WHICH ALGORITHMS ARE FEASIBLE AND WHICH ARE NOT DEPENDS ON THE GEOMETRY OF SPACE-TIME

by Dara Morgenstein
and Vladik Kreinovich[1]
Computer Science Department
University of Texas at El Paso, El Paso, TX 79968
email dara@cs.utep.edu and vladik@cs.utep.edu

Abstract. It is usually believed that adding computer parallelism makes algorithms faster, but does not change their feasibility. The usual proof of this claim is based on the assumption that our space is Euclidean. But according to modern physics, the actual space-time geometry is non-Euclidean. We show that for several physically meaningful non-Euclidean spaces (including Lobachevsky space) adding parallelism does enlarge the class of feasible algorithms, to the point that every problem becomes (in principle) solvable by a feasible algorithm.

In general, whether or not adding parallelism changes feasibility, depends on a simple geombinatoric property of space: namely, on the number of (non-intersecting) spheres of radius $r_0 < R$ that can be placed inside a sphere of radius R.

1. WHAT IS A FEASIBLE ALGORITHM?

In theory of computation, it is well known that not all algorithms are feasible (see, e.g., (Garey et al 1979), (Lewis et al 1981), (Martin 1991)): it depends on how many computational steps they need. If for input of length n, the algorithm requires computational time 2^n, then for an input of a reasonable length $n \approx 300$, we would need more computational steps than there can be during the lifetime of our Universe. So, such algorithms (called *exponential-time*) are usually considered not feasible. If, however, the running time grows

[1] This work was partially supported by NSF grants No. CDA–9015006 and No. EEC–9322370. The authors are thankful to Andrew Bernat and Michael Zakharevich for valuable comments.

only as a polynomial of n (i.e., an algorithm is *polynomial-time*), then the algorithm is considered feasible.

In mathematical terms, this leads to a following definition:

Definition 1. An algorithm U is called *feasible* if there exists a polynomial $P(n)$ such that for every input x, the running time $t_U(x)$ of this algorithm does not exceed $P(|x|)$, where by $|x|$, we denoted the length of the input x (i.e., the number of bits that form an input).

This definition is not perfect: There are cases when an algorithm is polynomial-time but not feasible (e.g., if the running time is $10^{300} \cdot n$), and vice versa (Garey et al 1979), (Lewis et al 1981), (Martin 1991), but the above definition is the best that we can get in formalizing "feasible".

The fact that an algorithm is not feasible, does not mean that it can never be applied: it simply means that there are cases when its running time will be too large for this algorithm to be practical.

To explain why non-feasible algorithm naturally arise, let us describe what a *problem* is. When we say that there is a problem, we mean that:

- we have some information (we will denote its computer representation by x), and
- we know the relationship $R(x, y)$ between the known information x and the desired object y.

In the computer, everything is represented by a binary sequence (i.e., sequence of 0's and 1's), so we will assume that x and y are binary sequences.

Let us give two examples:

- (*Mathematics*) We are given a mathematical statement x. The desired object y is either a proof of x, or a "disproof" of x (i.e., a proof of "not x"). Here, $R(x, y)$ means that y is a proof either of x, or of "not x".
- (*Physics*) x is the results of the experiments, and the desired y is the formula that fits all these data.

For a problem to be meaningful, we must have a way to check whether the proposed solution is correct. In other words, we must assume that there exists an algorithm that checks $R(x, y)$ (given x and y). If no such feasible algorithm exists, then this is not a problem in a scientific sense of this word, because there is no criterion to decide whether we achieved a solution or not.

Another requirement for a real-life problem is that in such problems, we usually know an upper bound for the length $|y|$ of the description of y. In the above examples:

- *In mathematics,* a proof must be not too huge, else it is impossible to check whether it is a proof or not.
- *In physics,* it makes no sense to have a formula $y = f(x, C_1, ..., C_{40})$ with, say, 40 parameters to describe the results $(x_1, y_1), ..., (x_{10}, y_{10})$ of 10 experiments, for two reasons:
  - First, one of the goals of physics is to discover the laws of nature. If the number of parameters exceeds the number of experimental data, then no matter what dependency $f(x, C_1, ...)$ we choose, in order to determine C_i, we have, say, 10 equations with 40 unknowns. Such underdetermined system usually has a solution, so the fact that, say, a linear formula with many parameters fits all the experimental data does not mean that the dependency is proven to be linear: a quadratic or cubic formula with as many parameters will fit the same data as well.
  - Second, another goal of physics (definitely related to the first one) is to find a way to compress the data, so that we will not need to store all billions of experimental results in order to make predictions. A dependency y that requires more storage space than the original data x is clearly not satisfying this goal.

