International Journal of
Science and Engineering Investigations vol. 7, issue 76, May 2018
ISSN: 2251-8843

A New Approach: A Hardware Device Model Solving Traveling Salesman Problem in $O(n^2)$ Time (Practical Application and Theoretical Consequences)

Óscar E. Chamizo Sánchez
Private Researcher
(chasanos@telefonica.net)

Abstract- Traveling salesman problem (TSP for short) is perhaps the most widely known and deeply investigated problem in computation. Given a set of cities, the simple goal is to find the cheapest way of visiting all cities and returning to the starting one. The optimal (in case of a simetric euclidean TSP the shortest) path from the starting city to itself through all the remaining cities is, in general, only one from the (n-1)!/2 set of possible *tours* or *circuits*. In this paper we present a hardware device model solving any instance of TSP in $O(n^2)$ time. In section 1 we go directly to the device model and prove mathematically its validity. In section 2 we explain the basic ideas behind the physical model. In section 3 we analyze complexity in such a device. In section 4 we outline the key aspects to put into practice the theoretical model in a feasible device. In section 5 we claim an almost sure equivalence between the Extended Church-Turing thesis and the conjecture P=NP. Finally in section 6 we discuss interesting implications in the field of computational complexity with special regard to the widely believed conjecture P≠NP.

TSP is one of those so intractable problems that the best, perhaps the only, way to solve is to get somebody else to solve it, if possible a "blind" law of nature, let´s say gravitation or electromagnetism.

Keywords- *Computational Complexity, TSP Problem, P versus NP, Church-Turing Thesis, Network Computing*

I. THE MODEL

Let be S a set of n emitter and receiver devices (e.g. computer network, cellular phones, etc., from now on cells) at any configuration in a 3-dimentional space. All cells are directly linked to each other whiteout antennas or intermediate devices of any kind and every cell can perform the usual operations, receiving, processing and sending a string of bytes, at the same speed. Now, any cell, let´s say cell 1, sends in $t=t_0$ its own number (1) to the remaining cells of the set. Every time a cell receives a string, appends its own number to the string and rebroadcasts it unless its own number is already in the string, thus avoiding redundant loops and subtours.

Proposition: The first string (A straightforward reasoning assures that in fact not one but two n-strings will arrive in first position simultaneously with the same stored sequence in reverse order) of length n to reach cell 1 in $t = t_1$ is the shortest path from 1 to itself through all points of S-{1} and $c(t_1-t_0)$ is the optimal tour length (c = speed of light), as we can state with no loss of generality that operation inside each cell takes no time.

Lemma 1: Every string of length n arriving to 1 is: 1,σ(S-{1}) i.e., element {1} at first position followed by a permutation of S-{1}.

Proof: Every string of length n reaching cell 1 started in 1 and has passed only once through each one of the n-1 remaining members of the set. Had it passed twice the string would not have been sent so the string would not have reached cell 1. Had it skipped any and the string would not have length n.

Lemma 2: Every possible permutation 1,σ(S-{1}) will reach 1 in finite time.

Proof: Every cell appends only once its number to every string and every string contains only once every cell, so given that every string begins with 1 and n is finite every 1, σ(S-{1}) string will reach 1 after n steps.

Lemma 1 and 2 prove that every tour reaches cell 1 and every n-string reaching cell1 is a tour. Now suppose the first n-string to arrive to 1 in time t and tour distance d is not the optimal tour i.e. the tour of minimum distance. Hence the optimal tour with tour distance d´ will reach 1 in $t´ > t$ but that is impossible given that: $t´= d´/c$, $d´< d$ and c constant. q.e.d.

II. THE IDEA BEHIND

How long it takes to solve the traveling salesman problem? It takes as long as it takes to the salesman himself to travel the optimal path. However, (n-1)! salesmen are required. Let´s see it: Each salesman takes a table, a tour sequence with n boxes to be filled out with the number of the cities he passed trough (Not necessarily *he*, but the table itself, or more accurately, the information stored at the table). So in the beginning n-1 salesmen leave at the same time the first city (say city one) bound for the remaining cities. In their tables the first box is filled with number one and all other boxes remain empty.

Whenever a salesman reaches a city the information stored at his table is copied to the n-1 tables of n-1 new salesmen (Obviously the arriving salesman could continue his travel thus reducing number of new salesmen needed to n-2 but that is completely irrelevant to our purpose). These n-1 salesmen append the number of their city to the table, leave their city bound for n-1 cities and so on and on. There is one important exception: No salesman leaves a city when its number is already stored at the arriving table.

Is almost trivial to prove that:

1) The first salesman to reach the starting city with a full n table has traveled through the optimal path, optimal path that has been stored at his table.

2) The optimal tour length is: time from the starting city multiplied by speed.

