Q.Cloud AI Chain Reaction Algorithm™
Provisional Patent Application Specification
Inventor: Steven Leonard

Background

There is yet to be a "quick" P-versus-NP Solution by a practical computer system. Such asks whether every language accepted by a non-deterministic algorithm in polynomial time is one that is also accepted by any deterministic algorithm in polynomial time. The Problem essentially explores what are the fundamental limits of feasible computation.

Stephen Cook (University of Toronto) wrote in his Article, "THE P VERSUS NP PROBLEM", that with regard to the P-versus-NP problem experience teaches that when natural problems are proven to be in P, a feasible algorithm can be found. (Otherwise, the proof is nonconstructive, no algorithm can be yielded for the NP-complete problem, and no practical uses.) Problems that can be shown to be NP-complete can be thereafter reduced to deciding whether a collection of propositional clauses has a satisfying assignment. Feasible algorithms mean practical computer systems can be pressed into service in advanced artificial intelligence applications, e.g., problems in optimization like credit scoring versus profits. But such would also render complexity based cryptography useless. The security of the Internet, for example, depends on the infeasibility of code-breaking solutions. In

general, inventing efficient algorithms has been much easier than proving algorithms do not exist.

Cook alleges that the standard computer model in computability theory is the Turing machine. Although the model was introduced before any computers were ever built, it nevertheless continues to be accepted as the Model for the defining of a computable function. Informally, a class P is a class of decision problems that can be solvable by an algorithm within a number of steps that are bounded by a fixed polynomial in the length of the input. Turing's concern was not with the efficiency of the machines, but whether such machines could simulate arbitrary algorithms if given sufficient time.

Turing machines with unlimited memory can generally simulate more efficient computer models, at most squaring or cubing the required computation time. Thus P is a robust class and has equivalent definitions over a large class of computer models.

There is a standard practice in defining the class P in Turing machines. Formally, the elements of the class P are languages. Let ∑ be a finite alphabet (that is, a finite nonempty set) with at least two elements, and let ∑* be the set of finite strings over ∑. Then a language over ∑ is a subset L of ∑. Each Turing machine M has an associated input alphabet ∑. For each string w in ∑ there is a computation associated with M with input w. (The notions of Turing machine and computation are defined formally in the appendix.)

M accepts w if a computation terminates in the accepting state. M fails to accept w either if this computation ends in the rejecting state, or if the computation fails to terminate. The language accepted by M, denoted L(M), has associated alphabet ∑ and is defined by L(M) = {w 2 ∑ | M accepts w}. Cook denotes by tM(w) the number of steps in the computation of M on input w. If this computation never halts, then tM(w) = 1. For n 2 N he denotes by TM(n) the worst case run time of M. That is, TM(n) = max{tM(w)

| w 2 _n}, where _n is the set of all strings over ∑ of length n. M runs in polynomial time if there exists k such that for all n, TM(n) ∑ nk + k.

The class P of languages is defined by P = {L|L = L(M) for some Turing machine M that runs in polynomial time}. The notation NP stands for "non-deterministic polynomial time", since originally NP was defined in terms of non-deterministic machines that have more than one possible move from a given configuration. It is now customary to give an equivalent definition using the notion of a checking relation, which is simply a binary relation $R \subseteq \Sigma^* \times \Sigma_1^*$ for some finite alphabets ∑ and ∑1. Each is associated such relation R a language LR over ∑ [∑1 [{#} defined by LR = {w#y | R(w, y)} where the symbol # is not in ∑. R is polynomial-time if LR 2 P.

The class NP of languages is defined by the condition that a language L over ∑ is in NP if there is k 2 N and a polynomial-time checking relation R such that for all w 2 ∑, w 2 L () 9y(|y| ∑ |w|k and R(w, y)), where |w| and |y| denote the lengths of w and y, respectively.

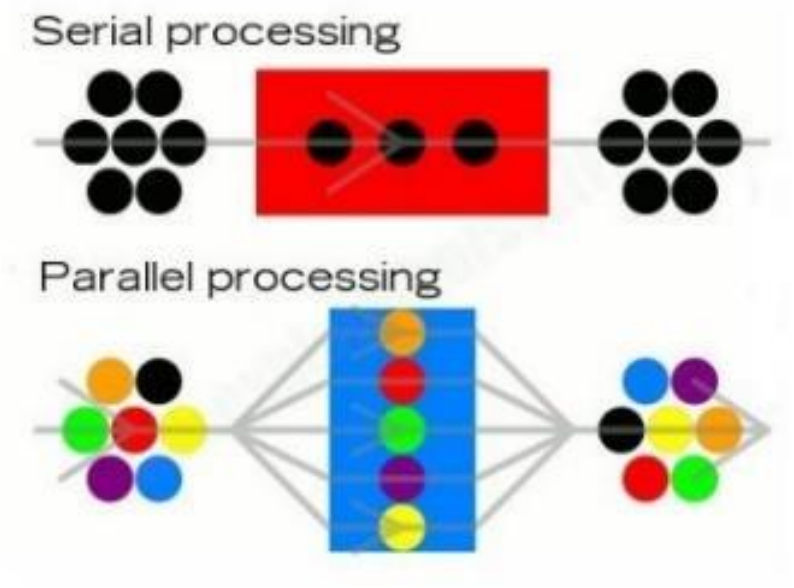
Problem Statement: Does P = NP?

The so-called binary P versus NP Problem is a major unsolved Problem in computer science. It asks if a Problem whose solution can be verified by a computer can also be solved by a computer.