In all cases, it is necessary for a user to be able to read the desired solution symbol-after-symbol, and the time required for that reading must be feasible. In Section 1, we have formalized "feasible time" as a time that is bounded by some polynomial of $|x|$. The reading time is proportional to the length $|y|$ of the answer y. Therefore, the fact the reading time is bounded by a polynomial of

$|x|$ means that the length of the output y is also bounded by some polynomial of $|x|$, i.e., that $|y| \le P_L(|x|)$ for some polynomial P_L. So, we arrive at the following formulation of a problem:

Definition 2. By a *class of problems*, we mean a pair (R, P_L), where $R(x, y)$ is a feasible algorithm that transforms two binary sequences into a Boolean value ("true" or "false"), and P_L is a polynomial. By a *particular case* of this class, we mean the following problem:

- *Given:* a binary sequence x.
- *To generate:*
  - either y such that $R(x, y)$ is true and $|y| \le P_L(|x|)$,
  - or, if such a y does not exist, a message saying that there are no solutions.

For example, in mathematics, a particular case would be: given a statement, find its proof or disproof.

What we called "problems" are usually described as "problems from the class NP" (to separate them from non-real life mathematical problems in which the solution may not be verifiable). Classes of problems for which there is a feasible algorithm that solves all particular cases are called *tractable*, *easily solvable*, or "problems from the class P" (P from *P*olynomial). It is widely believed that not all problems are easily solvable (i.e., that NP$\neq$P), but it has never been proved.

One way to solve an NP problem is to check $R(x, y)$ for all binary sequences y with $|y| \le P_L(|x|)$. This algorithm (called *British Museum* algorithm) requires $2^{P_L(|x|)}$ checks. This algorithm takes exponential time and is therefore, not feasible.

2. DOES ADDING COMPUTER PARALLELISM CHANGE FEASIBILITY? THE USUAL PROOF

What are parallel computations? If running an algorithm takes too much time on a computer, then the natural idea is to use several computers to implement it. An algorithm that uses

several computers that work simultaneously is called *parallel*, and the entire phenomenon is called *parallelism.*

The problem: does computer parallelism change the class of feasible algorithms? In other words, is it possible that an algorithm that was not feasible on a sequential machine becomes feasible if we use several processors working in parallel?

It is usually believed that the class does not change. At first glance, it seems possible that adding very many processors can enable us to run a previously non-feasible algorithm really fast. However, there are arguments (and we will present them right now) that although we can drastically decrease the computation time by using parallel processors, but we cannot make a non-feasible algorithm feasible. These arguments are presented, e.g., in (Feldman et al 1992).

These arguments are based on a reduction to a geombinatorical problem. Let's describe them in detail.

Main assumption. Suppose that an algorithm U runs on a parallel machine, and that this algorithm is feasible, i.e., there exists a polynomial $P(n)$ such that for every input x, it produces the result in time $t_U(x) \le P(|x|)$.

First geometrical conclusion: all processors are located inside a certain sphere. Let's describe this computation process in terms of space and time. Let's denote the location of the user by L. At some moment of time t_0, the user (located at the point L) inputs the input data. The algorithm takes time $t_U(x)$ to run. After the algorithm has done its job, i.e., in the moment of time $t_0 + t_U(x)$, the user (who is still located at the same spatial point L) get the result of this algorithm.

How far away can the computing processors be located? The fastest speed of any process is the speed of light c. So, even if a computer has a processor that is located at a distance $r > ct_U(x)$ from L, and this processor performs some computations, these computations will not influence the result of the algorithm, because

even when traveling with a speed of light, they will reach the user only *after* the time r/c, and from $r > ct_U(x)$, we conclude that $r/c > t_U(x)$. Therefore, all processors that participate in this computation are located at a distance r that is $\leq ct_U(x)$ from L.

In geometric terms, we can thus conclude that these processors are located inside a sphere of radius $ct_U(x)$ with a center in L.

