Revisiting the Decision Heuristics for the Look-Ahead Based SAT Solvers*

Ankit Shukla[1], Armin Biere[1], and Martina Seidl[1]

JKU, Linz, Austria
{ankit.shukla, biere, martina.seidl}@jku.at

Abstract. We consider the problem of finding the largest renamable Horn sub-CNF (MRHorn) of a given CNF.

Keywords: Renamable-Horn · SAT · Look ahead solvers.

1 Introduction

The look-ahead solvers use a completely different set of heuristics for the search compared to the CDCL solvers. These heuristics are used to gather information for better pertinence search spaces and decide the variable to branch. Look-ahead solvers are the first component of the cube-and-conquer paradigm and currently the best way to parallelize combinatorial problems. As seen in [8] choosing the decision heuristic for hard combinatorial problems requires manual interaction. The process of selection of heuristic is not completely automatic and generally, the decision requires sampling instances and running a set of decision heuristics to figure out the appropriate one. We present a complete formalize method to automate this task. The process of automation might allow us to use more powerful decision heuristics.

We revisit the decision heuristics for the lookahead solvers, and introduce renamable-Horn heuristic; an extension of the Horn-heuristic used in the SAT solver SATO [16]. Although theoretically attractive, Horn-based heuristics has not been used in the SAT solver since [16].

The sampling part of our method is inspired by the recent success of the Monte Carlo tree search (MCTS) algorithm in combinatorial reasoning [2, 9], maximum satisfiability (MaxSAT), [7] and constraint satisfaction problems (CSPs) [12]. In the area of propositional satisfiability (SAT), the MCTS-based approach [14,15] has been used to perform unit propagation [6] and Conflict-Driven Clause Learning (CDCL) [13]. The MCTS-based approaches target improved quality ("good") clause learning [10, 15] and guiding the CDCL search by random exploration [3, 4] of the search tree.

* Supported by organization XXX.

2 Preliminaries

The satisfiability problem, known as SAT, is a decision problem; given an input clausal formula Φ, determines whether Φ is satisfiable, i.e., if there exists an interpretation that satisfies the given formula. SAT is an NP-complete problem [5] and, it is widely believed that no efficiently (polynomial-time) algorithm exists to solve SAT. Although, there are subclasses of satisfiability with syntactic restrictions that are solvable in polynomial time, for example, 2-satisfiability (2SAT) and Horn-satisfiability (HORNSAT).

The major approaches to exploit the special structure of the CNF which are polynomial time solvable, either reduce the instance by an application of a partial assignment to a polynomial-time solvable subclasses, or identify the sub-formula of the instance belonging to a polynomial-time solvable subclass.

Maximum renamable Horn CNFs: We consider the problem of finding the largest renamable Horn sub-CNF (MRHorn) of a given CNF, i.e. to find the largest sub-family $H \subseteq C$ of a given CNF C such that H is renamable Horn. This problem is NP-Hard.

3 Estimating the size of the branching subtree

Early work by Knuth [11] attempted to predict the efficiency of a backtracking algorithm by estimating the size of the backtracking search tree. The size of a backtracking search tree was associated with the length of randomly sampled paths through the tree.

We present a new method to sample paths through the backtracking search tree size for the UNSAT formulas. The selection of variable and its assignment is a Prover-Delayer game. The Prover and Delayer take turns; the Prover picks an unassigned variable, and Delayer assigns it value. The game terminates if the formula become false by the current assignment. We use the number of assigned variables as the measure for the tree size. The Prover (Delayer) attempts to minimize (maximize) this measure. After each player's turn, we allow unit propagation to imply unit variable assignments and reduce the formula further. The implied assignments of the variables do not contribute to the tree size measure criteria.

We embed the MRHorn based heuristic in the above method to choose the decision variable from the non-horn part. The game terminates when the formula becomes renameable horn.

4 Maximum renamable Horn Heuristic

Based on MRHorn we present our new decision heuristic, MRH. The central idea of our heuristic is to compute a MRHorn of the given CNF and choose a decision variable from the non-horn part of the formula, hence reducing it's size. Iterate the procedure until the whole formula becomes a nameable horn and therefore

can be solved in polynomial time. The heuristic to chose a decision variable is decoupled from the algorithm of finding the MRHorn.