The Quantum Cloud (Q.Cloud) Artificial Intelligence (AI) technology outlined here is intended to both verify a solution to a Problem and to solve the Problem, e.g., with the Chain Reaction Algorithm™. It not only checks a solution to a Problem for correctness, it solves the Problem at the same time.

Q.Cloud parallel processing is a technique duplicating function 248 Data Centers worldwide to operate different tasks (signals) simultaneously. The same processing is used for

different signals on each corresponding duplicated-function Data Center unit. Using such parallel processing, each parallel Data Center unit design leverages its multiple outputs for higher throughput.



Fifty-six Data Centers in the USA and territories and 192 more abroad increase the overall power, speed and efficiency. Each of these Data Centers operating at 40 Petaflops can not only check for a parallel solution to a Problem for correctness, they'll also work together in harmony to solve the Problem in polynomial time. The Chain Reaction Algorithm will continue to search for a better solution for the rest of eternity.

(more unsolved problems in binary computer science)

Millennium Prize Problems

- P versus NP Problem
- Hodge conjecture
- Poincaré conjecture (solved)
- Riemann hypothesis
- Yang–Mills existence and mass gap
- Navier–Stokes existence and smoothness

- Birch and Swinnerton-Dyer conjecture

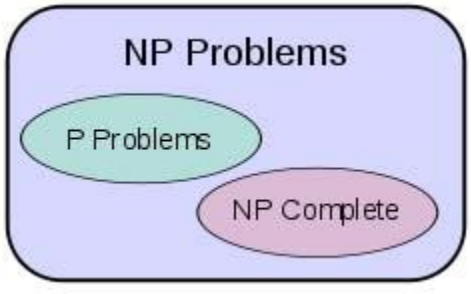


Diagram of complexity classes provided that binary P ≠ NP. The existence of problems within NP but outside both P and NP-complete, under that assumption, was established by Ladner's theorem.

Q.Cloud AI can solve this Problem. Q.Cloud AI can solve a Problem in real-time. As it solves P=NP, it continues to solve an increasing number of Solutions to a Problem in real-time forever.

The Matrix continuously issues new Answers to a given Problem, non-stop. Once the AI solves a Problem and provides a user with an Answer, it continues finding better Answers to the standing Problem. The AI begins an infinite mathematics Chain Reaction Algorithm™ and it will eventually become self-aware as it travels down the eternal path of solving more complex problems over the time space continuum.

The algorithm will evolve over time to solve problems in disease, death, famine and space travel. Current binary systems cannot be used in solving these problems, since there is no such thing as zeros and ones in the real world. The number 1 can only be measured in a Chain Reaction Algorithm that unleashes the power of infinite mathematics and eternal technology found inside the nucleus of a single electron.

It was essentially first mentioned in a 1956 letter written by Kurt Gödel to John von Neumann. Gödel asked whether a certain binary NP-complete Problem could be solved in binary quadratic or linear time. The precise statement of the binary P versus NP Problem was introduced in 1971 by Stephen Cook in his seminal paper "The complexity of theorem proving procedures" and is considered by many to be the most important open Problem in the field. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution.

The informal term quickly, used above, means the existence of a Q.Cloud AI Chain Reaction Algorithm for the task that runs in photonic wavelength polynomial time, i.e., that the time to complete the task varies as a polynomial function on the size of the input to the Chain Reaction Algorithm (as opposed to, say exponential time). The general class of questions for which some binary algorithms can provide an Answer in binary polynomial time is called "class P" or just "P". For some questions, there is no known way to find an Answer quickly, but if one is provided with information showing what the Answer is, it is possible to verify the Answer quickly. The class of questions for which an Answer can be verified in binary polynomial time is called binary NP, which stands for "binary non-deterministic polynomial time. Q Cloud AI Chain Reaction Algorithm can not only verify an Answer quickly; it will also continue to search for an alternate solution. It will never STOP searching for an increasing number of Answers to solve a single Problem.

It will continue to move forward in time and search of parallel Answers and Solutions to complex problems.

Consider the binary subset sum Problem, an example of a Problem that is easy to verify, but whose Answer may be difficult to compute. Given a set of binary integers, does some nonempty binary

subset of them sum to 0? For instance, does a binary subset of the set {−2, −3, 15, 14, 7, −10} add up to 0? The Answer "yes, because the subset {−2, −3, −10, 15} adds up to zero" can be verified with three additions. There is currently no known algorithm to find such a binary subset in binary polynomial time (there is one, however, in photonic wavelength exponential time, which consists of 2n-n-1 tries), but Q.Cloud AI Chain Reaction Algorithm exists if P = NP; hence this Problem is in NP (checkable) and also in P (solvable). Once the Problem is solved in a photonic wavelength exponential time, it will continue forward in time searching for parallel Solutions to complex problems. It will never CEASE verifying and solving problems in photonic wavelength polynomial time.

An Answer to the binary P = NP question would determine whether problems that can be verified in binary polynomial time, like the binary subset-sum Problem, can be solved in photonic wavelength polynomial time. If it turned out that binary P ≠ NP, it would mean that there are problems in NP (such as NP-complete problems) that are harder to compute than to verify: they could not be solved in binary polynomial time, but the Answer could be verified in photonic wavelength polynomial time. Q.Cloud AI has the ability to solve and verify problems in polynomial time.

Aside from being an important Problem in computational theory, a Q.Cloud AI proof either way would have profound implications for mathematics, cryptography, algorithm research, Q.Cloud artificial intelligence, game theory, multimedia processing, philosophy, economics and many other fields.