Salesman himself has solved salesman problem and the certificate is in salesman´s hand: The table of the n cities in the very order his table passed trough.

Obviously some assumptions must be made to assure the accuracy of the result:

1) All salesmen move at the same constant speed.

2) Time taken in copying and, when required, appending data to the tables is zero. Or at least is the same for every salesman at any city. Anyway this time must be negligible in the sense that is less than the time taken to travel between the two closest cities in the set, not a challenging assumption indeed.

Now, the well-known drawback, the very bottleneck of this approach is: when n grows the number of salesmen required grows exponentially until it reaches mammoth amounts of them even for humble sets of cities. Second drawback: The longer the path the longer the waiting time (The problem of the tour length is almost negligible (almost is not absolutely, see below) given that the set can always be resized with a change of scale shrinking distances while keeping relative positions.).

So all we need to improve our approach to TSP is to get very fast salesmen in industrial quantities. Do we have? Do we have an almost infinite supply of salesmen traveling at speed of light? We do. Photons. This brings us back to section 1.

III. Analyzing complexity in a FIN model

The model consists of a network of n fully interlinked Turing Machines (from now on Fully Interlinked Network, FIN for short). Worst case complexity of the model is easy to state: When the first n-string reaches the starting city three simple operations has been made n times: sequential search, O(n), and insertion, O(1) if required, plus operations intended for moving strings from/to ports, O(n). In order to preserve assumption 2 one of the following arrangements must be taken:

1) Keep always n+1+n cycles of running time to perform operations no matter if these cycles are used or they are not.

2) Set strings of n zeros from the beginning and replace zeros as required.

Thus in both cases worst, best and average complexity turns out to be $O(n)*[O(n)+O(1)+O(n)]->O(n)*O(n)->O(n^2)$.

Why is FIN model so efficient? It is so efficient -might be thought- because is a set of n TM´s working in parallel computation. Not at all. At the heart of a TM each tick from its clock allows to perform one and only one basic operation. At the heart of the universe each heartbeat of time allows infinite operations. A single salesman traveling n! tours or n! salesmen traveling one tour in turn is perfectly equivalent and perfectly ruinous in terms of efficiency. Why not n! salesmen traveling at a time? Is the universe a TM? Sure not. It would be better to say that Universe is a Super-TM. Universe can perform an unbounded number of operations during the same period of the time a TM performs one. Even more: shrinking the duration of a TM tick inexorably finds unreachable physical bounds, between others, speed of light [1]. However we don´t even know if there is something like a tick in the universal clock and in case yes which its duration is. And yes, as a matter of fact, we can improve the performance of an algorithm running in a single TM by adding new TM´s working together in parallel mode. But adding a new TM would be (as its best) equal to duplicate the clock frequency i.e. the number of ticks per second. Something useless when the number of operations to perform grows exponentially with the size of the input. What makes the difference is: In parallel computation model adding one more cell speeds the running time in at most a factor 2. In our network computation model adding one more cell speeds running time in a factor n. The same factor in which adding a new city to a TSP instance increases its complexity. Not only a quantitative change but a qualitative one.

Could this computation device be simulated by a TM? Of course. In fact, the underlying idea of Held Karp algorithm [2] performs exactly that simulation. However it runs in $O(2^n n^2)$ time (As a matter of fact from 1962 no one has been able to improve such performance with a better exact algorithm). In other words, there exists a physically realizable computation model that presumably cannot be simulated by a TM with polynomial overhead. Perhaps this has nothing to do with classic computation but with a brave new world, but all that will be further addressed.

IV. Outlining a real device in practice

Now, let´s go back to Earth and let´s try to figure out how this device could be implemented in practice.

The first observation to be made from a practical point of view concerns that so seemingly unfeasible idea: For each instance of the problem we must arrange physically in space a set of cells or computers, integrated circuits, RAM´s or whatever it be. But our hardware, like it or not, is nothing but a TM network with their interlinked devices in a concrete position in a physical space. Could we emulate this machine with some feasible and familiar hardware tool while keeping its interesting skills? I think so.

Two different approaches can be devised in order to replicate usefully the model:

1) A set of electronic interlinked devices packed as closely as semiconductor chip makers usually do.

2) A conventional computer network. It could be implemented with some technical adjustments in a global system like internet. This network with five billions interconnected devices provides boundless possibilities in order to solve unprecedented instances of the TSP and therefore of other NP problems.

In both cases n devices/computers are required. And in both cases their spatial configuration is meaningless at all if we make sure the transmission time between each other is exactly the same. The underlying idea here is to translate distance into time. Then the spatial configuration of the instance could be an input to be stored in its digital memory. Why? Because every device can delay the sending of each string to the remaining devices a given amount of time as a linear function of the physical distances in the TSP instance. Thus configuring n TM in physical space is equivalent to store their n(n-1)/2 distances in the digital space of the machine´s RAM. In fact every device needs to store only n-1 distances i.e., its own row from the matrix of distances. Better yet if, once and for all, in the very configuration procedure each device stores directly the, let´s say, timetable of dispatching times.

