Reformulation of SAT into a Polynomial Box-Constrained Optimization Problem

Stéphane Jacquet and Sylvain Hallé

Laboratoire d'informatique formelle
Université du Québec à Chicoutimi, Canada

Abstract. In order to leverage the capacities of non-linear constraint solvers, we propose a reformulation of SAT into a box-constrained optimization problem where the objective function is polynomial. We prove that any optimal solution of the numerical problem corresponds to a solution of the Boolean formula, and demonstrate a stopping criterion that can be used with a numerical solver.

1 Introduction

Boolean satisfiability (SAT) is probably the most well known NP-complete problem, which consists in its simplest form of finding appropriate values for variables of a propositional logic formula φ in Conjunctive Normal Form (CNF) such that it evaluates to true ($\top$). This problem is typically solved symbolically at the logical level through different techniques.

However, in recent years, different *reformulations* have been suggested to solve SAT by turning it into a numerical problem to be solved by numerical techniques. For example, a linear algebra approach has been attempted in [6]; the reformulation transforms a SAT instance into a system of linear equations. In [8], a relaxation of the Boolean variables is mixed with gradient-based algorithms. The work from [5] offers a reformulation through an optimization of degree 4 by adding as many variables as the number of clauses in the Boolean formula. In [7], the reformulation is done by defining an extension of the DeMorgan Laws.

The present work suggests a reformulation of SAT in order to use the capacities of *non-linear* solvers. The principle, illustrated in Figure 1, works as follows. First, a Boolean formula φ over $\mathbb{B}^n$ is transformed into a real-valued polynomial $\hat{\varphi}$ over the interval $[0;1]^n$, using a transformation called τ, described in Section 2 (top arrow). For example, a CNF formula $\varphi = (a \vee b) \wedge (a \vee \neg b)$ will result in the function $\hat{\varphi} = (a + b - ab) + (a + 1 - b - a(1 - b))$. The SAT problem turns into the problem of maximizing $\hat{\varphi}$, a box-constrained optimization task that can be offloaded to a numerical solver (right arrow). Section 3 then formally proves that a real-valued solution provided by such a solver, such as $\hat{a} = (0.99, 0.99)$, can be converted into an optimal solution over the integers 0 and 1 using a backwards

Corresponding author (S. Jacquet): stephane.jacquet1@uqac.ca

transformation ρ (bottom arrow); in our example, this would yield the point $(1, 1)$. Proposition 5 will show that the result applies even if a solver converges to a solution with coordinates that do not lie close to 0 or 1. Finally, Theorem 2 will show that such a solution exists if and only if the corresponding SAT instance admits it as a solution (left arrow).

In the context of this article, the objective function is non-linear (since $\hat{\phi}$ is polynomial of degree superior or equal to two as soon as there is a clause with two literals or more). The gradient of a function gives a direction where the function takes higher values [12]. When this gradient is not accessible, algorithms like the Nelder-Mead algorithm [11], genetic algorithms [13] or any algorithms from the derivative-free optimization field [1] are efficient. However, in the present work, the gradient is available, which makes it suitable for iterative algorithms. This paper lists a couple theoretical results that could be used when using a gradient-based algorithm. A few of those theoretical results have been listed in this paper; for example, some algorithms have been developed specifically for polynomial optimization [4, 10], which is what one gets after reformulating SAT using the construction presented in this article.

The paper is structured as follows. Section 2 describes the transformation rules to define the reformulation. Section 3 studies the properties of the polynomial obtained after the transformation. Section 4 analyses the links between the SAT problem and its reformulation. It also contains theoretical results that can be used to anticipate numerical results over reformulation. Section 5 summarizes the theoretical results and talks about other difficulties that could occur in future numerical tests.

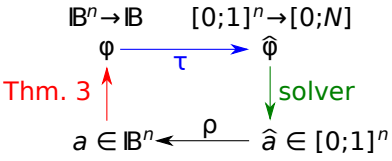

Fig. 1. A summary of the approach followed in this paper.

2 SAT as an Optimization Problem

In this section, we will describe how to transform a SAT instance into a polynomial function to maximize. Let $a_1, \ldots a_n$ be the n Boolean variables occurring in an arbitrary SAT instance. For convenience, we shall equate the values $\bot$ and $\top$ with integers 0 and 1, respectively. We will note $a = (a_1; \ldots; a_n)$. The set $\mathbb{B}$ will be interpreted as the subset of $\mathbb{R}$ containing only the values 0 and 1.

