Applying UCT to Boolean Satisfiability

Alessandro Previti[1], Raghuram Ramanujan[2], Marco Schaerf[1], and Bart Selman[2]

[1] Dipartimento di Informatica e Sistemistica Antonio Ruberti,
Sapienza, Università di Roma,
Roma, Italy.
`elsandro84@gmail.com, marco.schaerf@uniroma1.it`
[2] Department of Computer Science
Cornell University
Ithaca, New York
`raghu,selman@cs.cornell.edu` **

In this paper we perform a preliminary investigation into the application of sampling-based search algorithms to satisfiability testing of propositional formulas in Conjunctive Normal Form (CNF). In particular, we adapt the Upper Confidence bounds applied to Trees (UCT) algorithm [5] which has been successfully used in many game playing programs including MoGo, one of the strongest computer Go players [3].

Rather than explore the search space in a depth-first fashion, in the style of DPLL [2], UCT repeatedly starts from the root node and incrementally builds a tree based on estimates of node utilities and node visit frequencies computed from previous iterations. In most implementations of UCT, the estimated utility of a new node is computed using Monte-Carlo methods, i.e., by generating random completions of the search (termed "playouts") and averaging their outcomes. This utility is revised each time the search revisits the node using the estimated values of the children. This technique is especially effective when no adequate heuristic is available to perform this value estimation task.

In this paper, we introduce and study an algorithm called UCTSAT that employs the UCT search control mechanism but replaces the playouts with a heuristic to estimate the initial utility of a node. The heuristic we use is the fraction of the total set of clauses that are satisfied by the partial assignment associated with the node; this fraction is computed after the application of unit propagation. While we do not expect UCTSAT to outperform the highly-optimized, state of the art SAT solvers (especially with respect to CPU time), we believe that the development of an algorithm based on a radically different search technique is important for at least two reasons: (a) the hardness of SAT instances is related to the algorithm used [1], and hence UCTSAT, which uses a different search strategy, can provide useful and new insights into the complexity of SAT instances; and (b) because such an algorithm can be useful when included in a portfolio of algorithms (see, for example, [6]) where very different solution techniques can help expand the range of applicability of the portfolio.

** Supported by NSF Expeditions in Computing award for Computational Sustainability, 0832782; NSF IIS grant 0514429; and IISI, Cornell Univ. (AFOSR grant FA9550-04-1-0151)

As such, we focus our efforts on understanding whether UCTSAT is capable of solving SAT instances using smaller search trees than DPLL. To simplify the comparisons, we contrast our algorithm against a no-frills implementation of DPLL. We set the exploration bias parameter in UCTSAT to 0 as this yielded the best performance on average. We also experimented with varying the number of atoms that UCTSAT assigned at a given node in the search tree and discovered that setting more than one atom at once hurt the performance of the algorithm.

We compared the performance of DPLL and UCTSAT on problem instances drawn from the SATLIB repository [4]. On uniform random 3-SAT and flat-graph coloring instances of various sizes, we found little difference in the sizes of the search trees constructed by the two algorithms. We believe that this is due to the unstructured nature of these instances — UCTSAT works well when each exploration of the tree yields information that can be successfully used in subsequent iterations. In instances drawn from real-world problems (namely, single-stuck-at-fault analysis problems) that exhibit structure, we discovered that UCTSAT constructs significantly smaller search trees than DPLL — this is illustrated in table 1.

Table 1. Average tree sizes (number of nodes) for SSA circuit fault analysis instances

Instance	**DPLL**	**UCTSAT**
ssa-7552-038	9183	173
ssa-7552-158	6564	134
ssa-7552-159	5513	147
ssa-7552-160	4095	164

References

1. A. Aguirre and M.Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *Principles and Practice of Constraint ProgrammingCP 2001*, pages 121–136. Springer, 2001.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
3. Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1537–1540. AAAI Press, 2008.
4. H.H. Hoos and T. Stützle. SAT2000: Highlights of Satisfiability Research in the year 2000, chapter SATLIB: An Online Resource for Research on SAT. *Frontiers in Artificial Intelligence and Applications. Kluwer Academic*, pages 283–292, 2000. Web site available at: http://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html.
5. L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.
6. L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008.