Computing Rational Radical Sums in Uniform TC^0

Paul Hunter[1], Patricia Bouyer[2], Nicolas Markey[2], Joël Ouaknine[1], and James Worrell[1]

1 **Oxford University Computing Laboratory, UK**
{paul.hunter,joel.ouaknine,james.worrell}@comlab.ox.ac.uk
2 **Lab. Spécification et Vérification, CNRS & ENS Cachan, France**
{bouyer,markey}@lsv.ens-cachan.fr

Abstract

A fundamental problem in numerical computation and computational geometry is to determine the sign of arithmetic expressions in radicals. Here we consider the simpler problem of deciding whether $\sum_{i=1}^{m} C_i A_i^{X_i}$ is zero for given rational numbers A_i, C_i, X_i. It has been known for almost twenty years that this can be decided in polynomial time [2]. In this paper we improve this result by showing membership in uniform TC^0. This requires several significant departures from Blömer's polynomial-time algorithm as the latter crucially relies on primitives, such as gcd computation and binary search, that are not known to be in TC^0.

Keywords and phrases Sum of square roots, Threshold circuits, Complexity

1 Introduction

The SqrtSum problem is as follows: given integers $a_1, \ldots, a_n, b_1, \ldots, b_m$, is it the case that $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i} > 0$? This problem naturally arises in computational geometry whenever one needs to compare the length of two paths, as in the *Euclidean Travelling Salesman Problem* [8, 14] for example.

SqrtSum can be solved in polynomial time on a unit-cost RAM, i.e., counting only the number of algebraic operations [18]: one simply uses numerical methods to compute each root to a sufficient degree of accuracy. However the problem is not known to be in P when one accounts for the bit complexity of arithmetic operations. Hence certain ostensibly simple problems in computational geometry, such as computing minimal spanning trees, are not known to be in P. Moreover since SqrtSum is not even known to be in NP it is also not known whether the Euclidean Travelling Salesman problem is in NP.

The difficulty in solving SqrtSum hinges on the fact that the best root separation bounds to hand require that one compute a super-polynomial number of bits of the expression $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i}$ to determine its sign. The question of determining optimal separation bounds was posed at least as far back as [13]. More recent work on the problem includes [15, 16, 3, 5]; also [12] presents a conjecture that would imply P-membership of SqrtSum.

SqrtSum has found applications in numerical decision problems outside the area of computational geometry. For instance, it has recently been used as a complexity lower bound for several problems related to recursive probabilistic systems. Etessami and Yannakakis [7] show that SqrtSum is reducible in polynomial time to the problem of determining whether a stochastic context-free grammar produces a terminal string with probability greater than a given threshold. This latter problem is in turn equivalent to the reachability problem for a certain subclass of probabilistic pushdown automata [7]. In another paper Etessami and Yannakakis [6] consider a range of algorithmic problems in game theory and economics, several of which are shown to be as hard as SqrtSum. For example, SqrtSum can be reduced to various decision problems concerning Nash Equilibria and the value of Shapley stochastic games.

Since the decision problem for the existential fragment of the first-order theory of real-closed fields is known to be in PSPACE [4], it is straightforward that SQRTSUM is also in PSPACE. A more general problem than SQRTSUM is the problem POSSLP of determining whether an arithmetic circuit over the basis $\{+, -, *\}$ evaluates to a positive integer. POSSLP was shown to lie in the 4th level of the counting hierarchy [1]. To the best of our knowledge this result also yields the best upper bound for SQRTSUM.

The subject of this paper involves upper bounds for the easier problem SQRTSUMEQ: given integers $a_1, \ldots, a_n, b_1, \ldots b_m$, does $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i} = 0$? This problem is also of key importance in numerical analysis and exact geometric computation, as discussed in [20]. SQRTSUMEQ is apparently more tractable than SQRTSUM; a polynomial-time decision procedure has been given by Blömer [2]. In fact [2] gives a polynomial-time algorithm for a more general problem in which one considers arbitrary integer roots rather than just square roots, and with arbitrary rational coefficients in front of the radicals (i.e., not just 1 or -1 as in SQRTSUMEQ).

