Applied Computing and Informatics (2015) 11, 47–59

Saudi Computer Society, King Saud University
Applied Computing and Informatics
(http://computer.org.sa)
www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

An optimization approach for the satisfiability problem

S. Noureddine

Prince Sultan University, Riyadh, P.O. Box 66833, Saudi Arabia

Available online 9 January 2012

KEYWORDS
Satisfiability;
NP-complete problem;
Optimization;
Geometric programming

Abstract We describe a new approach for solving the satisfiability problem by geometric programming. We focus on the theoretical background and give details of the algorithmic procedure. The algorithm is provably efficient as geometric programming is in essence a polynomial problem. The correctness of the algorithm is discussed. The version of the satisfiability problem we study is exact satisfiability with only positive variables, which is known to be NP-complete.
© 2012 Production and hosting by Elsevier B.V. on behalf of King Saud University.

1. Introduction

The satisfiability problem is still the most important open problem in computer science. Many scientific disciplines highly depend on efficient solutions of this core problem. These span computer science itself, mathematical logic, pure and applied mathematics, physics, chemistry, economics, and engineering, just to mention some. Approximations and heuristics are more than welcome in applications, since

E-mail address: Soufiane@psu.edu.sa
Peer review under responsibility of King Saud University.

http://dx.doi.org/10.1016/j.aci.2011.11.002
2210-8327 © 2012 Production and hosting by Elsevier B.V. on behalf of King Saud University.

they oftentimes provide efficient solutions for special cases. Theoretically, however, the computer science community is still lacking a good understanding of the computational difficulties of this problem though many facts about it are already known. Complexity theory tells us that the actual problem is related to the $P = NP$? question. Indeed, this question is now becoming a *complex* offered to us by complexity theory. Much effort to solve it has been in vain so far. Regrettably, no real progress in the discipline of computer science can be made until the problem is solved. Most (senior) computer scientists today rather believe that P is not equal to NP. That is, no matter how much we try, the satisfiability problem, as a prominent NP-complete problem, will persist as a challenging problem with exponential known deterministic worst-case complexity.

It is also disappointing that the NP-complete problems are too many and essential in real applications. It is rather simple to find a new NP-complete problem: just try to solve one of them and describe the main difficulty you encounter as a new problem if possible. So, adding a new problem to the list is relatively easy, but removing one from the list seems to be very hard. The reason is that the list of NP-complete problems degenerates to the empty list as soon as one of the problems is discovered to be outside the list, that is, as soon as one of the problems is proved to be P-complete. This very fact is the main achievement of the theory of NP-completeness.

Prevalent approaches for the satisfiability problem have concentrated so far on: logical manipulation of formulas, graph-theoretic algorithms, probabilistic algorithms, and mathematical optimization. Two methods based on logical manipulations are worth mentioning: the Davis Putnam (DPLL) procedure and the method of resolution and reduction. DPLL (Bacchus, 2002) proved extremely efficient for most SAT instances arising in practice. Propositional resolution (Robinson, 1965) is a well-understood powerful method. Propositional reduction (Noureddine, submitted for publication) is also powerful and works well in conjunction with resolution. Graph-theoretic algorithms (Aspvall et al., 1979) are extremely helpful for analyzing special algorithms (e.g. algorithms for 2SAT). Probabilistic algorithms (Schoening, 1999) are also of extreme importance in practice and might deliver the best results for complex instances. Finally, the method of mathematical optimization (Fletcher, 1987) treats the satisfiability problem as an optimization problem and not as a decision problem. The method is very promising in a real setting though implementing reliable optimization code is a very challenging task.

We favor in this paper the method of mathematical optimization. In another paper, we focused on formulating the satisfiability problem as non-convex, exact, exterior, penalty-based problem with a coercive objective function. The method focused on exact satisfiability (XSAT), which is NP-complete. The method falls into the category of approximation schemes for solving the satisfiability problem and is sub-optimal and partially heuristic in nature. In this paper, we treat the problem by way of geometric programming. We still focus on XSAT or more precisely on 3XSAT, which has the following properties:

1. Each clause includes exactly three literals.
2. Each satisfying assignment satisfies exactly one literal in each clause.
3. No variable in the formula is negated, that is, all literals are positive.

