

On the Violation of Circuits in Decomposable Negation Normal Form

Lucas Bordeaux¹ and Nina Narodytska²

¹ Microsoft Research Cambridge, UK
lucasb@microsoft.com

² NICTA and University of NSW, Sydney, Australia
ninan@cse.unsw.edu.au

Abstract. Local Search approaches to constraint satisfaction are based on the ability to compute *violation* scores. These estimate 'how far' a given assignment is from satisfying the constraints, and aim at guiding the search towards assignments whose violation will, ultimately, be null. We study the computation of violation functions for Boolean predicates encoded in *Decomposable Negation Normal Form*, an important class of logical circuits that encode interesting constraints. We show that for these circuits an elegant and efficient violation function first proposed by Z. Stachniak is "ideal" in a certain sense. We discuss more broadly the notion of "ideal" violation and discuss the implications of this result.

1 Context and Summary of the Results

Local Search. Local Search (LS) is one of the most useful components in every constraint satisfaction and optimization toolbox. Two recent developments in stochastic local search that relate to this paper are the following:

- In the Constraint Programming (CP) community, Constraint-Based Local Search [19] has emerged as a formalism that integrates LS within a CP approach, clearly separating the *modeling* of a problem from its resolution.
- In the SAT community, most of the recent work on stochastic local search [17,13,14,18,7,8,15,2,3] is based on *circuits*, as opposed to the traditional Conjunctive Normal Form (CNF) representation. This is based on the belief that CNF somehow loses 'structure' compared to a natural modeling with circuits, and that this structure may be needed by the Local Search.

Common to both approaches is the idea to compute, for any assignment, a *violation* (a.k.a. *penalty*, a.k.a. *clash* function) that estimates 'by how much' it violates the constraint.

Violations in SAT. In SAT, the question is to assign a violation to arbitrary Boolean circuits. An elegant scheme for doing so was proposed by [13], following ideas initially proposed by [17] and used further in e.g. [18,2,3]. The basic idea of [17] is that the violation of an And gate is the *sum* of the violations of its inputs,

while for an Or gate it is their *minimum*. This follows the obvious intuition, e.g. to satisfy an And gate we have to cumulate the work needed to make both inputs true. To find the violation value of a Boolean circuit we start from the inputs of the circuit and compute the value of the output gate using the sum and minimum rules for And and Or gates, respectively. [13] modified this definition to make it fully *symmetric*¹: the trick is to use an integer violation score that goes *strictly* positive if the constraint is unsatisfied, and *strictly* negative if it is satisfied (i.e. the violation is always non-zero). The violation of a Not gate is now defined as $-[\text{violation of the input}]$.

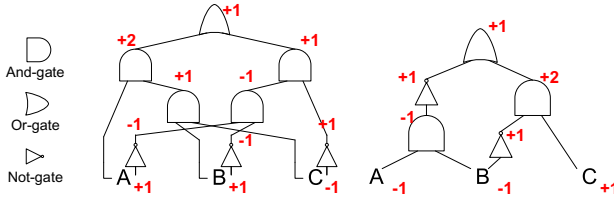


Fig. 1. Violations of circuits, arbitrary (left) and in Negation Normal Form (right)

Fig. 1 illustrates the computation of this violation; the exact definition of the rules we apply for each gate are given in Section 3. In the left-hand side example, the current assignment gives the values *true*, *true*, and *false*, respectively, to the inputs *A*, *B* and *C*. Seen as violations *true* translates to -1 and *false* to +1.

In this paper, our motivation is to better understand violation functions. These functions are usually seen as purely heuristic indicators in the literature, and little is said about the properties they should respect. For instance, the literature usually defines the violation of a constraint $\text{ALLDIFFERENT}(x_1, \dots, x_n)$ as $\sum_{a \in D} \max(0, \#\{h : v(x_h) = a\} - 1)$, where D is the domain of the variables, $v(x_h)$ is the value assigned in the current state v to variable x_h , and $\#S$ denotes the cardinality of set S . This definition seems sensible and satisfies the basic correctness property of being > 0 iff the constraint is indeed violated, but so do many alternative definitions—why this one? In the Constraint Programming literature we have a good answer to the question *what is the ideal propagation that can be reached for a constraint?* through the notion of domain-consistency (or “Generalized Arc-Consistency”). From the OR literature, we have a good answer to the question *what is the ideal relaxation of a constraint?* through the notion of convex Hull. We are looking for a similar notion for the third class of constraint solving represented by local search:

What is an ideal violation function for a constraint?

