QFT + NP = P
Quantum Field Theory (QFT): A Possible Way of Solving NP-Complete Problems in Polynomial Time

Vladik Kreinovich, Luc Longpré, and Adriana Beltran

Abstract It has been recently theoretically shown that the dependency of some (potential observable) quantities in quantum field theory (QFT) on the parameters of this theory is discontinuous. This discovery leads to the *theoretical* possibility of checking whether the value of a given physical quantity is equal to 0 or different from 0 (here, *theoretical* means that this checking requires very precise measurements and because of that, this conclusion has not yet been verified by a direct experiment).
This result from QFT enables us to do what we previously could not: check whether two computable real numbers are equal or not. In this paper, we show that we can use this ability to solve NP-complete ("computationally intractable") problems in polynomial ("reasonable") time.
Specifically, we will introduce a new model of computation. This new model is based on solid mainstream physics (namely, on quantum field theory). It is capable of solving NP-complete problems in polynomial time.

1 Pre-Introduction: Feasible and Intractable

Feasible. Some algorithms require lots of time to run. For example, some algorithms require the running time of $\geq 2^n$ computational steps on an input of size n. For reasonable sizes $n \approx 300$, the resulting running time exceeds the lifetime of the Universe and is, therefore, for all practical purposes, non-feasible.

Vladik Kreinovich, Luc Longpré, and Adriana Beltran
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
e-mail: vladik@utep.edu, longpre@utep.edu

In order to find out which algorithms are feasible and which are not, we must formalize what "feasible" means. This formalization problem has been studied in theoretical computer science; no completely satisfactory definition has yet been proposed.

The best known formalization is: an algorithm $\mathscr{U}$ is *feasible* iff it is *polynomial time*, i.e., iff there exists a polynomial P such that for every input x, the running time $t_{\mathscr{U}}(x)$ of the algorithm $\mathscr{U}$ on the input x is bounded by $P(|x|)$ (here, $|x|$ denotes the length of the input x).

This definition is not perfect, because there are algorithms that are polynomial time but that require billions of years to compute, and there are algorithms that require in a few cases exponential time but that are, in general, very practical. However, this is the best definition we have so far.

Intractable (NP-hard). For many mathematical problems, it is not yet known (2017) whether these problems can be solved in polynomial time or not. However, it is known that some combinatorial problems are as tough as possible, in the sense that:

- if we can solve any of these problems in polynomial time,
- then, crudely speaking, we can solve many practically important combinatorial problems in polynomial time.

The corresponding set of important combinatorial problems is usually denoted by NP, and problems whose fast solution leads to a fast solution of all problems from the class NP are called *NP-hard*.

The majority of computer scientists believe that NP-hard problems are not feasible. For that reason, NP-hard problems are also called *intractable*. For formal definitions and detailed descriptions, see, e.g., [19].

2 Introduction: Equality of Computable Real Numbers is Algorithmically Undecidable

Computable numbers is what theory of computing started with. In 1936, Alan Turing wrote his classical paper [35], one of the papers that started what we now know as *theory of computing*. This paper was motivated by the necessity to distinguish between *computable* and non-computable *real numbers*. To give a precise definition of what a computable real number is, Turing:

- invented the Turing machine and
- proved several positive and negative results about the Turing machines in general and computable real numbers in particular.

Since then, the definition of a computable real number has slightly changed, but one negative results still stands: it is algorithmically impossible to check whether a real number is equal to 0. Turing defined a computable real number as

a number for which an algorithm can compute the digits of its *decimal* expansion. This definition turned out to be not perfect, because some numbers computable in this sense stop being computable when we switch to *binary* expansions.

In view of this difficulty, nowadays, a slightly modified definition of a computable real is used:

a real number x is computable if there exist an algorithm that for every integer k, returns a rational number r_k for which

$$|x - r_k| \le 2^{-k}.$$

There is an area of mathematics called *constructive mathematics* which, crudely speaking, analyzes which problems with computable real numbers are algorithmically decidable and which are not (see, e.g., [1, 4, 10, 12, 27]).

Equality of real numbers is not decidable: a seemingly counter-intuitive but in reality, a rather clear result. In every exposition of constructive mathematics, one of the first negative results is a theorem that no algorithm can tell whether a given computable real number is equal to 0 or not.

From the viewpoint of a computer science student who learns about this result in his first (and often last) Theory of Computation course, this result may seem counter-intuitive. Indeed, checking equality of real numbers is something that programmers do on a daily basis, without giving it much thought. To a practical-oriented programmer, the fact that this simple problem is, in theory of computing, proven to be algorithmically undecidable, is an indication that theory of computing may be (somewhat) far away from the actual computing.

At first glance, this result may seem to be counter-intuitive, but on second glance, it is very natural.

- If we have an algorithmic binary sequence $f(n)$, then the real number

  $$x = \sum_{n=1}^{\infty} f(n) 2^{-n}$$

  is computable, because we can easily, given k, compute x's $2^{-k}-$rational approximation

  $$r_k = \sum_{n=1}^{k} f(n) 2^{-n}.$$

- If we were able to tell whether $x = 0$ or not, we would thus be able to tell whether the sequence $f(n)$ is all zeros or not.
- Thus, taking $f(n) = 0$ if a given algorithm U does not stop by time n, we would get an algorithm that solves the halting problem for algorithms, which is known to be impossible.

