Note: A better translation of the my article published at Transactions on Mathematics (TM) Vol. 3, No. 1, January 2017, pp. 34-37.

Languages Varying in Time and the P x NP Problem

Valdir Monteiro dos Santos Godoi
valdir.msgodoi@gmail.com

An original proof of P ≠ NP.

As we know, we cannot be correct 100% of the time on our predictions on the outcome of a game of even or odd, heads or tails, launch of dice, roulette, cards, lottery, stock market, etc., in short, all events where the probabilistic character dominates, has a strong influence.

It is not possible be correct always in our forecasts in the deterministic sense, assuming that we live in a world where there is free will. While there are rigid laws of physics, they are not able to predict the future in all its extension, and with all the precision.

So, it is doomed to failure any attempt to construct an algorithm or deterministic computer program, designed to hit unequivocally, without any error, and at all times, these probabilistic or random results (provided that it is not of addict dice, cards marked, teams and judges bought, etc.).

The same can no longer be said of non-deterministic algorithms. A non-deterministic algorithm can be considered as a process that, when faced with a choice between two (or more) alternatives, can create copies of itself for each alternative and further continue processing to each of them, independently of each other, in parallel. If exist a set of possibilities which lead to a positive response then this set is always chosen and the algorithm ends successfully ([1], [2], [3]).

In a simple example of launching dice, where the possible outcomes are elements of the set DICE = {1, 2, 3, 4, 5, 6}, our deterministic machine (of Turing), or modern computer, can launch your guess between the DICE elements, but will only have 1/6 probability of success.

As our idealized non-deterministic Turing machine (NDTM) can choose each of the six possible DICE alternatives (producing, say, six copies of itself), evidently one of the alternatives should match with the correct result of the launch of the dice, and the processing ends in the state of success or acceptance.

We see then that a non-deterministic machine or algorithms are capable of something the corresponding deterministic version cannot: hit evermore, always win.

For each launch the language related to the correct result varies depending on this result, i.e., if our dice showed face 1 up then the L = {1} set is the language that produces the corresponding result to the success/acceptance at that time, while other possible values, 2, 3, 4, 5 and 6, do not belong to L during that particular play with that particular player.

In the next moment we have L = {3}, in the next launch and again L = {6}, then L = {1}, L = {5}, etc., showing that we have an example of not constant language, and variable in time dependent on each new situation.

For a non-deterministic algorithm, or its corresponding NDTM, built to recognize and accept these languages, to be able to decide between to accept or reject a given input, between informing a YES or NO, SUCCESS or FAILURE, it must come external data to machine, in order to be able to decide, by comparison, on the acceptance or not of your guess, prediction, estimation, etc.

If we define a language that is of the form

$L = \{w = (x\ y\ z);\ x$ is the key to the problem, y is the guess of the machine, which was calculated and recorded before we know the value of z, z is the correct result of the problem, obtained after the calculation of $y\}$

the machine accepts the input whenever $y = z$, and reject otherwise, i.e., the elements of L in this problem are the strings w such that $y = z$.

This is a procedure in polynomial time: given x it is generated y (not deterministically for NDTM or deterministically for DTM), it is expected the full realization of coded event in x, we obtain z (by external environment, which will inform the result that actually occurred) and if $y = z$ it is accepts the input w. An example for x: PETR4-2018-03-08-CLOSE, which asks what value the Petrobras PETR4 stock will have on closing 08/March/2018 of BOVESPA, the Stock Exchange of Brazil (suppose possible values in the range of R$ 0.00 to R$ 10,000.00 only. Value R$ 0.00 means that there was no PETR4 negotiation that day, e.g., due to a holiday).

A NDTM, properly designed, will tend to accept one of your guesses y, because it will generate all possible values for z, one for each input string, and in polynomial time, then this is a class of problems that belongs to NP.

The deterministic version will not have the "ability", property, to produce copies of itself, like processing in parallel and simultaneous universes, and can only provide a single guess y for the key x, that is based on some kind of mathematical procedure and/or appropriate statistical solution to the issue. Given the uncertain nature of these problems (dice, roulette, cards, stock exchange, riddles, etc.) we cannot be said that has built an algorithm, or deterministic machine, to hit/solve the problem, with absolute certainty, always no errors, so these types of problems do not belong to P, but belong to NP, then $P \neq NP$.

We see that this is a demonstration that uses the most fundamental property of non-determinism, a characteristic that deterministic machines are unable to perform, which is the production of "copies" of itself and simultaneous processing with other "versions" of the machine (or program) even an exponential number of copies of itself (in relation to the size of the input).