¹ This definition is better-behaved, in particular w.r.t. duality: for instance the violations of $\neg(A \wedge B)$ and of $\neg A \vee \neg B$ are the same. For NNF formulae the initial formulation [17] and the symmetric one [13] are essentially equivalent.

Section 3 proposes an answer to this question. To be fair, although the literature has not asked the question explicitly, partial answers have been pointed to by some authors. We identify ‘ideal’ properties of violation functions, and observe that it is NP-complete to compute an ideal violation for arbitrary predicates.

Our main contribution is to show that such ideal violation functions can indeed be obtained, in a generic way, for an important class of Boolean circuits. Specifically the violation function described before is shown to be ideal for circuits in Negation Normal Form that have the well-known *Decomposability* property [4]. (Such a circuit is given in Fig. 1, right-hand side). Circuits in Decomposable NNF, or DNNF, are one of the most powerful ‘knowledge compilation languages’ [5] used in AI. Some properties of this language are: (1) it is *complete* in that it can represent any Boolean function (as for any class of circuits, most Boolean functions nevertheless have an exponential size representation); (2) important reasoning tasks including clausal entailment and model enumeration can be performed in time polynomial in the DNNF size; (3) it generalizes other widely used representations such as OBDDs; (4) it allows to concisely represent a large number of constraints, in particular grammar constraints [16,11].

The violation function of [17,13] seems like a “good” definition of violation for circuits but its properties were not previously well-understood. Our contribution, on the SAT side, is to show that it has indeed strong properties w.r.t. an important and natural class of circuits. We also discuss how, on the CP side, this result gives ideal representations of a number of constraints.

2 Basic Material

Local Minima. We denote by $\{0,1\}^n$ the set of Boolean tuples of dimension n . The n components of a tuple t are written $\langle t_1 \dots t_n \rangle$. We denote by $\mathcal{D}(t, t')$ the Hamming distance between two tuples $t, t' \in \{0,1\}^n$, which is equal to the number of positions at which the corresponding symbols in these tuples are distinct. We consider functions of signature $\{0,1\}^n \mapsto \mathbb{Z}$ or occasionally $\{0,1\}^n \mapsto \{0,1\}$, in which case the function is called *predicate*. A key LS notion is:

Definition 1. (*Local Minimum*) A tuple t is a local minimum of a function $\mathcal{F}$ if any tuple t' at Hamming distance 1 from t is such that $\mathcal{F}(t') \geq \mathcal{F}(t)$.

A straightforward greedy minimization algorithm² converges to a local minimum of any function $\mathcal{F}$ in time $\mathcal{O}(n^2KR)$, where R is the range of $\mathcal{F}$, defined as $\max_{t \in \{0,1\}^n} \mathcal{F}(t) - \min_{t \in \{0,1\}^n} \mathcal{F}(t)$, and K is the cost of evaluating the violation. It is important to note that the range R can be exponentially large w.r.t. n . There exists no local minimization algorithm of complexity directly polynomial in n and m unless the complexity class PLS is equal to P [10,9].

² A precise description of this algorithm is found in the Appendix.

Circuits. We focus on circuits with binary gates for notational simplicity (the definitions, techniques and results can be easily adapted to arbitrary gates):

Definition 2. (*Circuit*) A circuit with n inputs $x_1 \dots x_n$ is an ordered sequence of gates $\langle g_1 \dots g_m \rangle$ such that $m \geq n$ and:

- The n first gates are the inputs: $g_k := x_k$, for $k \in 1 \dots n$;
- Each gate g_k with $k > n$ is either:
 - A negation gate of the form $\neg g_i$ with $i < k$;
 - A conjunction gate of the form $g_i \wedge g_j$ with $i, j < k$;
 - A disjunction gate of the form $g_i \vee g_j$ with $i, j < k$.
- The last gate g_m is called output gate.

The predicate encoded by a circuit is noted $\mathcal{E}$ (for Boolean evaluation) is defined as follows. Given a tuple of Boolean values $t = \langle t_1 \dots t_n \rangle$ for the inputs, a value $\mathcal{E}_{g_k}(t)$ is assigned to each gate g_k : for each input $k \in 1 \dots n$, $\mathcal{E}_{g_k}(t) := t_k$; for non-input gates we apply the definition of the gate, for instance $\mathcal{E}_{g_i \wedge g_j}(t) := \mathcal{E}_{g_i}(t) \wedge \mathcal{E}_{g_j}(t)$. The value $\mathcal{E}(t)$ of the circuit is defined as the value of the output gate $\mathcal{E}_{g_m}(t)$. A tuple t is a *solution* if $\mathcal{E}(t) = 1$.

