Volt: A Lazy Grounding Framework for Solving Very Large MaxSAT Instances

Ravi Mangal[1], Xin Zhang[1], Aditya V. Nori[2], and Mayur Naik[1]

[1] Georgia Institute of Technology
[2] Microsoft Research

Abstract. Very large MaxSAT instances, comprising 10^{20} clauses and beyond, commonly arise in a variety of domains. We present VOLT, a framework for solving such instances, using an iterative, lazy grounding approach. In each iteration, VOLT grounds a subset of clauses in the MaxSAT problem, and solves it using an off-the-shelf MaxSAT solver. VOLT provides a common ground to compare and contrast different lazy grounding approaches for solving large MaxSAT instances. We cast four diverse approaches from the literature on information retrieval and program analysis as instances of VOLT. We have implemented VOLT and evaluate its performance under different state-of-the-art MaxSAT solvers.

1 Introduction

MaxSAT solvers have made remarkable progress in performance over the last decade. Annual evaluations to assess the state-of-the-art in MaxSAT solvers began in 2006. These evaluations primarily focus on efficiently solving difficult MaxSAT instances. Due to several advances in solving such instances, many emerging problems in a variety of application domains are being cast as large MaxSAT instances, comprising 10^{20} clauses and beyond.[1]

Large MaxSAT instances pose scalability challenges to existing solvers. Researchers in other communities, notably statistical relational learning and program analysis, have proposed various *lazy grounding* techniques to solve such instances that arise in their application domains [4, 9, 15–17, 19]. The high-level idea underlying these techniques is to use an iterative counterexample-guided approach that, in each iteration, poses a subset of clauses in the original large MaxSAT instance to an off-the-shelf MaxSAT solver. The construction of this subset of clauses is guided by means of *counterexamples*—these are clauses in the original problem that are unsatisfied by the current solution.

This paper presents a formal framework VOLT for systematically studying the class of lazy grounding techniques. We show how diverse existing techniques in the literature are instances of our framework (Table 1 in Section 2). In doing so, VOLT provides the first setting that formally compares and clarifies the relationship between these various techniques.

[1]Throughout the paper, we slightly abuse terminology by using MaxSAT to refer to the *weighted partial maximum satisfiability* problem, which asks for a solution that satisfies all hard clauses and maximizes the sum of weights of satisfied soft clauses.

$$\begin{array}{rlrl}
\text{(relation)} & r \in \mathbf{R} & \text{(argument)} & a \in \mathbf{A} = \mathbf{V} \cup \mathbf{C} \\
\text{(constant)} & c \in \mathbf{C} & \text{(fact)} & t \in \mathbf{T} = \mathbf{R} \times \mathbf{A}^* \\
\text{(variable)} & v \in \mathbf{V} & \text{(ground fact)} & g \in \mathbf{G} = \mathbf{R} \times \mathbf{C}^* \\
\text{(valuation)} & \sigma \in \mathbf{V} \to \mathbf{C} & \text{(weight)} & w \in \mathbb{R}+ = (0, \infty]
\end{array}$$

$$\begin{array}{rll}
\text{(hard constraints)} & H ::= \{h_1, ..., h_n\}, & h ::= \bigwedge_{i=1}^{n} t_i \Rightarrow \bigvee_{i=1}^{m} t'_i \\
\text{(soft constraints)} & S ::= \{s_1, ..., s_n\}, & s ::= (h, w)
\end{array}$$

$$\text{(weighted constraints)} \quad C ::= (H, S) \qquad \text{(input, output)} \quad P, Q \subseteq \mathbf{G}$$

Fig. 1: Syntax of weighted EPR constraints.

We have implemented the VOLT framework and its instantiations. It allows any off-the-shelf MaxSAT solver to be used in each iteration of the lazy grounding process. We evaluate the performance of VOLT under different state-of-the-art MaxSAT solvers using a particular instantiation. Our evaluation shows that existing lazy grounding techniques can produce instances that are beyond the reach of exact MaxSAT solvers. This in turn leads these techniques to sacrifice optimality, soundness, or scalability. VOLT is only a starting point and seeks to motivate further advances in lazy grounding and MaxSAT solving.