A Boolean variable a (which takes the values 0 or 1) will be assimilated with its bounded real variable *relaxation*, by allowing it to take a value in the interval $[0; 1]$. For the sake of readability, we shall use the same symbol for a Boolean variable and its relaxation; it should be clear enough in the formulas whether a variable is Boolean or real.

Equipped with this notation, we can lift the notion of relaxation from Boolean variables to Boolean formulas. The transformation will be done using a function

$\tau : (\mathbb{B}^n \to \mathbb{B}) \to ([0;1]^n \to [0;N])$, which takes as input a Boolean formula with N clauses, and produces as its output a real-valued polynomial expressed in terms of the relaxations of the Boolean variables.

Definition 1. *Let a be an arbitrary propositional variable, and φ_1 and φ_2 be arbitrary Boolean formulas. The transformation function $\tau : (\mathbb{B}^n \to \mathbb{B}) \to ([0;1]^n \to [0;N])$ defined recursively as follows:*

$$\tau(b) = b, \text{ for } b \in \mathbb{B} \tag{1}$$
$$\tau(a) = a \tag{2}$$
$$\tau(\neg a) = 1 - a \tag{3}$$
$$\tau(\varphi_1 \vee \varphi_2) = \tau(\varphi_1) + \tau(\varphi_2) - \tau(\varphi_1)\tau(\varphi_2) \tag{4}$$
$$\tau(\varphi_1 \wedge \varphi_2) = \tau(\varphi_1) + \tau(\varphi_2) \tag{5}$$

The introduction gave an example of a transformation using those five rules. It can be easily shown that applying them to a given CNF formula φ produces a unique polynomial, which will be written $\tau(\varphi)$. To simplify the notation, we shall also note this polynomial $\hat{\varphi}$. Three remarks should be made. First, one should be careful on the fact that two equivalent Boolean formulas (i.e. which have the same solutions) that have different CNF representations may have different transformation through τ. This is shown with the following example: $\tau(a \vee a) = 2a - a^2 \neq \tau(a) = a$ but $a \vee a$ is logically equivalent to a. Second, note how τ transforms logical conjunction into an *addition* instead of a multiplication; this goes against the "probabilistic" interpretation that $P(A \wedge B) = P(A)P(B)$ when A and B are independant. This decision has been done to reduce the degree of the polynomial. With a multiplication, the degree of the polynomial will be equal to the number of literals in the CNF representation of φ. With the addition, the degree will be much smaller and described in Proposition 1.

Finally, since $\hat{\varphi}$ takes as arguments elements of $[0;1]^n$, and not $\mathbb{B}^n$, the simplification $a^2 = a$ (commonly occurring in operations over $\{0,1\}$) is not used.

The objective of this work is to solve the following optimization problem:

$$(P) : \max_{a \in [0;1]^n} \hat{\varphi}(a).$$

This is a case of an optimization problem that has what are called *box constraints*, meaning that all its variables are bounded by real values –the interval $[0;1]$ in that case. In addition, the objective function is polynomial, which means that its gradient can be calculated and used in the solving process.

3 Properties of $\hat{\varphi}$

It remains to determine how solutions to (P) can be used to produce solutions to the original SAT instance, and under what conditions. This is the purpose of the next two sections. First, we need to establish a few results on the properties of the polynomial function $\hat{\varphi}$ on $[0;1]^n$. This will then help to solve (P). A first

observation can be made about the degree of the polynomial $\hat{\varphi}$ when φ is k-SAT (i.e. when each clause contains at most k literals).

Proposition 1. *Let $k \in \mathbb{N}^*$ (i.e. positive integer). If φ is k-SAT, then the degree of $\hat{\varphi}$ is k.*

The proof is trivial and can be done by induction. This result is important as it bounds the degree of the polynomial. Furthermore, since any SAT instance has a polynomial reduction into 3-SAT [9], this guarantees the existence of a reformulation of SAT into an optimization problem for a polynomial of degree at most 3.