Definition 3. (*Dependencies*) When we have a gate g_k of the form $\neg g_i$, $g_i \wedge g_j$ or $g_i \vee g_j$, we say that g_k directly depends on g_i and g_j , which we note $g_i \rightsquigarrow g_k$. We say that g_k depends on g_i if there is a chain of direct dependencies $g_i \rightsquigarrow \dots \rightsquigarrow g_k$. The set of inputs a gate g_k depends on is called $\text{Vars}(g_k)$.

Definition 4. (*Special Forms of Circuit*) The circuit is:

- in Negation Normal Form (NNF) if any negation $\neg g_i$ is such that g_i is directly an input, i.e. $i \in 1 \dots n$.
- in Decomposable NNF (DNNF) if it is in NNF and if every conjunction node $g_i \wedge g_j$ of the circuit is such that $\text{Vars}(g_i) \cap \text{Vars}(g_j) = \emptyset$.

3 Good Violation Functions

Throughout this section $\mathcal{E}$ denotes a predicate of n variables representing a non-empty relation, i.e. if there exists a tuple t that satisfies the relation $\mathcal{E}$ so that $\mathcal{E}(t) = 1$.

Basic Correctness Condition. Following a basic definition, used by e.g. [1], a violation is any poly-time computable function $\mathcal{F}$ that respects the condition:

$$\mathcal{F}(t) > 0 \text{ iff } \mathcal{E}(t) = 0 \quad (1)$$

This minimal requirement is needed for correctness. It is weak because it is satisfied by functions that are hard to minimize. For instance a straightforward violation function would define $\mathcal{F}(t)$ as $1 - \mathcal{E}(t)$. It is a correct function in the sense that minimizing this function will lead us to solutions; however the minimization in question won't be easy: if we are not within distance ≤ 1 of a solution, we have strictly no information on where to move.

What is a "ideal" Violation Function? Two basic desirable features of a good violation function are the following:

Definition 5. (*Properties of Violation Functions*) A function $\mathcal{F}$ is:

- **plateau-free** if all local minima are solutions: whenever we have a non-solution t , there exists a tuple t' at distance 1 from t such that $\mathcal{F}(t') < \mathcal{F}(t)$.
- **c -bounded** if $\mathcal{F}(t) \leq c$, for all tuples t .

Functions that have plateaus are not ideal because search can get stuck into local minima, where there is no information on where to move next. Functions that are bounded by a c that grows exponentially with the number of variables of the predicate are not ideal because in this case the number of moves needed to reach a local minimum grows exponentially³. Functions that are both plateau-free and c -bounded for a polynomially-bounded c are obviously ideal:

Observation 1. *If we have a violation function $\mathcal{F} : \{0, 1\}^n \mapsto \mathbb{Z}$ that is plateau-free and c -bounded for a predicate $\mathcal{E}$, then we can find a solution to $\mathcal{E}$ in time $\mathcal{O}(n^2 c K)$, where K is the cost of evaluating $\mathcal{F}$.*

We argue that being (1) efficiently computable, (2) plateau-free, and (3) polynomially-bounded are the properties that characterize an ideal violation function. In fact most definitions of violation function in the constraint-based literature follow this ideal, without necessarily making it clear (see e.g. [6,1,19]). Some remarks:

- This definition basically means that an ideal violation for a constraint should allow a trivial LS (and indeed greedy) search algorithm to find a solution quickly. This is reasonable and follows the approach used for propagation and relaxation: we cannot in general find an ideal propagation / relaxation / violation for an arbitrarily large system of constraints; but reasoning constraint by constraint allows us to do "the best work possible" for each constraint.
- For simplicity our definition is based on the Hamming distance, which is the natural distance for SAT. The notions of local minima and plateau-freeness can obviously be adapted to other distances (e.g. Euclidian distances or distances based on numbers of swaps rather than flips). This captures other notions of "neighborhood".

It is clear that ideal violation functions cannot be defined for all Boolean predicates: notably this cannot be done for some global constraints that are NP-complete, like the Cumulative constraint, that schedules a set of tasks with release times and deadlines so that resource capacity constraints are satisfied. For such predicates the best we can do is to use less-than-ideal violation functions

³ There are well-known optimization problems that are plateau-free but whose violation function has an exponential range, and for which no known method can reach a local minimum in (strongly) polynomial time: some examples are the computation of a stable state in a neural network and other PLS-complete problems [10,9].

that satisfy the correctness requirement of Eq. 1 and are 'as close as possible' to the ideal. This is, again, similar to what we do with other constraint processing techniques: propagation methods weaker than arc-consistency, relaxed LP representation.