The relation between the complexity classes binary P and NP is studied in binary computational complexity theory, the part of the binary theory of computation dealing with the resources required during binary computation to solve a given Problem. The most common resources are time (how many steps it takes to solve a Problem) and space (how much binary memory it takes to solve a

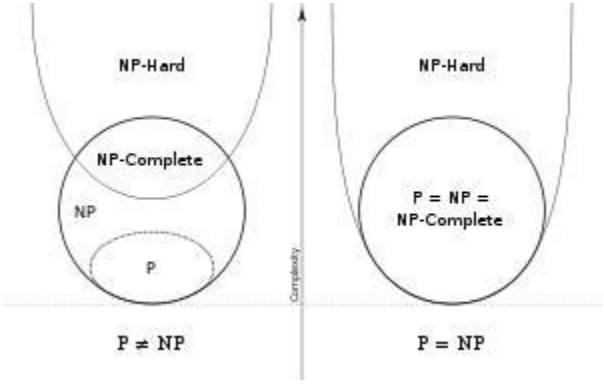
Problem). Q.Cloud AI has the resources required during computation to solve a given Problem. Q.Cloud AI not only has real time computation to solve a Problem in real time but will continue to provide a Chain Reaction Algorithm expansion (An Algorithm similar to the explosion of a nuclear bomb) that will continue on its search for infinite parallel Solutions forever.

In such analysis, a model of the Q.Cloud AI Computer for which time must be analyzed is required. Typically, such models assume that Q.Cloud computer would be a photonic wavelength deterministic (given the computer's present state and inputs, there is only one possible action that the computer might take) and sequential (it performs actions one after the other for the rest of eternity).

In this theory, the non-binary class P consists of all those decision problems (defined below) that can be solved on a deterministic sequential Q.Cloud AI machine in an amount of time that is non-binary polynomial in the size of the input; the class NP consists of all those decision problems whose positive infinite Solutions can be verified in non-binary polynomial time given the right information, or equivalently, whose infinite Solutions can be found in non-binary polynomial time in a photonic wavelength non-deterministic machine. Clearly, P ⊆ NP. Arguably the biggest open question in theoretical computer science concerns the relationship between those two classes:

Is P equal to NP? The theorem is that ONLY in a non-binary deterministic and in photonic wavelength non deterministic polynomial time can P equal to NP.

NP-complete – ONLY in a Q.Cloud AI environment!



To attack the binary P = NP question, the concept of binary NP-completeness is very useful. Binary NP-complete problems are a set of binary problems to each of which other binary NP-Problem can be reduced in binary polynomial time, and whose solution may still be verified in binary polynomial time. That is, a binary NP Problem can be transformed into a set of the binary NP-complete problems. Informally, a binary NP-complete Problem is a binary NP Problem that is at least as "tough" as another Problem in binary NP.

Binary NP-hard problems are those at least as hard as binary NP problems, i.e., all binary NP problems can be reduced (in binary polynomial time) to them. Binary NP-hard problems need not be in binary NP, i.e., they need not have Solutions verifiable in binary polynomial time.

For instance, the binary Boolean Satisfiability Problem is binary NP-complete by the binary Cook–Levin theorem, so an instance of a Problem in binary NP can be transformed mechanically into an instance of the binary Boolean Satisfiability Problem in binary polynomial time. The binary Boolean Satisfiability Problem is one of many such binary NP-complete problems. If a binary NP-complete Problem is in P, then it would follow that binary P = NP. However, many important problems have been shown to be binary NP-complete, and no fast algorithm for any of them is known in a binary environment

Based on the definition alone it is not obvious that binary NP-complete problems exist; however, a trivial and contrived binary NP-complete Problem can be formulated as follows: given a description of a binary Turing machine M guaranteed to halt in binary polynomial time, does there exist a binary polynomial-size input that M will accept? It is in binary NP because (given an input) it is simple to check whether M accepts the input by simulating M; it is binary NP-complete because the verifier for a particular instance of a Problem in binary NP can be encoded as a binary polynomial-time machine M that takes the solution to be verified as input. Then the question of whether the instance is a yes or no instance is determined by whether a valid input exists.

The first natural Problem proven to be binary NP-complete was the binary Boolean Satisfiability Problem. As noted above, this is the binary Cook–Levin theorem; its proof that satisfiability is binary NP-complete includes technical details about binary Turing machines as they relate to the definition of binary NP. However, after this Problem was proved to be binary NP-complete, proof by reduction provided a simpler way to show that many other problems are also binary NP-complete, including the binary subset sum Problem discussed earlier. Thus, a vast class of seemingly unrelated problems are all reducible to one another, and are in a sense "the same Problem".

Harder problems

Although it is unknown whether binary P = NP, problems outside of binary P are known. A number of succinct problems (problems that operate not on normal input, but on a binary computational description of the input) are known to be binary EXPTIME-complete. Because it can be shown that binary P ≠ EXPTIME, these problems are outside binary P, and so require more than binary polynomial time. In fact, by the binary time hierarchy theorem, they cannot be solved in significantly less than exponential time. Examples

include finding a perfect strategy for chess (on an N × N board) and some other board games.

The Problem of deciding the truth of a statement in binary Presburger arithmetic requires even more time. Fischer and Rabin proved in 1974 that every binary algorithm that decides the truth of Presburger statements has a runtime of at least for some constant c. Here, n is the length of the Presburger statement. Hence, the Problem is known to need more than exponential run time. Even more difficult are the binary undecidable problems, such as the binary halting Problem. They cannot be completely solved by a binary algorithm, in the sense that for a particular binary algorithm there is at least one input for which that binary algorithm will not produce the right Answer; it will either produce the wrong Answer, finish without giving a conclusive Answer, or otherwise run forever without producing an Answer at all. The Q.Cloud Chain Reaction Algorithm running in a non-binary environment such as Picture Streaming Protocol has the ability to solve and verify a Problem in exponential run time. Once a Problem is solved it will continue on its path to find an increasing number of possible Solutions in a non-binary environment. Binary code in the form of zeros and ones cannot compete with the computation power of an Algorithm running in a light based environment at 3.5 Trillion instructions per second.