The third property simplifies the formulation of the problem via geometric programming and preserves NP-completeness (Schaefer, 1978).

To the knowledge of the author, geometric programming has not been used to attack the satisfiability problem yet. Certainly, other optimization schemes are well-known, but geometric programming theory is very promising in the context of satisfiability as we shall see. The method we provide is new and, thus, interesting by its own means. The characteristic feature of geometric programming is that it helps overcome the non-convex nature of the optimization problem under study. This property is of extreme utility in our setting, since direct optimization methods for the satisfiability problem tend to be non-convex. Knowing that general non-convex optimization problems are (still) not tractable, the method of geometric programming offers new perspectives for attacking the problem.

We will need to briefly review the basic theory of geometric programming in Section 2. The focus will be on the refined theory based on *strong duality* theorems as described by the inventors of geometric programming in Duffin et al. (1967). Strong duality optimization is not the standard way of geometric programming. However, we shall see that the *weak duality* theory is not sufficient in our case. Applications of the strong duality theory are rare, if any, in the literature. This is why the algorithm we shall outline in Section 3 is of importance only if the theory behind it is proved to be efficient in practice. We shall not focus on experimenting with the algorithm in this paper. The aim is rather to outline the abstract approach and underpin it by known theoretical results. As any other optimization algorithm, the implementation of our algorithm is not straightforward. This is even more complicated in the case of the satisfiability problem, inasmuch as testing the algorithm for small instances can be very well misleading and is by no means sufficient to draw sound conclusions. Sufficient evidence will be given, however, that the algorithm we present is always of polynomial complexity in worst case and correct in most cases. Section 4 discusses the approach, its limitations, and highlights related to research perspectives.

2. Theoretical background

This section is devoted to the introduction of the needed theory of optimization and to fixing our notation. Vector quantities are written in bold if not obvious from the context (e.g. $\mathbf{x}$). Vector components are indexed accordingly (e.g. x_i). Unless stated differently, we assume throughout the paper continuity and differentiability of used functions. This assumption is not restrictive in our context and is beneficial computationally. An optimization problem (P) is to find the minimum (or maximum) of a real-valued function $f(\mathbf{x})$, so-called *objective function*, satisfying a set inequality constraints $g_i(\mathbf{x}) \leqslant 0$ for $i \in I$ and a set of equality constraints $h_j(\mathbf{x}) = 0$ for $j \in E$, where I and E are possibly empty index sets. Formally:

$$\begin{aligned} (P) \quad & \text{minimize } f(\mathbf{x}) \\ & \text{subject to} \\ & g_i(\mathbf{x}) \leqslant 0 \text{ for } i \in I \\ & h_j(\mathbf{x}) = 0 \text{ for } j \in E. \end{aligned}$$

The *feasibility set* $\mathcal{I}(P)$ of the problem (P) is the set of vectors satisfying the constraints, that is:

$$\mathcal{I}(P) = \{\mathbf{x} \in \mathbf{R}^n : \; g_i(\mathbf{x}) \leqslant 0 \; \forall j \in E \text{ and } h_j(\mathbf{x}) = 0 \; \forall i \in I\}.$$

A solution of (P) is any vector $\mathbf{x}^* \in \mathbf{R}^n$ such that:

$$f(\mathbf{x}^*) = \min\{f(\mathbf{x}), \mathbf{x} \in \mathcal{I}\}.$$

In our context, we will focus on a special type of optimization problems, namely, *geometric programs*. In geometric programs only positive polynomials, so-called *posynomials*, are permissible as objective functions. Posynomials have the form:

$$\begin{aligned} & f(x) = \sum_{k \in K} u_k(x) \\ & \text{where} \quad u_k(x) = c_k \prod_{l \in L_k} x_l^{a_l} \\ & c_k \geqslant 0. \end{aligned}$$

