A new approach: Solving the Hamiltonian circuit problem in $O(n^2)$ time in a computer network.

Óscar E.Chamizo Sánchez. (chasanos@telefonica.net). Dated: June-11-2018.

Abstract

Hamiltonian circuit problem (HCP for short) is one of the most famous and deeply investigated problem in computation. Given a (directed or undirected, 2 or 3 dimensional[1]) graph the simple goal is to answer to the question whether exists or not a circuit that visits each vertex exactly once. Since the problem of finding a Hamiltonian circuit is NP-complete, the only known way so far to find whether a general graph has a Hamiltonian circuit was to perform and exhaustive search with exponential execution time. In this paper we present an interesting supplement from our seminal paper *A new approach: A hardware device model solving Traveling Salesman Problem in $O(n^2)$ time. Practical application and theoretical consequences* [1] now solving any instance of HCP 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 model. In section 3 we analyze complexity and software and hardware design.

1. The Model.

Let be S a set of n computers/nodes (from now on *nodes*) in a computer network. Each node stores in its memory the numbers of the nodes connected to itself in the instance of the HCP. Now, any node, let´s say node 1, sends its own number (1) to the remaining nodes of the set. Every time a node receives a string, appends its own number to the string and resends it to the net **unless:**

1) its own number is already in the string, thus avoiding redundant loops and subtours.

2) the last byte of the string received is not stored in its memory, thus avoiding forbidden edges in the circuit.

Propositions: Any string of length n reaching node 1 is a Hamiltonian circuit. No string of length n will reach node 1 if the graph does not possess a Hamiltonian circuit.

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 feasible 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. Had it passed a non connected edge and the string would not have been sent.

Lemma 2: Every feasible permutation $1,\sigma(S-\{1\})$ will reach 1 in finite time.

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

Lemma 1 and 2 prove that every feasible tour reaches node 1 after n steps and every n-string reaching node 1 is a feasible tour. So a n-string will reach node 1 if and only if there is a Hamiltonian circuit. q.e.d.

1.- An alternative model for a directed graph would require only to change for each vertex the set of connected vertices in accordance. With regard to dimensions, our approach is suitable for 2 and 3Ds.

2. The idea behind.

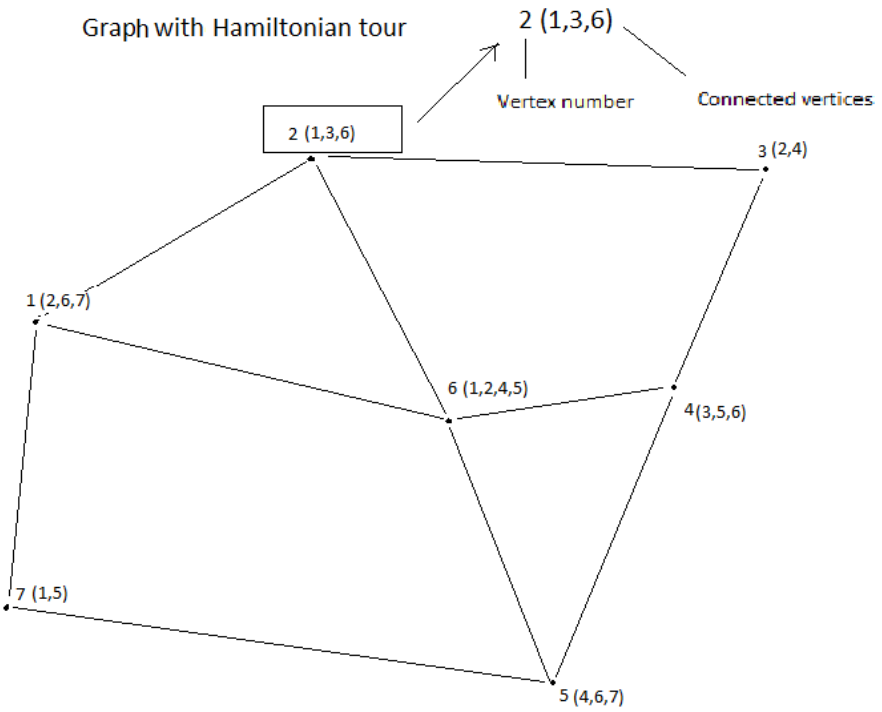


Figure 1

Let´s consider an instance of HCP like the one displayed in Figure 1: In our model each vertex is performed by a computer/node in the computer network. Each node is assigned with its vertex number and stores in its memory the set of vertices it connects with. (indicated between brackets in Figure 1). In any realistic and practical computer network every and each node sends to and gets from all other computers. Given that we are interested only in feasible circuits, non connected edges must be avoided so whenever a node receives a string from a node not stored at its memory (i.e. not connected in the graph) the string is not processed and fades away. The same thing happens if the string already contains its own vertex number, in order to avoid redundant loops and subtours.

