Two problems dealing with real numbers that are hard to parallelize

Felipe Cucker
Antonio Torrecillas

Report LSI–90–41
Two problems dealing with real numbers that are hard to parallelize

F. Cucker
Dept. L.S.I.
Univ. Politècnica de Catalunya
Barcelona 08028
SPAIN
e-mail: cucker@lsi.upc.es

A. Torrecillas
Dept. Matemàtica Aplicada II
Univ. Politècnica de Catalunya
Barcelona 08028
SPAIN
e-mail: torrecillas@ma2.upc.es

A classical subject in computational complexity is what is usually called algebraic complexity. This area deals with the complexity of algorithms that take their inputs on $\mathbb{R}^n$, where $\mathbb{R}$ is a ring, understood as the number of ring operations the algorithm performs as a function of $n$ and the main kind of results are both upper and lower bounds. A recent survey about this topic is [7].

A very special case in algebraic complexity is the one dealing with $\mathbb{R} = \mathbb{R}\cdot$. Here, the assumption that all the elements in the field have unit size, and that the cost of the field operations is also unitary, reflects the particular features of algorithms in numerical analysis. Very recently an article of L. Blum, M. Shub and S. Smale put into this scenery another approach to the complexity: the structural one (see [4]). To do so, they devised a model of real Turing machines, over the which the basis of a theory of computability and a theory of complexity is built. In particular analogs of the classes $\mathcal{P}$ and $\mathcal{NP}$ over the reals are introduced, and the problem of deciding whether a degree 4 real polynomial has a root is shown to be $\mathcal{NP}$-complete, thus, unlikely decidable in polynomial time.

On the other hand, for a wide variety of problems, fast parallel algorithms have been designed during the last years. These algorithms use as a model of parallel machine a family of circuits with polynomial size and polylogarithmic depth, for which a logspace uniformity condition is valid. However, this “is of course not possible for arbitrary constants over an uncountable field” (see [6]).

For the Boolean model, there are many problems for which algorithms working in polynomial time can be given that solve them and no good fast parallel algorithm is known that do the same. Some of these problems share the property that if a fast parallel algorithm is found that solves one of them, then every problem in $\mathcal{P}$ can be decided by such a fast parallel algorithm. These problems, that are said to be $\mathcal{P}$-complete constitute a class whose elements, even when no proof is known till now for $\mathcal{P} \neq \mathcal{NC}$, are usually considered as problems not admitting a solution in fast parallel time.

In section 2 of this paper we propose the parallel real RAM as a model for defining a class of problems dealing with real numbers that are decidable in fast parallel time. This formulation avoids uniformity considerations and the resulting class is trivially included in $\mathcal{P}$. We use reductions in this class (in fact, using constant parallel time) to give our main result, the existence of $\mathcal{P}$-complete problems for the real model. In section 3, two problems with this property are exhibited. On the one hand, the evaluation of algebraic circuits that have sign gates. On the other hand, the solution of systems of equalities and inequalities via substitutions. These results are a first step towards [4] 11.3. where the authors ask “to further develop ideas from recursive function theory such as fixed point theorems, reducibilities and hierarchies for machines over a ring $\mathbb{R}$".

1. Ground tools and notations.

In the sequel we shall denote the direct sum $\bigoplus^\infty \mathbb{R}\cdot$ by $\mathbb{R}^*$ according to the custom in language theory. We just recall that this direct sum is the set of sequences of real numbers having only a finite number of non-zero elements.