How many processors can we fit into that sphere? Processors can be small, and they are getting smaller and smaller, thinner and thinner, but on each technological level, there is a minimum thickness that we can achieve. We cannot make it arbitrarily thin: there is always some non-zero thickness d. In geometric terms, we can express this "minimum size" as follows: If a wooden board is, say, 3 *in* thick, and a worm starts in the middle of this board, then it must go at least 3/2 *in* to reach a surface. By a thickness of a body, we usually mean the smallest of its dimensions. So, if a board is 3 *in* thick, and a worm goes into any other direction, it will take the worm even longer to get out. So, if the thickness of the board is 3 *in*, then all the points located at a distance ≤ 1.5 *in* from its center are still inside this board. In other words, the entire sphere of radius 1.5 *in* is located inside the board.

Similarly, if a processor number i has a thickness d_i, then a sphere of radius $d_i/2$ is located inside this processor. We denote the smallest possible thickness of a processor by d. Therefore, for each processor i, $d_i \geq d$. So, $d/2 \leq d_i/2$. Therefore, if we take a sphere of radius $d/2$, it will be contained inside the (larger) sphere of larger radius $d_i/2$ and hence, inside the processor.

Reduction to a geombinatorical problem. So, we arrive at the following conclusion:

- all the processors are located inside the sphere of radius $R = ct_U(x)$;
- each processor contains inside itself a sphere of radius $r_0 = d/2$; small spheres that correspond to different processors do not intersect.

So, we can get an upper estimate on the total number of processors involved in computations, if we estimate the total number of non-

intersecting spheres of radius r_0 that one can fit into a sphere of radius $R = ct_U(x)$.

Solution of the geombinatorical problem. To find the desired upper bound, we can use the following idea: If there are totally N small spheres, and these small spheres do not intersect, then their total volume is equal to $N \cdot V(r_0)$ (here, for arbitrary r, $V(r)$ denotes the volume of a sphere of radius r; in Euclidean space, $V(r) = (4/3)\pi r^3$). On the other hand, this total volume $N \cdot V(r_0)$ cannot exceed the volume $V(R)$ of the large sphere (inside which all small spheres are located). Therefore, $N \cdot V(r_0) \le V(R)$, and so, $N \le V(R)/V(r_0)$.

Final step: what is feasible in parallel is also feasible on a sequential computer. So, for each input x, our parallel computer uses $N \le V(R)/V(r_0)$ processors. If we substitute the expression for the volume $V(r) = (4/3)\pi r^3$, we can conclude that $N \le (R/r_0)^3$. Since $R = ct_U(x)$, we conclude that

$$N \le (ct_U(x)/r_0)^3 = (c/r_0)^3 \cdot t_U(x)^3. \qquad (1)$$

We can simulate this algorithm on a sequential machine moment-after-moment: first, we simulate what each of N processors does in the first moment of time, then what they all do in the second moment of time, etc. To simulate each moment of time on a parallel machine, we thus need N moments of time on a sequential machine. So, the algorithm that took time $t_U(x)$ on a parallel machine with N processors, can be simulated on a sequential machine. The running time $t_V(x)$ of the resulting sequential algorithm V is equal to $t_V(x) = N \cdot t_U(x)$.

From (1), we conclude that

$$t_V(x) \le (c/r_0)^3 t_U(x)^4. \qquad (2)$$

We assumed that an algorithm U is feasible and therefore, $t_U(x) \le P(|x|)$ for some polynomial $P(n)$. Therefore, from (2), we conclude that $t_V(x) \le (c/r_0)^3 P(|x|)^4$. In other words, we can conclude that

$t_V(x) \le Q(|x|)$, where we denoted $Q(n) = (c/r_0)^3 P^4(n)$. Since $P(n)$ is a polynomial, $Q(n)$ is also a polynomial and therefore, the sequential algorithm V (that computes exactly the same results as U) is also feasible.

In other words, the usual conclusion is that *adding computer parallelism does not change the class of feasible functions.*

3. THE USUAL PROOF IS BASED ON EUCLIDEAN GEOMETRY BUT PHYSICAL SPACE-TIME IS NOT EUCLIDEAN

The above proof was based on Euclidean geometry: namely, we used the Euclidean formula for the volume of the sphere. But it is well known that the actual space-time is not Euclidean. Will the above conclusion change if we take the actual geometry into consideration?

4. LOBACHEVSKY SPACE: THE SIMPLEST NON-EUCLIDEAN GEOMETRY

Before we get into more realistic cases, let us describe what will happen in historically the first non-Euclidean geometry: Lobachevsky geometry.

