logic formula S in conjunctive normal form (CNF). Thus, S is a conjunction of clauses. In turn, each clause is a disjunction of literals, which are instances of possibly negated variables. An assignment for a set of variables maps truth values True/False to each variable in the set. The SAT problem consists of finding a solution of S or to show that no such solution exists.

Many real-life problems can be represented as a SAT problem. Nevertheless, this approach displays some limitations, because solutions are only given to those problems that are completely satisfiable. In some problems, however, it may be tolerable to accept a solution that violates some constraints. The SoftSAT problem is a generalization of SAT. It distinguishes between hard and soft clauses; in other words, clauses that must be satisfied and clauses that may be satisfied (and therefore, could be violated). The objective now becomes to find a solution that satisfies all hard clauses and the maximum number of soft clauses. The 2-SoftSAT problem is in particular interesting because it is a combination of the classical 2-SAT problem, which is in P, and 2-MaxSAT, which is in NP.

This paper describes some experiments on various aspects of the SoftSAT problem within the random 2-SAT domain. First, we investigate instances with only hard clauses to find out, per amount of variables (50, 100, 150, 200 and 250), what number of hard clauses caused 5%, 10% and 50% of the instances to be satisfiable. These numbers of hard clauses are the transition-densities of the problem, to which we refer to as *transitions*. With these transitions we continue to experiment in the SoftSAT domain by testing what happens when soft clauses are added. For these experiments in the SoftSAT domain we compare the performance of the SoftSAT-D [ARCH05] and the PM-SAT [MALI06] solver.

The paper is structured as follows. Section 2 describes the process of determining the interesting transitions for the 2-SoftSAT problem. In section 3 we present the two solvers SoftSAT-D and PM-SAT. Then in section 4 the results of the experiments are reported. Finally, some conclusions are drawn in section 5.

2 Transitions

Theoretically the threshold of the 2-SAT problem is 1. This only holds if the number of variables goes to infinity [BOLL01]. In this section we experimentally determine the transitions from 50 up to 250 variables. For every combination of variables and clauses we run 1,000 randomly generated instances. See figure 1 and table 1 for the results. From these results the transitions can be easily deduced.

2.1 Few #SAT solutions

#SAT is a variant on the SAT problem: Given a CNF-formula S, determine how many assignments to all variables exist that satisfy S. This problem is known to be NP-hard even for 2-#SAT. As can be seen in figure 2, as the density increases, the number of solutions decreases very rapidly. Please note that figure 2 is on a logarithmic scale.

The relevance of #SAT to our research comes from the fact that as the number of solutions to the hard clauses decreases, finding a solution to the SoftSAT problem is expected to be harder.

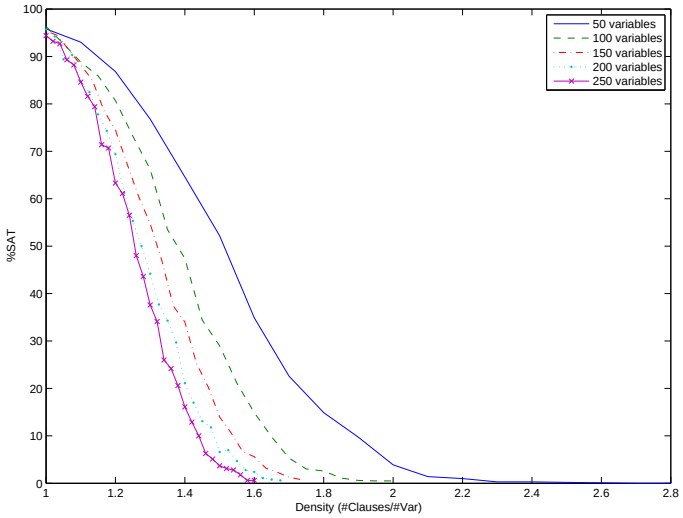

Figure 1: *Determining the SAT-UNSAT transitions:* Results are determined by solving (using zChaff [MOSK01]) 1,000 randomly generated 2-SAT instances for every #variables and #clauses combination and calculating what percentage is satisfiable. #Clauses are incremented by 5 for every next experiment. The transitions are then determined by linear interpolation.