In this paper we consider a further generalisation of SQRTSUMEQ, where we allow any rational exponent (less than 1) rather than exponents of the form $\frac{1}{N}$ for N a positive integer. In particular, we are interested in the following problem:

RADICALSUMEQ	
Instance:	Rational numbers C_i, A_i, X_i for $1 \leq i \leq m$ with $A_i > 0$ and $0 \leq X_i \leq 1$
Problem:	Does $\sum_{i=1}^{m} C_i A_i^{X_i} = 0$?

Our main result is that RADICALSUMEQ has very low complexity within P, and can be solved with fixed-depth circuits consisting of AND, OR, and threshold gates of unbounded fan-in:

▶ **Theorem 1.** RADICALSUMEQ $\in$ uniform TC^0.

The notion of uniformity referred to above is DLOGTIME-uniformity, which is the strongest uniformity requirement that is generally applicable.

Our TC^0 procedure adapts Blömer's polynomial-time algorithm for comparing radical expressions [2] and exploits the fact that division and iterated multiplication are in uniform TC^0; see [9, 10]. However we depart from [2] in several critical respects. Firstly [2] only considers the case of exponents of the form $\frac{1}{N}$. Whilst $A^{\frac{M}{N}}$ can be rewritten as $(A^M)^{\frac{1}{N}}$, the rational A^M cannot in general be explicitly computed in polynomial time and it is not clear how to apply Blömer's algorithm without doing so. Secondly at various points Blömer's algorithm requires the computation of the greatest common divisor of two numbers (specifically, denominators in the exponents) and binary search (to find integer d-th roots), two techniques not known to be in TC^0: indeed it is an open problem whether gcd computation is even in NC^1 [11].

One of the consequences of our work is that, unless $TC^0 = P$, SQRTSUMEQ has strictly lower complexity than EQSLP, the problem of determining whether an arithmetic circuit over the basis $\{+, -, *\}$ evaluates to zero or not. Indeed, the latter is easily seen to be P-hard, by reduction from the circuit value problem. In contrast, it is still open whether SQRTSUM has the same complexity as POSSLP or not.

The paper is organised as follows. In Section 2 we recall the definitions and notation we use throughout the paper. In Section 3 we present two procedures necessary for our main algorithm. The first of these procedures is a restatement of a known result but presented in a form suitable for our needs. The second shows how to compute the ratio of two radicals

in uniform TC^0, a result which we believe to be of independent interest. In Section 4 we present our main algorithm for deciding RadicalSumEQ, and in Section 5 we discuss further extensions of the problem.

2 Preliminaries

Throughout this paper we assume familiarity with standard notions of circuit complexity; an excellent reference on the subject is [19].

Recall that a circuit family $\{\mathsf{C}_n\}$ is DLogtime-uniform if there is a deterministic Turing machine that, given n and the name of a gate g, can determine g's label and neighbours in time $O(\log n)$.

In the sequel we always use n to represent the input size. In particular, we assume integers provided as part of the input to have absolute value bounded above by 2^n, and for there to be at most n terms in the sum (that is, $m \leq n$). This means the actual input size is $O(n^2)$, however this does not affect the overall result as the class TC^0 is closed under polynomial changes in input size. As is customary in this area of circuit complexity, we call $n^{O(1)}$-bit numbers *large* and denote them with uppercase letters, and we say $(\log n)^{O(1)}$-bit numbers are *small* and use lowercase letters to represent them.

We assume rational numbers are represented as ratios of two (not necessarily co-prime) integers. Whilst we do not require the rational numbers to be in reduced form, we use the fact that the size required to represent a rational number is bounded below by the size of its reduced form. For $A \in \mathbb{Q}$, we define $\|A\| := |M \cdot N|$ where $\frac{M}{N} = A$ and $\gcd(M, N) = 1$. The *height* of A is defined as $\mathrm{ht}(A) := 1 + \log \|A\|$. It is clear that $\|A\| = \min |M \cdot N|$ where the minimum is taken over all representations of A as $\frac{M}{N}$, thus the height of A provides a lower bound on the number of bits required to represent A.