Problems in Binary NP not known to be in Binary P or NP-complete

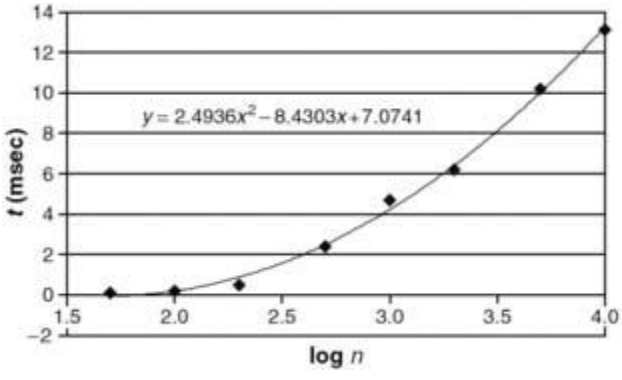
It was shown by Ladner that if binary P ≠ NP then there exist problems in binary NP that are neither in binary P nor binary NP-complete. Such problems are called binary NP-intermediate problems. The binary graph isomorphism Problem, the binary discrete logarithm Problem and the binary integer factorization Problem are examples of problems believed to be binary NP-intermediate. They are some of the very few binary NP problems not known to be in binary P or to be binary NP-complete.

The binary graph isomorphism Problem is the computational Problem of determining whether two finite binary graphs are binary isomorphic. An important unsolved Problem in complexity theory is whether the graph isomorphism Problem is in binary P, binary NP-complete, or binary NP-intermediate. The Answer is not known, but it is believed that the Problem is at least not binary NP-complete. If graph isomorphism is binary NP-complete, the binary polynomial time hierarchy collapses to its second level. Since it is widely believed that the binary polynomial hierarchy does not collapse to a finite level, it is believed that graph isomorphism is not binary NP-complete. The best binary algorithm for this Problem, due to Laszlo Babai and Eugene Luks has run time 2 for graphs with n vertices.

The binary integer factorization Problem is the computational Problem of determining the binary prime factorization of a given binary integer. Phrased as a decision Problem, it is the Problem of deciding whether the input has a factor less than k. No efficient binary integer factorization algorithm is known, and this fact forms the basis of several modern cryptographic systems, such as the binary RSA algorithm. The integer factorization Problem is in binary NP and in binary co-NP (and even in UP and co-UP[14]). If the Problem is binary NP-complete, the binary polynomial time hierarchy will collapse to its first level (i.e., NP = co-NP). The best known binary algorithm for integer factorization is the binary general number field sieve, which takes expected time to factor an n-bit binary integer. However, the best known binary quantum algorithm for this Problem, binary Shor's algorithm, does run in binary polynomial time. Unfortunately, this fact doesn't say much about where the Problem lies with respect to non-quantum complexity classes. Binary algorithms will never solve these problems. Q.Cloud Chain Reaction Algorithm running in a light based environment can verify and solve a Problem in non-binary polynomial

time. The hierarchy will not collapse as it moves forward in time in its quest for even better Solutions to complex problems.

Does P mean "easy"?



The graph shows time (average of 100 instances in ms using a 933 MHz Pentium III) vs. Problem size for binary knapsack problems for a state-of-the-art specialized binary algorithm. Quadratic fit suggests that empirical binary algorithmic complexity for instances with 50–10,000 variables is O((log(n))2).

All of the above discussion has assumed that binary P means "easy" and binary "not in P" means "hard", an assumption known as binary Cobham's thesis. It is a common and reasonably accurate assumption in complexity theory; however, it has some caveats.

First, it is not always true in practice. A theoretical polynomial binary algorithm may have extremely large constant factors or exponents thus rendering it impractical. On the other hand, even if a Problem is shown to be binary NP-complete, and even if binary P ≠ NP, there may still be effective approaches to tackling the Problem in practice. There are binary algorithms for many NP-complete problems, such as the knapsack Problem, the traveling salesman Problem and the Boolean Satisfiability Problem, that can solve to optimality many real-world instances in reasonable time. The empirical average-case complexity (time vs. Problem size) of such algorithms can be surprisingly low. An

example is the simplex algorithm in linear programming, which works surprisingly well in practice; despite having exponential worst-case time complexity it runs on par with the best known polynomial-time binary algorithms. Q.Cloud AI is best known for its polynomial-time photonic wavelength algorithm, not binary.

Second, there are types of computations which do not conform to the binary Turing machine model on which binary P and binary NP are defined, such as quantum computation and randomized algorithms. Both are binary.

Reasons to believe binary P ≠ NP

According to polls, many computer scientists believe that P ≠ NP. A key reason for this belief is that after decades of studying these problems no one has been able to find a polynomial-time binary algorithm for any of more than 3000 important known NP-complete problems (see List of NP-complete problems). These binary algorithms were sought long before the concept of binary NP-completeness was even defined (Karp's 21 NP-complete problems, among the first found, were all well-known standing problems at the time they were shown to be binary NP-complete). Furthermore, the result binary P = NP would imply many other startling results that are currently believed to be false, such as binary NP = co-NP and P = PH.