We recall from [4] that a real Turing machine consists of an input space $\mathbb{R}^*$, an output space $\mathbb{R}^*$ and a state space $S = \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{R}^*$, together with a connected directed graph whose nodes labelled $1 \ldots N$ (the set of different instructions) are of certain types and with associated functions. The internal content of $S$ at time $t$ is $(i, j, x_1, x_2, x_3, \ldots)$ where for $t = 1$ the input is in the $x_s$ with $s$ odd (thus we reserve the even coordinates to leave work space), and $x_2$ can denote the length of the input. The five types of nodes are as follows:

† Partially supported by DGICYT PB 860062 and the ESPRIT BRA Program of the EC under contract no. 3075, project ALCOM.
‡ Partially supported by the ESPRIT BRA Program of the EC under contract no. 3075, project ALCOM.
1) Exactly one input node: node 1. Associated with this node is a next node $\beta(1)$.

2) Exactly one output node: node N. Once it is reached the computation halts, the contents of the real part of $S$ being considered as the output.

3) Computation nodes. Associated with a node $m$ of this type there is a next node $\beta(m)$ and a map $g_m : S \to S$. The $g_m$ is of the form $g_m(i, j, x) = (i'(i), j'(j), x'(x))$, with $i'(i) = i + 1$ or 1, $j'(i) = j + 1$ or 1, and $x'$ is a polynomial or rational map.

4) Branch nodes. There are two nodes associated with this node: $\beta^+(m)$ and $\beta^-(m)$. The next node is $\beta^+(m)$ if $x_1 \geq 0$ and $\beta^-(m)$ otherwise.

5) Move nodes or fifth nodes. It has a unique next node $\beta(m)$. If the current element of $S$ is $(i, j, x_1, \ldots)$ it operates replacing $x_j$ by $x_1$ in the $j^{th}$ place of the vector $\mathbb{R}^*$ in $S$.

An instantaneous description of any moment of the computation can be given by providing an element in $S$ and the current node. The first one changes according to the function associated with the current node and the node itself according to the function $\beta$.

We also recall from [4] that a machine $M$ is said to work in polynomial time when there are constants $c, q \in \mathbb{Z}^+$ such that for every input $y \in \mathbb{R}^*$, $M$ reaches its output node after at most $c(size(y))^q$ steps. The class $P$ is then defined as the set of all subsets of $\mathbb{R}^*$ that can be accepted by a machine working in polynomial time.

Along the whole of this paper, we shall quote classes like $P$, $NP$, etc. that exist both in the Boolean and in the real model without adding any subscript to specify which case we are talking about. This should not create any confusion, the context being clear in all cases.

We shall now introduce a slight modification to the real Turing machine that will simplify further constructions.

We shall say that a real Turing machine $M$ is in reduced form when it satisfies

(i) all the computation nodes have an associated function of the type $g(x_1, x_2) = x_1 \ast x_2$ where $\ast \in \{+, -, \cdot, /\}$ and the result of this elemental arithmetical operation is stored in $x_3$,

(ii) if $m$ is a fifth node then $|i - j| \leq 2$.

We can consider the class of sets recognized by machines in reduced form that work in polynomial time, and no new class is introduced because of the following easy lemma.

Lemma. The machines in reduced form conserve the class $P$.

Proof. Easy because our two restrictions only produce a polynomial number of movements for each simulation of nodes in the not reduced model. ■

2. On parallel algorithms over the reals.

One of the first features concerning real Turing machines and the complexity defined upon them is that there are problems with no time complexity bounds. In fact, it is observed in [4] that the integers are a recursive set over the reals. But they can not be recognized in constant time because otherwise $\mathbb{Z}$ would be a semi-algebraic subset of $\mathbb{R}$. Thus, $\mathbb{Z}^*$ is a subset in $\mathbb{R}^*$ that has no upper bounds for its time complexity. However, it can be recognized in constant auxiliary space. This fact entails that we shall not have a relation of the kind

$$\text{DSPACE}(s) \subset \text{DTIME}(f(s))$$

where $f$ is a function (in the Boolean model it is well known that we can take $f$ to be the exponential function).