We shall then observe that $\hat{\varphi}$ is "well-behaved" —among other things, that it maps the real hypercube $[0;1]^n$ on $[0;N]$, that the discrete hypercube $\mathbb{B}^n$ on $\{0,\ldots,N\}$, where N is the number of clauses in φ and to understand how $\hat{\varphi}$ behaves on the boundaries of the set $[0;1]^n$, noted:

$$\partial([0;1]^n) = \{(a_1,\ldots,a_n) \in [0;1]^n : \exists i \in \{1,\ldots,n\}, a_i \in \mathbb{B}\}.$$

.

Proposition 2. *Let $\hat{\varphi}$ be a polynomial resulting from the transformation of a SAT instance φ containing $N \in \mathbb{N}^*$ clauses. Then: i) if $a \in [0;1]^n$ then $\hat{\varphi}(a) \in [0;N]$; ii) if $a \in \mathbb{B}^n$ then $\hat{\varphi}(a) \in \{0;\ldots;N\}$; iii) if $a \in]0;1[^n$ then $\hat{\varphi}(a) \in]0;N[$.*

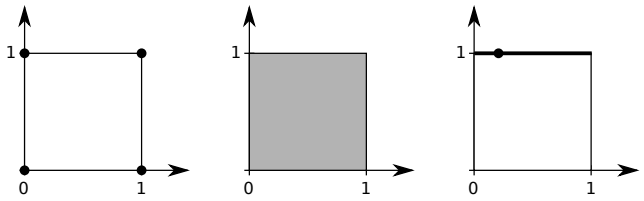

Fig. 2. On the left, studying $\hat{\varphi}$ on $\mathbb{B}^n$. In the middle, studying $\hat{\varphi}$ on $]0;1[^n$. On the right, studying $\hat{\varphi}$ on $\partial([0;1]^n)$.

Proof. It should be noted that, with two variables, the case *ii*) corresponds to the first graph on Figure 2, while *iii*) corresponds to the second one. For i), let $a \in [0;1]^n$. If φ is a clause, then it can be shown by induction on the length of the clause that $\tau(\varphi) \in [0;1]$. Then, if φ contains N clauses, using equation (5), it can be shown by induction on the number of clauses that $\tau(\varphi) \in [0;N]$. For *ii*), the proof is very similar to *i*). Let $a \in \mathbb{B}^n$. If φ is a clause, it can be shown by induction on the length of the clause that $\tau(\varphi) \in \mathbb{B}$. Then, if φ contains N clauses, then the transformation through τ of each clauses evaluated in a will be in $\mathbb{B}$ and thus the sum of the N terms being in $\{0;\ldots;N\}$. The proof of *iii*) is almost identical to *ii*). □

Let us now study the eventuality where there exists a solution to (P) on the boundaries of $\partial([0;1]^n)$. Of particular interest is the case where $\hat{\varphi}(a) = N$, $a \in [0;1]^n$ but a contains at least one variable that is neither 0 nor 1. This can be illustrated by the SAT instance $\varphi = (a \vee b) \wedge (\neg a \vee b)$, which yields $\hat{\varphi} = (a + b - ab) + ((1 - a) + b - (1 - a)b)$. This polynomial admits an optimal solution at $(1/2, 1) \in \partial([0;1]^2)$. However, one can observe that in this case, the value of a has no impact on the value of φ. A solver using the gradient may notice that $\partial\hat{\varphi}/\partial a = 0$, and thus never change this variable from any value it was initially set to. More importantly, it should be noted that in this case, both $(0;1)$ and $(1;1)$ are also optimal solutions of $\hat{\varphi}$. This result corresponds to the third plot of Figure 2; as a matter of fact, if $\hat{\varphi}$ takes the value N on the dot, then it takes the value N on the whole edge in bold.

Can this observation be generalized to any Boolean formula? It turns out that the answer is yes. In order to prove it, let us define the function $\Psi : [0;1]^n \mapsto \mathcal{P}(\mathbb{B}^n)$ defined for all $a \in [0;1]^n$ by $\Psi(a) = \{x \in \mathbb{B}^n : \forall i \in A(a), x_i = a_i, \forall i \notin A(a), x_i \in \mathbb{B}\}$, where $A(a) = \{i \in \{1, \ldots, n\} : a_i \in \mathbb{B}\}$. Intuitively, given a non-Boolean solution a, $\Psi(a)$ returns the set of "corners" of the hypercube $\mathbb{B}^n$ adjacent to a. If we use again the third plot of Figure 2, then the dot has coordinates $(0.25;1)$ and $\Psi(0.25;1) = \{(0;1),(1;1)\}$. First, we need to prove that the result is true for a formula containing only one clause.