For the sake of example, computer number 2 in Figure 1 will process only strings not containing number 2 sent from computers 1,3 or 6. If these conditions are fulfilled, computer 2 adds number 2 to the string and sends it to the net. And so on. It´s almost trivial to prove that computer 1 (the computer chosen to begin with) receives a string of length n if and only if the graph has a Hamiltonian circuit.

Le´s consider now the instance of HCP displayed in Figure 2. It´s easy to realize at a glance that it´s impossible to find out a tour that visits each vertex exactly once. So no string of length n will reach computer 1. Never. ¿What does it mean? Does it mean that we have to wait

indefinitely to solve the problem? Sure not . As explained above, strings that do not fulfill the scheduled conditions when reaching a vertex/computer disappear. They are never sent.

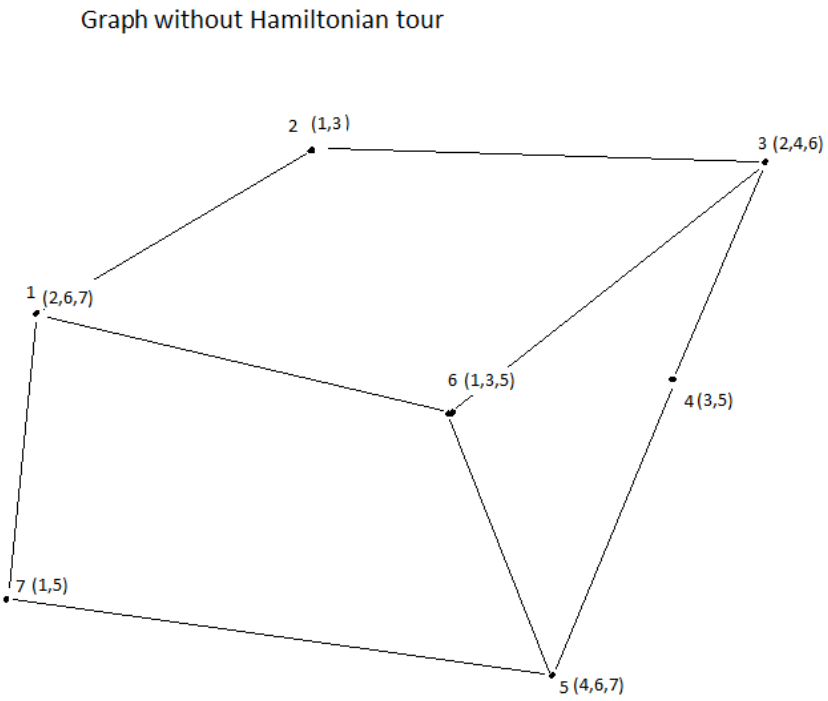


Figure 2

So it seems straightforward to figure out the behavior of our model when fed with an instance like the one displayed in Figure 2: All activity in the net will soon come to an end. No string will survive and no computer will send or get anything. Something very easy to check. But anyway ¿how long we might wait to get a *no* answer? It´s simple to realize that it cannot take more time than it would take if the answer to the question whether a Hamiltonian circuit exists would be *yes*.

3. Computational complexity and algorithmic design.

The model consists of a network of n computers. Complexity of the running model is easy to state: When the first n-string reaches computer 1 three simple operations has been made n times: sequential search, O(n), and insertion if required, O(1) plus operations intended for moving strings from/to ports, O(n).

Thus worst complexity turns out to be O(n)*[O(n)+O(1)+O(n)]->O(n)*O(n)->$O(n^2)$ if we take for granted that the *yes* answer running time is an upper bound for the general case, as explained above.