2 VOLT: A Lazy Grounding Framework

The first step in solving large MaxSAT instances is to succinctly represent them. VOLT uses a variant of *effectively propositional logic* (EPR) [11]. Our variant operates on relations over finite domains and has an optional weight associated with each clause. Figure 1 shows the syntax of a weighted EPR formula C, which consists of a set of hard constraints and a set of soft constraints. For convenience in formulating problems, we augment C with an *input* P which defines a set of ground facts (extensional database or EDB). Its solution, *output* Q, defines a set of ground facts that are true (intensional database or IDB).

Weighted EPR formulae are grounded by instantiating the relations over all constants in their corresponding input domains. We presume a grounding procedure $[\![\cdot]\!]$ that grounds each constraint into a set of corresponding clauses. For example, $[\![h]\!] = \bigwedge_{\sigma} [\![h]\!]_{\sigma}$ grounds the hard constraint h by enumerating all possible groundings σ of variables to constants, yielding a different clause for each unique valuation to the variables in h. The ground clauses represent a MaxSAT problem which can be solved to produce a solution that satisfies all hard clauses and maximizes the sum of the weights of satisfied soft clauses.

Enumerating all possible valuations, called *full grounding*, does not scale to real-world problems. Our framework VOLT, described in Algorithm 1, uses lazy grounding to address this problem.[1] The framework is parametric in procedures INIT, GROUND, and DONE. Diverse lazy grounding algorithms in the literature can be derived by different instantiations of these three procedures.

[1]We assume that any input P is encoded as part of the hard constraints H. For brevity, we assume that the hard constraints H are satisfiable, allowing us to elide showing UNSAT as a possible alternative to output Q.

approach	$(\phi,\psi) := \textsc{Init}(H,S)$	$(\phi,\psi) := \textsc{Ground}(H,S,Q)$	$\textsc{Done}(\phi,\phi',\psi,$ $\psi',w,i)$
SoftCegar [4]	$\phi := \mathit{true}$ $\psi := [\![S]\!]$	$\phi := \bigwedge_{h \in H} \bigwedge \texttt{Violate}(h,Q)$ $\psi := \mathit{true}$	$\phi' = \mathit{true}$
Cutting Plane [16, 17]	$\phi := \mathit{true}$ $\psi := \mathit{true}$	$\phi := \bigwedge_{h \in H} \bigwedge \texttt{Violate}(h,Q)$ $\psi ::= \bigwedge_{(h,w) \in S} \bigwedge \{\, (\rho,w) \mid$ $\rho \in \texttt{Violate}(h,Q) \,\}$	clauses in ϕ',ψ' $\subseteq$ clauses in ϕ,ψ
Alchemy [9] Tuffy [15]	$\phi := \bigwedge_{h \in H} \bigwedge \texttt{Active}(h,P)$ $\psi := \bigwedge_{(h,w) \in S} \bigwedge \{\, (\rho,w) \mid$ $\rho \in \texttt{Active}(h,P) \,\}$	$\phi := \bigwedge_{h \in H} \bigwedge \texttt{Active}(h,Q)$ $\psi := \bigwedge_{(h,w) \in S} \bigwedge \{\, (\rho,w) \mid$ $\rho \in \texttt{Active}(h,Q) \,\}$	$i > \mathit{maxIters}$ $\vee\ w > \mathit{target}$
AbsRefine [19]	$\phi := (\bigoplus_{a \in A} a) \wedge \neg q$ $\psi := \bigwedge_{a \in A} (a,w)$	$\phi := \bigwedge \{\, \bigvee_{i=1}^{n} \neg [\![t_i]\!]_\sigma \vee [\![t_0]\!]_\sigma \mid$ $(\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H\ \wedge$ $\forall i \in [0..n] \colon [\![t_i]\!]_\sigma \in G \,\}$ $\psi := \mathit{true}$ $\mathit{where}\, G = \mathsf{lfp}\ \lambda G'.\ G' \cup$ $\{\, [\![t_0]\!]_\sigma \mid (\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H\ \wedge$ $\forall i \in [1..n] \colon [\![t_i]\!]_\sigma \in (G' \cup Q) \,\}$	$\phi' = \mathit{true}$

Table 1: Instantiating lazy grounding approaches with `VOLT` where
$\texttt{Active}(h,Q) = \{\, [\![h]\!]_\sigma \mid (h = \bigwedge_{i=1}^{n} t_i \Rightarrow \bigvee_{i=1}^{m} t'_i)\, \mathit{and}\, (\exists i : [\![t_i]\!]_\sigma \in Q \vee [\![t'_i]\!]_\sigma \in Q)\}$
and $\texttt{Violate}(h,Q) = \{\, [\![h]\!]_\sigma \mid Q \not\models [\![h]\!]_\sigma \}$.