Proposition 3. *Let φ be a clause; if there is $a \in \partial([0;1]^n)$ such that $\hat{\varphi}(a) = 1$, then for all $x \in \Psi(a)$, $\hat{\varphi}(x) = 1$.*

Proof. Let $k \in \{1, \ldots, n\}$. We can consider that all the variables are positive literals in the formula. If not, the one with a negative literal can be redefined as the opposite of that variable. If needed, it is possible to rename the variables such that $\varphi = a_1 \vee \ldots \vee a_k$.

We show the result by finite induction on the length of the clause. For $k = 1$, then $\varphi = a_1$ and $\hat{\varphi}(a) = a_1$. If, for some $a \in \partial([0;1]^n)$, $\hat{\varphi}(a) = 1$, then $a_1 = 1$; so, for all $x \in \Psi(a)$, $a_1 = 1$. This shows that for all $x \in \Psi(a)$, we have that $\hat{\varphi}(x) = 1$.

Let $k \in \{1, \ldots, n-1\}$ and let us assume that for any clause of length k, if $a \in \partial([0;1]^n)$ is such that $\hat{\varphi}(a) = 1$, then for all $x \in \Psi(a)$, $\hat{\varphi}(x) = 1$. Consider a clause of length $k+1$. It can be written $\varphi \vee a_{k+1}$ where φ is a clause of length k. So $\tau(\varphi \vee a_{k+1}) = \tau(\varphi) + \tau(a_{k+1}) - \tau(\varphi)\tau(a_{k+1})$. If for some $a \in \partial([0;1]^n)$, $\tau(\varphi \vee a_{k+1})(a) = 1$, so necessarily, $\tau(\varphi)(a) = 1$ or $\tau(a_{k+1})(a) = 1$. In the first case, by the induction hypothesis, for all $x \in \Psi(a)$, $\tau(\varphi \vee a_{k+1})(x) = 1$. The second case is identical to the initial step ($k = 1$). □

Using Proposition 3, the result can now be generalized for logical formulas which are conjunctions of clauses.

Theorem 1. *If φ contains N clauses and $a \in \partial([0;1]^n)$ is such that $\hat{\varphi}(a) = N$, then for all $x \in \Psi(a)$, $\hat{\varphi}(x) = N$.*

Proof. Let $a \in \partial([0;1]^n)$ is such that $\hat{\varphi}(a) = N$. The theorem will be proven by induction on $N \in \mathbb{N}^*$, the number of clauses of φ written in CNF. The case $N = 1$ is solved with Proposition 3.

Let $N \geq 1$ and suppose that, for φ containing N clauses, if $a \in \partial([0;1]^n)$ is such that $\hat{\varphi}(a) = 1$, then for all $x \in \Psi(a)$, $\hat{\varphi}(x) = 1$. Let us define a formula with $N+1$ clauses. It can be written $\varphi \wedge C$, where φ contains N clauses and C is a clause. Then $\tau(\varphi \wedge C) = \tau(\varphi) + \tau(C)$. Let $a \in \partial([0;1]^n)$ such that $\tau(\varphi \wedge C)(a) = N+1$. Then $\tau(\varphi)(a) + \tau(C)(a) = N+1$. Using Property 2, then necessarily, $\tau(\varphi)(a) = N$ and $\tau(C)(a) = 1$. Since C is a clause, by Proposition 3, for all $x \in \Psi(a)$, we have $\tau(C)(x) = 1$. In addition, by the induction hypothesis, we can assert that for all $x \in \Psi(a)$, $\tau(\varphi)(x) = N$. This proves that for all $x \in \Psi(a)$, $\tau(\varphi \wedge C)(x) = \tau(\varphi)(x) + \tau(C)(x) = N+1$. □

4 From $\hat{\varphi}$ to φ

The previous result is important: it shows that, even when a non-Boolean optimum of $\hat{\varphi}$ is found, it can be turned into a solution that has only Boolean values and which is also optimal. It remains to prove that a solution to (P) can be used to construct a solution to the original SAT instance. The following theorem focuses about the case where a solver converges to a solution in $\mathbb{B}^n$.

