Proceedings of the 11th International Conference on Applied Informatics
Eger, Hungary, January 29–31, 2020, published at http://ceur-ws.org

The BWConverter Toolchain: An Incomplete Way to Convert SAT Problems into Directed Graphs*

Tamás Balla, Csaba Biró, Gábor Kusper

Eszterházy Károly University, Faculty of Informatics
{balla.tamas,biro.csaba,kusper.gabor}@uni-eszterhazy.hu

Abstract

In our previous works we showed, how to convert a directed graph into a SAT problem. We showed that if the directed graph is strongly connected, then the converted SAT problem is Black-and-White. A SAT problem is Black-and-White if it has exactly two solutions, the black assignment, where each variable is false, and the white one, where each variable is true. In this paper, we study the other way: How to convert a SAT problem into a directed graph. This direction seems to be very difficult, so we introduce the BWConverter toolchain which helps us to create Black-and-White SAT instances, and convert them to directed graphs using different strategies. The toolchain consist of two tools: the BWConverter and the NPP tool. As a first step, one has to use the BWConverter tool, which creates a Black-and-White SAT problem from any UNSAT problems. The resulting SAT problem is Black-and-White if the input was UNSAT . As a second step, we generate a directed graph using our `NPP` tool. This tool generates edges from clauses which contain either exactly one positive literal, or exactly one negative one using the observations of our previous models.

Keywords: SAT, Strongly Connected Directed Graphs, BWConverter

MSC: 03B05, 05C20

1. Introduction

The propositional satisfiability problem, for short the SAT problem, is the problem to find an assignment to the the Boolean variables of the input formula which

*This research was supported by the grant EFOP-3.6.1-16-2016-00001 “Complex improvement of research capacities and services at Eszterházy Károly University”.

makes it true. The formula is given in conjunctive normal form (CNF) which is represented as a clause set. If each clause consists of at most k literals, then it is a k-SAT problem. We know that the 2-SAT problem is solvable in linear time [1]. On the other hand 3-SAT is NP-complete [7]. So, there is a big the gap between the easiness of 2-SAT and hardness of 3-SAT. We know some classes which are between them:

Horn SAT is the restriction to instances where each clause has at most one positive literal. Horn SAT is solvable in linear time [9], as are a number of generalizations such as renamable Horn SAT [2], extended Horn SAT [6] and q-Horn SAT [5].

The hierarchy of *tractable* satisfiability problems [8], which is based on Horn SAT and 2-SAT, is solvable in polynomial time. An instance on the k-th level of the hierarchy is solvable in $O(n^{k+1})$ time.

r,r-SAT, where r,s-SAT is the class of problems in which every clause has exactly r literals and every variable has at most s occurrences. All r,r-SAT problems are satisfiable in polynomial time [15].

A formula is SLUR (Single Lookahead Unit Resolution) *solvable* if, for all possible sequences of selected variables, algorithm SLUR does not give up. Algorithm SLUR is a non-deterministic algorithm based on unit propagation. It eventually gives up the search if it starts with, or creates, an unsatisfiable formula with no unit clauses. The class of SLUR solvable formulae was developed as a generalization including Horn SAT, renamable Horn SAT, extended Horn SAT, and the class of CC-balanced formulae [14].

In this paper we restrict the general SAT problem such that each clause should contain at least one positive and one negative literal. We study how can we generate a directed graph from this kind of problem. Our goal is to give tools which help to do this work.

In our previous works we showed, how to convert a directed graph into a SAT problem [3, 12]. We have presented two SAT solvers [4, 10, 11] and introduced several models [13]. We showed that if the directed graph is strongly connected, then the converted SAT problem is Black-and-White.

A SAT problem is Black-and-White if it has exactly two solutions, the black assignment, where each variable is false, and the white one, where each variable is true. In this paper, we study the other way: How to convert a SAT problem into a directed graph.