4.1 MaxSAT translation of MRHorn

We present a translation of the MRHorn problem into a maximum satisfiability problem (MaxSAT). The translation idea is based on the IP-formulation of the MRHorn [1].

Example 1. Let us consider a clause $C = x_1 \vee x_2 \vee x_3$. The clause C will become Horn if at most one of the following events happen: "x_1 is not flipped", "x_2 is not flipped", or "x_3 is flipped". Correspondingly, C will become a Horn clause if and only if

$$\overline{f_1} + \overline{f_2} + f_3 \leq 1$$

where each new variable f_i represent the flipping of the corresponding variable x_i.

For $\mathcal{C} = \{C_1, ..., C_n\}$ we get set of linear constraints $\{F_1, ..., F_n\}$ to solve.

Next we present a partial MaxSAT translation of the MRHorn.

1. We translate each of the $F_i,\ 1 \leq i \leq n$ constraint as *at most one* constraint (AMO). The AMO constraint is true if at most one of the variables is assigned true. For example 1 the translation will create $\mathsf{AMO}(\neg f_1, \neg f_2, f_3)$, similarly for rest of the constraints.
2. For each of the AMO constraint F_i we use an additional switch variable s_i as a part of the constraint by adding negation of the switch variable to each clause of the AMO. For example 1 the AMO will create three clauses:

   $$(f_1 \vee f_2) \wedge (f_1 \vee \neg f_3) \wedge (f_2 \vee \neg f_3)$$

   we add the switch variable $\neg s_1$ to each of the three clauses.

   $$(\neg s_1 \vee f_1 \vee f_2) \wedge (\neg s_1 \vee f_1 \vee \neg f_3) \wedge (\neg s_1 \vee f_2 \vee \neg f_3) \tag{1}$$

   We represent the AMO constraint with the switch variable as AMO_s.
3. We add each AMO with switch variable AMO_s as a hard constraint (constraint with infinite weight) in MaxSAT. Note that every model of the MaxSAT problem must satisfy every hard constraints.
4. Next, we add each s_i to the MaxSAT with weight 1.
5. We then solve the MaxSAT problem to maximize the weights of switch variable $s_i,\ 1 \leq i \leq n$.

Note that the equation 1 and unit clause s_1 with their corresponding weights can be represented as

$$(s_1 \supset (f_1 \vee f_2))^{\infty} \wedge (s_1 \supset (f_1 \vee \neg f_3))^{\infty} \wedge (s_1 \supset (f_2 \vee \neg f_3))^{\infty} \tag{2}$$

$$(s_1)^1 \tag{3}$$

setting $s_1 = 1$ will satisfy and enforce $\mathsf{AMO}(\neg f_1, \neg f_2, f_3)$ to be *true*.

Algorithm 1: Estimate tree size based on sampling a path

Input: $\phi = \{C_1, C_2, \ldots, C_n\}$, v = no.of vars
Output: k = number of assigned variables

$k = 0, lit = var = 0$
while $(v = k) \wedge (\phi \neq UNSAT)$ **do**
  unitPropagation(ϕ)
  if $\phi == UNSAT$ **then** skip
  var = decVariable(MRH, ϕ)
  lit = phaseSelection(var)
  $k = k + 1$
  $\phi = \phi|_{lit}$
return k

5 Algorithm

The algorithm progresses down a single path of the complete branching tree, similar to the simulation path of the Monte Carlo tree search. To avoid the making decision on the implied variables, first we perform unit propagation and update the formula. Then we check the satisfiability of the formula: in case the formula turned UNSAT we end the while loop and return the assigned variable count. There will be multiple implied variables due to unit propagation but we do not consider these variables as part of the complexity for the tree size. Next, we use our newly introduced look-ahead heuristic, MRH, to pick the decision variable. Using a random phase selection, we decide on the phase of the selected variable. We increase the selected variable count and apply the literal application to the formula and continue. In case the formula is UNSAT, or the number of assigned variables equals the total number of variables in the formula we exist the loop and return the assigned variable count. The assigned variable count represents an abstracted size of the tree.