The following properties of $\|\cdot\|$ will prove useful:

▶ **Lemma 2.** *For $A, B \in \mathbb{Q}$ and $X, Y \in \mathbb{R}$:*

1. *If $A^X \in \mathbb{Q}$ then $\left\|A^X\right\| = \|A\|^{|X|}$, in particular $\left\|\frac{1}{A}\right\| = \|A\|$.*
2. *If $A^X \cdot B^Y \in \mathbb{Q}$ then $\left\|A^X \cdot B^Y\right\| \leq \|A\|^{|X|} \cdot \|B\|^{|Y|}$.*

Proof. For the first result, consider first $X \geq 0$. Let $A = \frac{M}{N}$ where $\gcd(M, N) = 1$. Clearly, $\left\|A^X\right\| = |M^X \cdot N^X| = |M \cdot N|^X = \|A\|^X$. To extend the result to $X < 0$ we observe from the symmetry of the definition of $\|\cdot\|$ that $\|A\| = \left\|A^{-1}\right\|$. Hence $\left\|A^X\right\| = \left\|(A^{-1})^{|X|}\right\| = \left\|A^{-1}\right\|^{|X|} = \|A\|^{|X|}$.

For the second result, we observe that for $A = 0$ the result holds trivially and for $A = 1$ the result follows from the first part of the proof. So assume $A \neq 0, 1$ and let $c = \frac{\log B}{\log A}$. Since $A^c = B$, it follows from the above result that $\|A\|^{|c|} = \|A^c\| = \|B\|$. Therefore,

$$\begin{aligned}
\left\|A^X \cdot B^Y\right\| &= \left\|A^{X+cY}\right\| \\
&= \|A\|^{|X+cY|} && \text{(from the first result)} \\
&\leq \|A\|^{|X|+|c|\cdot|Y|} && \text{(by the triangle inequality)} \\
&= \|A\|^{|X|} \cdot \|B\|^{|Y|} && \text{as required.}
\end{aligned}$$

◀

We will make use of some standard parallel algorithms and techniques known to be computable in uniform TC^0, notably:

- Existential guessing between $n^{O(1)}$ choices (in particular, guessing small integers),
- Universal (parallel) computation amongst $n^{O(1)}$ choices,

- Addition (and subtraction) of n n-bit numbers, and
- Iterated multiplication of n n-bit numbers and integer division of two n-bit numbers [9].

For ease of reference, in each *term* $C_i A_i^{X_i}$, C_i is the *coefficient*, A_i is the *base*, X_i is the *exponent* and $A_i^{X_i}$ is the *radical* (even though it may be rational).

3 TC^0 Tools

In this section we present two TC^0-procedures necessary for our algorithm. The first of these concerns determining whether an integer root of a given rational is itself rational, and if so, computing the root. The result follows from observations in [10] and [17] that functions given by a convergent power series, in particular $A^{\frac{1}{k}}$, can be approximated using iterated multiplication. We present it here in a form appropriate for our needs: computing the value if it is rational or failing if it is not rational. The algorithm uses the power series approximation to compute an approximant to sufficiently high accuracy and then tests if this estimation is the true value of the root by exponentiation. As we are dealing with rationals, it initially appears that some care is needed in extracting the approximant. However, as $\mathrm{ht}(A^{\frac{1}{k}}) \leq \mathrm{ht}(A)$, if $A^{\frac{1}{k}}$ is rational, its binary expansion will repeat after $O(\mathrm{ht}(A))$ bits. Thus to find an approximant it suffices to compute polynomially many bits to find the period of its binary expansion and then compute the rational using standard techniques. The situation appears clearer in the case of the integers: after sufficiently many computed bits we truncate our approximation and consider the integers around the value computed; but this is simply the rational case shifted by a factor of $2^{O(n)}$. Nevertheless, as the technique for extracting the approximant in the integer case is simpler, we adopt this procedure (see Lemma 3) for root computation and extend it to rationals (in Algorithm 2) by rationalising the denominator.