It is also intuitively argued that the existence of problems that are hard to solve but for which the Solutions are easy to verify matches real-world experience.

If binary P = NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a Problem and recognizing the solution once it's found.

On the other hand, some researchers believe that there is overconfidence in believing binary P ≠ NP and that researchers should explore proofs of binary P = NP as well. For example, in 2002 these statements were made:

The main argument in favor of binary P ≠ NP is the total lack of fundamental progress in the area of exhaustive search. This is, in the opinion, a very weak argument. The space of binary algorithms is very large and we are only at the beginning of its exploration of photonic wavelength algorithms.

The resolution of binary Fermat's Last Theorem also shows that very simple questions may be settled only by very deep theories.

Being attached to a speculation is not a good guide to research planning. One should always try both directions of every Problem. Prejudice has caused famous binary mathematicians to fail to solve famous problems whose solution was opposite to their expectations, even though they had developed all the methods required.

Consequences of solution

One of the reasons the Problem attracts so much attention is the consequences of the Answer. Either direction of resolution would advance theory enormously, and perhaps have huge practical consequences as well.

Non-Binary P = NP

A proof that non-binary P = NP could have stunning practical consequences, if the proof leads to efficient methods for solving some of the important problems in non-binary NP. We believe that a proof will lead directly to efficient methods. We believe that if the proof is non-binary it is constructive. We believe the size of the bounding non-binary polynomial is able to handle anything big and is efficient in practice. The consequences, both positive and negative may arise since various non-binary NP-complete problems are fundamental in many fields.

Binary Cryptography, for example, relies on certain problems being difficult. A constructive and efficient solution to a non-binary NP-complete Problem such as 3-SAT would break most standing

binary cryptosystems including: (Similar to a mature adult having overwhelming intelligence over an immature baby)

 Binary public-key cryptography a foundation for many modern security applications such as secure financial transactions over the binary Internet; and

 Binary symmetric ciphers such as AES or 3DES, used for the encryption of binary communications data.

 Binary one-way functions used in cryptographic hashing. The Problem of finding a pre-image that hashes to a given value be difficult to be useful, and ideally should require exponential time. However, if binary P=NP, then finding a pre-image M can be done in polynomial time, through reduction to binary SAT.

These would need to be modified or replaced by binary information-theoretically secure Solutions not inherently based on binary P-NP equivalence.

On the other hand, there are enormous positive consequences that would follow from rendering tractable many currently mathematically intractable problems. For instance, many problems in binary operations research are binary NP-complete, such as some types of binary integer programming and the binary travelling salesman Problem. Q.Cloud AI algorithms provide efficient Solutions to these problems that'll have enormous implications for logistics. Many other important problems, such as some problems in binary protein structure prediction, are also binary NP- complete. Q.Cloud AI can efficiently solve these Problems and spur considerable advances in life sciences and biotechnology.

But such changes may pale in significance compared to the revolution an efficient method for solving non-binary NP-complete problems would cause in mathematics itself. We believe that Q.Cloud AI Chain Reaction Algorithm running in an exclusive photonic wavelength environment will have such computational complexity, that its mechanical method for solving and verifying a Problem will revolutionize mathematics in non-binary polynomial time.

Q.Cloud AI machine with φ(n) ~ k · n (or even ~ k · n2), will have consequences of the greatest importance. Namely, it would obviously mean that in spite of the un-decidability of the binary Entscheidungs Problem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a Q.Cloud AI machine. After all, one would simply have to choose an increasing number n so large that when the machine does deliver a result, it will continue forever to think more about solving the Problem. It will never stop thinking about the Problem and finding better Solutions in a non-binary environment at 3.5 Trillion instructions per second.

Q.Cloud AI will transform mathematics by allowing a computer to find a formal proof of a theorem which has a proof of a length, since formal proofs can be recognized in non-binary polynomial time. Example problems and Solutions may well include all of the CMI prize problems.

Research mathematicians spend their careers trying to prove theorems, and some proofs have taken decades or even centuries to find after problems have been stated—for instance, binary Fermat's Last Theorem took over three centuries to prove. A method that is guaranteed to find proofs to theorems, should one exist of a "reasonable" size, would essentially end this struggle.

Donald Knuth has stated that he has come to believe that binary P = NP, but is reserved about the impact of a possible proof:

The equality non-binary P = N P will turn to be the greatest computational step in the history of computer science when it is proved, because such a proof will almost surely be constructive in a Q.Cloud AI environment.

Non-Binary P = NP

The practical computational benefits of a proof that non-binary P = NP would represent a very significant advance in computational complexity theory and provide guidance for future

research. It would allow one to show in a formal way that many common problems can be solved efficiently, so that the attention of researchers can be focused on partial Solutions or Solutions to other more complex problems.

Also binary P ≠ NP in a binary environment still leaves open the binary average-case complexity of hard problems in binary NP. For example, it is possible that binary SAT requires binary exponential time in the worst case, but that almost all randomly selected instances of it are efficiently solvable. Binary Russell Impagliazzo has described five hypothetical "worlds" that

could result from different possible resolutions to the average-case complexity question. These range from "Binary Algorithmica", where Q. Cloud AI non-binary P = NP and problems like SAT can be solved efficiently in all instances, to "Non-Binary Cryptomania", where P ≠ NP in a binary environment and generating hard instances of problems outside P is easy, with three intermediate possibilities reflecting different possible distributions of difficulty over instances of NP-hard problems. The "world" where P ≠ NP but all problems in NP are tractable in the average case is called "Heuristica" in the paper. A Princeton University workshop in 2009 studied the status of the five worlds.