Theorem 2. *For all $a \in \mathbb{B}^n$, $\varphi(a) = 1$ if, and only if, $\hat{\varphi}(a) = N$.*

The proof, very similar to the proof of Proposition 2, is omitted. Theorem 2 is what justifies the transformation of a Boolean formula φ to the function $\tau(\varphi)$. Finding a solution of the SAT problem described by φ is therefore equivalent to finding the optimal value (equal to N) of the function $\hat{\varphi}$.

However, in practice, a numerical solver will typically find a solution a that does not land perfectly on elements of $\mathbb{B}^n$, but more likely on values very close to 0 or 1. Likewise, the value taken by $\hat{\varphi}$ will be a real number close to, but not equal to N. In such a situation, Theorem 2 does not apply. A natural workaround would be to round each Boolean value to its nearest integer (0 or 1). To this end, let us define the "round" function ρ such that $\rho(x) = 0$ if $x < 1/2$, and $\rho(x) = 1$ otherwise. This function can be lifted to $\mathbb{R}^n$ by defining $\rho(x_1, \ldots, x_n) = (\rho(x_1), \ldots, \rho(x_n))$.

It is not clear at the onset that taking the round of each variable produces a solution that is optimal. Case in point, it is well known that in integer programming, rounding a solution after relaxation can lead to an non-optimal solution [3]. Fortunately, this is not the case with $\hat{\varphi}$, as we shall prove in Theorem 3. From this point, $||.||$ will be the euclidean norm.

Theorem 3. *Let $C > 0$ be a number that satisfies the Lipschitz condition of $\hat{\varphi}$ on $[0;1]^n$, and let $\hat{a} \in [0;1]^n$. If $\hat{\varphi}(\hat{a}) - C||\hat{a} - \rho(\hat{a})|| > N - 1$, then $\rho(\hat{a})$ is a solution of the logical formula φ.*

Proof. Since $\hat{\varphi}$ is a polynomial function on a compact set, it is Lipschitz. Let $C > 0$ be its Lipschitz constant. Suppose the solver found a solution $\hat{a} \in [0;1]^n$. Because $\hat{\varphi}$ is C-Lipschitz, $|\hat{\varphi}(\hat{a}) - \hat{\varphi}(\rho(\hat{a}))| \leq C||\hat{a} - \rho(\hat{a})||$, which means that:

$$\hat{\varphi}(\hat{a}) - C||\hat{a} - \rho(\hat{a})|| \leq \hat{\varphi}(\rho(a)) \leq \hat{\varphi}(\hat{a}) + C||\hat{a} - \rho(\hat{a})||.$$

But $\rho(\hat{a}) \in \mathbb{B}^n$, so using Proposition 2.*ii*) then $\hat{\varphi}(\rho(\hat{a})) \in \{0; \ldots; N\}$. This means that if $\hat{\varphi}(\hat{a}) - C||\hat{a} - \rho(\hat{a})|| > N - 1$, then necessarily $\hat{\varphi}(\rho(\hat{a})) = N$. Thus it guarantees that $\rho(\hat{a})$ is a solution of formula φ using Theorem 2. □

The proof of Theorem 3 claims the existence of a Lipschitz constant C without giving its value. However, using the definition of τ (in Definition 1), it is possible to find a Lipschitz constant of $\hat{\varphi}$, which depends only on n (the number of variables) and N (the number of clauses). Lemma 1, which is a consequence of the mean value theorem with several variables, provides a constant C, which we then use to prove that $\hat{\varphi}$ is C-Lipschitz.

Lemma 1. *If $\hat{\varphi}$ is differentiable and $C = \sup_{a \in [0;1]^n} ||\nabla \hat{\varphi}(a)||$, then $\hat{\varphi}$ is C-Lipschitz.*

Proposition 4. *If φ is a CNF Boolean formula containing N clauses depending of n variables, then $\hat{\varphi}$ is $N\sqrt{n}$-Lipschitz.*

Proof. This proof requires new notations. Consider a clause C where the variable a_i, $i \in \{1; \ldots; n\}$, appears. We will note C^{-a_i} the clause C where a_i has been removed. Also, considering the formula φ, we will define $V_{a_i} \subseteq \{1; \ldots; N\}$ the index of the clauses where a_i is a positive literal and $W_{a_i} \subseteq \{1; \ldots; N\}$ the index of the clauses where a_i is a negative literal. By definition of V_{a_i} and W_{a_i}, it can be observed that for all $i \in \{1; \ldots; N\}$, $|V_{a_i}| + |W_{a_i}| \leq N$.