▶ **Lemma 3.** *Let a be a $(\log n)$-bit positive integer and B be an n-bit positive integer. There exists a uniform TC^0-algorithm which computes $\sqrt[a]{B}$ if it is an integer or fails if it is not an integer.*

Proof. As mentioned above, the idea of the algorithm (presented in Algorithm 1) is to compute an integer approximation (technically three approximations) to $\sqrt[a]{B}$ and then check if the a-th power of the approximant is equal to B. The steps which are not clearly in uniform TC^0 are the computation of the first n bits of $\sqrt[a]{B}$, and the evaluation of the antecedent in the **if** statement. Membership of TC^0 for the computation of R follows from the result of [10] (Corollary 6.5) that polynomially many bits of $X^{\frac{1}{k}}$ can be computed in uniform TC^0. For the antecedent, iterated multiplication can be used to compute $\hat{R}^a$, $(\hat{R}+1)^a$, and $(\hat{R}-1)^a$ in uniform TC^0. To show correctness, we observe that as $a \geq 1$, $\mathrm{ht}(\sqrt[a]{B}) \leq \mathrm{ht}(B)$, and so $\sqrt[a]{B}$ requires at most n bits if it is an integer. Thus $|\hat{R} - \sqrt[a]{B}| \leq 1$, and the only possible integral values for $\sqrt[a]{B}$ are $\hat{R}-1$, $\hat{R}$, or $\hat{R}+1$. ◀

Although our final algorithm does not require the computation of large roots, the extension is trivial as a consequence of the following observation.

▶ **Lemma 4.** *Let $A \in \mathbb{Z}$, $B \in \mathbb{Q}$, $A, B > 0$, $B \neq 1$. If $\sqrt[A]{B} \in \mathbb{Q}$ then $A < \mathrm{ht}(B)$.*

Proof. Let $B = \frac{M}{N}$ where $\gcd(M, N) = 1$. As $B \neq 1$ there exists a prime p such that $p|M$ or $p|N$. Assume without loss of generality $p|M$. As $\sqrt[A]{B} = \frac{\sqrt[A]{M}}{\sqrt[A]{N}}$ is rational we have $p^A|M$. As $M < 2^{\mathrm{ht}(B)}$ and $p \geq 2$, it follows that $A < \mathrm{ht}(B)$. ◀

Our algorithm for computing roots of rationals is presented in Algorithm 2.

Algorithm 1 Computing integer roots

Input: $n, a, B \in \mathbb{Z}$, $0 < a < n$, $0 \leq B < 2^n$
Returns: $\sqrt[a]{B}$ or Fail if $\sqrt[a]{B} \notin \mathbb{Z}$.
  Compute R, the first n bits of $\sqrt[a]{B}$
  Let $\hat{R} = \lfloor R \rfloor$
  if $(\hat{R}-1)^a = B$ **or** $\hat{R}^a = B$ **or** $(\hat{R}+1)^a = B$ **then**
    return $(\hat{R}-1)$, $\hat{R}$, or $(\hat{R}+1)$ as appropriate
  else
    return Fail
  end if

▶ **Proposition 5.** *For $A \in \mathbb{Z}$, $B \in \mathbb{Q}$, $A, B > 0$ there exists a uniform TC^0-algorithm which computes $\sqrt[A]{B}$ if it is rational or fails if it is irrational.*

Algorithm 2 Computing rational roots

Input: $A \in \mathbb{Z}$, $B = \frac{M}{N} \in \mathbb{Q}$, $A, B > 0$
Returns: $\sqrt[A]{B}$ or Fail if $\sqrt[A]{B} \notin \mathbb{Q}$.
  if $B = 1$ **then**
    return 1
  else if $A \geq \mathrm{ht}(B)$ **then**
    return Fail
  else
    Compute $C = \sqrt[A]{M \cdot N^{A-1}}$
    if $C \notin \mathbb{Z}$ **then**
      return Fail
    end if
    return $\frac{C}{N}$
  end if

The second algorithm of this section, presented in Algorithm 3, overcomes the difficulties with Blömer's procedure for computing the ratio of two radicals.

The core of the correctness of Algorithm 3 is presented in the following lemma; if the ratio of two radicals is rational then one of two cases occurs, either the bases are powers of some common base, or the exponents have low height relative to their value. It is this case split that forms the basis of our algorithm: in the first case we can existentially guess the powers of the common base, and in the second we can guess reduced forms for X and Y and apply the algorithm of Blömer.

▶ **Lemma 6.** *For $A, B, X, Y \in \mathbb{Q}_{>0}$ if $\frac{A^X}{B^Y} \in \mathbb{Q}$ then either:*

- *There exists $Q \in \mathbb{Q}$ and $\alpha, \beta \in \mathbb{Z}$ with $\alpha X - \beta Y \in \mathbb{Z}$ such that $A = Q^\alpha$ and $B = Q^\beta$, or*
- $\|X\| < 4\mathrm{ht}(A)\mathrm{ht}(B)^2(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$ *and*
  $\|Y\| < 4\mathrm{ht}(A)^2\mathrm{ht}(B)(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$.