3 Introduction Continued: A Heuristic Use of the Result that Equality of Computable Real Numbers is Undecidable

It is possible to put a positive twist on this negative result. What this negative result basically says is that:

- if we were able to check whether two given computable numbers are equal or not,
- then we would be able to solve many problems that are now considered algorithmically undecidable.

In this form, this result sounds purely negative, because until recently, there was no known physical way of checking whether two real numbers (i.e., values of two physical quantities) are equal or not. To be able to check, say, equality of x and 0, we must have a physical device that:

- produces 1 ("true") when $x = 0$ and
- produces 0 ("false") when $x \neq 0$.

In other words, we need a physical process for which the dependence of some observable quantity on an input parameter is discontinuous. So far, all the known physical dependencies between observables are continuous. (The dependencies that correspond to phase transitions have "abrupt" changes, but these changes are still not discontinuous: they are continuous but exceptionally fast.)

With this physical impossibility to check whether two real numbers are equal or not, the result that we are discussing seems purely *negative*. However, it is possible to put the following *positive* spin on this result: The computer-based comparison of two real numbers can be viewed as a *heuristic* method of checking whether two real numbers are equal or not. Thus, we can try to design a heuristic method of solving difficult-to-solve problems as follows:

- first, we formulate an "algorithm" for solving these problems that includes checking equality of computable real numbers as an intermediate step;
- second, we transform this "algorithm" into an actual algorithm by checking easy-to-check "computer equality" (i.e., approximate equality) of computed real numbers instead of the algorithmically undecidable actual equality.

This idea has led to reasonably efficient heuristic algorithms in computational chemistry. This idea is actively being used in *computational chemistry*. One of the main problems of chemistry is identification of chemical substances. In non-organic and organic chemistry, there exist experimental techniques that enable us to describe a *graph* structure of the unknown substance, i.e., to describe which atoms it consists of, and which of these atoms are connected by chemical bounds. In order to identify this substance, we must compare it with graphs that describe known substances. In mathematical terms, we need to check whether an (experimentally obtained) graph is isomorphic to one of the graphs that describe known substances.

Unfortunately, graph isomorphism problem is known to be hard to solve. For some substances, different nodes correspond to different types of atoms; in this case,

it is relatively easy to check whether a given molecule coincides with this substance, because we can simply identify each atom with a similar atom in the standard substance and then check whether all connections are as in the standard model. For many other substances, however, atoms of the same type occur in different places of the structure in different roles; examples of such substances are organic substances and fullerenes. For these substances, we have to solve the difficult graph isomorphism problem.

One way of solving this problem is based on the idea described above:

- It is known that to every graph, we can assign a polynomial or several polynomials that uniquely determine this graph (i.e., the two tuples of polynomials coincide iff the graphs are isomorphic).
- Thus, to check whether the two graphs are isomorphic, we can compare the coefficients of the corresponding polynomials.

These methods are widely used in structural chemistry; see, e.g., [2, 5, 6, 24, 25, 30, 31].

We can further compress these polynomials into *numbers* (called *indices*) that also give complete information about the graph [32], and compare only these numbers.

The resulting method is, of course, only a *heuristic* method, because sometimes, due to computer inaccuracy, non-isomorphic substances get erroneously identified. However, the numerical experiments show that these errors are extremely rare (and that, therefore, this method works really well) [7]: among all the generated graphs, only 10^{-5} of them got mis-identified.

4 A New Physical Phenomenon

We have already mentioned that until recently, all known physical phenomena were *continuous*, and thus, the above idea could be applied only heuristically.

Recently, a *discontinuous* phenomenon has been discovered in quantum field theory [18, 21, 22, 23, 20]. Namely, it was shown that the discontinuous dependence on the parameter λ usually takes place in the theories with the so-called *spontaneous symmetry breaking* (see explanations below). Specifically, the authors of the above papers considered theories of a complex-valued scalar field φ in which a non-linear term

$$\lambda \cdot \varphi^4$$

is added to the standard Lagrangian L_0 of the wave equations (i.e., to

$$\varphi_{,a}\varphi^{,a} + m_0^2 \cdot \varphi^2,$$

where $\varphi_{,a}$ denotes the partial derivative $\dfrac{\partial \varphi}{\partial x_a}$, and the repetition of the index a means summation; for an introduction to classical and quantum field theory, see, e.g., [3, 28]).

This theory can be equivalently described as adding a non-linear term

$$2\lambda \cdot \varphi^3$$

to the right-hand side of the wave equation

$$\Box\varphi + m_0^2 \cdot \varphi = 0;$$

here, $\Box$ indicates the d'Alembertian operator

$$\Box \stackrel{\text{def}}{=} \frac{\partial^2}{\partial x_0^2} - \frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2} - \frac{\partial^2}{\partial x_3^2}.$$

For the resulting non-linear theory, the characteristics of the vacuum state, such as the energy density ρ and pressure p of quantum fluctuations, have a discontinuity at $\lambda = 0$, i.e., these (potentially observable) physical quantities have different values

- for $\lambda = 0$ and
- for $\lambda \neq 0$.

This result leads to a potential possibility of checking whether a given physical quantity is equal to 0 or not. In this paper, we will show that this possibility can help us solve NP-complete ("computationally intractable") problems in polynomial time. Before we describe how, we would like to make this physical result slightly less mysterious by describing in plain words why the dependence of ρ on λ is discontinuous.