Table 1: *Experimentally determined densities:* Densities and number of hard clauses for 50-50, 10-90 and 5-95 transitions. Number of hard clauses are given in parenthesis. Results are determined by solving (using zChaff [MOSK01]) 1,000 randomly generated 2-SAT instances for every #variables and #clauses combination mentioned. #Clauses are incremented by 5 for every next experiment. The transitions are then determined by linear interpolation.

	Transition		
#Variables	50-50	10-90	5-95
50	1.52 (76)	1.90 (95)	1.98 (99)
100	1.38 (138)	1.65 (165)	1.71 (171)
150	1.32 (198)	1.54 (231)	1.61 (241)
200	1.28 (255)	1.49 (297)	1.55 (309)
250	1.26 (314)	1.44 (360)	1.48 (370)

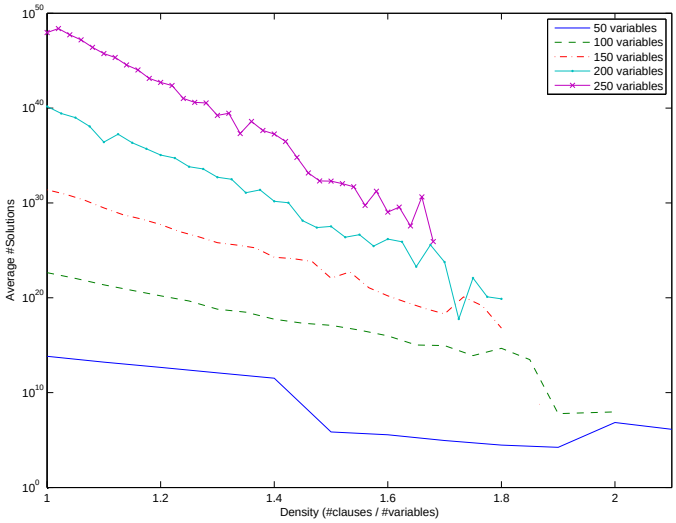

Figure 2: *Average number of solutions over 2-SAT instances:* Results are determined by #SAT-solving (using sharpSAT [THUR06]), 1,000 randomly generated 2-SAT instances for every #variables and #clauses combination depicted. The graphs stop at the point where there are no more satisfiable instances detected. #Clauses are incremented by 5 for every next experiment.

3 Solvers

3.1 SoftSAT solver

SoftSAT is a solver developed by Josep Archelich and Felip Manya. It comes in two variants; SoftSAT-S and SoftSAT-D, both explained in [ARCH05]. SoftSAT-S is a stripped version of SoftSAT-D which has been implemented with more efficient data-structures. We choose not to use SoftSAT-S as it turned out not to perform as well as SoftSAT-D.

3.1.1 Drawback of original local search module

The SoftSAT-D algorithm is a branch and bound algorithm. One of its parts consists of finding a feasible solution, in order to establish an upper bound on the number of unsatisfied soft constraints. As mentioned before; a solution is feasible if and only if all the hard constraints are satisfied and a good solution is feasible and has less unsatisfied soft constraints.

The current approach for finding an initial upper bound on the number of unsatisfied soft constraints is to generate 10 random variable-assignments. For

each assignment A the following process is repeated $100 \times \#variables$:

1. Randomly (with uniform distribution) choose variable V_i.
2. Change the assignment of V_i in A to the opposite value, resulting in A_{new}.
3. If A_{new} is feasible:
   (a) If A is not feasible: $A := A_{new}$
   (b) If A_{new} satisfies more soft constraints than A: $A := A_{new}$

After this is performed for every initial assignment, the feasible assignment satisfying the most soft constraints is used as upper bound on the number of unsatisfied soft constraints.