The terms $u_k(x)$ are called *monomials*. As an example, the function defined by:

$$f(x) = 2x_1^2 x_2^{-5} x_3^e + x_2 + \pi x_1^{-2} x_2^3$$

is a posynomial and, therefore, permissible as an objective function for (P). In geometric programs, an inequality constraint has the form $g_i(\mathbf{x}) \leqslant 1$ and any such function $g_i(\mathbf{x})$ must be a posynomial. The equality constraints are of the form $h_j(\mathbf{x}) = 1$ and the functions $h_j(\mathbf{x})$ have to be monomials. Finally, in any geometric program all mentioned variables x_k have to be positive. Thus, a geometric program (P) has the form:

$$\begin{aligned} (P) \quad & \text{minimize } f(\mathbf{x}) \\ & \text{subject to} \\ & \quad g_i(\mathbf{x}) \leqslant 1 \quad \forall i \in I \\ & \quad h_j(\mathbf{x}) = 1 \quad \forall j \in E \\ & \quad x_k > 0 \quad \forall k \in \{1, \ldots, n\} \\ & \text{where} \\ & f(\mathbf{x}) \text{ and } g_i(\mathbf{x}) \text{ are posynomials and the } h_j(\mathbf{x}) \text{ are monomials.} \end{aligned}$$

The original theory of geometric programming lies essentially on the simple *Arithmetic-Geometric-Mean* inequality (AG). Though other inequalities can also be used to develop the same theory, we focus here on the AG inequality due to its simplicity and widespread use. The AG inequality says:

(AG) Let $2n$ numbers $x_i > 0$ and $\delta_i > 0$ for $i \in \{1,\dots,n\}$ be given with:
$\sum_{i=1}^{n} \delta_i = 1$
Then: $\prod_{i=1}^{n} x_i^{\delta_i} \leqslant \sum_{i=1}^{n} \delta_i x_i$
with equality *iff* $x_1 = \cdots = x_n$.

A related inequality (necessary for the constrained case) is the *General Arithmetic-Geometric-Mean* inequality (GAG). The GAG inequality is:

(GAG) Let $2n$ numbers $x_i > 0$ and δ_i for $i \in \{1,\dots,n\}$, where the δ_i are all positive or all zero, be given with:
$\sum_{i=1}^{n} \delta_i = \lambda$
Then:
$\lambda^{\lambda}\left(\prod_{i=1}^{n}\left(\frac{x_i}{\delta_i}\right)^{\delta_i}\right) \leqslant \left(\sum_{i=1}^{n} x_i\right)^{\lambda}$
with $0^0 = (x_i/0)^0 \equiv 1$ and equality *iff* $\delta_1 = \cdots = \delta_n = 0$ or $x_i = \frac{\delta_i}{\lambda}\sum_{j=1}^{n} x_j$.

With the two inequalities in hand, a duality theory can be easily developed. The process is *sketched* (see e.g. Duffin et al., 1967; Peressini et al., 1988 for more details) in the proof of the following main theorem of geometric programming.

Theorem 2.1. *For every geometric program (P) with $E = \emptyset$, called* primal *program, there exists a* dual *program (D) of the form:*

$$(D) \quad \text{maximize } v(\boldsymbol{\delta}) = \left(\prod_{i=1}^{n}\left(\frac{c_i}{\delta_i}\right)^{\delta_i}\right)\prod_{i\in I}\lambda_i(\delta)^{\lambda_i(\delta)}$$

subject to

$$\sum_{k\in I_0} \delta_k = 1 \qquad \text{(a)}$$

$$\sum_{k\in D} a_{ij}\delta_k = 0 \quad \forall j \in \{1,\dots,n\} \qquad \text{(b)}$$

$$\delta_k \geqslant 0 \quad \forall k \in D$$

where

$$\lambda_i(\delta) = \sum_{k\in I_i} \delta_k \quad \forall i \in I \text{ and } D = \cup_{k\geqslant 0} I_k.$$

Moreover, $f(\mathbf{x}) \geqslant v(\delta)$ for any two feasible vectors $\mathbf{x} \in \mathcal{I}(P)$ and $\delta \in \mathcal{I}(D)$. □

Proof. First, the requirement $E = \emptyset$ can be omitted. We will focus on inequality constraints only. The index sets I_k follow from the structure of the used posynomials. We will prove the theorem in the case of a single inequality constraint $g_1(\mathbf{x}) \leqslant 1$. The case of multiple constraints follows in the same manner. By (AG), we have:

$$f(\mathbf{x}) \geqslant \left(\prod_{i=1}^{n_0} \left(\frac{c_i}{\delta_i} \right)^{\delta_i} \right) \prod_{i \in I_0} x_i^{L_i}. \tag{2.1}$$

