Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 2070–2078
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Semidefinite programming and arithmetic circuit evaluation

Sergey P. Tarasov, Mikhail N. Vyalyi

Dorodnitsyn Computing Center, Vavilova 40, Moscow 119991, Russia

Received 1 October 2005; received in revised form 11 April 2006; accepted 24 April 2007
Available online 17 May 2007
Communicated by E. Boros

Abstract

We address the exact semidefinite programming feasibility problem (SDFP) consisting in checking that intersection of the cone of positive semidefinite matrices and some affine subspace of matrices with rational entries is not empty. SDFP is a convex programming problem and is often considered as tractable since some of its approximate versions can be efficiently solved, e.g. by the ellipsoid algorithm.

We prove that SDFP can decide comparison of numbers represented by the arithmetic circuits, i.e. circuits that use standard arithmetical operations as gates. Our reduction may give evidence to the intrinsic difficulty of SDFP (contrary to the common expectations) and clarify the complexity status of the exact SDP—an old open problem in the field of mathematical programming.

Keywords: Semidefinite programming; Complexity; Succinct representation

1. Introduction

Algorithmic complexity provides a general framework to analyze complexity of computational problems. It works for many cases and gives results that are important for practical applications. Nevertheless, some basic assumptions of the theory are strange from practical point of view. By definition, all linear time algorithms are efficient though a $10^{1000}n$-algorithm is for sure practically inefficient. Also, some widely used algorithms have exponential running time in the worst case.

Numerical algorithms are especially in striking disagreement with complexity theory. Probably, the most popular exponential algorithm is the simplex algorithm for linear programming. It is widely used despite the existence of polynomial algorithms that were found after pioneering breakthrough of Khachiyan (see, e.g., [6,5,11]).

The complexity analysis of the semidefinite programming (SDP) problem involves even more difficulties. SDP is often considered as tractable due to various approximation algorithms. SDP is a convex optimization problem so the ellipsoid method can be applied to solve it approximately as well as a variety of interior points methods [4,8,12,18]. But there are amazingly few results on the complexity of the SDP problem.

Khachiyan and Porkolab [9] found a polynomial time algorithm for the SDP problem when the dimension is fixed. They established doubly exponential bounds on the solutions for the general SDP problem and on the discrepancies of infeasible programs.

E-mail addresses: serge99meister@gmail.com (S.P. Tarasov), vyalyi@mccme.ru (M.N. Vyalyi).

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2007.04.023

For our further considerations the most important are Ramana's results [10]. He developed the exact duality theory for SDP. Ramana's dual program can be constructed in polynomial time. His analogue of Farkas lemma has an immediate complexity-theoretic corollary: a complement to the SDP feasibility problem (SDFP) can be reduced to SDFP itself. It implies that SDFP cannot be NP-complete unless NP = coNP. More generally, SDFP can be put to a complexity class that is closed under complement. Examples are P, NP ∩ coNP, BPP, PSPACE. (For definitions of these classes and other useful information on complexity theory see, e.g., the book [15].)

Here we address the exact SDFP. The problem is to check that intersection of the cone of positive semidefinite matrices with some affine subspace of matrices is not empty. The subspace involved is defined by generators that are matrices with rational entries. It is well known that some pathological examples exist for the SDFP. For instance, it is possible that a feasible program has only doubly exponential solutions and that an infeasible program could have doubly exponentially small discrepancy. These examples show that in some cases a polynomially bounded machine cannot even write a feasible solution of SDFP, though it is of course possible that the existence of a solution could be checked in polynomial time.

Motivated by these examples, we relate SDP to *arithmetic circuits* (ACs) as they provide a nonstandard way of representing integers and rationals that enables to perform arithmetic operations on some doubly exponentially long numbers.

The common way to represent integers is to use a positional system, say, the binary system. Binary representation of a number N is a string of length $\theta(\log N)$. This bound is optimal due to the counting argument. But it is also possible to encode numbers in such a way that some numbers are encoded by very short strings. In this case we speak of a *succinct representation* of an integer. A natural way for succinct representation of a rational number r is to use an *arithmetic computation* or an AC. By definition, AC is a sequence of the elementary arithmetic operations starting from a fixed constant, say 1, and generating r as output.

In this paper we address a problem of the complexity of performing arithmetic operations and computing elementary predicates, e.g., "=" or "$\geqslant$", on rational numbers represented by AC.