How is Lobachevsky geombinatorics different from Euclidean one. In Lobachevsky space, the volume of a sphere of radius R is given by a formula

$$V(R) = 2\pi k^3(\sinh(R/k)\cosh(R/k) - R/k),$$

where $k > 0$ is a constant that characterizes the Lobachevsky space (this formula was first described by Taurinus; see, e.g., (Bonola 1955), Chapter III, Section 38). When $x \to \infty$,

$$\cosh(x) = \frac{\exp(x) + \exp(-x)}{2} \sim \frac{1}{2}\exp(x)$$

and similarly,

$$\sinh(x) = \frac{\exp(x) - \exp(-x)}{2} \sim \frac{1}{2}\exp(x).$$

So, when $R \to \infty$, we have $V(R) \sim 2\pi k^3((1/2)\exp(R/k))^2 = (1/2)\pi k^3 \exp(2R/k)$. Thus, if we fix $r_0 > 0$, then for sufficiently large R, we can easily place $\approx V(R)/V(r_0) \sim \text{const} \cdot \exp(2R/k)$ non-intersecting spheres of radius r_0 in a sphere of radius R.

Because of the above geombinatoric property of the Lobachevsky space, in this space, we can solve all (NP) problems in feasible (polynomial) time. Indeed, suppose that we have an input of size n. Then, British Museum algorithm consists of checking $2^{P_L(n)}$ sequences. So, if we have $2^{P_L(n)}$ processors working in parallel, then the total time t of this computation will consist of two parts $t = t_1 + t_2$:

- t_1 is the time that each processor takes to check whether for given x and y, $P(x, y)$ is true or not. Since we assumed that R is feasible, this time is bounded by a polynomial of $|x| + |y|$: $t_1 \le P(|x| + |y|)$. Since $|y| \le P_L(|x|)$, we have

  $$t_1 \le P(n + P_L(n)).$$

  So, t_1 is bounded by a polynomial of n.
- t_2 is the time necessary to communicate the results of these checks to the user (if one of the checks resulted in "true", then one of the corresponding y is returned; else, the no-solutions message). If we use speed of light, then this time is $t_2 \le R/c$, where R is the maximum distance between the user and the processor.

In Lobachevsky space, we can place $N \sim \text{const} \cdot \exp(2R/k)$ processors of size r_0 inside a sphere of radius R. So, to locate all $N = 2^{P_L(n)}$ processors, we need to choose R for which that $2^{P_L(n)} \sim \text{const} \cdot \exp(2R/k)$. To determine R, we must take the logarithms of both sides: $P_L(n)\ln(2) \sim 2R/k + \text{const}$. Hence, $R \sim (k/2)\ln(2)P_L(n) + \text{const}$. This radius does not exceed a polynomial of n. Therefore, the time t_2, that is $\le R/c$, is bounded by a polynomial. Since t_1 is also bounded by a polynomial, we can conclude that the total time $t_1 + t_2$ is bounded by a polynomial of n, i.e., that *this algorithm is feasible.*

In other words, for Lobachevsky space, *adding parallelism changes the class of feasible algorithms:*

- the British Museum algorithm is *not* feasible on sequential computers, but
- this same algorithm *is* feasible on a parallel computer.

As a corollary, we can conclude that *which algorithms are feasible and which are not depends on the geometry of space-time:*

- the British Museum algorithm is *not* feasible in Euclidean space, but
- this same algorithm *is* feasible in Lobachevsky space.

Historical comment. It is interesting that the very idea that living in Lobachevsky space may simplify some computations dates back to Lobachevsky himself: Namely, Lobachevsky noticed that many geometric formulas (for surfaces, volumes, angles, etc) are more complicated in his geometry than the corresponding formulas of Euclidean geometry. There are two ways to view this complexity:

- At first glance, it seems that this comparative complexity can only lead to a *pessimistic* conclusion: since similar geometric computations are more complicated in Lobachevsky space, live in Lobachevsky space is much more complicated: navigation, measurement, etc, would require more computational efforts.
- Lobachevsky himself discovered that this same complexity can be viewed positively, in an *optimistic* way: namely, that in Lobachevsky space, simple geometrical and physical experiments can easily compute the values of complicated functions that in Euclidean space require lengthy computations. E.g., by making a ball of a given radius R, filling it with water, and weighing it, we can thus easily get the value $2\pi k^3(\sinh(R/k)\cosh(R/k) - R/k)$ that would require lengthy computations in Euclidean space. So, due to Lobachevsky geometry, some computations can be faster in Lobachevsky space.