This direction seems to be very difficult, because if we would have a complete converter from SAT to directed graphs, then we could use our previous result to convert it to 2-SAT, i.e., we would have a SAT to 2-SAT converter. What we present is just an incomplete converter, so we do not attack the $\mathbf{P} = \mathbf{NP}$ question.

The BWConverter **toolchain** contains two tools: the BWConverter and the NPP tool. As a first step, one has to use the BWConverter tool, which creates a Black-and-White SAT problem from UNSAT problems.

The BWConverter makes sure that its output has the same set of solutions as the input, except the black and the white assignment. To do so, we have implemented several algorithms.

The most interesting one introduces two new variables, x and y; then it adds $(\neg y)$ to each clause which contains only positive literals; then it adds (x) to each clause which contains only negative ones; finally, it adds $(\neg x \vee y)$ as a new clause to the output. The resulting `SAT` problem is `Black-and-White` if the input was `UNSAT`.

As a second step, we generate a directed graph using our `NPP` tool. This tool generates edges from clauses which contain either exactly one positive literal, or exactly one negative one using the observations of our previous models.

From example from the clause $(\neg A \vee B \vee C)$ it generates the edges: $A \rightarrow B$ with weight 50%, and $A \rightarrow C$ with weight 50%. From the clause $(\neg A \vee \neg B \vee C)$ it generates the edges (the weights are in ()): $A \rightarrow C$ (50%), $B \rightarrow C$ (50%), $A \rightarrow B$ (50%), and $B \rightarrow A$ (50%). This conversation is too eager, because it creates a strongly connected graph even if the input was not a `Black-and-White SAT` problem. As a solution, we attach to each edge a weight number which indicates, how likely is that the edge is important. Then we iteratively remove the edges with the smallest and / or with the highest weight until we would have a non strongly connected directed graph in the next iteration. We call this directed graph to be the directed graph representation of the input SAT problem.

We studied where this state transition point is in case of unsatisfiable random 3-`SAT` problems.

2. The BWConverter tool

The `BWConverter` tool can create a `Black-and-White` SAT problem from an `UNSAT` problem. It has 3 main switches:

```
BWConverter -nooverlap num  |  BWConverter -n num
BWConverter -overlap num  |  BWConverter -o num
BWConverter -pigeonholeeater  |  BWConverter -p
```

After the first two ones we give a number, which shows how smart do we use these strategies. If the number is 0, then we do not use other input clauses to generate less clauses. 1 means that we use always the same clause to generate less clauses. 2 means that we use one of the best clause in order to generate as few clauses as possible.

They remove the black and the white clauses from each input clause. We introduce them by examples. Let us assume that the input has 6 variables: from a to f, these clauses was already processed: $\{a, d, e\}$, $\{b, c, d\}$, and the current clause is $\{a, b, c\}$. Then:

`-n 0 generates:` $\{a, b, c, \neg d\}$, $\{a, b, c, d, \neg e\}$, $\{a, b, c, d, e, \neg f\}$.
`-n 1 generates:` $\{a, b, c, \neg d\}$, $\{a, b, c, d, \neg e\}$.
`-n 2 generates:` $\{a, b, c, \neg d\}$.
`-o 0 generates:` $\{a, b, c, \neg d\}$, $\{a, b, c, \neg e\}$, $\{a, b, c, \neg f\}$.
`-o 1 generates:` $\{a, b, c, \neg d\}$, $\{a, b, c, \neg e\}$.
`-o 2 generates:` $\{a, b, c, \neg d\}$.

Note, that in case of `-n` there is no such full-length clause which is subsumed by the generated clauses, so they do not overlap.

`-p` introduces two new variables x and y; then it adds $(\neg y)$ to each clause which contains only positive literals; then it adds (x) to each clause which contains only negative ones; finally, it adds $(\neg x \vee y)$ as a new clause to the output. The resulting `SAT` problem is `Black-and-White` if the input was `UNSAT`.