Distance-Based Violation Functions. An important particular case of ideal violation function is the following:

Definition 6. *The Hamming Violation Function $\mathcal{H}$ is the distance from the current tuple t to the closest solution :*

$$\mathcal{H}(t) = \min_{t' \in \{0,1\}^n} \{\mathcal{D}(t, t') : \mathcal{E}(t') = 1\}$$

In particular $\mathcal{H}(t) = 0$ for any solution. By convention $\mathcal{H}(t) = \infty$ for every t when the circuit is unsatisfiable. This definition clearly relates to the literature on over-constrained problems, where notions of violation based on (Hamming) distances have been considered, see [20]. The Hamming violation function clearly has ideal properties: it is obviously plateau-free and n -bounded, which means that a local minimum can be found by a greedy algorithm in time $\mathcal{O}(n^3K)$; and is faithful to the intuition that "the higher the violation, the further-off we are from satisfying the constraint". Unfortunately for general circuits it is easy to see that $\mathcal{H}(t)$ cannot be computed efficiently.

Observation 2. *Given a predicate $\mathcal{E}(t)$ computing $\mathcal{H}$ is NP-hard.*

Proof. Follows from a reduction from the circuit satisfiability problem. We pick an arbitrary point t and ask to compute $\mathcal{H}(t)$. If $\mathcal{H}(t) = \infty$ then the circuit satisfiability problem is unsatisfiable, otherwise it is satisfiable. $\square$

A Practical Violation Function. Since we cannot in general produce the Hamming violation function or another plateau-free violation for all circuits, it is natural to consider alternatives that are less-than-ideal. We consider the violation function of [13], which generalizes for arbitrary circuits the definition proposed by [17] for NNF and makes it symmetric. We call this function *symmetric* violation function.

Definition 7. (*Symmetric Violation*) *Given a tuple of Boolean values $t = \langle t_1 \dots t_n \rangle$, the symmetric violation $\mathcal{V}_{g_k}$ of every gate g_k of the circuit is defined by induction on k as follows:*

- If $k \leq n$ (input gate): $\mathcal{V}_{g_k}(t) := +1$ if $t_k = 0$; $\mathcal{V}_{g_k}(t) := -1$ if $t_k = 1$.
- For a non-input gate, i.e. if $k > n$: the definition is by case depending on whether g_k is of the form $\neg g_i$, or $g_i \wedge g_j$, or $g_i \vee g_j$:

$$\begin{aligned}
\mathcal{V}_{\neg g_i}(t) &= -\mathcal{V}_{g_i}(t) \\
\mathcal{V}_{g_i \wedge g_j}(t) &= \begin{cases} \mathcal{V}_{g_i}(t) + \mathcal{V}_{g_j}(t) & \text{if } \mathcal{V}_{g_i}(t) > 0 \text{ and } \mathcal{V}_{g_j}(t) > 0 \\ \max(\mathcal{V}_{g_i}(t), \mathcal{V}_{g_j}(t)) & \text{otherwise} \end{cases} \\
\mathcal{V}_{g_i \vee g_j}(t) &= \begin{cases} \mathcal{V}_{g_i}(t) + \mathcal{V}_{g_j}(t) & \text{if } \mathcal{V}_{g_i}(t) < 0 \text{ and } \mathcal{V}_{g_j}(t) < 0 \\ \min(\mathcal{V}_{g_i}(t), \mathcal{V}_{g_j}(t)) & \text{otherwise} \end{cases}
\end{aligned}$$

- The violation of the circuit is noted $\mathcal{V}(t)$ and is defined as the violation of the output gate: $\mathcal{V}(t) = \mathcal{V}_{g_m}(t)$.

A straightforward induction shows that for every gate g_k , $\mathcal{V}_{g_k}(t) \neq 0$, and the violation $\mathcal{V}_{g_k}(t)$ is positive iff the evaluation of $\mathcal{E}_{g_k}(t)$ is false.

4 Our Main Result and Its Implications

The Symmetric Violation is Ideal for DNNF Circuits. With the context and all required material introduced it is now easy to state our result. The full proof is given in the Appendix.