6 Experiments

We have implemented the the MRHorn heuristic in a small codebase MaxRhorn[1]. To evaluate the effectiveness of the heuristic, we implement and compare the performance of MRHorn with four other decision heuristics, two DPLL based heuristics; Jeroslow-Wang and Max-Occurrence and two lookahead based heuristic, weighted binary heuristic (WBH) and clause reduction heuristic (CRH). The MaxRhorn includes 2,000 lines of C++ code.

All the experiments were performed on the cluster where each compute node has two Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz with turbo-mode disabled. Time limit was set to 2700 seconds and memory limit to 7 GB.

The selected benchmarks sets are from test suite of mellon, random 3-SAT instances, Marijn hard instance and structured instances. Table 1 shows the

[1] **https://github.com/arey0pushpa/dcnf-autarky**

Instances	Max Cls. Sat				Jeroslow-Wang				CRH (lookahead)				WBH (lookahead)				Max RHorn			
	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med
add16	15	45	22.27	19	16	46	21.41	20	15	45	22.27	19	19	42	31.65	32	5	46	16.95	15
add32	31	84	37.93	35	32	81	37.89	36	31	84	37.97	35	39	79	61.61	65	5	80	27.55	24
add64	63	133	69.27	67	64	128	69.73	68	63	133	69.37	67	80	153	116.41	122	5	154	46.33	41
prime65537	14	15	14.91	15	12	18	14.80	15	14	28	14.91	15	3	53	8.88	8	1	21	2.97	1
mulcom006	7	19	9.36	9	7	25	10.03	10	7	19	9.38	9	7	21	10.29	10	13	28	17.63	17
mulcom008	9	24	13.23	13	9	28	13.49	13	9	23	13.17	13	9	27	13.50	13	1	33	25.85	26
ph04	3	6	4.25	4	3	11	5.32	5	3	6	4.24	4	3	11	5.33	5	3	11	5.30	5
ph05	4	10	6.11	6	4	19	7.49	7	4	10	6.14	6	4	19	7.47	7	4	17	7.50	7
ph11	10	37	17.98	18	10	42	19.82	19	10	43	18.20	18	10	48	19.84	19	10	40	19.85	19
ph12	11	40	19.98	20	11	49	21.82	21	11	46	20.20	20	11	46	21.79	21	11	48	21.85	21
ph13	12	40	21.95	22	12	57	23.91	23	12	43	22.25	22	12	48	23.82	23	12	44	23.88	23
ph14	13	46	23.95	24	13	51	25.87	25	13	52	24.30	24	13	53	25.93	25	14	52	25.84	25
ph15	14	52	26.09	26	14	56	27.87	27	14	51	26.20	26	16	53	27.95	27	14	55	27.95	27
ph17	17	53	29.94	30	17	57	30.94	30	16	58	30.19	30	17	58	31.95	31	16	58	31.84	31
ph18	19	59	32.03	32	18	61	32.99	32	17	57	32.19	32	19	64	33.92	33	17	64	33.89	33
ph24	27	69	43.97	44	27	77	44.86	44	27	76	44.24	44	25	76	45.86	45	26	75	45.86	45
ph28	33	86	52.00	52	32	91	53.02	53	33	85	52.08	52	34	86	53.82	53	35	85	54.01	54
ph31	38	98	58.04	57	38	96	58.87	58	37	91	58.04	57	36	97	60.13	60	39	97	59.90	59
ph34	41	96	64.05	64	41	104	64.55	64	39	100	64.15	64	42	103	65.82	65	45	93	65.90	65
sdiv5prop	5	15	8.48	9	6	28	8.88	9	5	15	8.50	9	6	25	9.50	9	5	33	9.76	9
sdiv6prop	6	19	9.75	10	6	29	9.97	10	6	21	9.77	10	7	24	10.81	11	5	36	11.06	11
tph02	2	3	2.50	2	2	3	2.50	3	2	3	2.50	2	2	3	2.51	3	2	3	2.51	3
tph05	8	25	13.75	14	8	28	14.38	14	8	25	13.73	14	8	30	14.49	14	8	31	15.62	15
tph08	15	46	26.12	26	14	50	25.83	25	14	46	25.87	26	15	52	26.70	26	14	61	27.85	27
tph13	28	82	46.46	46	27	77	45.72	45	28	71	45.82	45	26	76	46.81	46	29	78	47.97	48
tph17	41	98	62.69	62	38	99	61.73	61	38	95	61.90	61	38	99	62.75	62	42	94	63.60	63
udiv10prop	11	26	11.83	11	10	48	12.34	10	11	26	11.84	11	10	40	13.19	12	10	50	19.43	16
udiv7prop	8	20	8.70	8	7	36	9.03	8	8	20	8.72	8	7	38	9.29	8	7	43	15.26	13
udiv8prop	9	24	9.78	9	8	41	10.59	9	9	20	9.76	9	8	38	10.51	9	8	49	16.97	14
udiv9prop	10	24	10.82	10	9	51	11.31	10	10	24	10.85	10	9	42	12.11	11	6	51	18.16	15
uniqinv3prop	2	9	3.44	3	3	5	4.24	4	2	9	3.43	3	2	8	4.26	5	3	5	4.25	4
uniqinv4prop	3	11	5.28	5	5	8	6.21	6	3	11	5.26	5	4	15	6.63	6	3	9	5.85	6
uniqinv5prop	4	16	7.16	7	7	11	8.22	8	4	15	7.15	7	5	20	8.53	8	3	15	7.40	8
uniqinv7prop	6	22	10.37	11	7	16	12.07	12	6	22	10.40	11	7	30	11.44	11	3	25	10.48	12