Proof. Suppose $\frac{A^X}{B^Y} = M \in \mathbb{Q}$. From Lemma 2 we observe that $\|M\| = \left\|\frac{A^X}{B^Y}\right\| \leq \|A\|^X \|B\|^Y$, so $\mathrm{ht}(M) < X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B)$. Let $A = \prod p_i^{a_i}$, $B = \prod p_i^{b_i}$ and $M = \prod p_i^{m_i}$ where for all i, p_i is prime and $a_i, b_i, m_i \in \mathbb{Z}$. We observe that $|a_i| < \mathrm{ht}(A)$, $|b_i| < \mathrm{ht}(B)$ and $|m_i| < \mathrm{ht}(M)$. Consider the (integral) vectors $\mathbf{a} = (a_i)$, $\mathbf{b} = (b_i)$, and $\mathbf{m} = (m_i)$. By equating prime powers

Algorithm 3 Computing rational radical ratios

Input: $A, B, X, Y \in \mathbb{Q}$, $A, B > 0$, $X, Y \in [0, 1]$
Returns: $\frac{A^X}{B^Y}$ or Fail if $\frac{A^X}{B^Y} \notin \mathbb{Q}$

```
Let n = max{ht(A), ht(B), ht(X), ht(Y)}
Existentially guess non-negative integers a, b < n
if aX − bY ∈ ℤ and Q = ∛A = ∛B ∈ ℚ† then
  return Q^{aX−bY}
end if
Existentially guess non-negative integers x, x′, y, y′ < 8n^4
if X = x/x′ and Y = y/y′ then
  Let z = gcd(x′, y′), x″ = x′/z, and y″ = y′/z
  Let R_A = ∛(A^x) and R_B = ∛(B^y)
  if R_A ∈ ℚ and R_B ∈ ℚ and R = ∛(R_A/R_B) ∈ ℚ then
    return R
  end if
end if
return Fail
```

† We allow the equality to hold here if $a = 0$ and $A = 1$ or if $b = 0$ and $B = 1$, setting $Q = 1$ if $a = b = 0$ and $A = B = 1$.

we have

$$\mathbf{a}X = \mathbf{m} + \mathbf{b}Y \cdot \qquad (*)$$

We consider two cases.

Case 1: $\mathbf{a}$ and $\mathbf{b}$ are linearly dependent (over $\mathbb{Q}$). In this case, there exist integers k and l (not necessarily co-prime) and an integral vector $\mathbf{q} = (q_i)$ such that $a_i = k \cdot q_i$, $b_i = l \cdot q_i$ and the q_i have no common factor. From $(*)$, $\mathbf{m} = (kX - lY)\mathbf{q}$. As $\mathbf{m}$ is integral and the q_i have no common factor, it follows that $(kX - lY) = c \in \mathbb{Z}$. Setting $Q = \prod p_i^{q_i}$ we have $A = Q^k$, $B = Q^l$, $M = Q^c$ and $kX = c + lY$.

Case 2: $\mathbf{a}$ and $\mathbf{b}$ are linearly independent (over $\mathbb{Q}$). In this case, there exist $i \neq j$ such that the vectors (a_i, a_j) and (b_i, b_j) are linearly independent. It therefore follows that X and Y satisfying $(*)$ are unique. Indeed $X = \frac{b_i m_j - b_j m_i}{b_i a_j - b_j a_i}$ and $Y = \frac{a_i m_j - a_j m_i}{b_i a_j - b_j a_i}$. Thus

$$\begin{aligned}
\|X\| &= \left\| \frac{b_i m_j - b_j m_i}{b_i a_j - b_j a_i} \right\| \\
&\leq \|b_i m_j - b_j m_i\| \cdot \|b_i a_j - b_j a_i\| \\
&= |b_i m_j - b_j m_i| \cdot |b_i a_j - b_j a_i| \\
&\leq (|b_i m_j| + |b_j m_i|)(|b_i a_j| + |b_j a_i|) \quad \text{(by the triangle inequality)} \\
&< (2\mathrm{ht}(B)\mathrm{ht}(M))(2\mathrm{ht}(B)\mathrm{ht}(A)) \\
&< 4\mathrm{ht}(A)\mathrm{ht}(B)^2(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B)),
\end{aligned}$$

and likewise $\|Y\| < 4\mathrm{ht}(A)^2\mathrm{ht}(B)(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$. ◀