On the other hand, a lower bound for the function $g_1(\mathbf{x})^\lambda$ (for $\lambda > 0$) is obtained by (GAG):

$$g_1(\mathbf{x})^\lambda \geqslant \lambda^\lambda \left(\prod_{i=n_0+1}^{n_1} \left(\frac{c_i}{\delta_i} \right)^{\delta_i} \right) \prod_{i \in I_1} x_i^{L_i}. \tag{2.2}$$

Here, the index sets I_0 and I_1 capture the indexes of the different variable x_i mentioned in $f(\mathbf{x})$ and $g_1(\mathbf{x})$, respectively, and each L_i is a linear combination of the form (a).

Since $g_1(\mathbf{x}) \leqslant 1$, it follows that:

$$f(\mathbf{x}) \geqslant f(\mathbf{x}) g_1(\mathbf{x})^\lambda \geqslant \left(\prod_{i=1}^{n_0} \left(\frac{c_i}{\delta_i} \right)^{\delta_i} \right) \prod_{i \in I_0} x_i^{L_i} \lambda^\lambda \left(\prod_{i=n_0+1}^{n_1} \left(\frac{c_i}{\delta_i} \right)^{\delta_i} \right) \prod_{i \in I_1} x_i^{L_i}. \tag{2.3}$$

By requiring that each $L_i = 0$ as in (a), the right-hand side of (2.3) becomes a function of the variables δ_k. Calling this function $v(\delta)$, (2.3) becomes:

$$f(\mathbf{x}) \geqslant v(\boldsymbol{\delta}).$$

Obviously, the above derivation is valid only if $\mathbf{x}$ and δ are feasible as required. □

The above theorem is the corner stone of the computational procedure for solving geometric problems. The main facts in this respect are summarized in the following theorem, which we state without proof (see e.g. Duffin et al., 1967).

Theorem 2.2

(i) *The logarithm of dual objective function $ln(v(\delta))$ is concave.*
(ii) *If the interior of $\mathcal{I}(P)$ is not empty and (P) has a (finite) solution $\mathbf{x}^*$, then (D) also has a solution δ^*. Moreover:*

$$f(x^*) = v(\delta^*).$$

□

In other words, fact (*i*) says (as $\ln(x)$ is monotone-increasing) that the dual program (*D*) is efficiently solvable, since it is in essence a (simple) convex program. Polynomial algorithms for convex programs are known (e.g. interior point methods (Forsgren et al., 2002)). Fact (*ii*) is saying that once we have solved the dual problem (*D*), the primal problem (*P*) is practically solved, too. Although only the minimum of $f(\mathbf{x}^*)$ is determined by solving the dual problem, a method for determining a minimizer $\mathbf{x}^*$ is known to be equivalent to a linear program and, therefore, the problem of determining $\mathbf{x}^*$ is solvable in polynomial time.

The theory outlined above is called *weak duality theory*. It is weak in the sense that fact (*ii*) requires the feasible set $\mathcal{I}(P)$ to have a non-empty interior. Such programs are called *super-consistent*. On the other hand, if $\mathcal{I}(P)$ is non-empty the program is called *consistent*. However, under further mild requirements, the super-consistency assumption can be omitted. This leads to the so-called *strong duality theory*. The following theorem is the main theorem of strong duality.

Theorem 2.3. *If the primal program (P) is consistent, then its dual (D) is consistent the its infimum, if it exists, coincides with the supremum of (D).* □

This theorem relaxes the requirement of super-consistency: it only requires that the primal is consistent. The price of this relaxation is reflected in the conclusion that just the primal infimum and the dual supremum coincide (if existent). Compared with Theorem 2.2 this is a weaker conclusion, since there the primal minimum and dual maximum coincide if they exist. However, Theorem 2.3 is only theoretically weaker. In practice, the conclusion is sufficient for devising an algorithmic procedure to approximate the minimum of the primal problem and/or the maximum of the dual problem.

Theorem 2.3 opens a new way for geometric programming if used effectively. The idea we now introduce shall try to attack the open problem of solving *geometric programs with equality constraints for posynomials*. So far, we have allowed equality constraints for monomials only. This requirement is related to the efficient solvability of the problem via convex techniques. Theorem 2.3 tells us, however, that consistency of the primal program is sufficient for the mentioned infimum–supremum correspondence if a suitable dual program is defined. As a matter of fact, we are facing the question: Is there a dual program for geometric programs with posynomial equality constraints? The answer to this question is yes as seen in the following theorem.