In this section, we have described another sense in which computations are faster in Lobachevsky space.

5. A MORE REALISTIC GEOMETRY OF SPACE-TIME IN WHICH ALL (NP) PROBLEMS ARE EASILY SOLVABLE

General case: open problem. The above result may be theoretically interesting, but it is not very realistic because the actual space-time, although non-Euclidean, is not a Lobachevsky space either. So, whether it is possible to solve all (NP) problems in polynomial time depends on what exactly geometry we have. For each new physical model of space-time, checking the corresponding geombinatoric properties (and correspondingly, checking whether all problems can be solved by a universal feasible algorithm) is an *open problem*.

In this paper, we will only consider two space-time models: open cosmological model and a model with "alsmot" black holes.

Open cosmological model and a related hypothesis. In a so-called *open cosmological model* (see, e.g., (Misner et al 1973)), the space is described (in the homogeneous case) by Lobachevsky geometry. So, it is reasonable to conjecture that in non-homogeneous open space-times, all problems are also easily solvable.

Black holes and "almost" black holes. Another (more realistic) example of a space-time model in which all problems *are* easily solvable is related to black holes and "almost" black holes. One of the well-known consequences of general relativity is the existence of the "black hole" solutions. A black hole is an area from which nothing comes out (in particular, light cannot escape it, hence it looks black). It is proved that if an object (e.g., a star) is massive enough, it will eventually be crushed by its own gravitational force and form a black hole. If the object is smaller, or if it has a significant electric charge, then it forms an "almost" black hole, i.e., an area from which it is possible but difficult to escape. If you enter this "almost" black hole, you go into the narrow throat (Misner et al 1973, Chapters 31, 44). From the outside, it looks like a small particle. So, a natural hypothesis (Misner et al 1973, Chapter 44)

is that all charged elementary particles *are* actually such "almost" black holes.

What happens after you enter this (highly curved) throat? According to the equations of general relativity, after a while, you end up in a space-time that is not so curved. So, this throat is like a *wormhole* connecting two regions of space-time. There are two possible interpretations of this mathematical result:

- *The second region belongs to the same space-time.* In this case, we have a wormhole (like a "tunnel" under the world) that connects two distant regions of space-time. Communication via this channel turns out to be much faster than sending light. As a result, we get the possibility of communication that is much faster than the speed of light. Therefore, we can also perform computations much faster. This wormhole interpretation (see, e.g., (Penrose 1989)) has, however, some problems with causality. So, let us turn to the second interpretation:
- *The second region belongs to a different part of space-time.* In other words, each "almost" black hole leads to a separate "world" that is connected with our world by a narrow "wormhole", and that, for our world, has an outside appearance of an elementary particle. Each of these worlds has its own elementary particles, which, in this interpretation, are gateways to yet another worlds, etc. So, the space-time is like a branching tree: in each elementary particle, our world is branching into a different world, etc.

In branching space-time, parallelized algorithms can solve all (NP) problems in feasible time. Let us show that we can use the branching structure to solve all (NP) problems in feasible time. Indeed, suppose that we have a problem. This means that we are given a sequence x of length $n = |x|$, and we need to find a binary sequence $y = (y_1 ... y_N)$ of size $\leq N = P_L(n)$ that satisfies the condition $R(x, y)$. To find y, we will use a parallelization of the British Museum algorithm. For this parallelization, we will need a *self-replicating* computer (i.e., a computer equipped with a robot that, being introduced into any environment, manufactures its own exact copy).

- We start with the self-replicating computer located in our world. This computer is responsible for checking all sequences y of length $\leq N$ (i.e., this checking will be done by this computer, the computers that it creates, the computers that its creations create, etc.).
- We pick two elementary particles from our world (i.e., two new worlds W_0 and W_1). The self-replicating computer:
  - makes two copies of itself, and
  - sends these copies to the worlds W_0 and W_1.

  Each of these new computers will be responsible for checking half of the sequences. Namely:
  - The computer that is located in the world W_0 is responsible for checking all the sequences with $y_1 = 0$.
  - The computer that is located in the world W_1 is responsible for checking all sequences with $y_1 = 1$.