Why discontinuous? In classical (pre-quantum) physics, vacuum means absence of matter. In quantum field theory, due to Heisenberg's uncertainty principle, we cannot have a complete absence of matter: there always are so-called *quantum fluctuations*. As a result of these fluctuations, in quantum field theory, vacuum is a complicated quantum state. One way to analyze the quantum vacuum is to neglect quantum effects and to consider the (quasi) classical approximation to the vacuum state.

Crudely speaking, in this approximation, the vacuum is the state with the smallest energy. In the classical vacuum, the equations do not explicitly depend on the coordinates, and therefore, the solution should also be independent on the coordinates. Thus, the gradient $\varphi_{,a}$ is equal to 0, and the vacuum energy can be described as the integral

$$E = \int \rho \, dx_0 \, dx_1 \, dx_2 \, dx_3,$$

where the energy density ρ is equal to

$$\rho = m_0 \cdot \varphi^2 + \lambda \cdot \varphi^4.$$

The minimum is attained at one of the stationary points, i.e., at one of the points where the derivative of the density w.r.t. φ is equal to 0.

- When $\lambda = 0$, the derivative is equal to $2m_0^2 \cdot \varphi$, and therefore, in the vacuum state, $\varphi = 0$. Hence, for $\lambda = 0$, the vacuum energy density $\rho(0)$ is also equal to 0.
- When $\lambda \neq 0$, the derivative is equal to

$$2m_0^2 \cdot \varphi + 4\lambda \cdot \varphi^3.$$

This derivative can be expressed as a product

$$\varphi \cdot (2m_0^2 + 4\lambda \cdot \varphi^2).$$

Thus, this derivative is equal to 0 if:

- either $\varphi = 0$;
- or $2m_0^2 + 4\lambda \cdot \varphi^2 = 0$, in which case $\varphi^2 = -m_0^2/2\lambda$, and $\varphi = \pm\dfrac{m_0}{\sqrt{-2\lambda}}$.

Thus, we have the three stationary points: $\varphi_1 = 0$ and $\varphi_{2,3} = \pm\dfrac{m_0}{\sqrt{-2\lambda}}$.

- For $\varphi_1 = 0$, the corresponding energy density ρ_1 is equal to 0.
- For two other stationary points, $\varphi_{2,3}^2 = \dfrac{m_0^2}{-2\lambda}$, and therefore, the corresponding vacuum energy density $\rho_2 = \rho_3$ is equal to the following expression:

$$\rho_2 = \rho_3 = m_0^2 \cdot \frac{m_0^2}{-2\lambda} + \lambda \cdot \frac{m_0^4}{4\lambda^2} = -\frac{m_0^4}{4\lambda}.$$

When $\lambda > 0$, $\rho_2 = \rho_3 < \rho_1$, and therefore, the true vacuum (i.e., the state with the smallest energy) corresponds to the one of the value $\varphi_{2,3}$. Thus, the energy density $\rho(\lambda)$ of the true vacuum is equal to

$$-\frac{m_0^4}{4\lambda}.$$

When $\lambda > 0$ and $\lambda \to 0$, we have $\rho(\lambda) \to -\infty \neq \rho(0) = 0$. Thus, the dependence of the vacuum energy density ρ on λ is indeed discontinuous.

Comment. In classical physics, there is usually a single vacuum state, and if the theory was invariant w.r.t. some symmetries, then this unique vacuum state is also invariant w.r.t. the same transformations. In our example, for $\lambda > 0$, we have *two* different vacuum states that correspond to the different value of α. The original theory is clearly invariant w.r.t. the transformation $\varphi \to -\varphi$, but neither of the two vacuum state φ_2 and φ_3 is invariant w.r.t. this transformation. Thus, we get a *symmetry breaking*. This symmetry breaking is called *spontaneous*, because it is not caused by any non-symmetric external fields.

Caution: what we described above was an oversimplified, qualitative explanation, *not* the proof. The actual proof requires much more complicated physical computations.

Discontinuity is not so surprizing after all. To some extent, quantum field theory can be viewed as a phenomenological theory, because it is based on the underlying notion of the non-quantized space-time. These discontuity results basically say that if we try to solve QFT equations within the natural assumption of a standard space-time in which all the fields are continuous, we inevitably run into discontinuities, which means that the underlying topology is no longer standard. Thus, to adequately describe quantuk fields, we need to quantize not only the fileds themselves, but also the underlying topology of space-time (see, e.g., [16, 17]).

When re-formulated in these terms, the discountuity stops being a strange mathematical phenomenon caused by a specific feature of the Lagrange function, but becomes a natural general effect of any sufficiently complicated quantum field theory.

5 How to Use the New Physical Phenomenon to Solve NP-Complete Problems in Polynomial Time: Definitions and the Main Result

5.1 Motivations for the new model of computation

Our goal is to describe the computational ability of a computer that can use the phenomena of Quantum Field Theory (QFT) in its computations; we will call such enriched computer a *QFT computer*. Before we introduce formal definitions, let us first describe informally what a computer can do if we equip it with the additional possibility stemming from the Quantum Field Theory.

General idea of QFT computer: analog operations in addition to discrete ones + ability to check whether two analog values are equal or not. To apply the physical phenomenon described above, we must have the real numbers represented as the actual values of physical quantities (i.e., we must have *analog* computations in addition to the usual ones). Thus:

- in addition to the standard memory, in which discrete values are stored, we need an additional (analog) memory for storing the actual values;
- in addition to the standard processing unit in which discrete values are processed, we need an additional (analog) processing unit for processing the analog values.