Table 1: Sampling score of 10,000 runs of each decision heuristic

comparison of all five selected heuristics on the mellon test suite. The MRHorn heuristic performs best on the structured instances. First four instances in gold are the ones where MRHorn overperform all other heuristic.

Instances	Max Cls. Sat				Jeroslow-Wang				CRH (lookahead)				WBH (lookahead)				Max RHorn			
	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med
filler	6	22	10.37	11	7	16	12.07	12	6	22	10.40	11	7	30	11.44	11	3	25	10.48	12

Table 2: Sampling score of 10,000 runs of each decision heuristic on random instances

Table 2: Run on Random instances. XXX

Instances	Max Cls. Sat				Jeroslow-Wang				CRH (lookahead)				WBH (lookahead)				Max RHorn			
	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med
filler	6	22	10.37	11	7	16	12.07	12	6	22	10.40	11	7	30	11.44	11	3	25	10.48	12

Table 3: Sampling score of 10,000 runs of each decision heuristic on random instances

Table 3: Run on Marijn's hard instances. XXX
Multiplication and Benchmarks.

Instances	Max Cls. Sat				Jeroslow-Wang				CRH (lookahead)				WBH (lookahead)				Max RHorn			
	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med	Min	Max	Mu	Med
filler	6	22	10.37	11	7	16	12.07	12	6	22	10.40	11	7	30	11.44	11	3	25	10.48	12

Table 4: Sampling score of 10,000 runs of each decision heuristic on random instances

Table 4: Run on structured instances. XXX

6.1 Average height tun time correlation

Update the code to show the average decision tree height corresponds to the run time. Sum up the time of sampling for the correlation. This experiment should be performed on the run of the solver on each 10000 instances.

6.2 Correctness

Check for the programming mistake. Heuristic bugs. Add Logging and verbose messages.

6.3 CDCL can solve MRHorn easily

If the formula is horn or renamable horn CDCL will just find it and solve it easily. Put in the paper. Empirical study where CDCL can solve HORN formula. This is an argument for Cube and Conquer but we just do just Look-Ahead.