Algorithm 1: `VOLT`

```
1:  input (H, S): Weighted constraints.
2:  output Q: Solution (assumes [[H]] is satisfiable).
3:  $(\phi, \psi) := \textsc{Init}(H, S)$
4:  $Q := \emptyset$;  $w := 0$;  $i := 0$
5:  loop
6:      $i := i + 1$
7:      $(\phi', \psi') := \textsc{Ground}(H, S, Q)$
8:      if $\textsc{Done}(\phi, \phi', \psi, \psi', w, i)$ return $Q$
9:      $(\phi, \psi) := (\phi \wedge \phi', \psi \wedge \psi')$
10:     $Q :=$ MaxSAT$(\phi, \psi)$
11:     $w :=$ Weight$(Q, \psi)$
```

In line 3, `VOLT` invokes the Init procedure to compute an initial set of hard clauses ϕ and soft clauses ψ. Next, `VOLT` enters the loop defined in lines 5–11. In each iteration of the loop, the algorithm keeps track of the previous solution Q, and the weight w of the solution Q by calling the `Weight` procedure that returns the sum of the weights of the soft clauses satisfied by Q. Initially, the solution is empty with weight zero (line 4). In line 7, `VOLT` invokes the Ground procedure to compute the set of hard clauses ϕ' and soft clauses ψ' to be grounded next. Typically, ϕ' and ψ' correspond to the set of hard and soft clauses violated by the previous solution Q. Next, in line 8, the algorithm checks if Q satisfies the terminating condition by invoking the Done procedure. If not, then in line 9, both sets of grounded clauses ϕ' and ψ' are added to the corresponding sets of grounded hard clauses ϕ and grounded soft clauses ψ respectively. In line 10, this updated set ϕ of hard clauses and set ψ of soft clauses are fed to the `MaxSAT` procedure to produce a new solution Q and its corresponding weight w.

Instantiations. Table 1 shows various lazy grounding algorithms from the literature as instantiations of the `VOLT` framework. SoftCegar [4] grounds all the

	brief description	# classes	# methods	bytecode (KB)	source (KLOC)
`antlr`	parser/translator generator	350	2,370	186	119
`luindex`	document indexing and search tool	619	3,732	235	170
`lusearch`	text indexing and search tool	640	3,923	250	178
`avrora`	microcontroller simulator/analyzer	1,544	6,247	325	178
`xalan`	XSLT processor to transform XML	903	6,053	354	285

Table 2: Benchmark program characteristics.

soft clauses upfront but lazily grounds the hard clauses. In each iteration, this approach grounds all the hard clauses violated by the current solution Q. Note that the `Violate` procedure takes as input a hard constraint h and a MaxSAT solution Q, and returns all grounded instances of h that are violated by Q. The algorithm terminates when no further hard clauses are violated.

Cutting Plane Inference (CPI) [16, 17], on the other hand, is lazier than SoftCegar and grounds no clauses upfront. In each iteration, both, hard and soft constraints are checked for violations, and any violated clauses are grounded. The algorithm terminates when no new constraints are violated.

A common approach, used in statistical relational learning tools like Alchemy [9] and Tuffy [15], relies on the observation that most ground facts are false in the final solution, and thereby most clauses are trivially true (since most clauses are Horn in these applications). An active ground fact is one that has a value of true. In each iteration, the clauses grounded are such that they contain at least one active fact as per the current solution. Initially, only the input facts P are considered active. This approach terminates after a fixed number of iterations or after the weight of the satisfied clauses is greater than a target weight.

Finally, the AbsRefine approach tackles a central problem in program analysis of efficiently finding a program abstraction that keeps only information relevant for proving properties of interest. In particular, this approach uses the <u>c</u>ounter<u>e</u>xample-<u>g</u>uided <u>a</u>bstraction <u>r</u>efinement (CEGAR) method [5] to efficiently find a suitable abstraction to prove a particular program property when the program analysis is expressed in Datalog. For such analyses, a set of hard Horn constraints expresses the analysis rules. A set of input ground facts A expresses the space of abstractions, with each ground fact in A representing a unique abstraction of cost w. The query q is a unique ground fact and proving the query implies having q as false in the final solution Q. The problem is to then find a solution with the lowest cost abstraction such that the query fact does not hold and all the analysis rules are satisfied. To lazily solve this problem, AbsRefine initially grounds hard constraints to ensure that in the final solution, the query fact q is false and only a single abstraction is true. Also, soft constraints specifying the abstraction costs are grounded upfront. Next, in the Ground procedure, AbsRefine grounds not only the hard clauses violated by the current solution, but uses the Horn nature of the constraints to ground additional clauses that would be necessarily grounded in future iterations. Specifically, it calls a Datalog solver, with the Horn constraints and the current solution Q as input, to compute the corresponding least fixed point (lfp) solution G. Any clause which has all of its ground facts in set G is added to the set ϕ' of hard clauses to be grounded. This approach terminates when no further hard clauses are grounded.