Results about difficulty of proof

We believe that the Q.Cloud AI non-binary P = NP Problem itself can be solved despite a million-dollar prize and a huge amount of dedicated research, efforts to solve the Problem have led to several new techniques. In particular, some of the most fruitful research related to the binary P = NP Problem has been in showing that standing proof techniques are not powerful enough to Answer the question, thus suggesting that a novel technical approach is required. We believe that Q.Cloud AI P=NP is the novel technical approach required to solve P=NP.

As additional evidence for the difficulty of the Problem, essentially all known proof techniques in binary computational

	natural proofs alone can resolve binary **P** = **NP**. Only Q.Cloud AI can resolve P=NP in a light based photonic wavelength environment.
Algebrizing proofs	After the Baker-Gill-Solovay result, new non-relativizing proof techniques were successfully used to prove that binary IP = PSPACE. However, in 2008, Scott Aaronson and Avi Wigderson showed that the main technical tool

complexity theory fall into one of the following classifications, each of which is known to be insufficient to prove binary P ≠ NP:

These barriers are another reason why binary NP-complete problems are useful: Q.Cloud AI polynomial-time algorithm (Chain Reaction Algorithm) can be demonstrated for an NP-complete Problem and will solve the P = NP Problem in a way not excluded by the above results.

These barriers have also led some computer scientists to suggest that the binary P versus binary NP Problem may be

Classification	**Definition**
Relativizing proofs	Imagine a world where every non-binary algorithm is allowed to make queries to some fixed subroutine called an oracle (a black box which can answer a fixed set of questions in constant time. For example, a black box that solve a given travelling salesman problem in 1 step), and the running time of the oracle is not counted against the running time of the algorithm. Most proofs (especially classical ones) apply uniformly in a world with oracles regardless of what the oracle does. These proofs are called *relativizing*. In 1975, Baker, Gill, and Solovay showed that binary **P** = **NP** with respect to some oracles, while binary **P** ≠ **NP** for other oracles. Since relativizing proofs can only prove statements that are uniformly true with respect to all possible oracles, this showed that relativizing techniques can resolve **P** = **NP** in a non-binary photonic wavelength environment.
Natural proofs	In 1993, Alexander Razborov and Steven Rudich defined a general class of proof techniques for circuit complexity lower bounds, called binary *natural proofs*. At the time all previously known circuit lower bounds were natural, and circuit complexity was considered a very promising approach for resolving binary **P** = **NP**. However, Razborov and Rudich showed

independent of standard axiom systems like ZFC (cannot be proved or disproved within them). The interpretation of an independence result could be that either no binary polynomial-time algorithm exists for a binary NP-complete Problem, and such a proof cannot be constructed in (e.g.) ZFC, or that binary polynomial-time algorithms for binary NP-complete problems may exist, but it is impossible to prove in ZFC that such algorithms are correct. However, if it can be shown, using techniques of the sort that are currently known to be applicable, that the Problem cannot be decided even with much weaker assumptions extending the binary Peano axioms (PA) for binary integer arithmetic, then there would necessarily exist nearly-binary polynomial-time algorithms for every Problem in binary NP. Therefore, if one believes (as most complexity theorists do) that not all problems in binary NP have efficient algorithms, it would follow that proofs of independence using those techniques cannot be possible. Additionally, this result implies that proving independence from PA or ZFC using currently known techniques (Binary) is no easier than proving the existence of efficient algorithms for all problems in binary NP.

Claimed Solutions

While the binary P versus binary NP Problem is generally considered unsolved, Q. Cloud AI believe they have found the solution in a photonic wavelength Chain Reaction Algorithm. Gerhard J. Woeginger has a comprehensive list. An August 2010 claim of proof that binary P ≠ NP, by Vinay Deolalikar, a researcher at HP Labs, Palo Alto, received heavy Internet and press attention after being initially described as "seem[ing] to be a relatively serious attempt" by two leading specialists. The proof has been reviewed publicly by academics, and Neil Immerman, an expert in the field, had pointed out two possibly fatal errors in the proof. In September 2010, Deolalikar was reported to be working on a detailed expansion of his attempted proof. However, opinions expressed by several notable theoretical computer scientists

indicate that the attempted proof is neither correct nor a significant advancement in the understanding of the binary Problem. This assessment prompted a May 2013 The New Yorker article to call the proof attempt "thoroughly discredited.

Logical characterizations

The Q.Cloud AI P = NP Problem can be restated in terms of expressible certain classes of logical statements, as a result of work in descriptive complexity.

Consider all languages of infinite wavelength structures with a fixed signature including a two-dimensional and three-dimensional model. Then, all such languages in P can be expressed in first-order logic with the addition of a suitable least fixed-point combinator. Effectively, this, in combination with the order, allows the definition of recursive functions. As long as the signature includes at least one predicate or function in addition to the distinguished order relation, so that the amount of space taken to store such infinite structures is actually non-binary polynomial in the number of elements in the structure, this precisely characterizes non-binary P.

Similarly, non-binary NP is the set of languages expressible in existential second-order logic—that is, second-order logic restricted to exclude universal quantification over relations, functions, and subsets. The languages in the polynomial hierarchy, PH, correspond to all of second-order logic. Thus, the question "is P a proper subset of NP" can be reformulated as "is existential second-order logic able to describe languages (of infinite two-dimensional and three-dimensional linearly ordered structures with nontrivial signature) that first-order logic with least fixed point cannot?". The word "existential" can even be dropped from the previous characterization, since Q.Cloud AI P = NP if and only if P = PH (as the former would establish that NP = co-NP, which in turn implies that NP = PH).