Theorem 2.4. *Let (P) be a primal program with posynomial equality constraints. Then (D) as defined above is the dual program of (P). Moreover, Theorem 2.1 applies for (P) and (D), in particular for feasible* $\mathbf{x}$ *and* δ *we have:* $f(\mathbf{x}) \geqslant v(\delta)$. □

Proof. The proof of Theorem 2.1 can be reused here without modification for posynomial inequality constraints. For posynomial equality constraints, we just point out that line (2.3) of that proof should include the equality symbol instead of the leftmost inequality symbol. However, this does not impair the validity of other inequalities. Thus, the main conclusion of the theorem, namely, that $f(\mathbf{x}) \geqslant v(\delta)$ for suitable $\mathbf{x}$ and δ, remains true here, too. □

The following theorem connects the preceding theorem to the task of finding the infimum of the primal program.

Theorem 2.5. *Let (P) be a consistent primal program with posynomial equality constraints and (D) be its dual program. Then (D) is consistent and the infimum of (P) coincides with supremum of (D).* □

Proof. Since per Theorem 2.4 the dual program (*D*) is well-defined, Theorem 2.3 can be used to infer the infimum–supremum correspondence as required. □

Observe that the requirement of consistency is crucial in Theorem 2.5. Just super-consistency would not suffice here, since the primal program is allowed to have posynomial equalities.

Corollary 2.6. *Let (P) be as in Theorem 2.5, then its infimum coincides with the constrained maximum of (D).*

Proof. As (*D*) is equivalent to a convex program, its supremum is actually its maximum, whence the desired proposition. □

The last claim is of no theoretical importance but useful in practice. It helps deal with the maximization algorithm of the dual (concave) program in the classical (efficient) way.

One remark is here in order. Paradoxically, the preceding theorems show that equality constraints are dealt with as inequality constraints if only the primal is consistent and the optimization is changed so as to find the infimum instead of the minimum. This is so, since Theorem 2.4 shows that the same lower bound is used for equality and inequality constraints. This is rather strange, as the dual program would lose information about the constraint types of the primal program. This is why the new introduced theorems are rather in the status of conjunctures until more theoretical and/or experimental evidence about them is demonstrated. One way of accepting this paradox, however, is to imagine that proving consistency of the primal program with posynomial equalities is by itself a serious difficulty in practice, so as to make the conclusions of the last theorems of theoretical interest only save perhaps for special (non-)interesting cases.

3. The algorithm

We finally come back to the original problem of the paper, the satisfiability problem. In the last section, we proved that equality constraints act theoretically like inequality constraints, if the original primal program is consistent. Algorithms for handling equality constraints are known. Actually, one of the inventors of geometric programming, Duffin pointed out (Duffin, 1970) that the method of *posynomial condensation* could be of great use in practice to approximate geometric programs by linear ones. This method is useful in our setting, too. The idea is to replace any posynomial equality constraint, e.g. $g(\mathbf{x}) = 1$, by the (in)equalities:

$$g'(\mathbf{x}) = 1,$$

where $g'(\mathbf{x})$ is the monomial obtained by direct use of (AG) for $g(\mathbf{x})$ i.e.: $g'(\mathbf{x}) \leqslant g(\mathbf{x})$.

It is clear that the method of condensation only approximates the original problem. However, good sub-optimal solutions can be achieved by this method.

The method of condensation does not rely on the strong duality theory. It circumvents the difficulty of equalities by direct approximation. We will not use it in this paper. We rather intend to apply the strong duality results of the last section. However, to really benefit from these results, we need to require that the consistency problem is (efficiently) solvable by some means. Though this requirement seems to be unachievable in general, there are special cases where it is easily achieved. A noteworthy case is when all constraints are posynomial equality constraints where each term is a simple term (i.e. consists of a single variable with 1 as exponent). We call such a function a *linear posynomial*. In this case, the consistency problem boils down to a linear programming problem, which is efficiently solvable by conventional techniques. With the help of the strong duality theory, an efficient method for minimization is also at hand. Though, this scheme only delivers infimum information, in applications this information does not differ much from minimum information, as optimization algorithms are approximative in nature. This is the way we are going to go for solving the satisfiability problem.