3.1.2 Change to local search module

Since (hard) 2-SAT is solvable in polynomial time, finding a feasible solution for the 2-SoftSAT problem can be done efficiently: The hard clauses of the 2-SoftSAT problem can be passed to a solver. If there is no solution, then there is no solution for the 2-SoftSAT problem - because 2-SoftSAT requires the hard clauses to be satisfied. Else an upper bound can be established performing a similar local search procedure as above, but starting with a feasible solution to the hard part. We call this adapted solver SoftSAT-Z.

3.1.3 Results local search

This section presents the results on the improvement obtained by the above adaptation. To test the improvement, the original solver and the modified solver were ran on a selection of 20 instances. These are instances for which SoftSAT-D needed more time to compute the optimal solution compared to the average for similar instances. The solve time of these instances ranges from 4 minutes to over 1.5 hours. Table 2 shows the results. For a couple of instances we see a huge difference between the upper bound found by SoftSAT-D and the upper bound found by SoftSAT-Z. These improved upper bounds resulted in a clear speed up.

Table 2: Local search results: The column "local search" is the time to calculate the initial upper bound, total time is time needed to solve the instance. UB stands for Upper Bound. Time units are seconds.

	SoftSAT-D			SoftSAT-Z		
	UB	local search	total time	UB	local search	total time
1	201	0.07	333.06	**30**	0.04	164.59
2	**19**	0.07	274.28	**19**	0.04	275.61
3	**19**	0.07	247.28	**19**	0.04	248.17
4	**6**	0.12	242.02	**6**	0.05	243.07
5	**10**	0.12	250.34	**10**	0.05	251.32
6	**9**	0.12	250.21	**9**	0.05	251.08
7	**11**	0.12	804.46	**11**	0.05	812.65
8	151	0.12	1,081.63	**10**	0.05	0.13
9	**8**	0.12	353.67	10	0.05	4,003.42
10	**11**	0.12	605.69	**11**	0.05	611.35
11	151	0.12	517.49	**8**	0.05	0.93
12	**6**	0.13	448.82	**6**	0.05	450.86
13	**10**	0.12	1,950.93	11	0.05	6,802.24
14	19	0.12	2,985.24	**18**	0.05	2,762.79
15	16	0.12	572.62	**15**	0.06	529.00
16	**28**	0.12	329.80	**28**	0.06	331.63
17	208	0.12	555.42	**35**	0.06	25.68
18	**16**	0.13	6,833.95	**16**	0.06	6,867.27
19	178	0.12	456.33	**17**	0.06	14.47
20	**16**	0.12	562.47	**16**	0.09	566.74

3.2 PM-SAT

In 2006 Fu and Malik [MALI06] proposed two alternative approaches to solve the SoftSAT problem. Both methods use the state-of-the-art SAT solver zChaff to assist with certain parts of the algorithm. The first is a diagnosis based approach that uses zChaff to find the UNSAT core of a problem. An UNSAT core indicates the subset of clauses that can be considered as the cause of the unsatisfiability. Once a core is found, it can be eliminated by adding distinct relaxation variables to each relaxable clause in the UNSAT core. The diagnosis based algorithm iteratively finds such UNSAT cores and eliminates them, until the problem is satisfiable. The second method is an encoding based approach that uses an auxiliary counter to constrain the number of simultaneously relaxed clauses. The algorithm then tries to find the minimum number of clauses that can be relaxed by using binary or linear search. After evaluating the results of their algorithms, we decided to use the diagnosis based algorithm because of it's better performance on random 2-SoftSAT instances.

An outline of the diagnosis based algorithm is given in algorithm 1. In each iteration the algorithm checks whether the SAT problem is satisfiable; if not,

it continues with finding an UNSAT core. It then proceeds to add a distinct relaxation variable to each relaxable clause in the UNSAT core. By setting this relaxation variable to true, we satisfy (and thus relax) the associated clauses. However, to guarantee the one-hot constraint that exactly one of the relaxation variables is relaxed, we must also add a couple of new clauses with the just introduced relaxation variables to the problem. For example, if the variables a, b and c were just added to three relaxable clauses in the UNSAT core, the one-hot constraint clauses would be $(\neg a \vee \neg b) \wedge (\neg b \vee \neg c) \wedge (\neg a \vee \neg c) \wedge (a \vee b \vee c)$. During every iteration one relaxation is set to true and thus one relaxable clause is relaxed. This algorithm stops after the exact minimum number of clauses have been relaxed.