This fact difficults the choice of a computational model for "fast parallel time". In the Boolean case, the usual computational model are families of circuits with polylogarithmic depth and polynomial size which are generated by a machine working in logarithmic space (see chapter 4 of [2]). If the circuits have depth $O(\log^k(n))$, the resulting class is called $NC^k$ and it is easily shown that for every $k$, $NC^k \subset P$. The bad behaviour of space bounds in the real case does not enable us to make a similar definition.

In section 3 of [6], a way of solving this problem is introduced that uses a hybrid encoding of the circuit and does not renounce to the notion of uniformity. It is intended to be used over any ring and no special emphasis is done on the reals.

2
In this paper, we shall use the PRAM machine as our computational model for parallel complexity classes. We will not give here a formal definition of a real RAM since it is essentially the same as the one given in [1] section 1.2, but allowing real numbers instead of integers. The only important remark we must do here is that no indirect addressing is done to a non-integer address. That is, it is a responsibility of the programmer to ensure that addressings are well done.

The real RAM is also treated in [9] section 1.4. where it is used as a model of computation for algorithms in computational geometry.

One feature to notice is that a real RAM can compute a big number such as $2^{2^n}$ in n steps, and so, it can store information after that time in doubly exponentially far registers. We shall see that this situation can be avoided.

**Lemma.** If a set can be recognized in polynomial time by a real RAM, it can be recognized by another real RAM that works in polynomial time and polynomial space.

**Sketch of the proof.** Given a machine $M$ that works in polynomial time, we simulate $M$ with $M'$ that uses pairs of registers, each pair consisting of an integer which denotes an active address, and a real number which is the content of that address. We substitute each old instruction which works with an address $m$ by a little program which first searches the value of $m_1$ as the first value of some pair and then works with the real content of this address (which is in the next position).

If $p$ is a polynomial bound for the running time of $M$, then the running time of $M'$ is bounded by $O(p^2)$ and its used space by $O(p)$.

As a consequence of these lemma we get the following result whose proof being easy is left to the reader.

**Proposition.** A set $L$ can be recognized in polynomial time by a real RAM if and only if it can be recognized in polynomial time by a real Turing machine.

A real PRAM consists of several independent real RAM's processors, each with its own private memory, communicating with another through a global memory. In one unit of time each processor can read one global or local memory location, execute a single real RAM instruction and write into global or local memory location. Many conventions can be taken about the protocol used for reading and writing in the global memory by different processors. We will consider the Concurrent-Read Exclusive-Write model.

We define the class $\text{PPT}^k$ (Parallel Polylogarythmic Time) to be the class of sets $L \subseteq \mathbb{R}^*$ such that there exists a real PRAM that accepts $L$ working in time $O(\log^k(n))$ with a polynomially bounded number of processors. Also, we define the class $\text{PPT}$ to be the union of the classes $\text{PPT}^k$, for $k \in \mathbb{N}$.

A final remark about this class concerns its relations with complexity classes defined by families of circuits. In the Boolean case it is well known that

$$\text{NC}^k \subset \text{PPT}^k \subset \text{NC}^{k+1}$$

(see [10]). In the real case, a straightforward complexity class defined by circuits with some uniformity restriction is the class $\text{PUNC}^k$ of sets accepted by a family of circuits with polynomial size and depth $O(\log^k(n))$ that can be generated by a real Turing machine in polynomial time. A slight variation of Stockmeyer and Vishkin arguments in [10] gives the following result.

**Proposition.** For every $k \geq 0$ we have the inclusion $\text{PPT}^k \subset \text{PUNC}^k$. ■

3. Two $\text{P}$-complete problems.

In this section we present two problems that are $\text{P}$-complete. We recall that a problem $A$ is hard for a class $C$ under reductions in a class $D$ when for every problem $B \in C$ there is a function $f: \mathbb{R}^* \rightarrow \mathbb{R}^*$ belonging to $D$ and such that for every $x \in \mathbb{R}^*$, $x \in B$ if and only if $f(x) \in A$. The problem is moreover complete for $C$ if it belongs to $C$. In general, the definition is used with classes satisfying that the resources allowed in $D$ are smaller than those allowed in $C$. For instance in [4], $C = \text{NP}$ and $D = \text{P}$. In our case, $C$ will be $\text{P}$ and $D$ will be $\text{PPT}$. One easy consequence of the existence of $\text{P}$-complete problems appears in the next result.