Proposition 1. *If the circuit is in Decomposable Negation Normal Form (DNNF), the symmetric violation function of Def. 7 is such that, for all $t \in \{0, 1\}^n$:*

$$\text{if } t \text{ is not a solution then } \mathcal{V}(t) = \mathcal{H}(t)$$

This result implies that when the considered circuit is in DNNF, (1) $\mathcal{H}(t)$ is equal to $\max(0, \mathcal{V}(t))$ and can be computed in linear time; and (2) the symmetric violation is ideal. In other words: when we have a circuit in DNNF, the violation score used in papers such as [13] is guaranteed to lead greedy or local search algorithms directly to a solution.

Example 1. Consider a Boolean formula $\mathcal{B}$ that is represented by DNNF in Figure 2 (borrowed from [5]). An assignment of A, B, C and D to *false* does not satisfy the Boolean formula $\mathcal{B}$. Fig. 2 shows the computation of the violation function $\mathcal{V}(t)$. The value of $\mathcal{V}(t)$ is 1.

A closest assignment by Hamming distance is A, B, C are *false* and D is *true*. The Hamming distance between these two assignments is one which is the value of the computed violation function $\mathcal{V}(t)$.

We note that for general circuits the symmetric violation is incomparable with the Hamming violation: i.e. can be higher or lower depending on cases. This implies that in this case the violation is a purely heuristic indicator respecting only the weak correctness property. It would be interesting to have useful violation functions that provably *under-approximate* the Hamming distance since these would be *admissible* heuristics in the classical A^* sense.

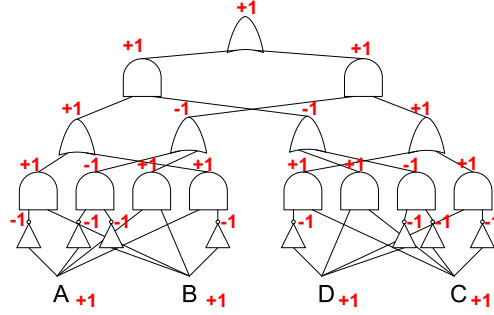


Fig. 2. Computation of the violation function $\mathcal{V}(t)$ for the assignment $[A, B, C, D] = [F, F, F, F]$

Ideal Violation Functions for Global Constraints. The main use of Prop. 1 is that it shows how good violation functions can be defined for a large class of tractable predicates. It has been shown in recent CP work that a large class of constraints can be encoded in DNNF. An example is the TABLE constraint, whose natural Boolean encoding as a disjunction of conjunctions (lines of the tables) is directly in DNNF. A more complex example is the GRAMMAR constraint, for which violation functions were only hinted at in [6]. From these decompositions it is easy to see that Prop. 1 guarantees that the symmetric violation is ideal.

Violation Functions and Weighted Propagators. To conclude on the computation of ideal violation functions for complex constraints, we note that an efficient computation of the Hamming violation for a constraint is implied by the existence of an efficient propagator for the *weighted* version of the constraint, in the sense used by e.g. [12]. In this framework, a function associates a weight to every variable/value pair. A variable is introduced to capture the weight of the assignment, defined as the sum of the weights of the values assigned to all variables. The propagator removes values that are not supported by any assignment with a weight that is less than the cost variable. While initially introduced for SAT, the symmetric violation algorithm can in the case of DNNF be thought of as a specialization of the weighted DNNF constraint propagator of [12]. Being specialized for violations (one value as opposed to two bounds) and purely 'bottom-up', the algorithm is slightly more efficient and incremental.

Proposition 2. *If a constraint has a (polynomially computable) weighted propagator, then its Hamming distance violation can be computed in polynomial time.*

Consider an application of Prop. 2 to the AllDiff constraint.

Example 2. Consider ALLDIFFERENT $[x_1, \dots, x_4]$ with the following domains: $D(x_1) = [2, 3]$, $D(x_2) = [2, 3]$, $D(x_3) = [1, 4]$ and $D(x_4) = [1, 4]$. Finding a solution of the ALLDIFFERENT constraint corresponds to finding a matching in

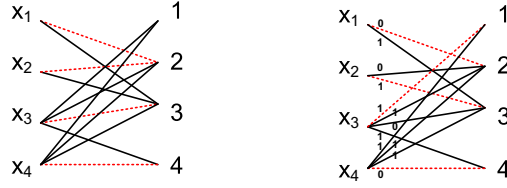


Fig. 3. Computation of Hamming distance violation for the $\text{ALLDIFFERENT}(x_1, \dots, x_4)$ constraint and the assignment $x = [x_1, x_2, x_3, x_4] = [2, 2, 3, 4]$

a bipartite variable-value graph G . To build the graph G we introduce a vertex for each variable $[x_1, \dots, x_4]$ and for each value $v \in [1, \dots, 4]$. We connect a variable-vertex x_i to the value-vertex v iff $v \in D(x_i)$. Fig. 3(left) shows graph G that corresponds to the ALLDIFFERENT constraint.