Algorithm 1 Iterative UNSAT Core Elimination, Source: [MALI06]

```
1:  while SAT solver returns UNSATISFIABLE do
2:    Let UC be the UNSAT core provided by the SAT solver
3:    S := ∅
4:    for all Clause c ∈ UC do
5:      if c is relaxable then
6:        Allocate a new relaxation variable v
7:        L(c) := L(c) ∪ v
8:        S := S ∪ v
9:      end if
10:   end for
11:   if S = ∅ then
12:     return CNF UNSATISFIABLE
13:   else
14:     Add clauses enforcing the One-Hot constraint for S to the SAT solver
15:   end if
16: end while
17: R := {v|v ∈ S, v = 1}; k := |R|
18: return Satisfying Assignment, k, R.
```

4 Results

The following parameters are relavant for our research on random 2-SoftSAT:

- Number of hard clauses
- Number of soft clauses
- Number of variables

We use two sorts of densities: The *hard-density* is defined as (#hard clauses / #variables) and the *soft-density* as ((#hard clauses + # soft clauses)/#variables).

Four types of experiments were conducted, all with a different hard-density and increasing number of soft clauses. These experiments were performed on the following hard-densities:

- Theoretical threshold; #hard clauses = # variables
- Experimentally determined 50-50 transition; 50% of instances SAT
- Experimentally determined 10-90 transition; 10% of instances SAT
- Experimentally determined 5-95 transition; 5% of instances SAT

See figure 1 for an overview of all transitions involved.

4.1 Experiments on theoretical threshold density

From figure 3 and 4 we see that for the SoftSAT-Z algorithm for 50, 100 and 150 variables the runtime starts to gradually increase. For 150 variables this increase shows exponentially with growing number of soft clauses around a soft-density of 1.5. For 100 variables the same shows around 2.5. For the PM-SAT algorithm quite similar runtimes are observed.

4.2 Experimentally determined 50-50 transition

From figure 5 we observe that for 50 variables the performance is much better than on the theoretical threshold density. This is especially the case for 50 variables, which now has modest runtime for densities up to 70. Also there is a small increase in these runtimes. If we look at figure 6 however, there is no clear increase in instances that are manageable for the PM-SAT algorithm.

4.3 Experimentally determined 10-90 transition

Figure 7 and 8 show the results of both solvers for the 10-90 transition. The greatest difference in results for both solvers are the results for 50 variables: While SoftSAT-Z can solves instances up to density 300, the PM-SAT solver could only handle instances till density 6.

4.4 Experimentally determined 5-95 transition

Figure 9 shows that for SoftSAT-Z problem instances with 50 variables can be computed within seconds even for densities up to 300, whereas the same runtimes are reached for densities below 10, for instances with 100 and 150 variables. On the other hand, figure 10 shows that for the PM-SAT on instances with 50, 100 and 150 variables the runtime is growing very rapidly with the density.

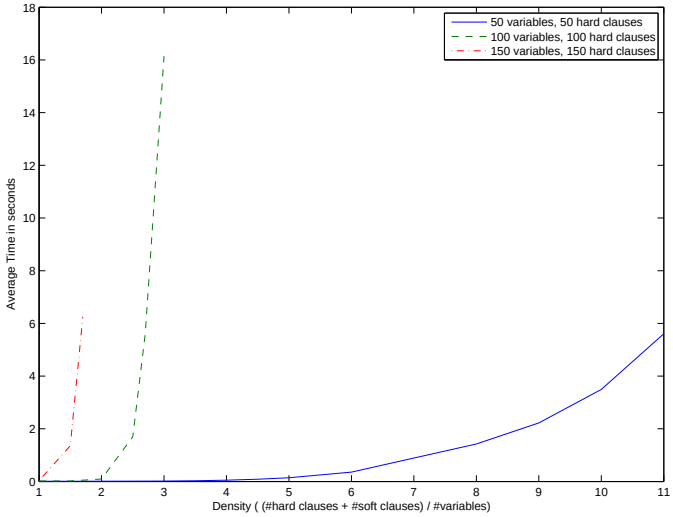

Figure 3: *Average Time for theoretical threshold density with* SoftSAT-Z*:* 1,000 runs for 50, 100 and 150 variables

