DC-SSAT: A Divide-and-Conquer Approach to Solving Stochastic Satisfiability Problems Efficiently

Stephen M. Majercik
Department of Computer Science
Bowdoin College
Brunswick, ME, USA 04011
smajerci@bowdoin.edu

Byron Boots
Center for Cognitive Neuroscience
Duke University
Durham, NC, USA 27708
bboots@duke.edu

Abstract

We present DC-SSAT, a sound and complete divide-and-conquer algorithm for solving stochastic satisfiability (SSAT) problems that outperforms the best existing algorithm for solving such problems (ZANDER) by several orders of magnitude with respect to both time and space. DC-SSAT achieves this performance by dividing the SSAT problem into subproblems based on the structure of the original instance, caching the viable partial assignments (VPAs) generated by solving these subproblems, and using these VPAs to construct the solution to the original problem. DC-SSAT does not save redundant VPAs and each VPA saved is necessary to construct the solution. Furthermore, DC-SSAT builds a solution that is already human-comprehensible, allowing it to avoid the costly solution rebuilding phase in ZANDER. As a result, DC-SSAT is able to solve problems using, typically, 1-2 orders of magnitude less space than ZANDER, allowing DC-SSAT to solve problems ZANDER cannot solve due to space constraints. And, in spite of its more parsimonious use of space, DC-SSAT is typically 1-2 orders of magnitude faster than ZANDER. We describe the DC-SSAT algorithm and present empirical results comparing its performance to that of ZANDER on a set of SSAT problems.

Introduction

Stochastic Boolean satisfiability (SSAT) (Papadimitriou 1985; Littman, Majercik, & Pitassi 2001) is a generalization of satisfiability (SAT) that is similar to quantified Boolean formulae (QBF). The ordered variables of the Boolean formula in an SSAT problem, instead of being existentially or universally quantified, are existentially or *randomly* quantified. Randomly quantified variables are `true` with a certain probability, and an SSAT instance is satisfiable with some probability that depends on the ordering of and interplay between the existential and randomized variables. The goal is to choose values for the existentially quantified variables that maximize the probability of satisfying the formula.

Like QBF, SSAT is PSPACE-complete, so it is theoretically possible to transform many probabilistic planning and reasoning problems of great practical interest into SSAT instances (Littman, Majercik, & Pitassi 2001). While such theoretically guaranteed translations are not always practical, previous work has shown that, in some cases, the

* Appears in the *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 416-422, AAAI Press, 2005.

ables in Φ

problems Φ_i and Φ_{i+1} will be some number of variables that

Problem (States in Plan Prob)	Size Statistic	Number of Steps = Number of ECs/SPs		
		5	t	50
COF (256)	NV	103	$19t + 8$	958
	NC	290	$56t + 10$	2810
	AVSP	27.0	27.0	27.0
	ACSP	58.0	57	56.2
SPF (256)	NV	103	$19t + 8$	958
	NC	362	$70t + 12$	3512
	AVSP	27.0	27.0	27.0
	ACSP	72.4	71	70.2
FAC (65536)	NV	276	$52t + 16$	2616
	NC	2007	$397t + 22$	19872
	AVSP	68.0	68.0	68.0
	ACSP	401.4	399	397.4
LI10 (1024)	NV	145	$27t + 10$	1360
	NC	660	$128t + 20$	6420
	AVSP	37.0	37.0	37.0
	ACSP	132.0	130	128.4
EX4 (16)	NV	59	$11t + 4$	554
	NC	178	$34t + 8$	1708
	AVSP	15.0	15.0	15.0
	ACSP	35.6	35	34.2

NV=Num Vars, NC=Num Clauses, EC=Equiv Class, SP=Subproblem, AVSP=Avg NV/SP, ACSP=Avg NC/SP

Table 1: Problem and decomposition characteristics.

$\varepsilon_2 = \overline{x_1}$ agrees with α_5, the VPA associated with $\varepsilon_5 = x_2$ and this solution ($x_1 = \texttt{false}, x_2 = \texttt{true}$) has a probability of satisfaction of 0.48 (y_1 must be `false` and y_2 must be `true`), so this is the optimal assignment.

Experimental Results

We tested DC-SSAT and ZANDER on a set of COPP-SSAT problems adapted from Majercik & Littman (2003) (COF = coffeebot, GOx = general-operations-x) and from (Hoey *et al.* 1999) (FAC = factory, EX4 = exponential04, LI10 = linear10), and on one additional problem (SPF = spearfishing). We omit problem descriptions, but provide information regarding problem size and the DC-SSAT decomposition of each problem in Tables 1 and 2. Tables 3 and 4 describe resource usage on problem instances of increasing size; the better time/space usage in each case is indicated in bold face. Reported results are representative of all results obtained. All tests were run on a Power Mac G5 with dual 2.5 GHz CPUs, 2 GB RAM, 512KB L2 cache/CPU, running Mac OS X 10.3.8. We omit probabilities of satisfaction due to lack of space.