Consider an assignment $x = [x_1, x_2, x_3, x_4] = [2, 2, 3, 4]$. This assignment does not satisfy the constraint. Hence, it does not correspond to a matching. The set of edges that correspond to the assignment are shown in red dashed line.

To find a solution that has the minimum Hamming distance violation to x we can use the variable-value graph G . We transform G into a weighted graph. We assign cost 0 to all edges that correspond to the assignment and 1 to the remaining edges (Fig. 3, right). The minimum cost matching corresponds to the solution $x = [2, 3, 1, 4]$ that minimizes Hamming distance violation.

5 Future Works

Our main result shows a new facet of the versatile DNNF class of tractable circuits. It was known that any constraint expressible in this language automatically benefits from *ideal propagators*, a meta-result that informs their integration in CP solvers. We showed that predicates expressible in DNNF also benefit from *ideal violation functions*, providing a meta-result that can now be used in constraint-based local search solvers. A third major class of constraint satisfaction tools is Linear Programming, and it would be interesting to study in a similar spirit *ideal (convex Hull) LP relaxations* of DNNF circuits. Another important open question left for future work is the empirical aspect: can we design an efficient circuit-based local search solver in which global constraints are effectively treated by an encoding to DNNF circuits? Lastly, a key question for all solvers is the *combination* of multiple constraints: problems can often be expressed using constraints that have, independently, good propagators / violation functions / relaxations; however by combining these constraints we lose the desired property. For local search it would be interesting to design schemas for combining constraints that are less naive than just summing violations.

Acknowledgements. This work benefitted from discussions with Matthew Kitching and Horst Samulowitz.

References

1. Ågren, M., Flener, P., Pearson, J.: Revisiting constraint-directed search. *Information and Computation* 207(3), 438–457 (2009)
2. Belov, A., Stachniak, Z.: Improving Variable Selection Process in Stochastic Local Search for Propositional Satisfiability. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 258–264. Springer, Heidelberg (2009)
3. Belov, A., Stachniak, Z.: Improved Local Search for Circuit Satisfiability. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 293–299. Springer, Heidelberg (2010)
4. Darwiche, A.: Decomposable negation normal form. *J. ACM* 48(4) (2001)
5. Darwiche, A., Marquis, P.: A knowledge compilation map. *J. of Artif. Intel. Research (JAIR)* 17, 229–264 (2002)
6. He, J., Flener, P., Pearson, J.: Toward an automaton constraint for local search. In: Workshop on LS Techniques for Constraint Satisfaction, pp. 13–25 (2009)
7. Jarvisalo, M., Junttila, T.A., Niemelä, I.: Justification-Based Local Search with Adaptive Noise Strategies. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 31–46. Springer, Heidelberg (2008)
8. Jarvisalo, M., Junttila, T.A., Niemelä, I.: Justification-based non-clausal local search for SAT. In: Euro. Conf. on Artif. Intel. (ECAI), pp. 535–539 (2008)
9. Johnson, D.S.: The NP-completeness column: Finding needles in haystacks. *ACM Trans. on Algorithms (TALG)* 3(2) (2004)
10. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: How easy is local search? *J. of Computer and System Sciences* 37(1), 79–100 (1988)
11. Jung, J.-C., Barahona, P., Katsirelos, G., Walsh, T.: Two encodings of DNNF theories. In: ECAI Workshop on Inference Methods Based on Graphical Structures of Knowledge (2008)
12. Katsirelos, G., Narodytska, N., Walsh, T.: The weighted grammar constraint. *Annals of Operations Research* (2010) (to appear)
13. Muhammad, R., Stuckey, P.J.: A Stochastic Non-CNF SAT Solver. In: Yang, Q., Webb, G. (eds.) PRICAI 2006. LNCS (LNAI), vol. 4099, pp. 120–129. Springer, Heidelberg (2006)
14. Pham, D.N., Thornton, J., Sattar, A.: Building structure into local search for SAT. In: Int. Joint. Conf. on Artif. Intel. (IJCAI), pp. 2359–2364 (2007)
15. Pham, D.N., Thornton, J., Sattar, A.: Efficiently exploiting dependencies in local search for sat. In: AAAI Conf. on Artif. Intel. (AAAI), pp. 1476–1478 (2008)
16. Quimper, C.-G., Walsh, T.: Decomposing Global Grammar Constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 590–604. Springer, Heidelberg (2007)
17. Stachniak, Z.: Going non-clausal. In: Conf. on Theory and Applications of Satisfiability Testing (SAT), pp. 316–322 (2002)
18. Stachniak, Z., Belov, A.: Speeding-Up Non-clausal Local Search for Propositional Satisfiability with Clause Learning. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 257–270. Springer, Heidelberg (2008)
19. Van Hentenryck, P., Michel, L.: Constraint-Based Local Search. MIT Press (2005)
20. van Hoeve, W.-J.: Over-constrained problems. In: Hybrid optimization: the 10 Years of CPAIOR, ch. 6 (2010) (to appear)

```

t      ← (an arbitrary value)
optimal ← false
while ¬ optimal do
  n ← t
  foreach t' at distance 1 from t do
    if  $\mathcal{F}(t') < \mathcal{F}(t)$  then {n ← t'; break}
  done
  optimal ← (n = t)
done