When Case 1 of Lemma 6 holds, it is clear Algorithm 3 correctly computes the ratio $\frac{A^X}{B^Y}$; the bounds on X and Y ensure that Q^{aX-bY} can be evaluated with iterated multiplication. To complete the correctness result we need to show the correctness of the algorithm when Case 2 of the above lemma occurs. This follows directly from the following result observed by Blömer [2]:

▶ **Lemma 7.** *For $q_1, q_2 \in \mathbb{Q}$, $d_1, d_2 \in \mathbb{N}$, the following are equivalent:*

- $\frac{\sqrt[d_1]{q_1}}{\sqrt[d_2]{q_2}} \in \mathbb{Q}$
- *If $d = \gcd(d_1, d_2)$ then*
  - $r_i = \sqrt[d_i]{q_i^d} \in \mathbb{Q}$ *for $i = 1, 2$, and*
  - $\sqrt[d]{\frac{r_1}{r_2}} \in \mathbb{Q}$.

It is straightforward to show that Algorithm 3 can be implemented with a uniform TC^0 circuit. The non-obvious steps are in the computation of Q, R_A, R_B and R where we use Algorithm 2, and the computation of z which can be calculated because x' and y' are small[1]. Lemmas 6 and 7 establish the correctness of the algorithm, giving us the following:

▶ **Theorem 8.** *Let $A, B, X, Y \in \mathbb{Q}$, with $A, B > 0$, $X, Y \in [0, 1]$. There exists a uniform TC^0-algorithm which computes the ratio $\frac{A^X}{B^Y}$ if it is rational, or fails if it is irrational.*

We note that in Theorem 8 we only need the upper bound on X and Y to compute the ratio: in the first case of Lemma 6 the bound ensures that Q^{aX-bY} is computable using iterated multiplication, and in the second case the bound ensures that $\|X\|$ and $\|Y\|$ are small. If instead we were provided with the ratio M and simply asked to check if $A^X = MB^Y$, we no longer require the bound on X and Y: in the first case $aX - bY \leq \mathrm{ht}(M)$ and in the second case we can use the bounds obtained in terms of $\mathrm{ht}(A)$, $\mathrm{ht}(B)$, and $\mathrm{ht}(M)$. This gives us the following additional result:

▶ **Theorem 9.** *For $A, B, M, X, Y \in \mathbb{Q}_{\geq 0}$, whether $A^X = MB^Y$ can be decided in uniform TC^0.*

4 Deciding RadicalSumEQ in TC^0

In this section we present our TC^0-algorithm for deciding RadicalSumEQ. Critical to our algorithm is the following result presented in Blömer:

▶ **Lemma 10** (Theorem 4 of [2])**.** *For $\rho_i \in \mathbb{Q}$, $d_i \in \mathbb{N}$, $1 \leq i \leq m$, the radicals $\sqrt[d_1]{\rho_1}, \ldots, \sqrt[d_m]{\rho_m}$ are linearly independent over $\mathbb{Q}$ if they are pairwise linearly independent.*

Clearly this result extends to arbitrary rational exponents, giving the following procedure (also presented in [2]) for determining if $S = \sum_{i=1}^{m} C_i A_i^{X_i} = 0$. Using Algorithm 3 partition the terms of S into linearly dependent groups $S_1, \ldots, S_{m'}$. For convenience let us assume $C_i A_i^{X_i} \in S_i$ for $1 \leq i \leq m'$. Again using Algorithm 3, replace each term in each group by the rational multiple of some common radical. For example, if $C_j A_j^{X_j} \in S_i$, replace it with $C_j R_{ij} A_i^{X_i}$ where $R_{ji} = \frac{A_j^{X_j}}{A_i^{X_i}}$ is computed with Algorithm 3. Then S can be written as

$$S = \sum_{i=1}^{m'} \left(\sum_j C_j R_{ji} \right) A_i^{X_i}$$

where j in the inner sum runs over all indices of terms in S_i. From the above result, as $A_1^{X_1}, \ldots, A_{m'}^{X_{m'}}$ are pairwise linearly independent, they form a linearly independent set. Thus $S = 0$ if and only if $\sum_j C_j \cdot R_{ji} = 0$ for all i, $1 \leq i \leq m'$, and this is easily checked. To simplify the parallelisation of this algorithm, rather than gathering linearly dependent terms under a common radical, we treat each radical as the common radical, repeating the coefficient check several times. The full algorithm is specified in Algorithm 4.