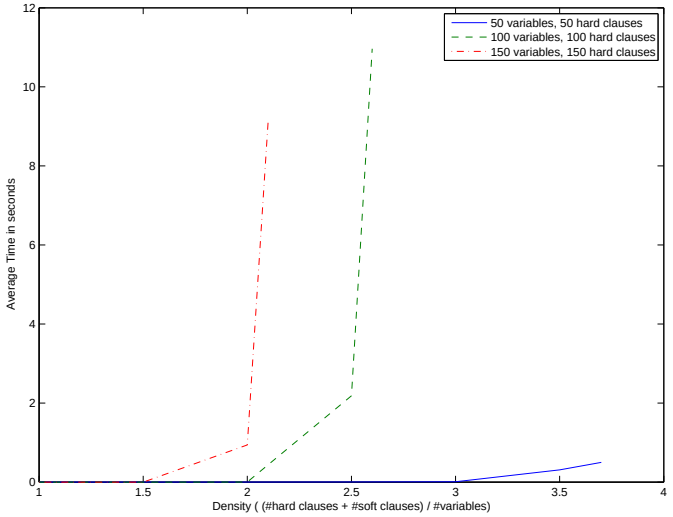

Figure 4: *Average Time for theoretical threshold density with* PM-SAT*:* 1,000 runs for 50, 100 and 150 variables

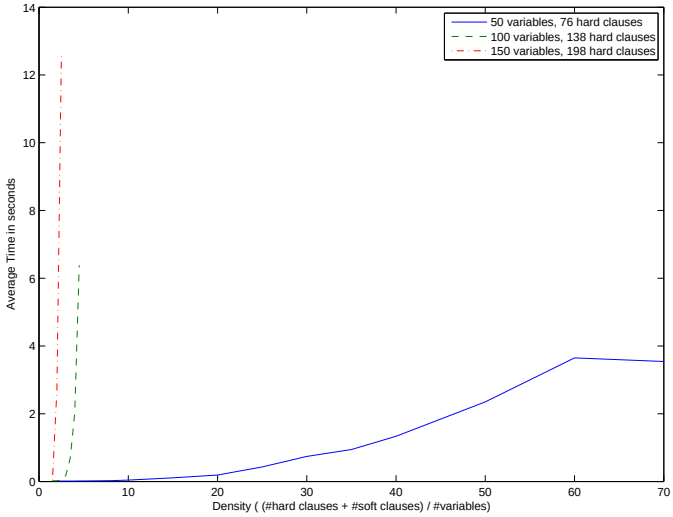

Figure 5: *Average Time for observed 50-50 transition with* SoftSAT-Z*:* 1,000 runs for 50, 100 and 150 variables

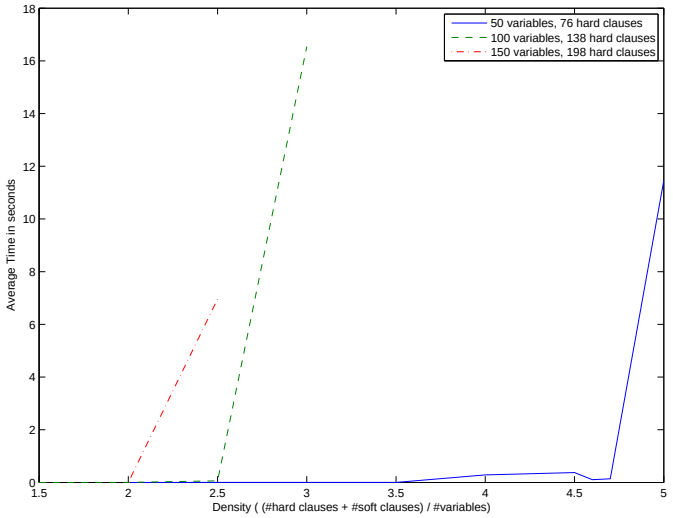

Figure 6: *Average Time for observed 50-50 transition with* PM-SAT*:* 1,000 runs for 50, 100 and 150 variables

