Efficiently solving Traveling Salesman Problem in a computer network. (Internet)

Dated: April-28-2018.

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 our seminal paper *A new approach: A hardware device model solving TSP in $O(n^2)$ time* we presented a physically realizable computation model solving any instance of TSP in $O(n^2)$ time, a model that can´t be efficiently simulated by a single TM, thus proving the fallacy of the so called *strong form of Church-Turing thesis*. In this paper we present a model of network computation that solves TSP in $O(n^2)$ time, model that 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 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 outline the key aspects to put into practice the theoretical model in a computer network.

1. 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 the 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[1] of length n to reach cell 1 in $t = t_1$ is the shortest path from 1 to itself trough 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: $1,\sigma(S-\{1\})$ 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.

1.- 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.

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.

2. 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 necessarly *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[2].

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.

2.- 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.

3. Outlining the model in a computer network .

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 peculiar TM network with their interlinked devices in a concrete position in a physical space. Could we emulate this model with some feasible and familiar device while keeping its interesting skills? I think so.

The answer is a conventional computer network. Their spatial configuration is meaningless at all if we make sure **the transmission time between computers 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 their digital memories. Why? Because every computer can **delay** the sending of each string to the remaining computers 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 computer´s RAM. In fact every computer in the net 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 computer stores directly the, let´s say, timetable of dispatching times.

Now, suppose we have our n 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: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 of the string. Search in the timetable the corresponding delay to that byte. Append m to the string. Store it to the dispatching queue. Send it to the net at scheduled time.

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 computer 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 an universal delay of r cycles for every computer 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.

The good news here is that such a device is already created. One could wonder if it´s worth making the effort to synchronize a n-computer network 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 makes it a general purpose device. In respect of hardware needs aside from the required space of memory, CPU resources are kept to a minimum. Each computer needs only to operate over n sized strings of integers the most simple imaginable procedures.

Anyway, electronic requirements and further technical details in order to achieve an almost perfect synchronization (same transmission time between computers) in a computer network are well beyond my capabilities in the subject.

PA^3

Dated: April, 28-2018.
Óscar E. Chamizo Sánchez.
chasanos@telefonica.net
Any comment, suggestion or critic is welcome.