GraphSAT Input Language

N. N.

March 18, 2017

Abstract

This document describes the GraphSAT solver that implements Boolean satisfiability testing combined with graph constraints.

Contents

1 Introduction

GraphSAT is a SAT modulo Graphs (..., 2017) solver. It solves Boolean satisfiability problems extended with graph constraints for acyclicity and s-t-reachability.

The syntax is based on the DIMACS CNF format for pure Boolean satisfiability problems. Additionally, a graph consisting of nodes and arcs can be included in an input file, with the requirement that the graph is acyclicity, or that certain pairs of nodes are reachable or unreachable from each other. GraphSAT then finds a truth-value assignment to all of the variables so that all clauses are true, and the graph induced by the arc variables assigned *true* are acyclic, and for pairs of nodes for which the corresponding *reachability* or *unreachability* variable is assigned *true*, the second node is respectively reachable or unreachable from the first.

GraphSAT has been designed with a broad range of applications in mind, including knowledge representation languages such as Answer Set Programming and logics for Inductive Definitions, as well as various other discrete combinatorial problems that can be expressed in terms of graphs.

Implementation techniques behind the solver are described in conference publications [3], similarly to applications [2, 1].

2 Syntax

The syntax includes the DIMACS CNF format. In the beginning of the input file there is a line indicating the number of atomic proposotions and the number of clauses, for example

```
p cnf 6 3
```

that says that the number of atomic propositions is 6 (numbered from 1 to 6), and the number of clauses is 3.

2.1 Example

The extension of the DIMACS syntax is achieved by supplying information about the graphs and graph constraints in the *comments* of the DIMACS syntax. Each comment line in the DIMACS syntax starts with the letter "c", and conventional SAT solvers ignore all the characters after "c" until the end of the line.

Here is an example of a SAT modulo Graphs problem, consisting of three clauses and an acyclicity constraint on a 4-node graph with 6 arcs. The 6 atomic propositions are associated with the 6 arcs: an arc proposition being true means that the arc is included in the graph. All *false* arcs, as well as arcs that are not mentioned, are absent from the graph.

```
p cnf 6 3
c graph 4
c node 0 1
c node 1 2
c node 2 2
c node 3 1
c arc 1 0 1
c arc 2 1 2
c arc 3 1 3
c arc 4 2 3
c arc 5 2 0
c arc 6 3 0
c endgraph
c acyc
c
c In this example the arcs are a cycle 0->1->2->3 with diagnonals 1->3 2->0.
c   EXAMPLE 0 -> 1
c   EXAMPLE ^    |
c   EXAMPLE |    V
```

```
c    EXAMPLE 3 <- 2
c
c
1  2  0
3  4  0
5  6  0
```

Given this problem instance, GraphSAT tries to assign either value *true* or *false* to each atomic proposition, to satisfy the clauses and the graph constraints. In this case, the clauses represent disjunctions of two arc variables,

$$A_{0,1} \vee A_{1,2}$$
$$A_{1,3} \vee A_{2,3}$$
$$A_{2,0} \vee A_{3,0}$$

Parts of GraphSAT input are described in the next sections.

2.2 Representing Graphs

Every graph specification consists of two parts, the nodes, and the arcs.
The graph specification starts with

p graph *N*

where N is an integer indicating the number of nodes in the graph. The nodes are numbered from 0 to $N-1$. This is followed by N declarations of the nodes' *arities*, that is, the number of arcs starting from the node in question:

c node *i arity*

where i is the index of the node in the range 0 to $N-1$, and *arity* is an integer ≥ 0.
This is followed by declarations of arcs of the form

c arc *v s t*

where v is an index of an atomic proposition, and s and t are indices of nodes in the range 0 to $N-1$, indicating that the atomic proposition v being true means that there is an arc from s to t in the graph. The number of arc declarations with s as the source node must match the arity indicated in the node declaration.
The graph specification ends with the following line.

c endgraph

2.3 Representing Acyclicity Constraints

For the specified graph the acyclicity constraint is required to hold by simply specifying the following.

c acyc

This means that GraphSAT considers only those truth-value assignments to all atomic propositions satisfiable that, in addition to satisfying all clauses, correspond to an acyclic graph formed from those arcs for which the corresponding atomic proposition is *true*.

2.4 Representing Reachability Constraints

Reachability constraints are specified as follows.

c greachable *s m* t_1 l_1 t_2 l_2 $\cdots$ t_m l_m

Here literals l_i are associated with the presence of a path from s to t_i in the graph, for every $i \in \{1, \ldots, m\}$. Formally, if l_i is true, then there is a path from s to t_i.

Unreachability constraints are specified as

c gnonreach *m* s_1 t_1 l_1 s_2 t_2 l_2 $\cdots$ s_m t_m l_m

which similarly associate literals l_i with the *absence* of a path from s_i to t_i in the graph, for every $i \in \{1, \ldots, m\}$. Formally, if l_i is true, then there is *no path* from s_i to t_i.

An arbitarary number of reachability and unreachability constraints may be given, and they can be combined with the acyclicity constraint.

Reachability from one node s to other nodes is preferably given on one line, as the constraint that contains all target nodes of paths from s is more powerful than separate constraints from s to some subsets of those nodes.

The motivation for having both unreachability and reachability is the need to express *don't care*: a negated reachability does not require that a node is reachable from another, it only states that the node is not required to be reached from another.

References

[1] K. Chatterjee, M. Chmelik, and J. Davies. A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 3225–3232. AAAI Press, 2016.

[2] M. Gebser, T. Janhunen, and J. Rintanen. Answer set programming by SAT modulo acyclicity. In *ECAI 2014. Proceedings of the 21st European Conference on Artificial Intelligence*, pages 351–356. IOS Press, 2014.

[3] M. Gebser, T. Janhunen, and J. Rintanen. SAT modulo graphs: Acyclicity. In *Logics in Artificial Intelligence, 14th European Conference, JELIA 2014, September 2014, Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 2014.