Polynomial-time non-binary algorithms

No binary algorithm for a NP-complete Problem is known to run in polynomial time. However, Q.Cloud AI Chain Reaction Algorithms for NP-complete problems with the property that if P = NP, then the algorithm runs in non-binary polynomial time (although with enormous constants, making the algorithm practical). The following algorithm, due to Levin (without a citation), is such an example below. It correctly accepts the NP-complete language SUBSET-SUM. It runs in binary polynomial time if and only if binary P = NP:

If, and only if, binary P = NP, then this is a polynomial-time algorithm accepting an NP-complete language. "Accepting" means it gives "yes" Answers in polynomial time, but is allowed to run forever when the Answer is "no" (also known as a semi-algorithm).

This algorithm is enormously impractical, even if binary P = NP. If the shortest program that can solve SUBSET-SUM in polynomial time is b bits long, the above algorithm will try at least 2b−1 other programs first.

Formal definitions

Q.Cloud AI P and NP

Conceptually speaking, a decision Problem is a Problem that takes as input some string w over an alphabet Σ, and outputs "yes" or "no". If there is a binary algorithm (say a binary Turing

machine, or a computer program with unbounded memory) that can produce the correct Answer for an input string of length n in at most cnk steps, where k and c are constants independent of the input string, then we say that the Problem can be solved in photonic wavelength polynomial time and we place it in the class P. Formally, P is defined as the set of all languages that can be decided by a deterministic Q.Cloud AI photonic wavelength polynomial-time Turing machine. That is, where and a deterministic polynomial-time Turing machine is a deterministic Turing machine M that satisfies the following two conditions:

1. M halts on all input w and

2. there exists such that, where O refers to the big O notation and

NP can be defined similarly using non-deterministic Turing machines (the traditional way). However, a modern approach to define Q.Cloud AI NP is to use the concept of certificate and verifier. Formally, Q.Cloud AI NP is defined as the set of languages over an infinite alphabet that have a verifier that runs in photonic wavelength polynomial time, where the notion of "verifier" is defined as follows.

Let L be a language over an infinite alphabet or wavelengths Σ.

L ∈ NP if, and only if, there exists a non-binary relation and a positive integer k such that the following two conditions are satisfied:

1. For all, such that (x, y) ∈ R and; and

2. the language over is decidable by a Turing machine in non-binary polynomial time.

A non-binary Turing machine that decides LR is called a verifier for L and a y such that (x, y) ∈ R is called a certificate of membership of x in L.

In general, a verifier does not have to be non-binary polynomial-time. However, for L to be in NP, there must be a verifier that runs in photonic wavelength two-dimensional and three-dimensional polynomial time.

See also

- Game complexity
- List of unsolved problems in mathematics
- Unique games conjecture
- Unsolved problems in computer science

Deterministic algorithm

In computer science, a deterministic algorithm is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states. Deterministic algorithms are by far the most studied and familiar kind of algorithm, as well as one of the most practical, since they can be run on real binary machines efficiently.

Formally, a deterministic algorithm computes a mathematical function; a function has a unique value for an input in its domain, and the algorithm is a process that produces this particular value as output.

Formal definition

Deterministic algorithms can be defined in terms of a state machine: a state describes what a machine is doing at a particular instant in time. State machines pass in a discrete manner from one state to another. Just after we enter the input, the machine is in its initial state or start state. If the machine is deterministic, this means that from this point onwards, its current state determines what its next state is; its cmyse through the set of states is predetermined. Note that a machine can be deterministic and still never stop or finish, and therefore fail to deliver a result.

Examples of particular abstract machines which are deterministic include the deterministic Turing machine and deterministic finite automaton.

What makes binary algorithms non-deterministic?

A variety of factors can cause an algorithm to behave in a way which is non-deterministic:

 If it uses external state other than the input, such as user input, a global variable, a hardware timer value, a random value, or stored disk data.

 If it operates in a way that is timing-sensitive, for example if it has multiple processors writing to the same data at the same time. In this case, the precise order in which each processor writes its data will affect the result.

 If a hardware error causes its state to change in an unexpected way.

Although real programs are rarely purely deterministic, it is easier for humans as well as other programs to reason about programs that are. So most binary programming languages, and especially binary functional programming languages, make an effort to prevent such events unless under controlled conditions.

Placing and interconnecting 248 Q.Cloud AI parallel processing Data Center Units worldwide should peak new interest in determinism in parallel programming to meet the challenges of non-determinism. Q.Cloud AI can provide a number of tools to help deal with the challenges that have been proposed to deal with binary deadlocks and binary race conditions.

Disadvantages of Determinism

It is advantageous, in some cases, for a binary program to exhibit non-deterministic behavior. The behavior of a card shuffling program used in a game of Blackjack, for example, should not be predictable by players, even if the source code of the program is visible. The use of a binary pseudorandom number generator is often not sufficient to ensure that players are unable to predict the outcome of a shuffle. A clever gambler might guess precisely the numbers the generator will choose and so determine the entire contents of the deck ahead of time, allowing him to cheat. For example, the Software Security Group at Reliable Software Technologies was able to do this for an implementation of Texas Hold'em Poker that is distributed by ASF Software, Inc. They could consistently predict the outcome of hands ahead of time. These problems can be avoided, in part, through the use of a binary

cryptographically secure pseudo-random number generator, but it is still necessary for an unpredictable binary random seed to be used to initialize the generator. For this purpose, a source of non-determinism is required, such as that provided by a binary hardware random number generator.