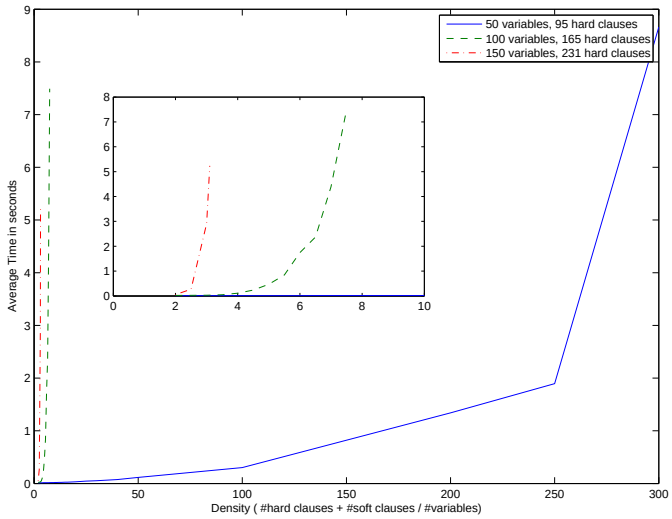

Figure 7: *Average Time for observed 10-90 transition with* SoftSAT-Z*:* 1,000 runs for 50, 100 and 150 variables

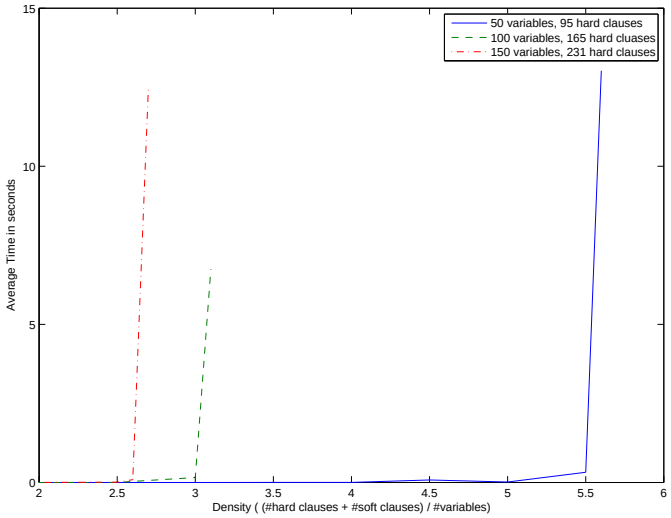

Figure 8: *Average Time for observed10-90 transition with* PM-SAT*:* 1,000 runs for 50, 100 and 150 variables

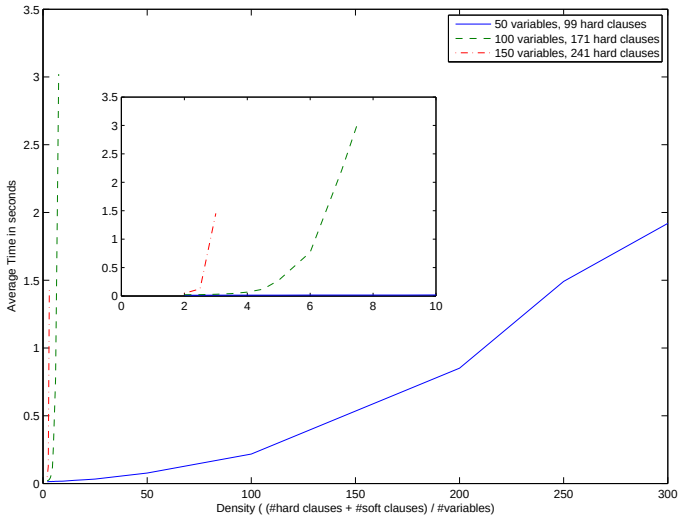

Figure 9: *Average Time for observed 5-95 transition with* SoftSAT-Z*:* 1,000 runs for 50, 100 and 150 variables