Formally, let the primal optimization problem (P) to be solved be of the form:

$$\begin{aligned}
(P)\quad & \text{minimize } f(\mathbf{x}) \\
& \text{subject to} \\
& \quad g_i(\mathbf{x}) = 1 \quad \forall i \in E \\
& \quad x_k > 0 \quad \forall k \in \{1, \ldots, n\} \\
& \text{where} \\
& f(\mathbf{x}) \text{ is a posynomial and the } g_i(\mathbf{x}) \text{ are linear posynomials.}
\end{aligned}$$

An algorithm for finding the infimum of the problem is as follows:

Algorithm 3.1

```
Input :   Problem $(P)$
Output :  Infimum of $(P)$ if it exists.

Step1: Consistency Check
Solve the linear programming problem (LP):
    $g_i(\mathbf{x}) = 1 \quad \forall i \in E$
    $x_k > 0 \qquad \forall k \in 1, \ldots, n.$
If (LP) is not solvable then $(P)$ is not solvable; stop.
Step 2: Solve the consistent $(P)$ by its dual $(D)$
Get $M$: = constrained maximum of $(D)$;
Return $M$;
```

Claim 3.1. *The algorithm is of polynomial complexity.* □

Proof. Step 1 is essentially a perturbed linear programming problem (due to the existence of strict inequalities). So, it is solvable in polynomial time. Step 2 is a convex problem, which is harder to solve, but is still of polynomial complexity. □

Claim 3.2. *The algorithm is correct.* □

Proof. The correctness of the algorithm relies directly on the results of Section 2. We need to emphasize, however, that only the infimum is approximated by the method and to refer to the remarks at the end of last section. □

The presented algorithm is of use for the exact satisfiability problem (XSAT), if we can convert (XSAT) into an optimization problem of the form (P). With Claim 3.1, we are guaranteed to have a polynomial-time algorithm then. In view of the remarks of the last section, Claim 3.2 is still to be justified experimentally. In any case, a conversion of (XSAT) to (P) is of great importance in our setting. To this end, we first recall that our XSAT instances are required to be positive (i.e. without negated literals). Let $C_1 = x_1 + x_2 + x_3$ be a clause of such an XSAT formula F. We first notice that the following system of equations:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ x_1 x_2 = 0 \\ x_1 x_3 = 0 \\ x_2 x_3 = 0 \end{cases}$$

has $S = \{(1,0,0),(0,1,0),(0,0,1)\}$ as a solution set. Remarkably, this is exactly the solution set of a permissible 3XSAT clause in a satisfiable formula F. We might better see the previous system of equations as an optimization problem (P_{C_1}) for clause C_1, where (P_{C_1}) is:

$$\begin{cases} \text{Minimize } x_1x_2 + x_1x_3 + x_2x_3 \\ \text{subject to} \\ \quad x_1 + x_2 + x_3 = 1 \\ \quad x_1 \geqslant 0 \\ \quad x_2 \geqslant 0 \\ \quad x_3 \geqslant 0. \end{cases}$$

Let the above objective function of clause C_1 be called $f_1(\mathbf{x})$ and the equality constraint function be called $g_1(\mathbf{x})$. Obviously, the same procedure can be used for any clause $C_i(1 < -i \leqslant m)$ if the formula F includes m clauses, and for each clause C_i functions $f_i(\mathbf{x})$ and $g_i(\mathbf{x})$ can be defined accordingly. To solve the XSAT problem, we need to combine the equations of the different programs (P_{C_i})'s. This yields to the following formulation of the XSAT problem as an optimization problem

$$\begin{aligned} \text{(XSAT)} \quad & \text{minimize } \sum_{i=1}^{m} f_i(\mathbf{x}) \\ & \text{subject to} \\ & \quad g_i(\mathbf{x}) = 1 \quad \forall i \in \{1,\ldots,m\} \\ & \quad x_k \geqslant 0 \quad \forall k \in \{1,\ldots,n\}. \end{aligned}$$

By construction of $f_i(\mathbf{x})$ and $g_i(\mathbf{x})$, we evidently see that these are posynomials and linear posynomials, respectively. Also, the positivity constraints of the variables can be easily perturbed to strict positivity constraints. Thus, in fact, we need to solve (XSAT) for positive x_k only. But now all assumptions of aforementioned algorithm are satisfied, whence the following claim.