If we prefer not to take this property of creation and parallelism, and instead admit that NDTM have an absolute luck, who are blessed with incredible luck, of manner always make the best choice [4], on the first try, as it seems defend Papadimitriou et al in [5], our demonstration would not change in essence: the nondeterministic algorithm would hit ever, this time by extreme luck, even without creating new versions of the machine, one for each alternative that is required for the algorithm. The deterministic algorithm, in turn, would have

no absolute luck, premonitory gift or "supernatural" power of instantaneous multiplication, and would only be able to hit, in general, in probabilistic terms.

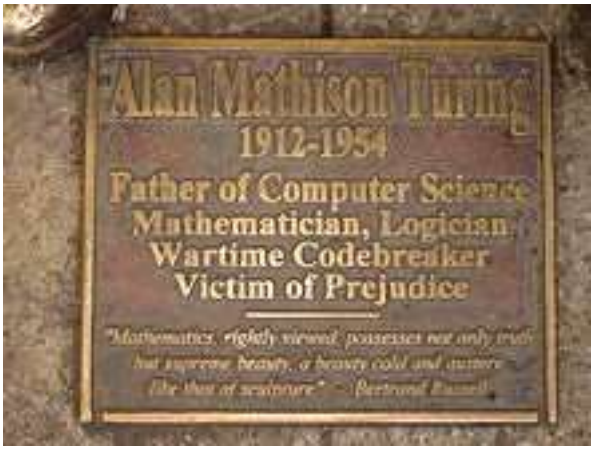
Realize that it is not a possible exponential order runtime for generate a guess the main cause that make these problems (DICE, BOVESPA, etc.) do not belong to P, but the impossibility of always resolve correctly an entry read by deterministic machine. It's like a passenger ship that runs aground or sinks 95% of the time and only complete a trip with 5% probability. It's a ship that obviously does not fit to travel, and no one should use it. Ditto a calculator machine that randomly change the functions of its buttons and does not warn us. In other words, prove that $P \neq NP$ does not imply that $SAT \notin P$ or generically $NPCOMPLETE \notin P$. You can have $P \neq NP$ and also $NPCOMPLETE \in P$.

It is also worth mentioning that in our example PETR4 we asked about a value that will be known only in two years or so. Of course it was an exaggerated example, because we can to input data much closer to happen, and easier to check (such as the launch of dice, our initial example).

Anyway, so we can process the correct result, the computer (or Turing machine, TM) must be informed by some input mechanism about this correct value. While the event did not finish, for example, the trading day of the shares or the play of the dice, the computer (or TM) will wait for the input of the correct value, but has already made his prediction. Of course, one NDTM, which multiplied in n copies or versions due to n alternative possibilities, it should receive the entry containing the correct value in all these n their versions. This is different from what is done, but after all our proof is unique, original. Where is it defined or prescribed that the TM and modern computers can only start processing its data after reading all the input characters until the end, sequentially, and only after the complete and uninterrupted reading of this entry it solve its "question"? This restriction is not formally described anywhere, for example, in [6]. Therefore, we assume a wait until the probabilistic event has occurred completely and the information of the respective result has been provided to the TM.

Certainly that is true $P \neq NP$.

To Alan Turing, In Memorian.



References

[1] R.M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher (eds.). New York: Plenum Press (1972), 85–103.

[2] J.E. Hopcraft, J.D. Ullman and R. Motwani, *Introdução à Teoria dos Autômatos, Linguagens e Computação*, Rio de Janeiro: Elsevier e Campus (2003), 452.

[3] N. Ziviani, *Projeto de Algoritmos com Implementações em Java e C++*, São Paulo: Thomson Learning (2007), 381-383, 388, 551.

[4] Devlin, Keith, *Os Problemas do Milênio – sete grandes enigmas matemáticos do nosso tempo*, cap. 3. Rio de Janeiro: editora Record (2004), 168-169.

[5] S. Dasgupta, C. Papadimitriou and U. Vazirani, *Algoritmos*, São Paulo: McGraw-Hill Interamericana do Brasil Ltda. (2009), 244.

[6] Cook, Stephen, *The P Versus NP Problem*, available in http://www.claymath.org/sites/default/files/pvsnp.pdf (2000).

Linguagens Variáveis no Tempo e o Problema P x NP

(Languages Varying in Time and the P x NP Problem)

Valdir Monteiro dos Santos Godoi
valdir.msgodoi@gmail.com

An original proof of P ≠ NP.

Conforme sabemos, não é possível acertar 100% das vezes nossas previsões sobre o resultado de um jogo de par ou ímpar, cara ou coroa, lançamento de dados, roleta, cartas, loteria, mercado de ações, etc., enfim, todos os eventos onde o caráter probabilista domina, tem forte influência.