benchmark	solver	total time (min)	# iterations	avg solver time (secs)	grounded clauses ($\times 10^6$)	total clauses
`antlr`	CCLS2akms	-	1	-	7.8	$8.5\text{x}10^{35}$
	Eva500a	124	15	64.8	10.4	
	MaxHS	117	14	71.2	10.1	
	wmifumax	109	14	44.4	10.3	
	MSCG	-	5	22.2	7.9	
	WPM-2014-co	115	14	40.3	10.3	
`lusearch`	CCLS2akms	-	1	-	4.6	$1\text{x}10^{37}$
	Eva500a	127	14	78.6	14.7	
	MaxHS	144	14	123.1	19.1	
	wmifumax	119	15	51.2	10.2	
	MSCG	-	6	17	7.5	
	WPM-2014-co	196	14	332.7	16	
`luindex`	CCLS2akms	-	1	-	5.2	$4.5\text{x}10^{36}$
	Eva500a	172	23	45.2	5.9	
	MaxHS	161	22	52.5	5.9	
	wmifumax	169	23	34.1	6.9	
	MSCG	-	6	17.8	9	
	WPM-2014-co	216	21	226.3	5.7	
`avrora`	CCLS2akms	-	1	-	7	$4\text{x}10^{37}$
	Eva500a	-	4	80.2	17.6	
	MaxHS	-	13	136.7	15.5	
	wmifumax	-	13	115.1	9.1	
	MSCG	-	5	31.6	16.9	
	WPM-2014-co	-	12	2135.9	14.8	
`xalan`	CCLS2akms	-	1	-	10	$3.8\text{x}10^{39}$
	Eva500a	-	5	96.6	19.2	
	MaxHS	-	18	571.6	> 4290	
	wmifumax	-	14	78.7	42.9	
	MSCG	-	5	47.6	19.7	
	WPM-2014-co	-	12	505.7	44.3	

Table 3: Results of `VOLT` on program analysis benchmarks. Highlighted rows indicate cases where the MaxSAT solver used finishes successfully in all iterations.

Implementation. We have implemented the `VOLT` framework in Java. To compute the set of clauses to be grounded when the hard constraints are in the form of Horn clauses, as in [19], we use `bddbddb` [18], a Datalog solver. To compute `Violate`, the grounded constraints that are violated by a solution, we follow existing techniques [15, 16] and use SQL queries implemented using `PostgreSQL`.

3 Empirical Evaluation

We evaluate `VOLT` by instantiating it with the AbsRefine approach for the problem of finding suitable abstractions for proving safety of downcasts in five Java benchmark programs. A safe downcast is one that cannot fail because the object to which it is applied is guaranteed to be a subtype of the target type. Our experiments were done using a Linux server with 64GB RAM and 3.0GHz CPUs.

Table 2 shows statistics of the five Java programs (`antlr`, `lusearch`, `luindex`, `avrora`, `xalan`) from the DaCapo suite [3], each comprising 119–285 thousand lines of code. Note that these are fairly large real-world programs and allow us to study the limits of `VOLT`'s scalability with existing MaxSAT solvers.

We use complete weighted partial MaxSAT solvers that were available from the top performers in Random, Crafted and Industrial categories of the 9th MaxSAT Evaluation [1]. In particular, the solvers we use are CCLS2akms [10, 12], Eva500a [14], MaxHS [6], wmifumax [7], MSCG [8, 13] and WPM-2014-co [2].