Note that if numbers are represented by AC then to perform an elementary arithmetic operation one should simply merge the ACs of the operands in the appropriate way, thus arithmetic operations are easy to perform, despite the evident fact that AC representation is succinct for some numbers (e.g., 2^{2^n} can be obtained by repeated squaring). On the other hand, it is unclear how to compute efficiently elementary predicates "=" or "$\geqslant$", i.e. how to check efficiently whether two ACs compute the same value or how to compute the maximum. In the sequel we denote these predicates by $\mathrm{AC}_=$ and $\mathrm{AC}_\geqslant$, respectively.[1] The equality predicate $\mathrm{AC}_=$ is contained in the complexity class coRP (see [13]) and thus can be efficiently checked probabilistically. It is shown in [1] that the inequality predicate is contained in the counting hierarchy and thus can be checked in polynomial space. Also [1] gives some evidence in favor of the computational intractability of the predicate $\mathrm{AC}_\geqslant$. Unfortunately, no hardness result for this predicate is known.

Here we reduce the predicate $\mathrm{AC}_\geqslant$ to SDFP. More exactly, we prove that some restricted version of AC can be efficiently simulated by the exact SDP and construct a polynomial reduction of $\mathrm{AC}_\geqslant$ to SDFP based on this simulation.

So any nontrivial lower bound for the complexity of the $\mathrm{AC}_\geqslant$ predicate would imply lower bounds for the exact SDP—one of the main open problems in the field of mathematical programming.

It is worth mentioning that the complexity issues of the AC representation were studied in the framework of the algebraic complexity theory over a field introduced by Smale (see, e.g., [14,7]). Namely, let $\tau(k)$ be the minimum number of arithmetic operations required to build the integer k from the constant, say 1, i.e. the length of the minimal AC computing k. A sequence x_k of integers is said to be "ultimately easy to compute" if there exists another sequence a_k and a polynomial $p(\cdot)$ such that $\tau(a_k x_k) \leqslant p(\log k)$ for all k (for instance, in can be shown that the sequence 2^k is ultimately easy to compute). Otherwise the sequence is said to be "ultimately hard to compute". The counting argument shows that ultimately hard to compute sequences do exist. A central open problem here is to show that some explicit sequences, say, $n!$, $\lfloor (3/2)^n \rfloor$ are ultimately hard to compute. For instance, if $n!$ is ultimately hard to compute then over the field of complex numbers the analogues of the classes P and NP coincide [14].

In [7] the complexity of AC representation is related to the circuit model of computation for polynomials that was introduced and studied by Valiant [16].

[1] In [1] they are denoted by EquSLP and PosSLP, respectively.

In [1] the complexity of $\mathrm{AC}_{\geqslant}$ is related to some fundamental problems of numerical computations and to the Smale complexity.

The rest of paper is organized as follows. In Section 2 we give a definition of the circuit representation for rationals and state the basic results concerning it. In Section 3 we construct a reduction of $\mathrm{AC}_{\geqslant}$ to SDFP. In Section 4 we discuss open questions around the $\mathrm{AC}_{\geqslant}$ and SDFPs problems.

2. AC representation of numbers

Let $\mathscr{B}$ be a finite collection of functions of type $\mathbb{Q}^k \to \mathbb{Q}$ ($k \in \{1, 2\}$). It is called a *basis*. A *circuit* over basis $\mathscr{B}$ is a sequence $\mathscr{S} = s_0, s_1, \ldots, s_\ell$ of assignments such that $s_0 := 1$ and for each $i \geqslant 1$ either $s_i := f_i(s_j, s_k)$ or $s_i := f_i(s_j)$ where $f_i \in \mathscr{B}$ and $j, k < i$. The *size* $\ell(\mathscr{S})$ of a circuit $\mathscr{S}$ is the number ℓ.

A natural basis consists of four arithmetic operations $\{+, -, /, \cdot\}$. Circuits over this basis are called AC. Circuits over the basis $\{+, -, \cdot\}$ are called *division-free* circuits. *Monotone* (arithmetic) circuits are circuits over the basis $\{+, \cdot\}$.

We will represent rationals by circuits. For each circuit we define the *value* of the circuit by induction. We will use notation $v(\mathscr{S})$ for the value of a circuit $\mathscr{S}$. Value of 1 is 1. The value of a circuit $\mathscr{S} = s_0, s_1, \ldots, s_\ell$ where $s_\ell = s_j * s_k$ is $v(\mathscr{S}_j) * v(\mathscr{S}_k)$. Here $\mathscr{S}_j = s_0, s_1, \ldots, s_j$. Note that each prefix of a circuit is a circuit by definition. If some operations cannot be performed (e.g., a division by 0) the value of such circuit is undefined. Let $X = v(\mathscr{S})$. We say that such AC *represents X*.

It is easy to see that AC representation can be much more compressed than the usual binary representation. A circuit $\mathscr{S} = 1, s_1, s_2, \ldots, s_\ell$ where $s_1 = 1 + 1$ and $s_{j+1} = s_j \cdot s_j$ for $j \geqslant 1$ represents $2^{2^{\ell-1}}$. On the other hand, this example is asymptotically optimal. The following statement can be easily verified by induction.