Claim 3.3. *The optimization problem (XSAT) with strict positivity constraints is solvable in polynomial time.* □

We emphasize that by "solvable" we mean that the solution can be approximated to any desired degree of accuracy. This so because our algorithm delivers the infimum and (XSAT) needs to be perturbed numerically.

Claim 3.4. *The 3XSAT with positive literals is solvable in polynomial time.* □

Proof. A procedure for solving positive XSAT is straightforward:

```
1.   Solve the corresponding geometric programming problem (XSAT)
     by Algorithm 3.1 and get its infimum $M$.
2.   **If** $M \approx 0$ **then**
          print *satisfiable*
     **else**   print *unsatisfiable*.
```

□

The previous procedure is a decision procedure. To obtain a solution vector, bisection in variables' vector space is the immediate approach. Thus, if F is satisfiable, the procedure in the previous proof needs to be called O(n) times, where n is the number of variables mentioned in F. We cannot rely here on the approach of geometric programming for determining minimizers (via linear programming), since this method relies on the super-consistency assumption of the problem to be minimized, which is not allowed in our setting.

4. Conclusion

This paper has two main contributions:

We revived the theory of strong duality in geometric programming. We easily extended the theory to handle posynomial equality constraints. At this seemed to be only of theoretical interest, but we argued that in special cases (linear posynomials) the theory extremely useful in practice and we outlined a polynomial-time algorithm for solving this sort of problems.

The second main contribution is related to satisfiability research. We proposed a new format of the (positive) 3XSAT problem as a geometric program. We then proved that our format adheres to the requirement of linear posynomial equalities. Thus, we were able to apply the proposed optimization algorithm for XSAT.

There is no doubt that both Algorithm 3.1 and the procedure outlined in the proof of Claim 3.4 are polynomial-time algorithms. However, the issue that still needs further investigation is the (practical) correctness of these procedures. We intend to verify correctness experimentally. The main drawback of this approach is that faults in the implementation of mentioned optimization algorithms inevitably lead to erroneous conclusions. The paper shows, however, that the theory behind the algorithms is sound and that it predicts polynomial-time performance (with exact upper bounds). Despite this fact, a warning is here in order. Experience with other optimization methods has shown that theoretical investigation is not the whole story in this domain. One need only consider the Ellipsoid Method of optimization, which is provably of polynomial-time complexity, and which, however, has poor performance in practice compared to the theoretically inferior Simplex Method with its exponential worst-case complexity.

We finally want to point out that the optimization problem for XSAT as defined in this paper calls for another totally different way of solution. We could, namely, try to attack the problem via quasi-convex programming. Quasi-convexity of constraint functions is sufficient for minimization algorithms based on the Karush–Kuhn–Tucker theory and may be implemented efficiently, if the objective function is convex or at least pseudo-convex. The performance and reliability (i.e. performability) of this approach will be part of our future research.

References

Aspvall, B. et al., 1979. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. Information Processing Letters 8 (3), 121–123.

Bacchus, F., 2002. Enhancing Davis Putnam with extended binary clause reasoning. In: 18th National Conference on Artificial Intelligence.

Duffin, R.J., 1970. Linearizing geometric programs. SIAM Review 12 (2), 211–227.

Duffin, R.J. et al., 1967. Geometric Programming: Theory and Applications. Wiley, New York.

Fletcher, S.R., 1987. Practical Methods of Optimization, second ed. Wiley.

Forsgren, A. et al., 2002. Interior methods for nonlinear optimization. SIAM Review 44 (4), 925–997.

Noureddine, S., submitted for publication. An approach for the satisfiability problem via exterior penalty optimization, Journal of Computer Science.

Peressini, A.L. et al., 1988. The Mathematics of Nonlinear Programming. Springer-Verlag, New York.

Robinson, J.A., 1965. A machine-oriented logic based on the resolution principle. Journal of the Association for Computing Machinery 12, 23–41.

Schaefer, T.J., 1978. The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, pp. 216–226.

Schoening, U., 1999. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS'99, pp. 410–414.