Now, suppose we have our n device/computer network configured, i.e., each computer has stored its own row of n-1 distances from the table of distances in its memory translated to delaying times. Once any of them chosen to begin with, let´s say computer 1, we set the following straightforward procedures:

Algorithm for computer 1:

-Send 1 to the net.
-Waiting to receive a string.
-A string is received.
¿Has n length?
No: Keep waiting
Yes: Output: The n length string. End of program.

Algorithm for any other computer *m*:

-Waiting
-A string is received.
-Is *m* in the string?
Yes. Keep waiting.
No. Store the last byte (say k) of the string. Search in the timetable the corresponding delay to that byte. (Distance m-k). Append m to the string.

¿Has the string n length?
No. Store it to the buffer/dispatching queue.
Yes. Search in the timetable the corresponding delay to the distance m-1 and add this delay to the previous one. Store it to the buffer/dispatching queue.
Send at scheduled time.
-Keep waiting.

Some issues and technical challenges still need to be addressed.

1) How to match distances (distances are floating points) to time delays in ticks from the clock (ticks are integers) i.e. how to round numbers without jeopardizing the accuracy of the overall process.

2) How to deal with exactly simultaneous arriving or departing strings.

3) How to avoid unintended time delays when a string reaches a device before the previous string has been entirely dispatched.

Solution for the first problem would be resizing if necessary. Given that any set of points can be resized *ad libitum* and distances increased or decreased, a trade-off between accuracy and efficiency must be taken.

For the second issue the simplest approach could be to set a universal delay of r cycles for every device whenever it gets a string, in order to ensure enough time for data processing and orderly dispatch at the scheduled time. This general delay would imply an increase of n*r cycles in execution time. But $O(nr)->O(n)$ and $O(n)+O(n^2)->O(n^2)$. Once more, network processing pays off.

In respect the third issue I cannot envisage another solution but a (most likely) exponential space of memory, the very bottleneck of this approach.

One could wonder if it´s worth making the effort to synchronize an n-computer network (the good news here is that such a device is already created) or to design and build an n-device machine for a n-input problem. The answer would be yes indeed if only for TSP, where fields of such a paramount importance as ongoing work in genome sequencing are involved, but the fact that any NP complete problem can be reduced (i.e. translated) to another one in polynomial time [3] makes it a general purpose machine. In respect of hardware needs aside from the required space of memory, CPU resources are kept to a minimum. Each device needs only to operate over n sized strings of integers the simplest imaginable procedures.

Anyway, electronic requirements, computer architecture and further technical details are well beyond my capabilities in the subject.

V. Upon the equivalence between the Extended Church-Turing Thesis and P=NP

Aside from any practical purpose, our hardware device model allows us to make some interesting statements that we can infer from.

Since the mere Church-Turing thesis states that Turing machines can simulate any computational device, but perhaps at the cost of an exponential overhead in time and/or space, the Extended Church-Turing Thesis (ECT for short) says that every physically realizable computational device can be efficiently simulated by Turing machines[4]. Unlike the Church-Turing Thesis, which computer science general

agreement believes to be right, the Extended Church-Turing Thesis is widely believed false just as it happens with the P=NP conjecture[5].

Now we can prove that:

1) Extended Church-Turing thesis implies P=NP.

As shown, exists a physically realizable computation model solving TSP in $O(n^2)$ time. If ECT thesis holds, this computation model can be simulated by a TM with polynomial overhead, i.e., in $O(n^2)^k$ time, so it follows that P=NP. q.e.d.

2) If Church-Turing thesis holds, P=NP implies Extended Church-Turing thesis. Indeed if P=NP, anything computable is solvable in polynomial time. If the mere Church-Turing thesis (widely tested and believed to be true) holds, every physically realizable model can be simulated by a Turing machine. If it is computable in $O(n^k)$ time, it follows that the ECT is true. q.e.d.

Thus, if Church-Turing thesis holds, ETC and P=NP are equivalent.

VI. P, NP AND BEYOND

After all, things would be really easier if P=NP let´s say in the classical sense. (From now on P=NP c.s.) P=NP c.s. means an efficient algorithm exists solving TSP in a TM computation model. But for years we have been waiting for someone to turn up with an efficient algorithm or for someone to turn up with a proof that such algorithm does not exist. We have surrendered. In our approach we really do nothing. It is not an algorithm, it is not a simulation, it´s the very speed of light traveling the very paths of the set solving the problem in real time. We have thrown in the towel, we have looked for someone in nature to do the task, speed of light, and told her: I can´t deal whit this. Do it yourself. So meanwhile ¿Is TSP solvable in polynomial time $O(n^c)$ in a single TM? We Don´t know.