Comment. For simplicity, in this paper, we will add the operations with the analog values to the standard Turing machine. It is reasonably easy, however, to add these operations to RAM or to any other existing model of computation.

Operations with analog values. What operations can we perform with the analog quantities? First, we can *compare* these quantities. This possibility to check whether the two values are equal or not is a completely new phenomenon, and this is what make our computer more powerful that other known computation models.

In addition to this new operation of comparison, there are several physically meaningful operations that we can perform with the actual (analog) values:

- First of all, there are many physically reasonable ways to implement the *sum* of two given quantities. Let us give three examples:
  - If we have two values v_1 and v_2 stored as masses, then we can simply add these masses together and get the mass
  $$v = v_1 + v_2$$
  that is equal to the sum of the two masses.
  - Similarly, if we have two currents v_1 and v_2, we can place the corresponding two wires (where these currents are) as inputs to the third wire, then this third wire will carry the current
  $$v = v_1 + v_2$$
  that is equal to the sum of these two currents.
  - If we have two values v_1 and v_2 represented as distances marked on straight-line rulers, then we can align these two rulers so that the second instance goes right after the first one, and as a result, get the distance
  $$v = v_1 + v_2.$$
- If the two values v_1 and v_2 are represented as distances marked on the straight-line rulers, then we align the rulers in such a way that the second ruler goes in the different direction than the first one, then we get a physical implementation of the *difference*.
- In addition to physical processes that implement addition and difference, there are also known physical processes that implement, e.g.:
  - *multiplication*: e.g., Ohm's law, according to which the voltage V is a product of the current and the resistance:
  $$V = I \cdot R;$$
  - *division*;
  - *absolute value* $|x|$;
  - *sine*: e.g., for a perfect resonator, the signal $x(t)$ is proportional to the sine of time t:
  $$x(t) = A \cdot \sin(\omega \cdot t);$$
  - *arcsine*, etc.

The above described implementations are currently *approximate*, because so far, there was no way to check whether the given value is exactly correct. The new physical phenomenon enables us to check exactly this, and thus, will, hopefully, lead to a precise implementation of these analog operations.

5.2 Informal description of the QFT computer

Memory. In contrast to the standard (one-tape-one-head) Turing machine, this computer will have *two* different *tapes*:

- a *standard* tape (for storing symbols from a given finite alphabet), and
- a new *analog* tape, i.e., a potentially infinite sequence of storage devices in which the actual values are stored.

Initial contents of the memory. In the standard Turing machine, it is usually assumed that all the cells whose contents is not initially specified are empty.

Similarly, we will assume that on the *analog tape*, the initial stored values are all zeros.

Processing unit: states. The *head* (*processing unit*) must be able to store not only the symbol written on the standard tape, but also the value stored in the analog tape. It is convenient to make the processing unit able to store not only a single value, but several values. Thus, we want to equip the head with three *registers* in which the analog values can be stored;

- these registers will be denoted R_0, R_1, and R_2, and
- their contents will be denotes, correspondingly, by r_0, r_1, and r_2.

At any given moment of time, the state of the head is characterized:

- by the state of its discrete part (as in the standard Turing machine) +
- by the contents of the three registers.

Similarly to the tape, we assume that the initial values stored in these registers are zeros.

What influences the processing unit. In the standard Turing machine, the next action of the processing unit is determined by two things: by the state of the head and by the symbol that it is currently observing on the standard tape.

In the QFT computer, we can also make our decision based on whether the contents r_0 of the main register R_0 is equal to 0 or not.

Possible actions. At each moment of time, the standard Turing machine has the following options:

- First, it can change its state (in particular, it can get into the halting state, or into the states that correspond to "yes" and "no" answers).
- Second, it can move the cursor to the left, to the right, or it can *stay*, i.e., leave the cursor at the same place.

- Third, it can also write one of the possible symbols to the cell at which we are currently looking.

The new machine has two tapes and has, therefore, more options:

- In addition to moving the cursor of the standard tape, it can also move the cursor of the analog tape.
- We can also perform some operations with the register values. We can perform no operations at all; this case will be denoted by $\emptyset$. We can perform the simplest possible *copying* operations:
  - we can copy the contents of the main register R_0 into the corresponding cell on the analog tape (we will denote this operation by $C_{0\to}$);
  - we can copy the contents of the corresponding cell on the analog tape into the main register R_0 (we will denote this operation by $C_{\to 0}$); or
  - we can copy the contents of one of the registers into another one (copying from register R_i to the register R_j will be denoted by $C_{i\to j}$).

  In addition to copying, we can perform some *operations* with the analog values stored in the registers; e.g.:
  - the operation 1 will be performed as
$$r_0 := 1.0;$$
  - the operation + will be performed as
$$r_0 := r_1 + r_2;$$
  - the operation $-$ will be performed as
$$r_0 := r_1 - r_2;$$
  - the operation sin will be performed as
$$r_0 := \sin(r_1),$$
    etc.

Now, we are ready for a formal definition. In this definition, we will extend the formal notion of a Turing machine as defined in [29].