Now, suppose we have our n node/computer network configured, i.e., each computer has stored the set of vertices it connects with. Once any of them chosen to begin with, let´s say computer 1, we set the following straightforward procedures:

Pseudocode:

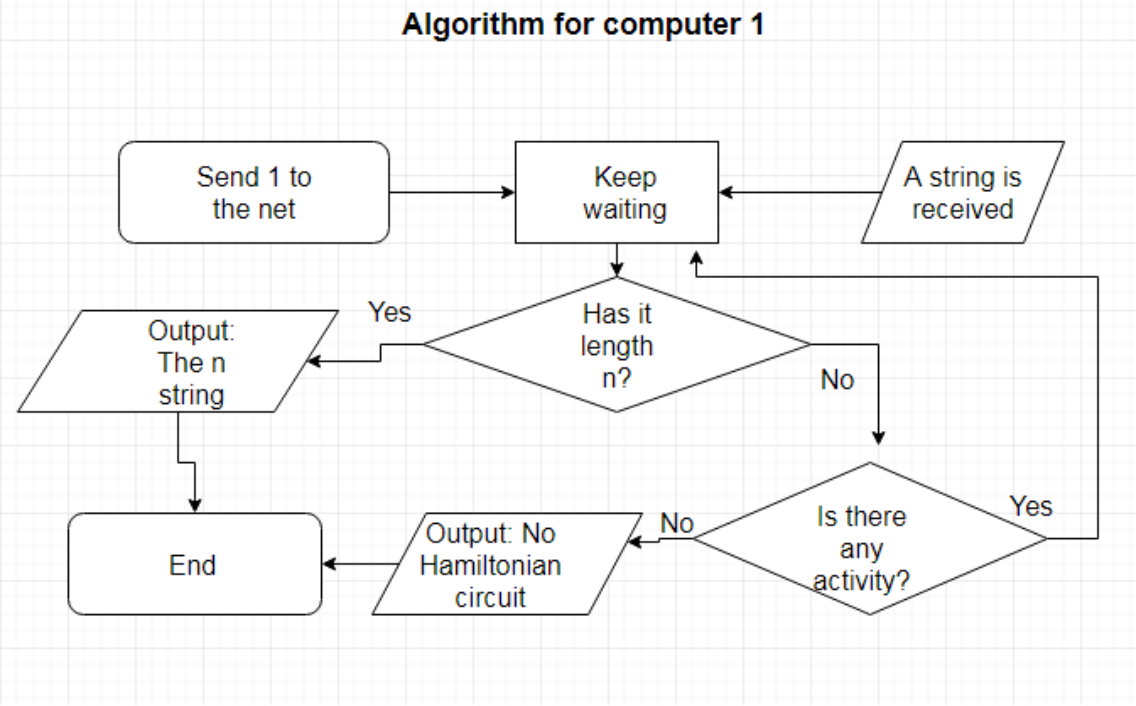
Algorithm for computer 1:

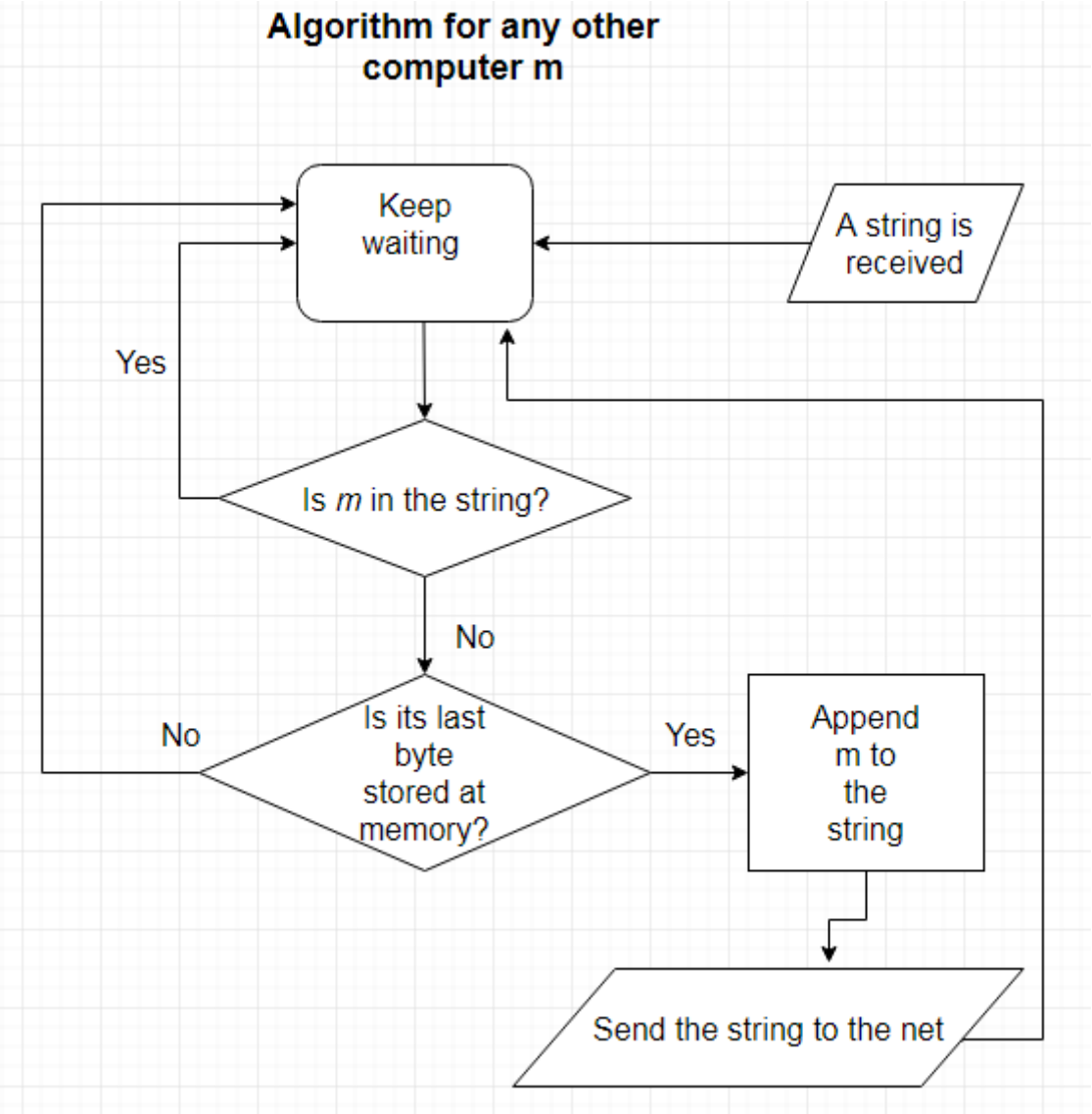
-Send 1 to the net.
-Keep waiting.
-A string is received.
¿Has n length?
No:
Is there any activity?
No: Output: No Hamiltonian circuit. **End of program**.
Yes: Keep waiting.
Yes: Output: The n length string. **End of program**.

Algorithm for any other computer m:

-Keep waiting.
-A string is received.
-Is *m* in the string?
Yes. Keep waiting.
No.
Is its last byte stored at memory?
No. Keep waiting.
Yes. Append m to the string. Send it to the net. Keep waiting.

Flow diagrams:





If this software design stands out for an almost childish easiness, hardware requirements are also kept to a minimum. In contrast with our abovementioned approach[1] to the closely related Traveling Salesman Problem[2], there is no need here to set synchronization between computers and time delays. Simply we must make sure that all strings are really sent, received and, when required, forgotten, no matter when or in which order. At this point it´s worth pointing out that in an inter-algorithmic computation model like this Traveling Salesman Problem is not only as hard as HCP [3]. While both run in $O(n^2)$ time, in regard with hardware needs TSP it´s really harder.

Anyway, transmission network capacities and inter communication speed and efficiency are well beyond my capabilities in the subject.

4. Conclusions.

As far as I know, while single TM computation, parallel computation or network standard computation are deeply investigated fields, theory for an inter-algorithmic computation model like this does not exist. I think I have proved that, at least on a theoretical level, HCP

can be solved in $O(n^2)$ time and hence P=NP. Outstanding consequences and implications have been further discussed in my abovementioned paper [1] which interested or curious readers are invited to address.

5. References.

[1] Óscar E. Chamizo Sánchez, *A New Approach: A Hardware Device Model Solving Traveling Salesman Problem in $O(n^2)$ Time (Practical Application and Theoretical Consequences)*. International Journal of Science and Engineering Investigations, vol. 7, issue 76, May 2018.

[2] Applegate, Bixby, Chvatal, *The traveling salesman problem*, Princeton University. 2006.

[3] Steven S. Skienna, *The algorithm design manual*, Second Edition, Springer, 2008, p. 324.

PA[3]

Dated:June,11,2018.

Óscar E. Chamizo Sánchez.

chasanos@telefonica.net. (Any comment, suggestion or critic is welcome).