```

Fig. 4. Greedy local minimization algorithm

A Appendix

A.1 Greedy Local Minimization Algorithm

A.2 Proof of Proposition 1

Proposition 1. *If the circuit is in Decomposable Negation Normal Form (DNNF), the symmetric violation function of Def. 7 is such that, for all $t \in \{0, 1\}^n$:*

$$\text{if } t \text{ is not a solution then } \mathcal{V}(t) = \mathcal{H}(t)$$

Proof. Note that a DNNF is always satisfiable, i.e. represents a non-empty Boolean relation; so we don't have to deal with the case where $\mathcal{H}(t) = \infty$. Given an assignment t we prove by induction on k for all gates g_k that:

$$\text{if } \mathcal{E}_{g_k}(t) = 0 \text{ then } \mathcal{V}_{g_k}(t) = \min\{\mathcal{D}(t, t') : \mathcal{E}_{g_k}(t') = 1\}$$

- if $k \leq n$, i.e. the gate g_k is an input. If g_k is violated by the assignment t then $\mathcal{V}_{g_k}(t) = 1$; the closest tuple t' that satisfies g_k is obtained by flipping the value assigned by t to g_k , and $\mathcal{D}(t, t') = 1$.
- if $k > n$, the proof is by case on the type of gate g_k (In all cases the induction hypothesis holds for the inputs g_i, g_j since $i, j < k$):
 - Negation of the form $\neg g_i$. The Negation Normal Form means that g_i is an input. If $\mathcal{E}_{g_k}(t) = 0$ then $\mathcal{V}_{g_k}(t) = -\mathcal{V}_{g_i}(t) = +1$. The closest tuple t' that satisfies g_k is obtained by flipping the value on g_i , and $\mathcal{D}(t, t') = 1$.
 - Conjunction of the form $g_i \wedge g_j$. The decomposability property applies, i.e. we have $\text{Vars}(g_i) \cap \text{Vars}(g_j) = \emptyset$. If $\mathcal{E}_{g_k}(t) = 0$, we have the following cases:

Case 1: $\mathcal{V}_{g_i}(t) > 0$ and $\mathcal{V}_{g_j}(t) > 0$. Both inputs are violated and $\mathcal{V}_{g_k}(t) = \mathcal{V}_{g_i}(t) + \mathcal{V}_{g_j}(t)$.

By the induction hypothesis there exist (1) a tuple u such that $\mathcal{E}_{g_i}(u) = 1$ and $\mathcal{D}(t, u) = \mathcal{V}_{g_i}(t)$; and (2) a tuple w such that $\mathcal{E}_{g_j}(w) = 1$ and $\mathcal{D}(t, w) = \mathcal{V}_{g_j}(t)$. Consider now the tuple t' whose value on every input $l \in 1 \dots n$ is defined as follows:

$$t'_l = \begin{cases} u_l & \text{if } l \in \text{Vars}(g_i) \\ w_l & \text{if } l \in \text{Vars}(g_j) \\ t_l & \text{otherwise} \end{cases}$$

(This is well-defined since $\text{Vars}(g_i)$ and $\text{Vars}(g_j)$ are disjoint.) It is easy to show that (1) $\mathcal{E}_{g_k}(t') = 1$; (2) t' is at distance $\mathcal{V}_{g_k}(t) = \mathcal{V}_{g_i}(t) + \mathcal{V}_{g_j}(t)$ from t ; (3) any tuple τ at distance $\mathcal{D}(t, \tau) < \mathcal{V}_{g_k}(t)$ is such that $\mathcal{E}_{g_k}(\tau) = 0$. This shows that $\mathcal{V}_{g_k}(t) = \min\{\mathcal{D}(t, t') : \mathcal{E}_{g_k}(t') = 1\}$.