5.3 Definitions

Definition 1. *By a* Quantum Field Theory computer *(or* QFT computer*, for short), we will mean a quadruple $M = (K, \Sigma, \delta, s)$, where:*

K *is a finite set whose elements are called* states*;*
s *is an element of the set K;*
Σ *is a finite set of symbols (that will be called an* alphabet *of M); we assume that K and Σ are disjoint sets, and that the set Σ contains the special symbols $\sqcup$ and $\triangleright$ that are called the* blank *and the* first *symbol.*
δ *is a function (called* transition function*) from $K \times \Sigma \times \{T, F\}$ into*

$$S \times \Sigma \times M_s \times M_a \times (\{\emptyset\} \cup C \cup O),$$

where:

- *$S = K \cup \{h$, "yes", "no"$\}$ is called the set of possible* resulting states*;*
- *$M_s = \{\leftarrow_s, \rightarrow_s, -_s\}$ is called* the set of possible cursor motions on the standard tape*;*
- *$M_a = \{\leftarrow_a, \rightarrow_a, -_a\}$ is called* the set of possible cursor motions on the analog tape*;*
- *$C = \{C_{0\rightarrow}, C_{\rightarrow 0}, C_{i\rightarrow j}\,(0 \leq\ i, j \leq\ 2)\}$ is called* the set of copying operations*;*
- *$O = \{1, +, -, \cdot, /, \sin, \arcsin\}$ is called the* set of analog operations.

We assume that the following symbols are not in $K \cup \Sigma$:

- *h (called the* halting state*);*
- "yes" *(called the* accepting *state);*
- "not" *(called the* rejecting *state);*
- *the* cursor directions*:*
  - *$\leftarrow_s, \leftarrow_a$ (called* left*);*
  - *$\rightarrow_s, \rightarrow_a$ (called* right*); and*
  - *$-_s, -_a$ (called* stay*), and*
- *the operation symbols for the analog tape.*

Definition 2. *By a* configuration *of a QFT computer, we mean a tuple*

$$(q, w_s, u_s, w_a, u_a, r_0, r_1, r_2),$$

where:

q *is a state (i.e., an element of the set K);*
w_s *is a finite sequence of symbols from Σ (it describes all the symbols to the left of the cursor, including the symbol scanned by the cursor);*
u_s *is a finite sequence of symbols from Σ (it describes all the symbols to the right of the cursor);*

w_a *is a finite sequence of real numbers (it describes all the real numbers on the analog tape that are to the left of the cursor, including the real number currently scanned by the cursor);*
u_a *is a finite sequence of real numbers (it describes all the real numbers on the analog tape that are to the right of the cursor; we can always add 0's to this sequence);*
r_i *are real numbers;* r_i *is called the* current contents of the register *i.*

Comment. We will not go into the technicality of defining the initial configuration and a step of the newly defined machine.

Basically, we assume that in the initial configuration, only the standard tape contains the data, i.e., that in the initial configuration we have:

- $q = s$,
- $w_s = \triangleright$,
- $r_0 = r_1 = r_2 = 0$, and
- all zeros are written on the analog tape.

The transition to a new configuration is done according to the description of the operations given above.

The decision is based:

- on the current state ($q \in K$),
- on the symbol ($\sigma \in \Sigma$) currently scanned on the standard tape, and
- on whether the current contents r_0 of the main register R_0 is equal to 0 (this case is denoted by a symbol T), or not (this case is denoted by the symbol F).

Based on this information, the function δ determines what to do next. For each tuple (q, σ, t), where $t \in \{T, F\}$, this function δ returns a tuple

$$(q', \sigma', m_s, m_a, o),$$

where:

q' is a new state of the head;
σ' is a new symbol overwriting the old symbol on the standard tape's cell that is currently scanned by the cursor;
m_s is a motion of the cursor that scans the standard tape;
m_a is a motion of the cursor that scans the analog tape;
o is an operation that is performed with the registers.

(The knowledgeable reader can easily fill in the details.)

Comment. If we are not using the new tape and the new registers at all, then we have a standard Turing machine. Thus:

- every function computable on a standard Turing machine (i.e., every function computable in the usual sense of this word)
- is computable on a QFT computer as well.

5.4 Main result and its proof

Theorem. *Every problem from the class NP can be solved on a QFT computer in polynomial time.*

Proof. We will show that by using this new computer, we can solve the partition problem (known to be NP-complete) in polynomial time. Since partition problem is NP-complete, from the fact that we can solve it in polynomial time, it follows that we can solve any other problem from the class NP in polynomial time.

The partition problem is listed as Problem SP12 in [19]; its NP-completeness was proven in the pioneer paper by R. M. Karp [26]. This problem is as follows (we will slightly change notations):

GIVEN: an integer n, and n positive integers

$$s_1, \ldots, s_n.$$

QUESTION: do there exist values

$$\varepsilon_1, \ldots, \varepsilon_n \in \{-1, 1\}$$

for which

$$\varepsilon_1 \cdot s_1 + \ldots + \varepsilon_n \cdot s_n = 0?$$

We will show how to solve this problem on a QFT computer in polynomial time for $n \geq 3$. Let us first describe how we can solve several auxiliary problems in polynomial time:

1°. According to the definition of QFT computer, we can check *equality* of two real numbers in one step. Using this possibility, we can *check inequality* between real numbers by using the following simple property: for any real numbers a and b,

$$a \leq b \text{ if and only if } b - a = |b - a|.$$

Thus, we can check in three steps whether $a \leq b$ or not:

1. apply the subtraction operation to compute

$$b - a;$$

2. apply the absolute value to compute

$$|b - a|;$$

3. compare the real numbers obtained on Steps 1 and 2.