An example for `-p` is the following: Let us assume that the input is $(a \vee b) \wedge (c \vee d) \wedge (e \vee f) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg e) \wedge (\neg c \vee \neg e) \wedge (\neg b \vee \neg d) \wedge (\neg b \vee \neg f) \wedge (\neg d \vee \neg f)$, which is actually the Pigeon Hole problem[1] with 3 pigeons and 2 holes. Then the output is: $(\neg y \vee a \vee b) \wedge (\neg y \vee c \vee d) \wedge (\neg y \vee e \vee f) \wedge (x \vee \neg a \vee \neg c) \wedge (x \vee \neg a \vee \neg e) \wedge (x \vee \neg c \vee \neg e) \wedge (x \vee \neg b \vee \neg d) \wedge (x \vee \neg b \vee \neg f) \wedge (x \vee \neg d \vee \neg f) \wedge (\neg x \vee y)$.

There are some more minor switches:
`-nob` means that the output will not subsume the black clause.
`-now` means that the output will not subsume the white clause.
`-bw` means that the generated output will be a black-and-what SAT problem, if the input was an UNSAT problem. This is the same as `-nob -now`, and this is a default switch.
`-same` means that the output problem will have the same set of solutions as the input, because it adds the black and the white clauses at the end of the output.
`-help` prints the help.

3. The NPP Tool

The `NPP` tool work as follows: it reads a `CNF` file in `DIMACS` format. For each clause C it adds the edges (n, p, w) to the graph, where n is a negative literal, p is a positive literal, and w is $1/pos$, where pos is the number of positive literals in C. It also adds the edges (n_1, n_2, nw) to the graph, where n_1 and n_2 are different negative literals, and $nw = 1/neg$, where neg is the number of positive literals in C.

For example if $C = \{\neg a, \neg b, \neg c, d, e\}$, then the following edges is added to the graph: $(a, d, 0.5)$, $(b, d, 0.5)$, $(c, d, 0.5)$, $(a, e, 0.5)$, $(b, e, 0.5)$, $(c, e, 0.5)$, $(a, b, 0.33)$, $(a, c, 0.33)$, $(b, a, 0.33)$, $(b, c, 0.33)$, $(c, a, 0.33)$, $(c, b, 0.33)$.

The tool has the following switches:

```
NPP -autominlimit  |  NPP -a
NPP -automaxlimit  |  NPP -A
NPP -setminlimit num  |  NPP -s num
NPP -setmaxlimit num  |  NPP -S num
NPP -normalized |  NPP -N
```

`-a` finds the biggest limit, such that by deleting edges with weight $\leq$ limit the directed graph is still strongly connected. *`-A`* finds the lowest limit, such that by deleting edges with weight $\geq$ limit the directed graph is still strongly connected. *`-s`* sets the lower limit to num, i.e., it deletes all edges where the weight $\leq$ num.

[1]`https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/PHOLE/descr.html`

-*S* sets the upper limit to num, i.e., it deletes all edges where the weight ≥ num.

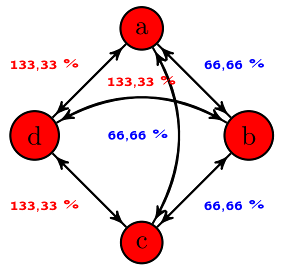
The most typical usage is ***NPP -a -A*** which finds automatically both limits.

The output of the NNP tool is the adjency matrix of the constructed and modified graph. If $M_{ij} = -1$ that means that there is no directed vertex from i to j, otherwise the value is the weight of vertex from i to j. If we use -N switch the tool will normalize the weight values to $[0; 1]$ interval before it removes any edge from graph.

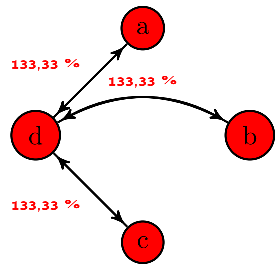
4. An Example

The initial **UNSAT** `3-SAT` formula let be as follows: $(\neg a \vee b \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$.