**Proposition.** If some $\text{P}$-complete problem has an algorithm in $\text{PPT}$, then $\text{P} = \text{PPT}$. ■
In the reductions given in the sequel, the parallel algorithms are given in high level language being clear that a PRAM can be devised that does the same task.

3.1. Evaluation of real decision circuits.

A very useful model in algebraic complexity is the algebraic circuit (also called "straight-line" program) which is involved mainly in arguments for computing lower bounds. In fact, this model is related to the birthday of algebraic complexity theory and the optimality of Horner's rule for evaluating polynomials (see the section 2 of [7] for an account of such question). In the real case, a similar model, the algebraic computation tree, has been recently used by Ben-Or in order to get several lower bounds in computational geometry (see [3]). A natural question in this setting is the complexity of evaluating such circuits. In this section we shall see that for real decision circuits, the problem of deciding whether the circuit returns 1 on a given input is P-complete.

We recall that an algebraic circuit \( B \) is a sequence of gates \((G_1, \ldots, G_n)\) belonging to one of the following types:

1) **Input gates**: \( G_i = x_i \), takes the input \( x_i \) from \( \mathbb{R} \).
2) **Arithmetic operation gates**: perform the operation \( * \) to the outputs of gates \( G_j \) and \( G_l \), \( j, l < i \) and \( * \in \{+, -, \cdot, /\} \).
3) **Operations-with-a-constant gates**: perform the operation \( * \) to the output of gate \( G_i \) and \( k, j < i, k \in \mathbb{R} \) and \( * \in \{+, -, \cdot, /\} \).
4) **Sign gates**: \( \{G_i = 1 \text{ if } G_j \geq 0\} \) and \( \{G_i = 0 \text{ if } G_j < 0\}, j < i \).

If the circuit has \( s \) input gates, we can suppose that they are the first ones, \( G_1, \ldots, G_s \). If moreover the node \( G_n \) is of the fourth type, we shall say that \( B \) is a decision circuit.

We define the real circuit decision problem to be the set

\[
\text{RCDP} = \{(B, a) \mid \text{the decision circuit } B \text{ with input } a \text{ returns } 1\}
\]

**Theorem.** The real circuit decision problem is P-complete.

**Proof.** Clearly, the problem is in \( P \). We shall then show that it is \( P \)-hard.

Let \( M \) be a real Turing machine in reduced form that works in time bounded by a polynomial \( p \). For every \( n \in \mathbb{N} \) we will associate a real decision circuit \( B_n \) such that for every \( a = a_1, \ldots, a_n \), \( M \) accepts \( a \) if and only if the circuit \( B_n \) returns 1 with input \( a \).

The circuit \( B_n \) is constructed from a family of smaller ground circuits \( I_t, J_t, Y_t, X_{i,t} \) such as \( i, t \leq s(n) \), and the intended meaning of each circuit is the following:

\[ I_t \text{ computes the value of the component } i \text{ in } S \text{ at time } t, \]
\[ J_t \text{ computes the value of the component } j \text{ in } S \text{ at time } t, \]
\[ Y_t \text{ computes the current node at time } t, \]
\[ X_{i,t} \text{ computes the value of the component } x_i \text{ in } S \text{ at time } t. \]

Their outputs are denoted by \( i_t, j_t, y_t \) and \( x_{i,t} \) respectively.

Note that, since \( M \) is in reduced form, each one of these circuits has a number of inputs bounded by 8.

For instance, for a circuit \( X_{i,t} \) with \( i \geq 4 \) and \( t \geq 2 \) we have the following situation.

![Diagram](image)

We go now into the description of these ground circuits, that we shall do by cases.