Quote "Actually from Oliver's 2013 paper and 2014 JAR article. There is a Schlipf et.al. paper IPL 1995 (also with John Franco) which introduced SLUR. SLUR: decides all those classes of formulas (even if you do not know that it falls in this class). I looked at that before (actually was the editor in charge for the JAR paper by Oliver). The SLUR algorithm is a strange DLL variant where you do look-ahead and commit otherwise to decisions. If probing of both phases of variables does not lead to a conflict you commit to (an arbitrary) phase. If only one phase leads to a conflict you force the probing variable to the other phase. If initially unit propagation gives a conflict you determine UNSAT, otherwise if (after a decision - which is not in the paper) both phases of the probing yield a conflict, then you give up (because you committed to another decision earlier). In the Franco DIMACS'96 paper he refers to Truemper claiming the two propagations can be interleaved/done-in-parallel which reduces the overall complexity to linear (which I partially can follow only). Anyhow, now to our problem. If you get to the point in our cube and conquer algorithm where the formula becomes RHORN (or even one of those even larger classes they have which are motivated from LP relaxations), then doing the SLUR algorithm would give you indeed a linear algorithm for deciding the formula (does not give up but correctly either says UNSAT or SAT). If you do DLL instead, then it still somewhat unclear what would happen (same for CDCL), but my conjecture is that the search space in DLL below this point can not be exponential. Here is my reasoning. Let us say we pick decision variable X. If unit propagation on X leads to a conflict DLL does the same what SLUR does and forces X to false. If SLUR at this point after propagating !X would not yield a conflict, then also this decision is something SLUR could also do. So the remaining case is where we would pick X as decision, so branch on it, propagate it, and it does not lead to a conflict, while SLUR at this point would also test the propagation of !X and detect a conflict. In this case the SLUR algorithm would force X to true, which however has the same effect as our decision (setting X to true in the left branch). So I would say DLL would simply solve or refute the formula in a linear number of decisions. Now to CDCL. This becomes more complicated since we back-jump, which however in the worse case only gives a linear blow-up (we will need to redo the assignments we were jumping over), so in the worst case quadratic. It might be interesting to look at chronological backtracking (Sibylle's last SAT paper based on the Intel work) in this case, since it avoids this quadratic blow-up of CDCL. If you are doing look-ahead with at least failed literal probing until completion instead of plain DLL or CDCL, then of course this simulates SLUR completely. Then however you usually invest linear time for the failed literal probing anyhow and thus would be able to afford testing this SLUR algorithm without real overhead ...."

Check if this is the case by comparing the run times of the renamable horn formulas. Generate formulas that are renamable horn and run CDCL solver.

If the formula is Horn we'll know that the formula is true. As we have already performed the unit propagation. If it is re-namable horn then we don't know this. Figuring out of the formula is RHorn is hard. SLUR related argument. That will give us the basis!

7 Related Work

XXX

8 Conclusion

XXX

References

1. Boros, E.: Maximum renamable horn sub-cnfs. Discret. Appl. Math. **96-97**, 29–40 (1999)
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012)
3. Chowdhury, M.S., Müller, M., You, J.: Guiding cdcl sat search via random exploration amid conflict depression. In: AAAI. pp. 1428–1435 (2020)
4. Chowdhury, M.S., Müller, M., You, J.H.: Exploration via random walks in cdcl sat solving amid conflict depression
5. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC. pp. 151–158. ACM (1971)
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
7. Goffinet, J., Ramanujan, R.: Monte-carlo tree search for the maximum satisfiability problem. In: CP. Lecture Notes in Computer Science, vol. 9892, pp. 251–267. Springer (2016)
8. Heule, M.J.H.: Schur number five. In: AAAI. pp. 6598–6606. AAAI Press (2018)
9. Jooken, J., Leyman, P., De Causmaecker, P., Wauters, T.: Exploring search space trees using an adapted version of monte carlo tree search for a combinatorial optimization problem. arXiv preprint arXiv:2010.11523 (2020)
10. Keszocze, O., Schmitz, K., Schloeter, J., Drechsler, R.: Improving sat solving using monte carlo tree search-based clause learning. In: Advanced Boolean Techniques, pp. 107–133. Springer (2020)
11. Knuth, D.E.: Estimating the efficiency of backtrack programs. Mathematics of computation **29**(129), 122–136 (1975)

12. Loth, M., Sebag, M., Hamadi, Y., Schoenauer, M.: Bandit-based search for constraint programming. In: International Conference on Principles and Practice of Constraint Programming. pp. 464–480. Springer (2013)
13. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC. pp. 530–535. ACM (2001)
14. Previti, A., Ramanujan, R., Schaerf, M., Selman, B.: Monte-carlo style UCT search for boolean satisfiability. In: AI*IA. Lecture Notes in Computer Science, vol. 6934, pp. 177–188. Springer (2011)
15. Schloeter, J.: A monte carlo tree search based conflict-driven clause learning sat solver. INFORMATIK 2017 (2017)
16. Zhang, H.: SATO: an efficient propositional prover. In: CADE. Lecture Notes in Computer Science, vol. 1249, pp. 272–275. Springer (1997)