[1] Existentially guess z and verify in parallel that it is the greatest of all common divisors of x' and y'.

Algorithm 4 Deciding RadicalSumEQ

Input: $\{A_i, C_i, X_i : 1 \leq i \leq m\} \subseteq \mathbb{Q}$ with $A_i > 0$ and $X_i \in [0, 1]$
Returns: True if and only if $\sum_{i=1}^{m} C_i A_i^{X_i} = 0$

```
for all i, j ≤ m do
  Let R_ij = A_i^{X_i} / A_j^{X_j}
  if R_ij ∉ ℚ then
    Let R_ij = 0
  end if
end for
for all j ≤ m do
  if Σ_{i=1}^{m} C_i R_ij ≠ 0 then
    return False
  end if
end for
return True
```

The only step of Algorithm 4 which is not clearly in uniform TC^0 is the computation of R_{ij}. Theorem 8 establishes its membership in TC^0. The correctness of the algorithm follows from Lemma 10 and the discussion above. Combining these together gives our main result.

▶ **Theorem 1.** RadicalSumEQ $\in$ uniform TC^0.

5 Further Work

It is clear from Lemma 6 that we can extend Algorithm 3 (and hence Algorithm 4) to exponents bounded (in value) by some polynomial in n. This raises the question of whether we can remove the upper bound on the exponents completely. Theorem 9 shows that we can do so in the special case where $m = 2$. By rewriting A^X as $A^{\lfloor X \rfloor} \cdot A^{\{X\}}$ where $\{X\}$ denotes the fractional part of X, we can absorb the "rational part" $A^{\lfloor X \rfloor}$ of the radical into the coefficient, and run our algorithm up to the point where we check if $\sum_{i=1}^{m} C_i R_{ij} = 0$. Thus we have reduced the problem to deciding if a given rational-valued point is a root of a sparse, multivariate polynomial (a natural sub-instance of the unbounded version of RadicalSumEQ). Whether or not this problem is in TC^0, or even in P, is part of ongoing work.

References

1 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

2 Johannes Blömer. Computing sums of radicals in polynomial time. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 670–677. IEEE, 1991.

3 C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55(1):14–28, 2009.

4 John F. Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 460–467. ACM, 1988.

5 Qi Cheng, Xianmeng Meng, Celi Sun, and Jiazhe Chen. Bounding the sum of square roots via lattice reduction. *Math. Comput. (to appear)*, 2010.

6 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points (extended abstract). In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 113–123. IEEE Computer Society, 2007.

7 Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1–66, 2009.

8 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

9 William Hesse. Division is in uniform TC^0. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2001.

10 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65:695–716, 2002.

11 Yu Lin-Kriz and Victor Y. Pan. On parallel complexity of integer linear programming, gcd and the iterated mod function. In *Proceedings of the Third Annual Symposium on Discrete Algorithms*, pages 124–137. ACM/SIAM, 1992.

12 Gregorio Malajovich. An effective version of Kronecker's theorem on simultaneous Diophantine approximation. *Preprint*, 2001.

13 Joseph O'Rourke. Advanced problem 6369. *Amer. Math. Monthly*, 88(10):769, 1981.

14 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.

15 Sylvain Pion and Chee Yap. Constructive root bound method for k-ary rational input numbers. *Theor. Comput. Sci.*, 369(1-3):361–376, 2006.

16 Jianbo Qian and Cao An Wang. How much precision is needed to compare two sums of square roots of integers? *Inf. Process. Lett.*, 100(5):194–198, 2006.

17 John H. Reif and Stephen R. Tait. On threshold circuits and polynomial computation. *SIAM J. of Comput.*, 21:896–908, 1992.

18 Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. Complexity*, 8(4):393–397, 1992.

19 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.

20 Chee K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, 2nd edition, 2004.