  As a result, each of these computers has 2^{N-1} sequences to check.
- The computer located in each world W_i will, in its turn, do the following:
  - find two elementary particles in his world W_i; these particles correspond to two new worlds W_{i0} and W_{i1};
  - make two copies of itself; and
  - send these copies to worlds W_{i0} and W_{i1}.

  Each of these new computers will be responsible for checking a half of the sequences allocated to its parent-computer W_i. Namely:
  - The computer that is located in the world W_{i0} is responsible for checking all sequences with $y_1 = i$ and $y_2 = 0$.
  - The computer that is located in the world W_{i1} is responsible for checking all sequences with $y_1 = i$ and $y_2 = 1$.

  Now, in addition to the main computer, and to computers W_i, we have four computers in four worlds:
  - The computer in the world W_{00} that is responsible for checking all sequences with $y_1 = 0$ and $y_2 = 0$ (after checking, this computer reports to its parent computer that is located in the world W_0).

  - The computer in the world W_{01} that is responsible for checking all sequences with $y_1 = 0$ and $y_2 = 1$ (after checking, this computer reports to its parent computer that is located in the world W_0).
  - The computer in the world W_{10} that is responsible for checking all sequences with $y_1 = 1$ and $y_2 = 0$ (after checking, this computer reports to its parent computer that is located in the world W_1).
  - The computer in the world W_{11} that is responsible for checking all sequences with $y_1 = 1$ and $y_2 = 1$ (after checking, this computer reports to its parent computer that is located in the world W_1).
- The computer located in each world W_{ij}, will, in its turn, do the following:
  - find two elementary particles in his world W_{ij}; these particles correspond to two new worlds W_{ijk}, $k = 0.1$;
  - make two copies of itself; and
  - send these copies to the worlds W_{ijk}.

  Each of these new computers will be responsible for checking a half of the sequences allocated to its parent-computer W_{ij}; namely, for all sequences with $y_1 = i, y_2 = j$, and $y_3 = k$. After checking, this computer reports to its parent computer that is located in the world W_{ij}.
- ...
- After N steps, we get 2^N computers in 2^N worlds $W_{i_1 \ldots i_N}$, each of which is responsible for only one sequence y to check: $y_1 = i_1, ..., y_N = i_N$. So, it checks that sequence, and sends the result of that checking back to its parent-computer (that is located in the world $W_{i_1 \ldots i_{N-1}}$).
- Each computer located in the world $W_{i_1 \ldots i_{N-1}}$ receives two messages from its creations $W_{i_1 \ldots i_{N-1} i_N}$. There are three possibilities:
  - If both checks failed, this means that among sequences supervised by this computer there are no solutions. In this case, a no-solutions message is sent to its parent computer that is located in the world $W_{i_1 \ldots i_{N-2}}$.

- If exactly one of checks succeeded, then the corresponding sequence $y = (i_1...i_N)$ is passed to the parent computer.
- If both checks succeeded, then one of the corresponding sequences $y = (i_1...i_N)$ is passed to the parent computer.

- ...
- Each computer located in the world $W_{i_1...i_k}$ receives two messages from its creations $W_{i_1...i_k0}$ and $W_{i_1...i_k1}$. There are three possibilities:
  - If both creations sent a no-solutions message, then a no-solutions message is passed to the parent computer that is located in the world $W_{i_1...i_{k-1}}$.
  - If one of the creations sent a no-solutions message, and another a binary sequence, then this sequence is passed to the parent computer.
  - If both creations sent a binary sequence, then one of these sequences is passed to the parent computer.
- ...
- At the end, the original computer receives two messages from its creations located in the worlds W_i. If both are no-solutions messages, then a no-solution message is passed to the user. If one of these messages is a solution, then one of these solutions is passed to the user.

The total time of this computation is $t_1 + t_2 + t_3$, where:

- t_1 is the time that is necessary to create the computers, and to send them into their corresponding worlds. Computers of each generation (i.e., both computers in W_i, all four computers in W_{ij}, ..., all 2^k computers in the worlds $W_{i_1...i_k}$, ...) are created simultaneously. Therefore, the total time that is necessary for this creation is proportional to the total number of generations, i.e., to N. Since $N = P_L(n)$, this time is bounded by a polynomial in n.
- t_2 is the check time (that is polynomial in $n = |x|$).
- t_3 is the time for a message to get back to us. This message passing requires N world-to-world communications: from $W_{i_1...i_N}$ to $W_{i_1...i_{N-1}}$, ..., from $W_{i_1...i_{k+1}}$ to $W_{i_1...i_k}$, ..., and finally, from W_{i_1} to our world. So, it takes time $\text{const} \cdot N \leq$

const $\cdot P_L(n)$ (here, const is the time required for a communication between the worlds). Hence, t_3 is also a feasible time. Therefore, the total time $t_1 + t_2 + t_3$ is feasible.