Não é possível acertar sempre nossas previsões no sentido determinista, admitindo-se que vivemos num mundo onde existe o livre arbítrio. Embora existam as rígidas leis da Física, elas não são capazes de predizer o futuro em toda a sua extensão, e com toda a precisão.

Sendo assim, está condenada ao fracasso qualquer tentativa de construir um algoritmo ou programa de computador determinísticos, destinados a acertar inequivocamente, sem erro algum, e em todas as vezes, estes resultados probabilísticos ou aleatórios (desde que não se trate de dados viciados, cartas marcadas, times e juízes comprados, etc.).

O mesmo já não pode ser dito de algoritmos não determinísticos. Um algoritmo não determinístico pode ser considerado como um processo que, quando confrontado com uma escolha entre duas (ou mais) alternativas, pode criar cópias de si mesmo para cada alternativa e prosseguir o processamento para cada uma delas, independentemente das demais, em paralelo. Se existir um conjunto de possibilidades que levem a uma resposta positiva então esse conjunto é sempre escolhido e o algoritmo terminará com sucesso ([1], [2], [3]).

Num exemplo simples de lançamento de dados, onde os resultados possíveis são os elementos do conjunto DADOS = {1, 2, 3, 4, 5, 6}, nossa máquina (de Turing) determinística (ou computador moderno) poderá lançar seu palpite entre os elementos de DADOS, mas terá apenas 1/6 de probabilidade de acerto.

Como nossa idealizada máquina de Turing não determinística (MTND) poderá escolher cada uma das seis alternativas possíveis de DADOS (produzindo, digamos, seis cópias de si mesma), uma das alternativas deverá evidentemente coincidir com o resultado correto do lançamento do dado, e o processamento terminará no estado de sucesso ou aceitação.

Vimos assim que uma máquina ou algoritmo não determinísticos são capazes de algo que a correspondente versão determinística não é capaz: acertar sempre.

Para cada lançamento a linguagem relacionada ao resultado correto variará em função deste resultado, ou seja, se nosso dado mostrou a face 1 para cima então a linguagem L = {1} é a linguagem que produzirá o resultado correspondente ao sucesso/aceitação naquele

momento, enquanto os demais valores possíveis, 2, 3, 4, 5 e 6, não pertencerão a L durante aquela jogada específica, com aquele jogador específico.

Num momento seguinte poderemos ter L = {3}, no próximo lançamento e a seguir L = {6}, depois L = {1}, L = {5}, etc., evidenciando que temos um exemplo de linguagem não constante, e variável no tempo, dependente de cada nova situação.

Para que um algoritmo não determinístico, ou sua correspondente MTND, construído para reconhecer e aceitar estas linguagens seja capaz de decidir entre aceitar ou rejeitar um dado de entrada, entre informar um SIM ou NÃO, SUCESSO ou INSUCESSO, é necessário que venham dados do ambiente externo, a fim de se poder decidir, pela comparação, sobre a aceitação ou não de seu palpite, previsão, estimativa, etc.

Se definirmos uma linguagem da forma

$L = \{w = (x\ y\ z); x$ é a chave do problema, y é o palpite da máquina, calculado e registrado antes de se saber o valor de z, z é o resultado correto do problema$\}$

a máquina aceitará a entrada sempre que $y = z$, e rejeitará caso contrário, ou seja, os elementos de L neste problema são os strings w tais que $y = z$.

Este é um procedimento em tempo polinomial: dado x gera-se y (de maneira não determinística para as MTND ou determinística para as MTD), espera-se a realização completa do evento indicado em x, obtém-se z (do ambiente externo, que informará o resultado que efetivamente ocorreu) e se $y = z$ aceita-se a entrada w. Um exemplo para x: PETR4-2018-03-08-CLOSE, onde se pergunta qual o valor que a ação PETR4 da Petrobrás terá no fechamento de 08/março/2018 (suponhamos valores possíveis na faixa de R$ 0,00 a R$ 10.000,00 apenas. Valor R$ 0,00 significa que não houve negócios de PETR4 naquele dia, p.ex., devido a um feriado).

Uma MTND, corretamente projetada, tenderá por aceitar (em tempo polinomial) um de seus palpites y, pois gerará todos os valores possíveis para z, um para cada string de entrada, então esta é uma classe de problemas que pertence a NP.

A versão determinística não terá a “habilidade”, propriedade, de produzir cópias de si mesma, como se fossem processamentos em universos paralelos e simultâneos, e poderá apenas fornecer um único palpite para a chave x, que seja baseado em algum tipo de procedimento matemático e/ou estatístico mais adequado para a solução da questão. Dado o caráter incerto destes problemas (dados, roletas, cartas, bolsa de valores, adivinhações, etc.) não se poderá dizer que se construiu um algoritmo, nem máquina determinística, para se acertar/resolver o problema, com certeza absoluta, sendo assim estes tipos de problemas não pertencem a P, mas pertencem a NP, então $P \neq NP$.