Statement 1. *If $p/q = v(\mathscr{S})$ where p, q are integers then* $\max\{p, q\} = \mathrm{O}(2^{2^\ell})$.

2.1. The complexity of equality and inequality predicates over AC

Implementation of arithmetic operations with rationals represented by AC is straightforward and can be done in linear time. Formally we write

$$\mathscr{S}(X * Y) = \mathscr{S}(X), \quad \mathrm{tail}(\mathscr{S}(Y)), \quad s_{\ell(\mathscr{S}(X))+\ell(\mathscr{S}(Y)-1)},$$
$$\text{where } s_{\ell(\mathscr{S}(X))+\ell(\mathscr{S}(Y))-1} := s_{\ell(\mathscr{S}(X))} * s_{\ell(\mathscr{S}(X))+\ell(\mathscr{S}(Y))-2}.$$

Here $\mathrm{tail}(\mathscr{S})$ denotes a circuit $\mathscr{S}$ without the starting 1.

But how difficult can be the computation of the equality predicate and of the inequality predicate? Let us state these algorithmic problems formally. We always assume that a circuit is represented by a list of triples $(*, j, k)$ where $* \in \mathscr{B}$ and j, k are positive integers. The nth element in the list corresponds to an assignment $s_n = s_j * s_k$. So a circuit of size ℓ is written as a $\mathrm{O}(\ell \log \ell)$ binary word.

Predicate $\mathrm{AC}_{=}(\mathscr{B})$**.** It is true for a pair of circuits $\mathscr{S}_1, \mathscr{S}_2$ over the basis $\mathscr{B}$ iff $v(\mathscr{S}_1) = v(\mathscr{S}_2)$.

Predicate $\mathrm{AC}_{\geqslant}(\mathscr{B})$**.** It is true for a pair of circuits $\mathscr{S}_1, \mathscr{S}_2$ over the basis $\mathscr{B}$ iff $v(\mathscr{S}_1) \geqslant v(\mathscr{S}_2)$.

Note. If $\mathscr{B}$ contains division then these predicates are partially defined (values of some circuits are undefined). Partially defined predicates are called *promise problems*. Most complexity classes can be easily redefined to include promise problems and most results remain true in this, more general, setting. We omit the discussion of promise problems but indicate that their use is safe in our considerations.

We denote the equality and the inequality predicates over the arithmetic basis by $\mathrm{AC}_{=}$ (resp. $\mathrm{AC}_{\geqslant}$).

It is clear that both predicates fall into the complexity class EXPTIME but fortunately they are known to be in the lower levels of the computational hierarchy. By results of [13] $\mathrm{AC}_{=} \in \mathrm{coRP}$.

In other words, the equality check can be performed by a probabilistic Turing machine with one-sided error in polynomial time.

We are unable to give the exact characterization of the computational complexity of the $\mathrm{AC}_{\geqslant}$-predicate. Computing $\mathrm{AC}_{\geqslant}$ looks as a computationally hard problem. An obvious way to solve it is to make all calculations indicated in the circuits that form the input of the problem. Using binary representation we need exponentially large memory to do it. It follows from [3] that $\mathrm{AC}_{\geqslant} \in \mathrm{PSPACE/poly}$. This result is strengthened in [1], where $\mathrm{AC}_{\geqslant}$ is put in the third level of the counting hierarchy CH introduced in [17] and hence in PSPACE.

2.2. Equivalent bases

Two bases $\mathscr{B}_1$ and $\mathscr{B}_2$ are called "=" *equivalent* (resp. "$\geqslant$" *equivalent*) iff the predicates $\mathrm{AC}_{=}(\mathscr{B}_1)$ and $\mathrm{AC}_{=}(\mathscr{B}_2)$ (resp. $\mathrm{AC}_{\geqslant}(\mathscr{B}_1)$ and $\mathrm{AC}_{\geqslant}(\mathscr{B}_2)$) are mutually polynomially reducible.

Theorem 1. *The following bases are* "=" *and* "$\geqslant$" *equivalent*: *arithmetic*, *division-free*, *monotone and* $\{+, x \mapsto x^2/2\}$.

The last basis in the list is added for technical purposes. It is used in the reduction of the problem $\mathrm{AC}_{\geqslant}$ to the feasibility problem for SDP.

Reductions of all mentioned bases to the arithmetic basis are straightforward. To prove Theorem 1 we establish reductions in the opposite direction.