In the COF problem, DC-SSAT is 1-2 orders of magnitude faster and uses 1-2 orders of magnitude less space (Table 3). In the largest COF problem that ZANDER was able to solve before exhausting memory (the 110-step problem, not shown, with 2098 variables and 6170 clauses), ZANDER required 43.61 CPU seconds and used 1411.32 MB of memory, while DC-SSAT was able to solve the problem in 0.35 CPU second using 25.12 MB of memory, 2 orders of magnitude faster and using 2 orders of magnitude less space.

Problem (States in Plan Prob)	Size Statistic	Number of Steps = Number of ECs		
		5	t	50
GO6 (128)	NV	107	$20t + 7$	1007
	NC	374	$72t + 14$	3614
	AVSP	27.0	27.0	27.0
	ACSP	74.8	73	72.3
GO7 (256)	NV	123	$23t + 8$	1158
	NC	451	$87t + 16$	4366
	AVSP	31.0	31.0	31.0
	ACSP	90.2	88	87.3
GO8 (512)	NV	139	$26t + 9$	1309
	NC	533	$103t + 18$	5168
	AVSP	35.0	35.0	35.0
	ACSP	106.6	104	103.4
GO9 (1024)	NV	155	$29t + 10$	1460
	NC	620	$120t + 20$	6020
	AVSP	39.0	39.0	39.0
	ACSP	124.0	121	120.4

NV=Num Vars, NC=Num Clauses, EC=Equiv Class, SP=Subproblem, AVSP=Avg NV/SP, ACSP=Avg NC/SP

Table 2: Problem and decomposition characteristics.

Table 3 shows that, in most cases, ZANDER exhausted memory on the SPF problem and was timed out after 20 minutes on the FAC problem (which provided the largest SSAT instances in our test set; see Table 1). In the 10-step SPF problem, the one problem that both solvers completed that had a non-zero probability of satisfaction, the difference is dramatic: DC-SSAT was 3 orders of magnitude faster and used 2 orders of magnitude less space. Although ZANDER frequently uses much of the time and space it consumes to rebuild its solution tree, this was not the case in SPF and FAC, where the resources were consumed mostly in the solution calculation phase. We argue below that this is due to the larger number of actions (which increases the size of the existential blocks) in SPF (6 actions) and FAC (14 actions) compared to COF (4 actions).

Neither the SSAT problem size nor the size of the subproblems in the DC-SSAT decomposition seems to be a reliable indicator of DC-SSAT's performance; e.g. the 50-step FAC problem is approximately 2-3 times larger than the 50-step LI10 problem on all size statistics (Table 1), but DC-SSAT's solution time for that FAC problem was approximately $\frac{1}{4}$ its solution time for the LI10 problem. LI10 (like EX4) is an artificial problem; together, they shed some light on this issue and on the generally extreme differences between the performances of DC-SSAT and ZANDER. The LI10 problem requires a number of time steps (existential blocks) that is *linear* in the number of variables in each of these blocks in order to obtain a nonzero probability of satisfaction, although the actions must be executed in just the right order. Aided by unit propagation, ZANDER quickly establishes the correct sequence of actions, starting with the final action and working its way back to the initial action (although ZANDER

Prob-	Sol-	Re-	Resource Usage by Number of Steps in Plan									
lem	ver	source	5	10	15	20	25	30	35	40	45	50
COF	DC	CS	**0.02**	**0.02**	**0.03**	**0.04**	**0.07**	**0.07**	**0.08**	**0.12**	**0.12**	**0.14**
		MB	0.04	0.27	**0.63**	**1.08**	**1.63**	**2.27**	**3.01**	**3.84**	**4.64**	**5.65**
	ZA	CS	0.06	0.03	0.07	0.25	0.84	2.19	3.85	5.62	7.53	9.48
		MB	**0.00**	**0.14**	1.55	9.46	38.78	111.59	192.60	273.65	354.72	435.83
SPF	DC	CS	**0.01**	**0.03**	**0.06**	**0.08**	**0.11**	**0.13**	**0.17**	**0.20**	**0.22**	**0.25**
		MB	0.07	**0.48**	**1.07**	**1.80**	**2.70**	**3.74**	**4.94**	**6.30**	**7.59**	**9.23**
	ZA	CS	0.02	29.87	–	–	–	–	–	–	–	–
		MB	**0.01**	28.69	M	M	M	M	M	M	M	M
FAC	DC	CS	**0.17**	**2.65**	**7.81**	**14.26**	**19.95**	**26.01**	**30.51**	**37.20**	**43.36**	**50.80**
		MB	**0.70**	**16.44**	**50.47**	**100.27**	**162.08**	**237.36**	**326.11**	**428.33**	**544.02**	**673.18**
	ZA	CS	3.47	1200+	1200+	1200+	1200+	1200+	1200+	1200+	1200+	1200+
		MB	0.99	–	–	–	–	–	–	–	–	–
LI10	DC	CS	0.11	8.71	25.12	44.55	66.94	92.01	**119.16**	**146.81**	**178.58**	**205.37**
		MB	1.20	24.53	66.75	126.72	**196.84**	**280.08**	**376.46**	**497.12**	**621.39**	**758.78**
	ZA	CS	**0.02**	**0.02**	**0.47**	**2.60**	**7.60**	**24.83**	–	–	–	–
		MB	**0.00**	**0.00**	**3.83**	**37.48**	206.83	1002.53	M	M	M	M
EX4	DC	CS	0.00	**0.01**	**0.01**	**0.02**	**0.02**	**0.03**	**0.04**	**0.04**	**0.06**	**0.06**
		MB	0.01	0.07	0.19	0.37	0.56	**0.81**	**1.09**	**1.35**	**1.70**	**2.08**
	ZA	CS	0.00	0.03	39.89	201.93	291.22	355.95	1200+	1200+	1200+	1200+
		MB	**0.00**	**0.00**	**0.00**	**0.04**	**0.27**	0.99	–	–	–	–