Comment. Instead of using "self-replicating" computers, we could as well assume that the computers that are used in this algorithm are already in place in different "worlds".

In geometric terms, the fact that we managed to get 2^N processors located at a linear-time distance (i.e., distance $\leq$ const $\cdot N$) from the user means that for this "branching" space-time, the volume $V(R)$ of the sphere grows at least exponentially with R, and therefore, we can fit exponentially many spheres of radius r_0 into a large sphere of radius R.

In other words, for this "branching" space-time, *adding parallelism changes the class of feasible algorithms:*

- the British Museum algorithm that is *not* feasible on sequential computers, but
- this same algorithm *is* feasible on a parallel computer.

This fact is one more proof that *which algorithms are feasible and which are not depends on the geometry of space-time:*

- the British Museum algorithm is *not* feasible in Euclidean space, but
- this same algorithm *is* feasible in a "branching" space-time.

6. PHILOSOPHICAL CONCLUSIONS: GEOMBINATORICS AND LEIBNIZ'S DREAM

Leibniz dreamed of a universal language in which all discussions would disappear: if there is a disagreement, we would write down the disputed statement in this universal language, and compute the (using modern language) truth value of this statement (i.e., determine whether it is true or false).

In other words, Leibniz dreamed of a feasible algorithm that would solve all real-life problems. Is such an algorithm possible? I.e., in our terms, is the class of all problems (NP) equal to the class

of all easily solvable problems (P)? It is usually believed that these classes are different. There is no proof, but there are arguments in favor of P≠NP. However, as we have shown, these arguments are only valid in Euclidean space. If we live in Lobachevsky space, or in a branching space, where each elementary particle is an "almost" black hole, then every problem *is* easily solvable.

So, whether Leibniz's dream is a reality or not depends on the geombinatoric properties of the space-time in which we live. And even if we are optimistic, what to believe in, depends on what we mean by optimism:

- If we wish to reach the ultimate knowledge, then being optimistic means to believe that some day, Leibniz's dream will come true, and we will able to get a (practically immediate) answer to any meaningful question. According to this belief:
  - Science (as we know it) will stop, and only spiritual life will continue.
  - Of two main possible cosmological models:
    - *open* (with Lobachevsky space), and
    - *closed* (with a spherical space),

    we would then prefer an open one.
- If we wish to be challenged every day by the mysteries of this world, then being optimistic means that we will never know everything, that we will never find an algorithm that answers all the questions. According to this belief:
  - Science and quest for knowledge will never stop.
  - And of two main possible cosmological models, we would then prefer a closed one.

Both are noble and nontrivial beliefs, and to think of it that both are related to something as seemingly un-inspiring as the number of small spheres packable into a large one!

REFERENCES

Bonola, R. *Non-Euclidean geometry*, Dover Publications, Inc., New York, 1955.

Feldman, Y., and Shapiro, E. *Spatial machines: a more realistic*

approach to parallel computations, Communications of the ACM, 1992, Vol. 35, No. 10, pp. 61–73.

Garey, M. R., and Johnson, D. S. *Computers and intractability: a guide to the theory of NP-completeness*, W. F. Freeman, San Francisco, 1979.

Guth, A. H., and Steinghardt, P. J. *The inflationary Universe*, Scientific American, 1984, Vol. 250, No. 5, p. 116–129.

Lewis, H.R. and Papadimitriou, C.H. *Elements of the Theory of Computation*, Prentice-Hall, Inc., New Jersey, 1981.

Martin, J. C. *Introduction to languages and the theory of computation*, McGraw-Hill, N.Y., 1991.

Misner, C.W., Thorne, K.S. and Wheeler, J.A. *Gravitation*, W.H. Freeman and Company, San Francisco, 1973.

Penrose, R. *The Emperor's new mind*, Oxford Univ. Press, Oxford, 1989.