Note that if a basis contains a subtraction then a general predicate $\mathrm{AC}_{\geqslant}(\mathscr{B})$ is reducible to its particular case when one of the compared numbers is zero. Indeed, suppose we are going to compare $v(\mathscr{S}_1)$ and $v(\mathscr{S}_2)$. We can merge the circuits $\mathscr{S}_1$, $\mathscr{S}_2$ into one circuit. This merged circuit contains all assignments of $\mathscr{S}_1$, $\mathscr{S}_2$ and ends by the assignment $d := a - b$, where a and b are the last assignments in the circuits $\mathscr{S}_1$, $\mathscr{S}_2$. The same argument can be also applied to the predicate $\mathrm{AC}_{=}(\mathscr{B})$.

All reductions described below have similar form. A circuit $\mathscr{S}$ over some basis is converted to a circuit $\mathscr{S}'$ over another basis using step-by-step substitution of constant-sized groups of assignments instead of each assignment in $\mathscr{S}$.

Lemma 1. $\mathrm{AC}_{\geqslant}(\{+, -, /, \cdot\})$ (*resp.* $\mathrm{AC}_{=}(\{+, -, /, \cdot\})$) *is reducible to* $\mathrm{AC}_{\geqslant}(\{+, -, \cdot\})$ (*resp.* $\mathrm{AC}_{=}(\{+, -, \cdot\})$).

Proof. Informally speaking, the lemma is very simple: we can keep numerators and denominators separately. Below we present a more detailed description of the reduction.

Let $\mathscr{S}$ be a circuit of size ℓ over the arithmetic basis. We construct a circuit $\mathscr{S}'$ of size $\mathrm{O}(\ell)$ over the division-free basis in the following way.

The circuit $\mathscr{S}'$ consists of four sequences of assignments $\mathscr{A}^1$, $\mathscr{A}^2$, $\mathscr{N}$, $\mathscr{D}$. For the assignment $s_i := s_j \pm s_k$ in $\mathscr{S}$ add the assignments

$$\mathscr{A}^1_i := \mathscr{N}_j \cdot \mathscr{D}_k, \quad \mathscr{A}^2_i := \mathscr{D}_j \cdot \mathscr{N}_k, \quad \mathscr{D}_i := \mathscr{D}_j \cdot \mathscr{D}_k, \quad \mathscr{N}_i := \mathscr{A}^1_i \pm \mathscr{A}^2_i.$$

Similarly, for the assignment $s_i := s_j \cdot s_k$ in $\mathscr{S}$ add the assignments

$$\mathscr{D}_i := \mathscr{D}_j \cdot \mathscr{D}_k, \quad \mathscr{N}_i := \mathscr{N}_j \cdot \mathscr{N}_k,$$

and for the assignment $s_i := s_j/s_k$ in $\mathscr{S}$ add the assignments

$$\mathscr{D}_i := \mathscr{D}_j \cdot \mathscr{N}_k, \quad \mathscr{N}_i := \mathscr{N}_j \cdot \mathscr{D}_k.$$

The last assignment in the circuit $\mathscr{S}'$ is

$$s'_N := \mathscr{N}_\ell \cdot \mathscr{D}_\ell.$$

It is easy to see that

$$v(\mathscr{S}) = \frac{v(\mathscr{N}_\ell)}{v(\mathscr{D}_\ell)}.$$

So, $v(\mathscr{S}) \geqslant 0$ iff $s'_N \geqslant 0$.

Note that due to our assumptions the case $\mathscr{D}_\ell = 0$ is impossible. So, the same reduction is valid for the predicate $\mathrm{AC}_=$. □

Lemma 2. $\mathrm{AC}_\geqslant(\{+,-,\cdot\})$ (*resp.* $\mathrm{AC}_=(\{+,-,\cdot\})$) *is reducible to* $\mathrm{AC}_\geqslant(\{+,\cdot\})$ (*resp.* $\mathrm{AC}_=(\{+,\cdot\})$).

Proof. Informally, we do the same trick as above using identities

$$(A-B)+(C-D)=(A+C)-(B+D), \tag{1}$$

$$(A-B)-(C-D)=(A+D)-(B+C), \tag{2}$$

$$(A-B)\cdot(C-D)=(A\cdot C+B\cdot D)-(B\cdot C+A\cdot D). \tag{3}$$

Now let us consider the details of the reduction.

At first we convert an input $(\mathscr{S}_1,\mathscr{S}_2)$ of $\mathrm{AC}_\geqslant(\{+,-,/,\cdot\})$ into $(\mathscr{S},0)$ as it was explained above.