2°. Let us now show that, given a integer k, a QFT computer can compute the real number

$$P(k) = 2^{2^k}$$

in time bounded by a polynomial of k.

Indeed:

- We can easily generate a real number 1.0.
- Then, we can compute $P(0) = 2.0$ as $1.0 + 1.0$.
- From $P(j)$, we can compute $P(j+1)$ by a single multiplication operation, as

$$P(j) \cdot P(j).$$

- So, we can compute $P(k)$ by computing sequentially

$$P(1), P(2), \ldots, P(k).$$

This computation takes $k+1$ steps.

3°. Let us now use this auxiliary result to show that, given an integer s, we can compute the real number 2^s in *polynomial time* (i.e., in time bounded by a polynomial of the length of the binary number s).

Indeed, if an integer is given in its binary form

$$s = d_k d_{k-1} \ldots d_0,$$

this means that

$$s = d_k \cdot 2^k + d_{k-1} \cdot 2^{k-1} + \ldots + d_0 \cdot 2^0,$$

and hence, 2^s is equal to the product of $k+1$ factors $2^{d_j \cdot 2^j}$, $0 \le j \le k$. When $d_j = 0$, the corresponding factor is equal to 1, so it is sufficient to consider only the factors for which $d_j = 1$. Hence,

$$2^s = \prod_{j:\ d_j = 1} 2^{2^j},$$

where the product is taken over all j for which $d_j = 1$.

According to Part 2 of this proof, the time for computing each factor is bounded by a linear function of k.

- There are at most k factors.
- Since the length of s is $k+1$, we can thus compute 2^s in quadratic time.

4°. Similarly, for a given integer s, we can compute 2^{-s} in polynomial (quadratic) time, as $1/2^s$.

5°. To solve the partition problem, we will compute $\pi = 2 \cdot \arcsin(1.0)$,

$$\Pi = \prod_{i=1}^{n} (2^{2n \cdot s_i} + 2^{-2n \cdot s_i}),$$

and

$$\alpha = \sin(2\pi \cdot \Pi \cdot 2^{-2n}).$$

Then:

- If $\alpha \geq \sin(2\pi \cdot 2^{-2n})$, then we conclude that the desired partition exists, i.e., that there exist ε_i for which
$$\sum \varepsilon_i \cdot s_i = 0;$$
- otherwise, we conclude that the desired partition does not exist.

Due to Parts 3 and 4 of the proof, computing each term in the product Π takes polynomial (quadratic) time. Thus, the entire product can be actually computed in cubic time. The remaining parts of the algorithm are even faster, hence, the entire algorithm takes *cubic time* to run.

To complete the proof, we must show that this algorithm returns a correct answer. Indeed, the product Π of the sums can be represented as the sum of all possible products:
$$\Pi = \sum 2^{2n\cdot(\varepsilon_1\cdot s_1+\ldots+\varepsilon_n\cdot s_n)},$$
where the sum is taken over all 2^n possible combinations of signs
$$(\varepsilon_1,\ldots,\varepsilon_n).$$
This sum, in its turn, can be represented as
$$\Pi = \Sigma_+ + \Sigma_0 + \Sigma_-,$$
where:

- Σ_+ is the sum of all terms for which $\sum \varepsilon_i \cdot s_i > 0$;
- Σ_0 is the sum of all terms for which $\sum \varepsilon_i \cdot s_i = 0$;
- Σ_- is the sum of all terms for which $\sum \varepsilon_i \cdot s_i < 0$.

Hence,
$$\alpha = \sin(A),$$
where
$$A = 2\pi \cdot \Pi \cdot 2^{-2n} = 2\pi \cdot \Sigma_+ \cdot 2^{-2n} + 2\pi \cdot \Sigma_0 \cdot 2^{-2n} + 2\pi \cdot \Sigma_- \cdot 2^{-2n}.$$
Each term in Σ_+ has the form $2^{2n\cdot p}$ for a positive integer p and is, therefore, an integer multiple of 2^{2n}. Hence, the expression
$$2\pi \cdot \Sigma_+ \cdot 2^{-2n}$$
is an integer multiple of 2π and can, therefore, be omitted in the argument of sin. Thus,
$$\alpha = \sin(B),$$
where
$$B = 2\pi \cdot \Sigma_0 \cdot 2^{-2n} + 2\pi \cdot \Sigma_- \cdot 2^{-2n}.$$

- If a particular instance of the partition problem *has no solutions*, then $\Sigma_0 = 0$. Each terms in Σ_- is of the type $2^{-2n\cdot p}$ for a positive integer p and, therefore, it

cannot exceed 2^{-2n}. There are, totally, no more than 2^n terms in the entire sum; therefore,

$$\Sigma_- \leq 2^n \cdot 2^{-2n} = 2^{-n}.$$

Hence, the argument B of the expression $\alpha = \sin(B)$ is $\leq 2\pi \cdot 2^{-3n}$. For $n > 0$, we have $2^{-3n} < 2^{-2n}$, and therefore,

$$B < 2\pi \cdot 2^{-2n}.$$

For all $n \geq 1$, we have

$$2^{-2n} \leq 2^{-2} = 1/4,$$

hence,

$$2\pi \cdot 2^{-2n} \leq 2\pi \cdot (1/4) = \pi/2.$$

Thence, B belongs to the interval $[0, \pi/2]$ on which the function $\sin(B)$ is strictly increasing. Hence, from