After running the `BWConverter tool`, the modified formula is as follows: $(\neg a \vee b \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c \vee \neg d) \wedge (\neg a \vee \neg b \vee \neg c \vee d)$. Note, that this is not **UNSAT** any more, it is `Black-and-White`.



The generated graph from the formula



Result of using the tool with `-a -A (NPP -a -A)`

5. Usage and Conclusion

The toolchain can be found here: `http://fmv.ektf.hu/tools.html`. It consists of two tools: `BWConverter`, and `NPP`. Its typical usage is the following:

- `java BWConverter -o 2 unsat.cnf bw.cnf`
- `java NPP -a -A -N bw.cnf`

This work was an inner project, because we used to need a tool to test our theories. For us the toolchain was very useful, so we decided to publish it. As future work we should prove the lemmas what we found by the help of this toolchain.

References

[1] B. Aspvall, M. F. Plass, R. E. Tarjan, *A Linear-Time Algorithm For Testing The Truth Of Certain Quantified Boolean Formulas*, Information Processing Letters, pp. 121–123, 1979.

[2] B. Aspvall, *Recognizing Disguised NR(1) Instances of the Satisfiability Problem*, J. of Algorithms, 1, pp. 97–103, 1980.

[3] Cs. Biró, G. Kusper, *Equivalence of Strongly Connected Graphs and Black-and-White 2-SAT Problems*, Miskolc Mathematical Notes, Vol. 19, No. 2, pp. 755-768, 2018.

[4] Cs. Biró and G. Kusper, *BaW 1.0 - A Problem Specific SAT Solver for Effective Strong Connectivity Testing in Sparse Directed Graphs*, CINTI-2018, pp. 137-142, 2018.

[5] E. Boros, P. L. Hammer, X. Sun, *Recognition of q-Horn Formulae in Linear Time*. Discrete Applied Mathematics, v. 55, pp. 1–13, 1994.

[6] V. Chandru, J. Hooker, *Extended Horn Sets in Propositional Logic*, J. of the ACM, 38(1), pp. 205–221, 1991.

[7] S. A. Cook, *The Complexity of Theorem-Proving Procedures*, Proc of STOC'71, pp. 151–158, 1971.

[8] M. Dalal, D. W. Etherington, *A Hierarchy of Tractable Satisfiability Problems.* Information Processing Letters, v.44, pp. 173–180, 1992.

[9] W. F. Dowling, J. H. Gallier, *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae.* J. of Logic Programming, 1(3), pp. 267–284, 1984.

[10] G. Kusper and Cs. Biró, Solving SAT by an Iterative Version of the Inclusion-Exclusion Principle, *SYNASC 2015*, IEEE Computer Society Press pp. 189–190, 2015.

[11] G. Kusper, Cs. Biró, Gy. B. Iszály, *SAT solving by CSFLOC, the next generation of full-length clause counting algorithms*, Future IoT 2018, pp.1–9, 2018.

[12] G. Kusper, Cs. Biró, *Convert a Strongly Connected Directed Graph to a Black-and-White 3-SAT Problem by the Balatonboglár Model*, submitted to Theory of Computing, 17 pages, arrived on 24.08.2019, status: under review.

[13] G.Kusper, Cs. Biró, T. Balla, *Representing Directed Graphs as 3-SAT Problems Using The Simplified Balatonboglár Model*, The 11th International Conference on Applied Informatics to be held in Eger, Hungary January 29–31, 2020.

[14] J. S. Schlipf, F. Annexstein, J. Franco, R. P. Swaminathan, *On finding solutions for extended Horn formulas*, Information Processing Letters, v. 54, pp. 133–137, 1995.

[15] C. A. Tovey, *A Simplified NP-complete Satisfiability Problem*, Discrete Applied Mathematics, v. 8, pp. 85–89, 1984.