Then we construct two circuits $\mathscr{L}$, $\mathscr{R}$ such that $v(\mathscr{S})=v(\mathscr{L})-v(\mathscr{R})$. So, $v(\mathscr{S})\geqslant 0$ iff $v(\mathscr{L})\geqslant v(\mathscr{R})$ as well as $v(\mathscr{S})=0$ iff $v(\mathscr{L})=v(\mathscr{R})$. Again, the same reduction will work for both predicates. The size of $\mathscr{L}$, $\mathscr{R}$ will be $\mathrm{O}(s(\mathscr{S}))$ and they will be the circuits over the basis $\{+,\cdot\}$.

Both circuits $\mathscr{L}$ and $\mathscr{R}$ consist of six series of assignments $\mathscr{L}^1$, $\mathscr{L}^2$, $\mathscr{L}^3$, $\mathscr{R}^1$, $\mathscr{R}^2$, $\mathscr{R}^3$, where the circuit $\mathscr{L}$ has a form

$$\ldots,\mathscr{R}^3_i,\mathscr{R}^2_i,\mathscr{R}^1_i,\mathscr{L}^3_i,\mathscr{L}^2_i,\mathscr{L}^1_i,\ldots$$

and the circuit $\mathscr{R}$ has a form

$$\ldots,\mathscr{L}^3_i,\mathscr{L}^2_i,\mathscr{L}^1_i,\mathscr{R}^3_i,\mathscr{R}^2_i,\mathscr{R}^1_i,\ldots$$

The series of assignments $\mathscr{L}^i_j$ and $\mathscr{R}^i_j$ are specified as follows.

For the assignment $s_i := s_j + s_k$ in $\mathscr{S}$ set

$$\mathscr{L}^1_i := \mathscr{L}^1_j + \mathscr{L}^1_k, \quad \mathscr{R}^1_i := \mathscr{R}^1_j + \mathscr{R}^1_k$$

(see Eq. (1)).

For the assignment $s_i := s_j - s_k$ in $\mathscr{S}$ set

$$\mathscr{L}^1_i := \mathscr{L}^1_j + \mathscr{R}^1_k, \quad \mathscr{R}^1_i := \mathscr{R}^1_j + \mathscr{L}^1_k$$

(see Eq. (2)).

For the assignment $s_i := s_j \cdot s_k$ in $\mathscr{S}$ set

$$\mathscr{L}^3_i := \mathscr{L}^1_j\cdot\mathscr{L}^1_k, \quad \mathscr{L}^2_i := \mathscr{R}^1_j\cdot\mathscr{R}^1_k, \quad \mathscr{L}^1_i := \mathscr{L}^2_i + \mathscr{L}^3_i,$$

$$\mathscr{R}^3_i := \mathscr{R}^1_j\cdot\mathscr{L}^1_k, \quad \mathscr{R}^2_i := \mathscr{L}^1_j\cdot\mathscr{R}^1_k, \quad \mathscr{R}^1_i := \mathscr{R}^2_i + \mathscr{R}^3_i$$

(see Eq. (3)).

By induction, we see that for each i the value of s_i is $v(\mathscr{L}^1_i)-v(\mathscr{R}^1_i)$. □

Lemma 3. $\mathrm{AC}_\geqslant(\{+,\cdot\})$ (*resp.* $\mathrm{AC}_=(\{+,\cdot\})$) *is reducible to* $\mathrm{AC}_\geqslant(\{+,x\mapsto x^2/2\})$ (*resp.* $\mathrm{AC}_=(\{+,x\mapsto x^2/2\})$).

Proof. Informally, we use the identity

$$(A-B)(C-D)=\tfrac{1}{2}((A+C)^2+(B+D)^2-(A+D)^2-(B+C)^2).$$

Let $\mathscr{S}$ be a circuit over the monotone basis. We construct a circuit $\mathscr{S}'$ over the basis $\{+,x\mapsto x^2/2\}$ consisting of 10 series of assignments $\mathscr{P}$, $\mathscr{N}$, $\mathscr{A}^t$, $t\in[1,8]$ such that for each k the equality

$$v(s_k)=v(\mathscr{P}_k)-v(N_k) \tag{4}$$

holds. The construction is step-by-step substitution as in the above lemmas.