$$B < 2\pi \cdot 2^{-2n},$$

it follows that

$$\alpha = \sin(B) < \sin(2\pi \cdot 2^{-2n}).$$

- If a given instance of the partition problem *has a solution*, then the sum Σ_0 has at least one term. By definition of Σ_0, each of these terms is equal to 1, so, the fact that we have at least one term means that $\Sigma_0 \geq 1$ and hence, that

$$B \geq 2\pi \cdot \Sigma_0 \cdot 2^{-2n} \geq 2\pi \cdot 2^{-2n}.$$

Let us show that B is within the interval $[0, \pi/2]$ on which the sine function is strictly increasing. We have already proven that

$$\Sigma_- \leq 2^{-n}.$$

In the total sum, there are 2^n terms, so

$$\Sigma_0 \leq 2^n.$$

Hence,

$$B = 2\pi \cdot \Sigma_0 \cdot 2^{-2n} + 2\pi \cdot \Sigma_- \cdot 2^{-2n} \leq$$
$$2\pi \cdot 2^n \cdot 2^{-2n} + 2\pi \cdot 2^{-n} \cdot 2^{-2n} = 2\pi \cdot (2^{-n} + 2^{-3n}).$$

From the condition $n \geq 3$, we can conclude that

$$B \leq 2\pi \cdot (2^{-n} + 2^{-3n}) \leq 2\pi \cdot (2^{-3} + 2^{-9}) < 2\pi \cdot (1/4) = \pi/2.$$

From $B \in [0, \pi/2]$ and

$$B \geq 2\pi \cdot 2^{-2n},$$

we can now conclude that

$$\alpha = \sin(B) > \sin(2\pi \cdot 2^{-2n}).$$

The theorem is proven.

5.5 This result is not so surprising since our "computer" uses quantum effects

The fact that our newly proposed "computer" can solve the problems faster than the traditional computers is not so surprising if we take into consideration that the proposed computation model uses specifically *quantum* effects (namely, quantum vacuum fluctuations).

The idea that the use of quantum effects can make computers faster and more efficient was first proposed by R. Feynman [15]. Shortly after a quantum computing was formally described by D. Deutsch [13], it was shown that *some* computational problems can indeed be drastically sped up by using such computers [14].

The most convincing example of such a speed-up was given by P. Shor [33], who showed that quantum computers can solve, in polynomial time, the problem of factoring integers into prime factors, the problem known to be very tough.

In view of this success of computers that use *quantum mechanics*, it is no wonder that computers that use the more powerful physical processes of *quantum field theory* can do even more: namely, solve NP-complete problems in polynomial time.

Comment. In the context of quantum computing, our result can be viewed as a partial answer to the question raised in [34]: can other models of quantum computing solve some problems that the existing models cannot yet solve. Our answer is: yes, the new QFT model can do what previous models could not: namely, it can solve all NP-problems in polynomial time.

6 Open Problems

6.1 Engineering problem: how can we implement these computations?

The first important problem is of engineering nature: how can we implement these computations?

This is a very tough engineering problem, because, as we have already mentioned, even the discontinuity phenomenon itself, although it follows directly from

the equations of mainstream physics, has not yet been confirmed by a direct experiment.

6.2 Theoretical problem: what else can we compute on a Quantum Field Theory computer?

In this paper, we have proposed a new computation model, and we have shown that within this computation model, we can solve all problems from the class NP is polynomial time.

Similar to the class QP that was introduced in [9] as the class of all the problems that can be solved on a quantum computer in polynomial time, we can introduce a new class QFTP of all the problem that can be solved on the above-described Quantum Field Theory computer in polynomial time.

In terms of this denotation, our main result says that

$$\text{NP} \subseteq \text{QFTP}.$$

Since complement is easily handled by this computer, the class coNP also belongs to QFTP. So,

$$\text{NP} \cup \text{coNP} \subseteq \text{QFTP}.$$

The main open problem is:

What else is in QFTP?

Specifically:

- Does this new complexity class QFTP contain any higher-level classes in the polynomial hierarchy?
- For example, does this new complexity class contain the class Σ_2P?
- Does this complexity class contain all Turing computable functions?
- Can this new complexity class contain any function that is *not* Turing computable?
- What is the relation between this class QFTP and different structural complexity classes of *quantum computing* (e.g., the class QP and the classes introduced in [8])?

Acknowledgments. This work was supported in part:

- by the National Science Foundation grants
  - HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and
  - DUE-0926721, and
- by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

We are thankful:

- to Andrei A. Grib and Vladimir M. Mostepanenko who patiently explained their physical results to us,
- to Michael G. Gelfond and Yuri Gurevich for valuable discussions of constructive real numbers, and
- to James M. Salvador for valuable discussions of the corresponding chemical algorithms.