However, I dare to make the following remarks:

First: We have proved that assuming Church-Turing thesis, P=NP c.s. implies the Extended Church-Turing thesis (One cannot fail to note that the *Extended form of the Church-Turing thesis* has nothing to do with Church/Turing.[7]) and, as shown, it seems unlikely that an efficient algorithm in TM could simulate our device model. A proof that such a device model cannot be simulated by a TM in polynomial time would invalidate the extended form of CT thesis and then P≠NP c.s.

Second: If the answer is yes (P=NP c.s.) and parallel computation thesis holds true, ¿is it solvable in O(n) time in a parallel computation mode? But what would that mean? Would it mean that we could turn up with the optimal tour *before* the salesman, traveling at speed of light, have passed through? Would it mean that something somewhere someway moves faster than light? That would be nonsense. Then I conjecture not. I conjecture such an algorithm for a single TM could exist only if there is in the universe some $v > c$. I can´t even figure out where to start proving that impossibility by means of such a bizarre statement, rooted not (only) in mathematical arguments but in the deepest structures of physical nature of reality.

The outstanding and visionary works by Turing, Church, Gödel and others giants of science in the 1930´s allowed us to take full advantage of a model that became quickly standard and paradigm of computation. Eighty years later, the time has come to ask the question whether this familiar paradigm fully encapsulates everything machines can do or, on the contrary, this model has become outdated in view of the hardware tools nowadays available. In particular with regard to unexplored areas of connectivity and intercommunication in running time between algorithms instead of a single algorithm running disconnected from outside world (i.e., disconnected from other algorithms) and self-absorbed from start to finish.

At this point it´s worth wondering why a "natural brain" is much more efficient than an electronic one [6] performing certain kind of tasks, pattern recognition, future forecasting and so on, tasks that share in common the following feature: They don´t fit well with our computation model of a step-by-step instructions set. Instead, it seems that the power of a human brain lies, not in the, let´s say, computational capacity of neurons themselves but in their interlinking density and speed.

Our old CPU, or TM, or RAM approach is a general purpose, ideal tool to find exact solutions in unbounded time with complete information (By no means am I stating that our TM model cannot turn up with an acceptable solution within a (well...) reasonable period of time, as is proved, for example, by very efficient heuristic algorithms for TSP. I limit myself to remind what TM is designed to and what is mainly for). For evolutionary reasons our brain became a general purpose machine to quickly find with sketchy information acceptable answers for straightforward, hard and even imposing challenges. Joining the best of both worlds would be such a giant leap that it would indeed improve dramatically, perhaps in a dreadful manner, our capacity to shape world and ourselves.

VII. CONCLUSIONS

As far as I know, while single TM computation, parallel computation or network standard computation are deeply investigated fields, theory for a FIN model like this does not exist. I think I have proved that, at least on a theoretical level, TSP can be solved in $O(n^2)$ time and hence P=NP. The key issue is whether this arrangement is technically feasible or not in which case perhaps it will remain like an inconsequential oddity (*gedankenexperiment*). If such a roughly outlined computer network or machine is technically feasible and it could be made, it will be made, perhaps sooner than expected. Other modern devices such as drones, inexpensive, with fast 3-D mobility and intercommunication in real time, seem really intended to replicate our device model in the most straightforward and effective way, to the extent that a drone computing model could be founded and developed. Perhaps by then we might not know yet whether an efficient algorithm in TM could be implemented but... would it really matter?

REFERENCES

[1] Seth Lloyd, "Ultimate physical limits to computation", Nature 406, 1047–1054 (31 August 2000).

[2] Michael Held and Richard M. Karp, "A dynamic programming approach to sequencing problems", Journal for the Society for Industrial and Applied Mathematics 1:10. 1962.

[3] Sanjeev Arora and Boaz Barak, "Computational complexity. A modern approach", Princeton University. Downloadable version from January 2007. pp., 50-55.

[4] B. Jack Copeland. "The Church-Turing Thesis". Stanford Encyclopedia of Philosophy, 1997 (substantially revised in 2017).

[5] William I. Gasarch, "The P vs. NP poll". *SIGACT News* **33** (2) June 2002: 34-47.

[6] Arunodhayan Sam Solomon D, Melwin, Banu, Sashikala. "Superiority of the Human Brain over the Computer World in terms of Memory, Network, Retrieval and Processing". American Journal of Engineering Research, volume-03, Issue-05, pp-230-239. 2014.

[7] Enrique L. Dáder, Luis Soria Romero, Luis C. Yepes, "What Turing stated, what he really meant and what others understood", unpublished.