1) **Circuit** \( X_{3,t+1} \).

The inputs are \( i_t, j_t, y_t, z_{1,t}, z_{2,t}, z_{3,t}, z_{4,t}, z_{5,t} \). Now, if \( y_t \) is a node of type 1,2 or 4, the output of \( X_{3,t+1} \) will be \( z_{3,t} \). On the other hand, if \( y_t \) is a node of type 3 (i.e. a node that performs an arithmetical
operation \( \ast \) the output of \( X_{3,t+1} \) will be \( x_{1,t} \ast x_{2,t} \). Finally, let us consider the possibility that \( y_t \) is a node of type 5. In that case,

- if \( j_t \neq 3 \) then the output of \( X_{3,t+1} \) is \( x_{3,t} \),
- else, it is the value of \( x_{i_t} \) (note that \( i_t \in [1, \ldots, 5] \) because of the reduced form of \( M \)).

Thus, the output of \( X_{3,t+1} \), as a function of some trivial expressions in \( x_{1,t}, x_{2,t}, x_{3,t}, x_{4,t} \) and \( x_{5,t} \), depends on the values of \( y_t, i_t \) and \( j_t \).

Since the only thing we are interested on the value of \( j_t \) is whether it is equal to 3, we can, using two sign gates, map it to

\[
\tilde{j}_t = \begin{cases} 
1, & \text{if } j_t = 3; \\
0, & \text{otherwise.}
\end{cases}
\]

We finally have that the above mentioned output is determined according to \( 6N \) different possibilities in the values of \( y_t, i_t \) and \( j_t \). We can then profit the existence of Lagrange's interpolator polynomials to construct a polynomial that realizes the desired computation whose degree is exactly \( 6N \). Let us see how this polynomial is constructed in this case.

First, for any \( 1 \leq l \leq N \) we consider the polynomial

\[
a_l(X) = \prod_{k \neq l}^{N} \frac{X - k}{l - k}
\]

takes the value 1 if \( X = l \) and 0 if \( X = 1, \ldots, l-1, l+1, \ldots, N \).

Now, for every \( 1 \leq l \leq N \) we define the polynomial \( g_l \) such that

- (i) if \( l \) is of type 1, 2 or 4, \( g_l = x_{3,t} \),
- (ii) if \( l \) is of type 3, \( g_l = x_{1,t} \ast x_{2,t} \) where \( \ast \) is the operation performed at node \( l \), and
- (iii) if \( l \) is of type 5,

\[
g_l = (1 - \tilde{j}_t) x_{3,t} + \tilde{j}_t \left( \sum_{s=1}^{5} a_s(i_t) x_{s,t} \right)
\]

Clearly the polynomial

\[
G_{3,t+1} = \sum_{l=1}^{N} a_l(y_l) g_l
\]

computes the value \( x_{3,t+1} \) with inputs \( i_t, \tilde{j}_t, y_t, x_{1,t}, x_{2,t}, x_{3,t}, x_{4,t}, x_{5,t} \) and so, we take as \( X_{3,t+1} \) an arithmetic circuit that computes \( G_{3,t+1} \) preceded by two sign gates for getting \( \tilde{j}_t \). A final remark is that since the degree of \( G_{3,t+1} \) only depends on \( N \) the size of \( X_{3,t+1} \) is constant as a function of \( n \).

(2) Circuit \( X_{i,t+1} \ i \neq 3 \).

It is done in a similar way, but now, since the only kind of node that changes the value of this component is the fifth one, the \( g_l \) for \( l \) of type 3 is like the one for \( l \) of type 1, 2 or 4 in the preceding case.

(3) Circuit \( Y_{i,t+1} \).