DC=DC-SSAT, ZA=ZANDER, CS=CPU secs, MB=MB mem, 1200+=Timed out, M=Mem exhausted, 0.00=Rounding

Prob-lem	Sol-ver	Re-source	Resource Usage by Number of Steps in Plan									
			5	10	15	20	25	30	35	40	45	50
GO6	DC	CS	**0.03**	**0.11**	**0.22**	**0.34**	**0.43**	**0.53**	**0.68**	**0.76**	**0.84**	**0.96**
		MB	0.31	2.44	5.43	**9.22**	**13.78**	**19.13**	**26.07**	**33.12**	**40.95**	**49.56**
	ZA	CS	0.07	0.65	0.73	1.17	2.95	9.08	26.28	–	–	–
		MB	**0.04**	**0.77**	**3.25**	20.75	104.82	402.50	1254.17	M	M	M
GO7	DC	CS	**0.05**	**0.29**	**0.57**	**0.87**	**1.21**	**1.57**	**1.99**	**2.46**	**2.95**	**3.46**
		MB	0.49	**5.38**	**13.05**	**22.22**	**34.37**	**47.42**	**64.06**	**83.09**	**102.12**	**125.64**
	ZA	CS	0.14	12.81	13.42	15.30	21.03	45.04	–	–	–	–
		MB	**0.06**	9.34	13.41	51.00	293.12	1379.37	M	M	M	M
GO8	DC	CS	**0.06**	**1.02**	**2.54**	**4.59**	**7.11**	**10.24**	**13.70**	**17.87**	**20.71**	**23.60**
		MB	0.81	**11.54**	**29.20**	**52.23**	**80.65**	**114.44**	**153.61**	**198.16**	**248.09**	**303.40**
	ZA	CS	0.18	368.04	393.09	427.61	451.85	1200+	1200+	1200+	1200+	1200+
		MB	**0.09**	228.36	234.54	301.16	878.43	–	–	–	–	–
GO9	DC	CS	**0.10**	**6.73**	**20.85**	**35.97**	**55.11**	**76.22**	**100.63**	**122.56**	**141.97**	**162.15**
		MB	1.18	**24.07**	**64.16**	**120.40**	**185.86**	**270.46**	**370.00**	**474.25**	**602.14**	**731.77**
	ZA	CS	0.26	1200+	1200+	1200+	1200+	1200+	1200+	1200+	1200+	1200+
		MB	**0.00**	–	–	–	–	–	–	–	–	–

DC=DC-SSAT, ZA=ZANDER, CS=CPU secs, MB=MB mem, 1200+=Timed out, M=Mem exhausted, 0.00=Rounding

Table 4: ZANDER's performance deteriorates significantly as the number of actions in the GOx problems increases.

structure of a COPP-SSAT problem ensures that all the subproblems, except for the first and last ones, are isomorphic, it might be possible to use the solution from the first of these to extend the time horizon of the overall solution/plan an arbitrary number of steps at very little incremental cost, producing a planner that is able to solve very large problems efficiently. We plan to implement this idea and test DC-SSAT against other probabilistic planning techniques. Also, in a future paper, we will show how DC-SSAT can be extended to partially observable probabilistic planning problems and to general SSAT problems. One difficulty here is that a VPA in subproblem Φ_i may contain variables that are also in subproblem $\Phi_j, j > i + 1$; hence, checking VPA compatibility only between adjacent subproblems is no longer sufficient. One consequence is that the depth-first search that calculates $Pr[\Phi]$ must maintain a list of VPAs that contain variables in later subproblems so that compatibility between non-adjacent subproblems can be checked as needed.

There are other interesting directions for further work. In the case of general SSAT problems, can we characterize when the DC-SSAT approach will be beneficial? It will fail if all the existential variables are related and form a single equivalence class, but what is the impact of the connection topology in general? Variables that are far apart in the prefix, but share a clause, may induce the solver to explore a long, unproductive path. Assignment compatibility issues that were local in a COPP-SSAT problem become global in the general case. Walsh (2001) investigated the impact of the connection topologies of graphs associated with real-world search problems and related this to the notion of a *backbone* (variables that have the same value in all solutions). What role do these notions play in SSAT problems?

Finally, although QBF problems do not require a solver to consider all possible satisfying assignments, SSAT and95(other)-296(interesting)-295(directions)-295(for)-295(further)-295(w)10(osignm9r

Walsh, T. 2001. Search on high degree graphs. In *IJCAI-01*, 266–274.