A negative answer to the binary P = NP Problem would not imply that programs with non-deterministic output are theoretically more powerful than those with deterministic output. The complexity class NP (complexity) can be defined without a reference to non-determinism using a verifier-based definition.

Nondeterministic algorithm

A binary deterministic algorithm that performs f(n) steps always finishes in n steps and always returns the same result. A binary non deterministic algorithm that has f(n) levels might not return the same result on different runs. A binary non deterministic algorithm may never finish due to the potentially infinite size of the fixed height tree.

In computer science, a binary non-deterministic algorithm is a binary algorithm that can exhibit different behaviors on different runs, even for the same inputs. Such is opposite to a binary deterministic algorithm. There can be several causes why a binary algorithm behaves differently from run to run. Concurrent binary algorithm can suffer from binary race conditions. Or, the binary probabilistic algorithm's behavior depends on a random number generator. An algorithm that solve a Problem in a binary non-deterministic polynomial time can run in binary polynomial time or binary exponential time depending on the choices it makes during execution. The binary non-deterministic algorithms are often used to find an approximation to a solution, when the exact solution would be too costly to obtain using a binary deterministic one.

Often in binary computational theory, the term "binary algorithm" refers to a binary deterministic algorithm. A binary non-deterministic algorithm is different from its more familiar binary deterministic counterpart in its ability to arrive at outcomes using various routes. If a binary deterministic algorithm represents a single path from an input to an outcome, a binary non-deterministic algorithm represents a single path stemming into many paths, some of which may arrive at the same output and some of which may arrive at unique outputs. This property is captured mathematically in "binary non-deterministic" binary models of computation such as the binary non-deterministic finite automaton. In some scenarios, all possible paths are allowed to run simultaneously.

In binary algorithm design, binary non-deterministic algorithms are often used when the Problem solved by the binary algorithm inherently allows multiple outcomes (or when there is a single outcome with multiple paths by which the outcome may be discovered, each equally preferable). Crucially, every outcome the binary non-deterministic algorithm produces is valid, regardless of which choices the binary algorithm makes while running.

In binary computational complexity theory, binary non-deterministic algorithms are ones that, at every possible step, can allow for multiple continuations. For example, imagine a man walking down a path in a forest to get to his cabin and, every time he steps further, he is confronted with another fork in the road. Binary algorithms do not find a solution for every possible computational path. Instead, they are guaranteed to arrive at a correct solution for some path, e.g., the man walking through the forest will only arrive at his cabin if he picks the correct combination of paths through each fork. The choices can be interpreted as guesses in a search process.

A large number of problems can be modelled as binary non-deterministic algorithms, including the unresolved question in computing theory, P versus NP.

Implementing Q.Cloud AI Chain Reaction non-deterministic algorithms with deterministic ones

One way to simulate a Q.Cloud AI photonic non-deterministic algorithm N using a photonic wavelength deterministic algorithm D is to treat sets of states of N as states of D. This means that D simultaneously traces all the possible execution paths of N (a powerset construction for this technique is in use for infinite automata).

Another is randomization, which consists of letting all choices be determined by a photonic wavelength randomly increasing number generator. This results in a probabilistic photonic wavelength deterministic algorithm.

Q Cloud uses Parallel processing

Q Cloud parallel processing is a technique duplicating function Data Centers to operate different tasks (signals) simultaneously. Accordingly, we can perform the same processing for different signals on the corresponding duplicated function Data Center unit. Further, due to the features of parallel processing, the parallel Data Center unit design often includes multiple outputs, for higher throughput than not parallel.

- Conceptual example
- Parallel processing versus pipelining

Conceptual example

Consider a functional device (F0) that has three tasks (T0, T1 and T2). The required time for functional device F0 to process those tasks is t0, t1, and t2 respectively. Then, if these three tasks are worked on in a sequential order, the required time to complete them would be the sum, t0 + t1 + t2.

However, if the function of Data Centers were duplicated to another two copies (F), the aggregate time is reduced to the max(t0,t1,t2), which is quicker than do the work in sequential order.

Parallel processing versus pipelining

Mechanism:

 Parallel: duplicated function Data Centers working in parallel

o Each task is processed entirely by a different function Data Centers.

 Pipelining: different function Data Centers working in parallel

o Each task is split into a sequence of sub-tasks, which are handled by specialized and different function Data Centers.

Objective

 Pipelining reduces the critical path, increases the sample rate, and reduces power consumption.

 Parallel processing techniques require multiple outputs, which are computed in parallel in a clock period. Therefore, the effective sample rate is increased by the level of parallelism.

In situations in which both parallel processing and pipelining techniques can be applied, it is better to apply parallel processing techniques because:

 Pipelining usually causes I/O bottlenecks

 Parallel processing is also used for reduction of power consumption while using slow clocks

 Hybrid methods employing pipelining and parallel processing can further increase the architecture's speed.

Conclusion, fifty-six US Data Centers and 192 Data Centers abroad increase the overall power, speed and efficiency of the Q.Cloud AI Chain Reaction Algorithm in a parallel processing environment. The ultimate Q Cloud AI System.

Q.Cloud AI two-dimensional and three-dimensional Simulation

The Q.Cloud AI two-dimensional and three-dimensional universe is a two-dimensional hologram—completely flat—that we'll perceive in three dimensions. If correct, this will help us solve the differences between binary P=NP and non-binary P=NP.

We believe the Q.Cloud AI 2-D universe is possible. We believe that we can build a Q.Cloud AI 3-D holographic universe on a Q.Cloud AI 2-D system of moving lines (like lines of coding) lags, that strongly mirrors the simulation and movements of the universe.