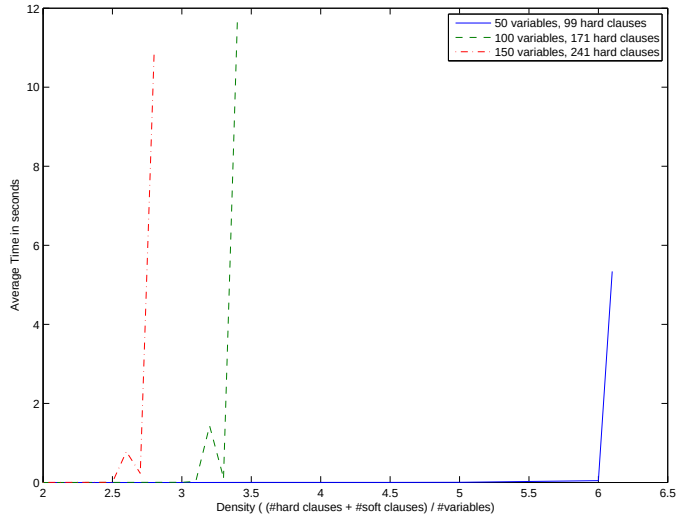

Figure 10: *Average Time for observed 5-95 transition with* PM-SAT*:* 1,000 runs for 50, 100 and 150 variables

5 Conclusion

We investigated the performance of two solvers, SoftSAT-D and PM-SAT, on the 2-SoftSAT problem. SoftSAT-D takes a classical approach in tackling 2-SoftSAT using a branch and bound algorithm. A branch is abandoned if no solution can be found on that branch for which the number of satisfied soft clauses is higher than that of a previous found solution. PM-SAT has a totally different approach. It starts by making all the clauses hard and tries to solve the SAT instance. If it is satisfiable it has found a solution. If there is no solution, it finds the clauses that causes the unsatisfiability and relaxes these. Then it retries to solve the new SAT instance. This process is performed repeated until a solution is found.

We modified the local search module of SoftSAT-D by finding a better start solution - since finding a feasible solution to the hard part is easy for 2-SAT. The performances of the solvers are measured at four different SAT/UNSAT transition-densities with increasing number of soft clauses. The experiments showed that SoftSAT-Z performs better than PM-SAT for all interesting domains within the context of 2-SoftSAT.

6 Acknowledgments

We are grateful to Josep Argelich, Felip Manya, Zhaohui Fu and Sharad Malik for providing us with the sources of their solvers.

References

[ARCH05] Josep Argelich and Felip Manyà, *Solving Over-Constrained Problems with SAT Technology*, F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 1−15, 2005.

[MALI06] Zhaohui Fu and Sharad Malik, *On Solving the Partial MAX-SAT Problem*, To appear LNCS SAT 2006.

[SIPS97] Michael Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, U.S, pp. 254−259, 1997.

[BOLL01] Béla Bollobás, Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, David B. Wilson, *The scaling window of the 2-SAT transition*, Random Structures and Algorithms, Volume 18, no. 3, pp. 201−256, 2001.

[MOSK01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik *Chaff: Engineering an Efficient SAT Solver*, 39th Design Automation Conference (DAC 2001), Las Vegas, June 2001. zChaff download: `http://www.princeton.edu/~chaff/zchaff.html`.

[THUR06] M. Thurley, *The sharpSAT #SAT solver*, Version 1.0. April 04, 2006, download: `http://www2.informatik.hu-berlin.de/~thurley/sharpSAT/index.html`