For the assignment $s_i := s_j + s_k$ in $\mathscr{S}$ add the assignments

$$\mathscr{P}_i := \mathscr{P}_j + \mathscr{P}_k, \quad \mathscr{N}_i := \mathscr{N}_j + \mathscr{N}_k \tag{5}$$

and for the assignment $s_i := s_j \cdot s_k$ add the assignments

$$\begin{aligned}
&A_i^1 := \mathscr{P}_j + \mathscr{P}_k, \quad A_i^2 := \mathscr{N}_j + \mathscr{N}_k, \quad A_i^3 := \mathscr{P}_j + \mathscr{N}_k, \quad A_i^4 := \mathscr{N}_j + \mathscr{P}_k,\\
&A_i^5 := \frac{(A_i^1)^2}{2}, \quad A_i^6 := \frac{(A_i^2)^2}{2}, \quad A_i^7 := \frac{(A_i^3)^2}{2}, \quad A_i^8 := \frac{(A_i^4)^2}{2},\\
&\mathscr{P}_i = A_i^5 + A_i^6,\\
&\mathscr{N}_i = A_i^7 + A_i^8.
\end{aligned} \tag{6}$$

In the latter case the following equations hold:

$$\begin{aligned}
v(\mathscr{P}_i) &= \frac{(v(\mathscr{P}_j) + v(\mathscr{P}_k))^2}{2} + \frac{(v(\mathscr{N}_j) + v(\mathscr{N}_k))^2}{2},\\
v(\mathscr{N}_i) &= \frac{(v(\mathscr{P}_j) + v(\mathscr{N}_k))^2}{2} + \frac{(v(\mathscr{N}_j) + v(\mathscr{P}_k))^2}{2}.
\end{aligned} \tag{7}$$

Eq. (4) is verified by induction using Eq. (7)

$$\begin{aligned}
v(s_i) &= (v(\mathscr{P}_j) - v(\mathscr{N}_j))(v(\mathscr{P}_k) - v(\mathscr{N}_k))\\
&= (v(\mathscr{P}_j) \cdot v(\mathscr{P}_k) + v(\mathscr{N}_j) \cdot v(\mathscr{N}_k)) - (v(\mathscr{P}_j) \cdot v(\mathscr{N}_k) + v(\mathscr{N}_j) \cdot v(\mathscr{P}_k))\\
&= \left(\frac{(v(\mathscr{P}_j) + v(\mathscr{P}_k))^2}{2} + \frac{(v(\mathscr{N}_j) + v(\mathscr{N}_k))^2}{2}\right) - \left(\frac{(v(\mathscr{P}_j) + v(\mathscr{N}_k))^2}{2} + \frac{(v(\mathscr{N}_j) + v(\mathscr{P}_k))^2}{2}\right)\\
&= v(\mathscr{P}_i) - v(\mathscr{N}_i).
\end{aligned} \tag{8}$$

Now we are able to construct a reduction. Take an instance $(\mathscr{S}_1, \mathscr{S}_2)$ of the problem $\mathrm{AC}_{\geqslant}(\{+, \cdot\})$. Convert the circuits $\mathscr{S}_1$, $\mathscr{S}_2$ into the circuits $\mathscr{S}'$, $\mathscr{S}''$, over the basis $\{+, x \mapsto x^2/2\}$ as described above. Merge them into one circuit $\mathscr{S}$.

Now form the circuits $\mathscr{F}_1$ and $\mathscr{F}_2$ starting from the assignments of $\mathscr{S}$. The last assignments in $\mathscr{F}_1$ and $\mathscr{F}_2$, respectively, are $f_1 := \mathscr{P}^1_{s_1} + \mathscr{N}^2_{s_2}$ and $f_2 := \mathscr{P}^2_{s_2} + \mathscr{N}^1_{s_1}$, where s_1 is a size of $\mathscr{S}_1$ and s_2 is a size of $\mathscr{S}_2$.

By construction

$$v(\mathscr{F}_1) - v(\mathscr{F}_2) = (\mathscr{P}'_{s_1} + \mathscr{N}''_{s_2}) - (\mathscr{P}''_{s_2} + \mathscr{N}'_{s_1}) = (\mathscr{P}'_{s_1} - \mathscr{N}'_{s_1}) + (\mathscr{P}'_{s_2} - \mathscr{N}'_{s_2}) = v(\mathscr{S}_1) - v(\mathscr{S}_2).$$

The reduction is given by the mapping $(\mathscr{S}_1, \mathscr{S}_2) \mapsto (\mathscr{S}', \mathscr{S}'')$. From Eq. (4) we see that it also works for both $\mathrm{AC}_{\geqslant}$ and $\mathrm{AC}_{=}$. □

3. Reduction of the problem $\mathrm{AC}_{\geqslant}$ to SDFP

Linear optimization on the intersection of the cone of positive semidefinite matrices with an affine subspace of matrices is called SDP. Let us denote SDFP the corresponding feasibility problem: to check whether the cone of positive semidefinite matrices has nonempty intersection with an affine subspace.

SDFP can be stated in the following form.

Input: A list $Q_0, \ldots, Q_m$ of symmetric $(n \times n)$ matrices with rational entries. Matrices are represented by lists of entries, each entry is represented by a pair (numerator, denominator), integers are given in binary.