Table 3 summarizes the results of running `VOLT` with the different MaxSAT solvers on our benchmarks. The 'total time' column shows the total running time of `VOLT`. A '-' indicates an incomplete run either because the underlying MaxSAT solver crashed or timed out (ran for >18000 seconds) on a particular instance. The next column '# iterations' provides the number of iterations needed by the lazy `VOLT` algorithm. In cases where `VOLT` did not terminate, this indicates the iteration in which the MaxSAT solver failed. The 'avg solver time' column provides the average time spent by the MaxSAT solver in solving an instance. It does not include the time spent by the solver on a failed run. The 'ground clauses' column provides the distinct number of clauses grounded by `VOLT` in the process of solving the weighted constraints. In other words, it indicates the size of the problem fed to the MaxSAT solver in the final iteration of the `VOLT` algorithm. The 'total clauses' column reports the theoretical upper bound for the number of ground clauses if all the constraints were grounded naively.

The evaluation results indicate that the MaxSAT instances generated by `VOLT` are many orders of magnitude smaller than the full MaxSAT instance. It is clear from these numbers that any approach attempting to tackle problems of this scale needs to employ lazy techniques for solving such instances. On the other hand, we also observe that many of the solvers are unable to solve these relatively smaller instances generated by `VOLT`. For example, `VOLT` does not terminate using any of the solvers for **`avrora`** and **`xalan`**.

The lack of scalability of existing solvers on the larger MaxSAT instances from our evaluation suggests the need for further research in both, lazy grounding approaches as well as MaxSAT solvers. A possible next step is to make lazy grounding more demand-driven. This is motivated by the fact many applications including ours are only concerned with the value of a particular variable instead of the entire MaxSAT solution. We intend to make the MaxSAT instances generated in our evaluation publicly available to facilitate future research.

4 Conclusion

Emerging problems in fields like statistical relational learning and program analysis are being cast as very large MaxSAT instances. Researchers in these areas have developed approaches that lazily ground weighted EPR formulae to solve such instances. We have presented a framework `VOLT` that captures the essence of lazy grounding techniques in the literature. `VOLT` not only allows to formally compare and clarify the relationship between diverse lazy grounding techniques but also enables to empirically evaluate different MaxSAT solvers. We hope that `VOLT` will stimulate further advances in lazy grounding and MaxSAT solving.

Acknowledgements. We thank the anonymous referees for helpful feedback. This work was supported by DARPA contract #FA8750-15-2-0009 and by NSF awards #1253867 and #1526270.

References

1. http://www.maxsat.udl.cat/14/index.html
2. http://web.udl.es/usuaris/q4374304/
3. Blackburn, S.M., Garner, R., Hoffman, C., Khan, A.M., McKinley, K.S., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A., Jump, M., Lee, H., Moss, J.E.B., Phansalkar, A., Stefanović, D., VanDrunen, T., von Dincklage, D., Wiedermann, B.: The DaCapo benchmarks: Java benchmarking development and analysis. In: OOPSLA (2006)
4. Chaganty, A., Lal, A., Nori, A., Rajamani, S.: Combining relational learning with SMT solvers using CEGAR. In: CAV (2013)
5. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. JACM 50(5) (2003)
6. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: CP (2013)
7. Janota, M.: MiFuMax — a literate MaxSAT solver (2013)
8. Joao Marques-Silva and Alexey Ignatiev and António Morgado: MSCG - Maximum Satisfiability: a Core-Guided approach (2014)
9. Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Domingos, P.: The alchemy system for statistical relational AI. Tech. rep., Department of Computer Science and Engineering, University of Washington, Seattle, WA (2007), http://alchemy.cs.washington.edu
10. Kügel, A.: Improved exact solver for the weighted MAX-SAT problem. In: POS-10. Pragmatics of SAT (2010)
11. Lewis, H.R.: Complexity results for classes of quantificational formulas. J. Comput. Syst. Sci. 21(3), 317–353 (1980)
12. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Computers 64(7), 1830–1843 (2015)
13. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided maxsat with soft cardinality constraints. In: CP (2014)
14. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: AAAI (2014)
15. Niu, F., Ré, C., Doan, A., Shavlik, J.W.: Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. In: VLDB (2011)
16. Noessner, J., Niepert, M., Stuckenschmidt, H.: RockIt: Exploiting parallelism and symmetry for MAP inference in statistical relational models. In: AAAI (2013)
17. Riedel, S.: Improving the accuracy and efficiency of MAP inference for Markov Logic. In: UAI (2008)
18. Whaley, J., Lam, M.: Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In: PLDI (2004)
19. Zhang, X., Mangal, R., Grigore, R., Naik, M., Yang, H.: On abstraction refinement for program analyses in Datalog. In: PLDI (2014)