Vemos que esta é uma demonstração que utiliza a mais fundamental propriedade do não determinismo, uma característica que as máquinas determinísticas são incapazes de realizar, que é a produção de “cópias” de si mesma e o processamento simultâneo com as outras “versões” da máquina (ou programa), até mesmo uma quantidade exponencial de cópias de si mesma (em relação ao tamanho da entrada).

Se preferirmos não adotar esta propriedade de criação e paralelismo, e ao invés disso admitirmos que as MTND têm uma sorte absoluta, que são abençoadas com uma sorte inacreditável, de modo que sempre fazem a melhor escolha [4], na primeira tentativa, como parece defender Papadimitriou *et al* em [5], nossa demonstração não mudaria em essência: o algoritmo não determinístico acertaria sempre, desta vez por sorte extrema, mesmo sem criar novas versões da máquina, uma para cada alternativa que é necessária ao algoritmo. O algoritmo determinístico, por sua vez, não teria nenhuma sorte absoluta, faculdade premonitória, nem poder "sobrenatural" de multiplicação instantânea, e só seria capaz de acertar, em geral, em termos probabilísticos.

Percebam que não é um eventual tempo de execução de ordem exponencial para se gerar um palpite a causa principal que faz com que estes problemas (DADOS, BOVESPA, etc.) não pertençam a P, mas sim a impossibilidade de resolver sempre e corretamente uma entrada lida pela máquina determinística. É como um navio de passageiros que encalha ou afunda 95% das vezes e só completa uma viagem com probabilidade de 5%. É um navio que obviamente não serve para se viajar, e ninguém deveria usá-lo. Idem uma máquina de calcular que troca aleatoriamente as funções de seus botões e não nos avisa. Ou seja, provar que $P \neq NP$ não implica que $SAT \notin P$ ou de forma genérica $NPCOMPLETO \notin P$. É possível ter $P \neq NP$ e também $NPCOMPLETO \in P$.

Também vale a pena mencionar que em nosso exemplo de PETR4 perguntamos sobre um valor que será conhecido somente daqui a dois anos aproximadamente. Claro que foi um exemplo exagerado, pois podemos entrar com dados muito mais próximos de acontecer, e mais simples de verificar (como o lançamento de dados, o nosso exemplo inicial).

De qualquer forma, para que possamos processar o resultado correto, o computador (ou máquina de Turing, MT) deve ser informado através de algum mecanismo de entrada sobre este correto valor. Enquanto o evento não terminar, por exemplo, o pregão de ações do dia ou o lançamento do dado, o computador (ou MT) ficará aguardando a entrada do valor correto, mas já fez sua previsão. Claro, uma MTND, que se multiplicou em n cópias ou versões, devido às n alternativas de possibilidades, deverá receber a entrada contendo o valor correto em todas estas suas n versões. Isso é diferente do que se faz, mas afinal nossa prova é original. Onde está definido ou prescrito que as MT e os computadores modernos só podem começar a processar seus dados após lerem todos os caracteres de entrada até o fim, sequencialmente, e só após a leitura completa e ininterrupta desta entrada resolver sua "questão"? Tal restrição não está descrita formalmente em lugar algum, por exemplo, em [6]. Portanto, admitimos uma espera até que o evento probabilístico tenha ocorrido por completo e a informação do respectivo resultado tenha sido fornecida às MT.

Certamente que tem de ser $P \neq NP$.

A Alan Turing, In Memorian.

Referências

[1] R.M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher (eds.). New York: Plenum Press (1972), 85–103.

[2] J.E. Hopcraft, J.D. Ullman and R. Motwani, *Introdução à Teoria dos Autômatos, Linguagens e Computação*. Rio de Janeiro: Elsevier e Campus (2003), 452.

[3] N. Ziviani, *Projeto de Algoritmos com Implementações em Java e C++*. São Paulo: Thomson Learning (2007), 381-383, 388, 551.

[4] Devlin, Keith, *Os Problemas do Milênio – sete grandes enigmas matemáticos do nosso tempo*, cap. 3. Rio de Janeiro: editora Record (2004), 168-169.

[5] S. Dasgupta, C. Papadimitriou and U. Vazirani, *Algoritmos*. São Paulo: McGraw-Hill Interamericana do Brasil Ltda. (2009), 244.

[6] Cook, Stephen, *The P Versus NP Problem*, available at http://www.claymath.org/sites/default/files/pvsnp.pdf (2000).