It can be then shown that for all $i \in \{1, \ldots, n\}$ and all $a \in [0; 1]^n$,

$$\frac{\partial \hat{\varphi}}{\partial a_i}(a) = \sum_{k \in V_{a_i}} (1 - \hat{C}_k^{-a_i}(a)) - \sum_{k \in W_{a_i}} (1 - \hat{C}_k^{-a_i}(a)).$$

Then,

$$\left|\tfrac{\partial \hat{\varphi}}{\partial a_i}(a)\right| \leq \left| \sum_{k \in V_{a_i}} (1 - \hat{C}_k^{-a_i}(a)) \right| + \left| \sum_{k \in W_{a_i}} (1 - \hat{C}_k^{-a_i}(a)) \right| \leq |V_{a_i}| + |W_{a_i}|$$

Hence $||\nabla \hat{\varphi}(a)||^2 \leq \sum_{i=1}^{n} \left(\frac{\partial \hat{\varphi}}{\partial a_i}(a) \right)^2 \leq \sum_{i=1}^{n} (|V_{a_i}| + |W_{a_i}|)^2 \leq \sum_{i=1}^{n} N^2 \leq nN^2$.

By Lemma 1, this means that $\sup_{a \in [0;1]^n} ||\nabla \hat{\varphi}(a)|| \leq N\sqrt{n}$. □

We note that the Lipschitz constant $C = N\sqrt{n}$ is elegantly simple, especially considering the rather complex polynomials produced by τ in practice. Combining Theorem 3 with Proposition 4 leads to the following proposition:

Proposition 5. *If there is $\hat{a} \in \mathbb{B}^n$ such that $\hat{\varphi}(\hat{a}) - N\sqrt{n}||\hat{a} - \rho(\hat{a})|| > N - 1$, then $\rho(\hat{a})$ is a solution of the Boolean formula φ.*

This result can be used by an iterative numerical solver as a *stopping criterion*. As soon as a solution $\hat{a}$ satisfying the condition of Proposition 5 is found, the solver can stop; there is no point in iterating further to find a better solution, since we already have the guarantee that $\rho(\hat{a})$ is a solution of the SAT instance. This criterion is specific to the transformation τ we use in this work.

5 Conclusion

In this work, a reformulation of a SAT instance has been suggested. This transformation leads to an optimization problem where the objective function is a polynomial function. The link between feasible solutions of the initial SAT problem and the optimal solution of that reformulation has been made.

As future work, numerical experiments should be performed to test the capacity of this reformulation to solve SAT instances. It is known that in such optimization problems, algorithms using the gradient as a stopping criterion can converge to local optimums; multi-start solving using Latin hypercubes [2] to select starting points could help handle that difficulty. In addition, it would be interesting to combine this reformulation with ideas from other SAT solving techniques, such as backtracking methods. This would help fixing some values of the variables. Combining the reformulation with a complete method could help improve convergence.

References

1. C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2017.
2. J. Chen and P. Z. G. Qian. Latin hypercube designs with controlled correlations and multi-dimensional stratification. *Biometrika*, 101(2):319–332, 2014.
3. R. Cont and M. Heidari. Optimal rounding under integer constraints. *CoRR*, abs/1501.00014, 2015.
4. M. Dressler, S. Iliman, and T. de Wolff. An approach to constrained polynomial optimization via nonnegative circuit polynomials and geometric programming. *J. Symb. Comput.*, 91:149–172, 2019.
5. J. Duda. P =? NP as minimization of degree 4 polynomial, or Grassmann number problem. *CoRR*, abs/1703.04456, 2017.
6. C. Fang and J. Liu. A linear algebra formulation for boolean satisfiability testing. *CoRR*, abs/1701.02401, 2017.
7. J. Gu. Global optimization for satisfiability (SAT) problem. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):361–381, June 1994.
8. J. P. Inala, S. Gao, S. Kong, and A. Solar-Lezama. REAS: combining numerical optimization with SAT solving. *CoRR*, abs/1802.04408, 2018.
9. R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.
10. Z. Li. *Polynomial Optimization Problems*. PhD thesis, The Chinese University of Hong Kong, 2011.
11. J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.
12. S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.
13. A. Thengade and R. Dondal. Genetic algorithm – survey paper. *IJCA Proc National Conference on Recent Trends in Computing, NCRTC*, 5, January 2012.