Case 2: $\mathcal{V}_{g_i}(t) > 0$ and $\mathcal{V}_{g_j}(t) < 0$. (The **Case 3:** $\mathcal{V}_{g_j}(t) > 0$ and $\mathcal{V}_{g_i}(t) < 0$ is symmetric.) g_i is the only input that is violated and $\mathcal{V}_{g_k}(t) = \mathcal{V}_{g_i}(t)$.

There exists a tuple u such that $\mathcal{E}_{g_i}(u) = 1$ and $\mathcal{D}(t, u) = \mathcal{V}_{g_i}(t)$. Consider the tuple t' defined for all inputs $l \in 1 \dots n$ by: $t'_l = u_l$ if $l \in \text{Vars}(g_i)$; otherwise: $t'_l = t_l$. The tuple t' is at distance $\mathcal{V}_{g_k}(t)$ from t and is the closest tuple that satisfies g_k .

- Disjunction of the form $g_i \vee g_j$. If $\mathcal{E}_{g_k}(t) = 0$, then we have $\mathcal{V}_{g_i}(t) > 0$ and $\mathcal{V}_{g_j}(t) > 0$. W.l.o.g. let g_i be the input that is "less violated", i.e. let us assume that $\mathcal{V}_{g_i}(t) \leq \mathcal{V}_{g_j}(t)$. Then $\mathcal{V}_{g_k}(t) = \mathcal{V}_{g_i}(t)$.

By the induction hypothesis there exists a tuple u such that $\mathcal{E}_{g_i}(u) = 1$ and $\mathcal{D}(t, u) = \mathcal{V}_{g_i}(t) = \mathcal{V}_{g_k}(t)$. It is clear that $\mathcal{E}_{g_k}(u) = 1$. Additionally no tuple t' with $\mathcal{D}(t, t') < \mathcal{V}_{g_k}(t)$ satisfies g_k : if such a tuple did exist it would either satisfy g_i and be at distance $< \mathcal{V}_{g_i}(t)$ from t ; or satisfy g_j and be at distance $< \mathcal{V}_{g_j}(t)$. Both cases are contradictory to the induction hypothesis.

A.3 Proof of Proposition 2

Proposition 2. *If a constraint has a (polynomially computable) weighted propagator, then its Hamming distance violation can be computed in polynomial time.*

Proof. We show that it is possible to compute the Hamming violation function for many useful global constraints C . Let $C(X)$, $X = [X_1, \dots, X_n]$ be a global constraint and $C(X, W, z)$ be the weighted version of the constraint $C(X)$. The weight matrix $W(X_i, t_j)$, $t_j \in D(X_i)$ $i = 1, \dots, n$ of integer values specifies the weight of each variable-value pair, and the cost variable z restricts possible weights of a solution to be less than or equal to z . The weight of a solution $X = t$ is $W(X, t) = \sum_{i=1}^n W(X_i, t_i)$. $C(X, W, z)$ holds iff $C(X)$ holds and $W(X, t) \leq z$.

We assume that $C(X, W, z)$ admits a polynomial time propagator that achieves domain consistency on the variables X and bounds consistency on the cost variable z . This is the case for many important global constraints, like ALLDIFFERENT or REGULAR constraints.

We show that we can use $C(X, W, z)$ to compute the Hamming distance function. Suppose t' is the current assignment that violates the constraint $C(X)$. We construct $C(X, W, z)$ in the following way:

$$W(X_i, t_k) = \begin{cases} 0 & \text{if } t_k = t'_i \\ 1 & \text{otherwise} \end{cases}$$

for $t_k \in D(X_i)$, $i = 1, \dots, n$ and $D(z) = [0, n]$. As the propagator for $C(X, W, z)$ enforces bounds consistency on z , the lower bound of z gives the value of $\mathcal{H}$ and its support $X = t$, such that $W(X, t) = lb(z)$, is the closest, by Hamming distance, solution to t' . This follows from the correctness of the propagator for $C(X, W, z)$.