Output: "Yes" if there exist reals $x_1, \ldots, x_m$ such that $Q = Q_0 + \sum_{i=1}^m x_i Q_i$ is a positive semidefinite matrix, otherwise the output is "no".

The proof below uses Ramana's results on the exact duality theory for SDP [10]. Ramana found a special form of dual program for SDP. It is called the extended Lagrange–Slater dual program (ELSD). Ramana proved that

- ELSD can be constructed from the primal program in polynomial time;

- if the optimum value of the primal program is bounded then the dual program is feasible;
- if both the primal and the dual are feasible then their optimum values are equal.

Theorem 2. *The predicate* $\mathrm{AC}_{\geqslant}(\{+, x \mapsto x^2/2\})$ *is reducible to* SDFP.

Applying Theorem 1 we obtain reductions of the predicates $\mathrm{AC}_{\geqslant}$ over all bases discussed in the previous section to SDFP.

Proof. The first step is to represent a circuit value as an optimal value of some semidefinite program in the form

$$\begin{aligned} &t \to \inf, \\ &\text{semidefinite conditions on } t \text{ and other variables.} \end{aligned} \tag{9}$$

Let $\mathscr{S}$ be a circuit over the basis $\{+, x \mapsto x^2/2\}$. We construct a semidefinite program $P(\mathscr{S})$ such that for each assignment s_i there is a variable x_i in $P(\mathscr{S})$ and a matrix M_i. Matrices M_i are built as follows:

$$\begin{aligned} s_i := s_j + s_k \quad &\longrightarrow \quad M_i = (x_i - x_j - x_k), \\ s_i := \frac{s_j^2}{2} \quad &\longrightarrow \quad M_i = \begin{pmatrix} 2x_i & x_j \\ x_j & 1 \end{pmatrix}. \end{aligned}$$

Note that in the former case $M_i \succeq 0$ iff $x_i \geqslant x_j + x_k$ and in the latter $M_i \succeq 0$ iff $x_i \geqslant x_j^2/2$.

Let ℓ be the size of $\mathscr{S}$. Consider an SDP

$$\begin{aligned} &x_\ell \to \inf, \\ &M_i \succeq 0 \text{ for each } i, \\ &x_0 = 1. \end{aligned} \tag{10}$$

To convert the program (10) to the standard form we can replace the equality $x_0 = 1$ by the inequality

$$\begin{pmatrix} x_0 - 1 & 0 \\ 0 & -x_0 + 1 \end{pmatrix} \succeq 0. \tag{11}$$

The optimal value of $P(\mathscr{S})$ is the value of $\mathscr{S}$. Indeed, all operations in the process of circuit evaluation are monotone with respect to each variable. So, any feasible solution of (10) satisfies the condition $x_\ell \geqslant v(\mathscr{S})$. On the other hand, $x_i = v(s_i)$ is a feasible solution.

Take now an instance of $\mathrm{AC}_{\geqslant}(\{+, x \mapsto x^2/2\})$. Its input is a pair of circuits $\mathscr{S}^{(1)}$, $\mathscr{S}^{(2)}$. The positive answer in the problem $\mathrm{AC}_{\geqslant}(\{+, x \mapsto x^2/2\})$ means that $v(\mathscr{S}^{(1)}) \geqslant v(\mathscr{S}^{(2)})$ which is equivalent to $-v(\mathscr{S}^{(2)}) \geqslant -v(\mathscr{S}^{(1)})$. The value $-v(\mathscr{S}^{(2)})$ is the optimal value for SDP

$$\begin{aligned} &-x_\ell^{(2)} \to \sup, \\ &M_i^{(2)} \succeq 0, \\ &x_0^{(2)} = 1. \end{aligned} \tag{12}$$

To represent $-v(\mathscr{S}^{(1)})$ as the infimum of SDP we use the ELSD to the program of type (10). In Ramana's paper ELSD has a mixed form (linear equations are permitted). To convert it to the standard form each linear equation should be replaced by a positive semidefinite condition on a 2×2 matrix as in Eq. (11). After conversion ELSD can be written as follows:

$$\begin{aligned} &c(Y) \to \inf, \\ &D \succeq 0. \end{aligned} \tag{13}$$

Here $c(Y)$ is a linear functional on dual variables Y.

Thus, the program

$$\begin{aligned} & -x_\ell^{(2)} - c(Y) \geqslant 0, \\ & M_i^{(2)} \succeq 0, \\ & x_0^{(2)} = 1, \\ & D \succeq 0 \end{aligned} \tag{14}$$

is feasible iff $-v(\mathscr{S}^{(2)}) \geqslant -v(\mathscr{S}^{(1)})$.