Let us consider the value

\[
\sigma = \begin{cases} 
1, & \text{if } x_{1,t} \geq 0; \\
0, & \text{otherwise.}
\end{cases}
\]

which we can compute with one sign gate from \( x_{1,t} \). From this value and \( y_t \) we can calculate \( y_{t+1} \) with the polynomial

\[
H_{t+1} = \sum_{l=1}^{N} a_l(y_l) h_l
\]

where

\[
h_l = \begin{cases} 
(1 - \sigma) \beta^-(l) + \sigma \beta^+(l), & \text{if } l \text{ is of type 4; } \\
\beta(l), & \text{otherwise.}
\end{cases}
\]

and as before these computations can be performed by a circuit whose size does not depend on \( n \).
(4) Circuits $I_{t+1}$ and $J_{t+1}$.
Let us remember that in any node $i$ of type 3, the value of the component $i$ is modified according to a polynomial $i_{ij}$. We define for nodes $i$ of another type $i_{ij} = i_t$. Clearly, the polynomial we are looking for $I_{t+1}$ is

$$I_{t+1} = \sum_{i=1}^{N} a_i(y_i)i_{ij}$$

and a similar construction applies for $J_{t+1}$.

Thus, each circuit can be produced by a processor that works in constant time as a function of $n$. Since we do it for $O(p(n)^3)$ circuits, we get the stated result. ■

3.2. Systems of equations solvable by substitution.

We go now into the subject of semi-algebraic systems of equations (equalities and inequalities), first recalling that by such a system we understand one of the form

$$\bigwedge_{i=1}^{S} \bigvee_{j=1}^{r_i} f_{ij}(x_{ij}) \sigma_{ij} 0$$

where the $x_{ij}$ are subsets of $\{x_1, \ldots, x_n\}$, the $f_{ij}$ are polynomials and the $\sigma_{ij}$ are sign conditions taken from $\{=, >, \geq\}$. Each operand

$$\bigvee_{j=1}^{r_i} f_{ij}(x_{ij}) \sigma_{ij} 0$$

will be called clause.

In fact, any Boolean combination of equations of the kind $f_{ij}(x_{ij})\sigma_{ij} 0$ determines a semi-algebraic system of equations, but it is well known that such a system can be written in the above defined form (see chapter 2 of [5] for an introduction to semi-algebraic sets).

We shall also say that a system $S$ has degree $d$ if all the $f_{ij}$ have degree smaller than $d$. An important result from [4] states that for every $d \geq 2$ the problem of deciding whether a system of degree $d$ is satisfiable (dSAS in the sequel) is NP-complete.

On the other hand, there is a straightforward approach for deciding systems (widely used in linear algebra) by means of substitutions. We shall see that for every $d \geq 2$ the problem of deciding whether a system of degree $d$ is solvable by substitution is P-complete. To do so, let us begin with some definitions.

**Definition.** If a clause consists on a single equation of the form $ax - b = 0$, and a second clause contains the variable $x$ in some equation, then the substitution of these two clauses is the new one obtained by the following process:

- We substitute every occurrence of $x$ in the second clause by $\frac{b}{a}$.
- We perform the operations with real constants that are possible to do.
- In case we get an inequality without variables we decide if it is TRUE or FALSE.
- We simplify the clause according with the rules

  $$\text{FALSE} \cup E_1 \cup \cdots \cup E_k = E_1 \cup \cdots \cup E_k, \text{ and}$$
  $$\text{TRUE} \cup E_1 \cup \cdots \cup E_k = \text{TRUE}.$$

We shall say that a semi-algebraic system given by (*) is solvable by substitution when we can obtain a point $\{x_1, \ldots, x_n\}$ satisfying (*) after a finite number of substitutions to pairs of clauses in the system.

**Lemma.** For every $d \geq 2$ the problem of deciding whether a system of degree $d$ is solvable by substitution is in P.