References

1. O. Aberth, *Precise numerical analysis*, Wm. C. Brown Publishers, Dubuque, Iowa, 1988.
2. J. Aihara and H. Hosoya, *Bull. Chem. Soc. Japan*, 1988, Vol. 61, pp. 2657–ff.
3. A. I. Akhiezer and V. B. Berestetskii, *Quantum electrodynamics*, Pergamon Press, N.Y., 1982.
4. M. J. Beeson, *Foundations of constructive mathematics*, Springer-Verlag, N.Y., 1985.
5. R. A. Beezer and E. J. Farrell, "The matching polynomial of a regular graph", *Discrete Mathematics*, 1995, Vol. 137, No. 1, pp. 7–18.
6. R. A. Beezer, E. J. Farrell, J. Riegsecker, and B. Smith, "Graphs with a minimum number of pairs of independent edges I: Matching polynomials", *Bulletin of the Institute of Combinatorics and Its Applications*, 1996 (to appear).
7. A. Beltran and J. M. Salvador, "The Ulam index", *Abstracts of the Second SC-COSMIC Conference in Computational Sciences, October 25–27, El Paso, TX*, Rice University Center for Research on Parallel Computations and University of Texas at El Paso, 1996, p. 6.
8. E. Bernstein and U. Vazirani, "Quantum complexity theory", *Proceedings of the 25th ACM Symposium on Theory of Computing*, 1993, pp. 11–20.
9. A. Berthiaume and G. Brassard, "The quantum challenge to structural complexity theory", *Proceedings of the 7th IEEE Conference on Structure in Complexity Theory*, 1992, pp. 132–137.
10. E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, 1967.
11. E. Bishop and D. S. Bridges, *Constructive Analysis*, Springer, N.Y., 1985.
12. D. S. Bridges, *Constructive Functional Analysis*, Pitman, London, 1979.
13. D. Deutsch, "Quantum theory, the Church-Turing principle, and the universal quantum computer", *Proceedings of the Royal Society of London, Ser. A*, 1985, Vol. 400, pp. 96–117.
14. D. Deutsch and R. Jozsa, "Rapid solution of problem by quantum computation", *Proceedings of the Royal Society of London, Ser. A*, 1992, Vol. 439, pp. 553–558.
15. R. Feynman, "Simulating physics with computers", *International Journal of Theoretical Physics*, 1982, Vol. 21, pp. 467–488.
16. D. Finkelstein and J. M. Gibbs, "Quantum relativity", *International Journal of Theoretical Physics*, 1993, Vol. 32, p. 1801.
17. D. Finkelstein, *Quantum relativity*, Springer-Verlag, Berlin, Heidelberg, 1996.
18. V. M. Frolov, A. A. Grib, and V. M. Mostepanenko, "Conformal symmetry breaking and quantization in curved space-time", *Phys. Lett. A*, 1978, Vol. 65, pp. 282–284.
19. M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.
20. A. A. Grib, S. G. Mamayev, and V. M. Mostepanenko, *Vacuum Quantum Effects in Strong Fields*, Friedmann Laboratory Publishing, St.Petersburg, 1994 (Chapter 11).
21. A. A. Grib and V. M. Mostepanenko, "Spontaneous breaking of gauge symmetry in a homogeneous isotropic universe if the open type", *Sov. Phys.-JETP Lett.*, 1977, Vol. 25, pp. 277–279.
22. A. A. Grib, V. M. Mostepanenko, and V. M. Frolov, "Spontaneous breaking of gauge symmetry in a nonstationary isotropic metric", *Theor. Math. Phys.*, 1977, Vol. 33, pp. 869–876.

23. A. A. Grib, V. M. Mostepanenko, and V. M. Frolov, "Spontaneous breaking of CP symmetry in a nonstationary isotropic metric", *Theor. Math. Phys.*, 1978, Vol. 37, No. 2, pp. 975–983.
24. H. Hosoya, *Comp. Math. Appls.*, 1986, Vol. 12B, pp. 271–ff.
25. H. Hosoya and K. Balasubramanian, "Computational Algorithms for Matching Polynomials of Graphs from the Characteristic Polynomials of Edge-Weighted Graphs", *Journal of Computational Chemistry*, 1989, Vol. 10, No. 5, pp. 698–710.
26. R. M. Karp, "Reducibility among combinatorial problems", In: R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, N.Y., 1972, pp. 85–103.
27. B. A. Kushner, *Lectures on Constructive Mathematical Analysis*, Translations of Mathematical Monographs, Vol. 60, American Mathematical Society, Providence, RI, 1984.
28. L. D. Landau and E. M. Lifschitz, *Quantum mechanics: non-relativistic theory*, Pergamon Press, Oxford, N.Y., 1965.
29. C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.
30. M. Randic, H. Hosoya, and O. E. Polansky, "On the Construction of the Matching Polynomial for Unbranched Catacondensed Benzenoids", *Journal of Computational Chemistry*, 1989, Vol. 10, No. 5, pp. 683–697.
31. V. R. Rosenfeld and I. Gutman, "A novel approach to graph polynomials", *Match*, 1989, Vol. 24, pp. 191–ff.
32. J. M. Salvador, "Topological indices and polynomials: the partial derivatives", *Abstracts of the 5th International Conference on Mathematical and Computational Chemistry, May 17–21, 1993*, Kansas City, Missouri, p. 154.
33. P. W. Shor, "Algorithms for quantum computations: discrete logarithms and factoring", *Proceedings of the 35th Annual Symposium on Fundamentals of Computer Science (FOCS)*, 1994, pp. 124–134.
34. P. Simon, "On the power of quantum computation", *Proceedings of the 35th Annual Symposium on Fundamentals of Computer Science (FOCS)*, 1994, pp. 116–123.
35. A. M. Turing, "On computable numbers, with an application to the em Entscheidungsproblem", *Proc. London Math. Society*, 1936, Vol. 42, pp. 230–265; see also *Proc. London Math. Society*,1937, Vol. 43, pp. 544–546.