This gives a reduction of $\mathrm{AC}_{\geqslant}(\{+, x \mapsto x^2/2\})$ to SDFP. □

4. Open questions

The main question remained open is the complexity of SDFP. We suggest the problem $\mathrm{AC}_{\geqslant}$ as a "lowerbound" for SDFP. If it is hard then SDFP is hard too. The maximal result in this direction would be putting SDFP on some level of the counting hierarchy CH.

Complexity of $\mathrm{AC}_{\geqslant}$ itself is the next question. To be a good "lowerbound" it should be hard. But up to our present knowledge nothing prohibits inventing an efficient algorithm for its solution. Such an algorithm would be interesting by other reasons. It would justify using circuit representation for rationals instead of binary system in complexity theoretic questions when time is estimated up to a polynomial slowdown. As it is shown in [1] it might simplify complexity analysis of numeric algorithms very much. So any definite result about $\mathrm{AC}_{\geqslant}$ would lead to interesting conclusions.

From algorithmic point of view some intrinsic difficulty of numerical algorithms is related to nonconstructive nature of real numbers. BSS model of computation [2] directly operates with real numbers and thus completely ignores the question of number representation. Presently the relation of BSS model to the common algorithmic complexity is not well understood. In particular, in BSS model SDFP falls into the complexity class $\mathrm{NP}_{\mathbb{R}}$—an analogue of the class NP over reals. Referring to our way of number representation it is natural to ask about the inclusion $\mathrm{SDFP} \in \mathrm{NP}^{\mathrm{AC}_{\geqslant}}$. This question is also open.

In the opposite direction, it may be interesting to represent numbers as optimal solutions of SDP. For example, the well-known $n!$ conjecture by Shub and Smale [14] in our setting asks about monotone circuit complexity of $n!$. It can be shown that monotone circuit complexity is lowerbounded by the dimension of an SDP representing $n!$ as an optimal solution. Is this dimension polylogarithmic on n?

Acknowledgments

We would like to thank A. Kitaev for helpful discussions and the unknown referee for the helpful and insightful comments. The work of both authors was supported by RFBR Grant 05-01-01019. The second author was also supported by Grant NSh-5833.2006.1.

And the last but not the least: we are much indebted to our late friend and teacher L. Khachiyan who stimulated our interest to this problem.

References

[1] E. Allender, P. Brügiser, J. Kjeldgaard-Petersen, P. Mitersen, On the complexity of numerical analysis, ECCC, TR-37, 2005.
[2] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and Real Computation, Springer, New York, 1998.
[3] F. Cucker, D. Grigoriev, On the power of real Turing machines over binary inputs, SIAM J. Comput. 26 (1997) 243–254.
[4] M. Grötshel, L. Lovaśz, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer, New York, 1988.
[5] N. Karmarkar, A new polynomial algorithm for linear programming, Combinatorica 4 (1984) 373–395.
[6] L. Khachiyan, A polynomial algorithm in linear programming, Soviet Math. Dokl. 20 (1979) 191–194.
[7] P. Koiran, Valiant's model and the cost of computing integers, ECCC, TR-03, 2004.
[8] Yu.E. Nesterov, A.S. Nemirovskii, Interior Point Methods in Convex Optimization: Theory and Applications, SIAM Publications, Philadelphia, 1994.
[9] L. Porkolab, L. Khachiyan, On the complexity of semidefinite programs, J. Global Optim. 10 (1997) 351–365.
[10] M.V. Ramana, An exact duality theory for semidefinite programming and its complexity implications, Math. Program. 77 (1997) 129–162.
[11] J. Renegar, A polynomial-time algorithm based on Newton's method for linear programming, Math. Program. 40 (1988) 59–94.

[12] J. Renegar, A Mathematical View of Interior-point Methods in Convex Optimization, SIAM Publications, Philadelphia, 2001.
[13] A. Schönhage, On the power of random access machines, in: H. Maurer (Ed.), Automata, Languages and Programming ICALP 79, Lecture Notes in Computer Science, vol. 71, Springer, New York, 1979, pp. 520–529.
[14] M. Shub, S. Smale, On the intractability of Hilbert's Nullstellensatz and an algebraic version of "NP $\neq$ P", Duke Math. J. 81 (1996) 47–54.
[15] M. Sipser, Introduction to the Theory of Computation, second ed., Thomson Course Technology, Boston, 2005.
[16] L.G. Valiant, Completeness classes in algebra, in: Proceedings of the11th ACM Symposium on Theory of Computing, ACM Press, New York, 1979, pp. 249–261.
[17] K. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Inform. 23 (1986) 325–356.
[18] H. Wolkowicz, R. Saigal, L. Vandenberghe (Eds.), Handbook of Semidefinite Programming, Kluwer Academic Publishers, Dordrecht, 2000.