**Proof.** Let $S$ be a semi-algebraic system, $S$ be its set of clauses consisting of a single equation of the type $ax - b = 0$, and $T$ the set of its remaining clauses. We suppose that FALSE is not a clause in $S$, and that we have an order in $S$ and $T$. Then the following procedure decides whether $S$ is solvable by substitution:

$$\bigwedge_{i=1}^{S} \bigvee_{j=1}^{r_i} f_{ij}(x_{ij}) \sigma_{ij} 0$$

...
while $S \neq \emptyset$ do
  let $E$ be the first equation in $S$
  let $x$ be the variable appearing in $E$
  for all clause $C \neq E$ in $S \cup T$ do
    if $x$ appears in $C$ then
      let $w$ be the simple substitution of $E$ and $C$
      remove $C$
      if $w = \text{FALSE}$ then REJECT and halt
      elif $w = \text{TRUE}$ then remove $w$
      else add $w$ at the end of $S$
    od
  fi
od
remove $E$
if $T = \emptyset$ then ACCEPT else REJECT

Notice that this algorithm works for any order in the sets $S$ and $T$.

Theorem. For every $d \geq 2$ the problem of deciding whether a system of degree $d$ is solvable by substitution is P-complete.

Proof. We shall reduce the RCDP to it. So, given a real decision circuit $(B_1, \ldots, B_n)$ with $s$ input gates, and an input $a = a_1, \ldots, a_s$, we associate to it the system

$$S := \bigwedge_{i=1}^{n} E_i \land E_{n+1}$$

where, for every $i$:

- if $B_i$ is an input gate, $E_i := \{X_i - a_i = 0\}$,
- if $B_i$ is an arithmetic gate that performs the operation $*$ on the outputs of $B_j$ and $B_k$ then $E_i := \{X_i - (X_j \ast X_k) = 0\}$,
- if $B_i$ is an arithmetic gate that performs the operation $\ast c$ on the outputs of $B_j$ and $c$ a real constant then $E_i := \{X_i - (X_j \ast c) = 0\}$,
- if $B_i$ is a sign gate returning 1 for an input $\geq 0$ and 0 otherwise, whose input is the output of gate $B_j$, then $E_i := E_{i1} \lor E_{i2}$ where

  $$E_{i1} := \{X_j \geq 0 \land X_i - 1 = 0\} \quad \text{and} \quad E_{i2} := \{X_j < 0 \land X_i = 0\}$$

that can be readily transformed into the conjunction of four clauses.
- finally, $E_{n+1} := \{X_n - 1 = 0\}$.

This construction can be done with $n + 1$ processors (one for each gate and another for the equation $E_{n+1}$) each of them working in constant time.

Moreover, it is clear that the resulting system has a solution if and only if the circuit $B$ returns 1 with input $a$, and in this case the solution can be found by substitution.

From these facts we get that RCDP is P-complete.

Remarks. i) We want to attract the reader’s attention to the fact that this last problem has with dSAS the same relation that UNIT has with SAT in the Boolean case; we consider a certain feasibility problem that happens to be NP-complete and restricting ourselves to finding solutions using a concrete procedure we fall into a P-complete problem (for the Boolean case see [8]).

ii) Notice that both reductions use constant time, a phenomenon that looks strange under the Boolean model point of view. The reason of such a phenomenon is maybe related to the correspondence between parallel time and sequential space and the above quoted properties of space in the real model. A straightforward
consequence of such a small time for the reduction is that if an algorithm in $PPT^k$ is found for any of the
two $P$-complete problems exhibited above, then $P = PPT^k$.

Acknowledgement. Thanks are due to Ricard Gavaldà for helpful discussions as well as for a careful
reading of the manuscript.

References.
[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman; *The design and analysis of computer algorithms*. Addison-Wesley,
[4] L. Blum, M. Shub and S. Smale; "On a theory of computation and complexity over the real numbers: $NP$-
completeness, recursive functions and universal machines". *Bulletin of the Amer. Math. Soc.*, vol.21, n.1,
[9] F.P. Preparata and M.I. Shamos; *Computational Geometry: an introduction*. Texts and Monographs in Com-