Hamiltonian Paths in Graphs

Atul Mehta
India

December 3, 2016

Abstract

In this paper, we explore the connections between graphs and Turing machines. A method to construct Turing machines from a general undirected graph is provided. Determining whether a Hamiltonian cycle does exist is now shown to be equivalent to solving the halting problem. A modified version of the classical Turing machine is now developed to solve certain classes of computational problems.

1 Introduction

Currently there are no known polynomial-time algorithms which allow us to determine whether a Hamiltonian cycle in a graph exists. Current methods often reduce to variants of brute force computation or developing a solution limited to a subset of the general problem. Both these approaches do not yield desired results in real life situations. In this paper we stray from representing a graph G as just an object (V, E). We transform graphs into Turing machines and investigate the consequences of such a transformation.

Outline The remainder of this paper is organized as follows. Section 2 illustrates our approach of viewing graphs as Turing machines. Some of the results directly obtained from this new viewpoint are listed in Section 3. Section 4 shows ways of constructing a newer class of computing machines. Section 5 indicates lines of future research.

2 Graph Transformation

We need to formalize the process of converting a graph $G = (V, E)$ to its equivalent Turing machine counterpart $M =< Q, \Gamma, b, \Sigma, \delta, q_0, F >$[1]. The following is just one of several ways to do it.

- Q is the finite, non-empty set of states and is same as set V.
- Γ is a finite, non-empty set of tape alphabet symbols. Every element in $|E|$ is represented by a unique (non-zero) symbol.

[1] We use the Hopcroft/Ullmann representation of a Turing machine here

- $b \in \Gamma$ is the blank symbol. We denote that by 0.
- $\Sigma \subseteq \Gamma \backslash b$ is the set of input symobols.
- $q_0 \in Q$ is the initial state. Without any loss of generality we denote the first element in V, v_0 as our initial state.
- $F \subseteq Q$ is the set of *final* or *stopping* state. This is same as v_0.
- $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is our transition function. This can be derived easily from the set E. Without any loss of generality we denote an element in E say (v_i, v_j) and the edge corresponds to the input character α. This will contribute to building our δ with 2 rows: $v_i \times \alpha \to v_j \times \alpha \times R$ and $v_j \times \alpha \to v_i \times \alpha \times L$.

A sample Turing machine T is illustrated (Figure 1) alongside its graph counterpart. The node in blue represents v_0. This denotes the start/stop state by which we know that the computation is over at least once.

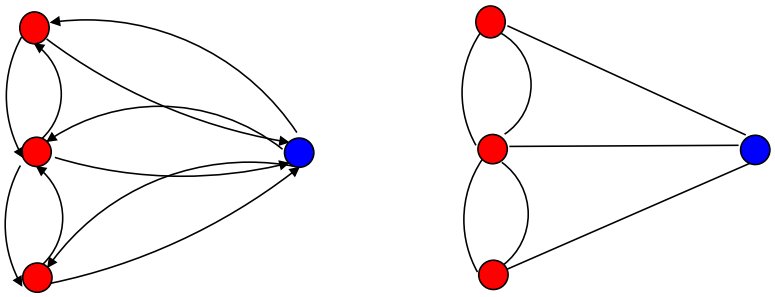

Figure 1: A Turing Machine modelling the bridges of Königsberg

An illustration of some Turing machines fashioned from arbitrary graphs is given below(Figure 2). The numbers on the edges represent the input. Every edge in the graph(s) below is bi-directional (as per our earlier construction). We just show it as a single line without the directional arrows solely for simplicity.

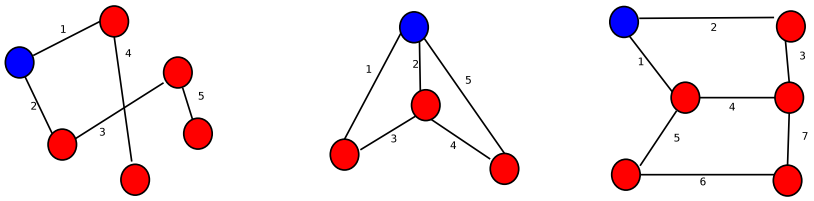

Figure 2: Turing machines constructed from various graphs

The idea behind our construction is that every path in a graph can now be linked to the processing of some input m by Turing machine T. There is no Hamiltonian path in the undirected graph counterpart of our first Turing machine. So there cannot be any input of size $|V|$ where there are no repeating alphabets for which the first Turing machine stops. The remaining two do indeed stop for input 1345 and 237651 respectively. *If T_g is the*

Turing representation of a graph G and it halts on a input of size $|V|$ with no repeating alphabets, then it is obvious that the graph G has a Hamiltonian Path.

3 Results

There exist an infinite number of graphs and consequently an infinite number of Turing machines which can be constructed out of them. A solution to the hamiltonian path problem will qualify as a partial halting solver.

Lemma 3.1 *A brute force approach to solving the Hamiltonian path problem does not qualify as a partial halting solver.*

Proof Consider the graph $G = (V, E)$ and its equivalent Turing counterpart T_g. Note that we cannot determine *a priori* if an arbitrary Turing machine halts on a particular input. This would mean solving the halting problem. So the brute force approach to solve the problem is to run $V!$ copies of T_g each with one input from the alphabet set $V!$[2]. If at least one copy of T_g halts in V steps, then we've a solution to the hamiltonian problem for graph G. But this implicitly assumes that a operator is present and the rate of computation is known *prior* to the start of the computation process. If we assume that a machine can consume a single alphabet of the input in one time interval, then the job of the operator is as follows:

1. At time $t = N$ if even one machine has completed computation, then report success. Optionally shut down all the machines.

2. After N time units have elapsed and none of the machines have completed, then report failure and shut down all the machines.

The requirement of an operator ensures that this does not qualify as a partial solution to the halting problem. We can only get rid of the operator and our dependence on a predefined rate of computation if we *renormalize* the input provided to the Turing machine. This is also a manual process. Strictly speaking if $(v_i, v_j) \notin E$, then $E(v_i, v_j) = \infty$ simply because it is undefined. Such an input cannot be consumed by a Turing machine and we need to manually replace the infinities arising with a constant c. So for example if edge (v_i, v_j) exists, then $E(v_i, v_j) = 1$, else $E(v_i, v_j) = 0$. With the input restructured, we can make a Turing machine run automatically till we figure out an answer, either 0 or 1 whether an Hamiltonian path does indeed exist in the graph input. The reliance of manually *renormalizing* the input again rules it out as a proper partial halting solver.

The general question remains? Is there a computable function to determine if a Hamiltonian path is present in graph G.

Definition If G has a hamiltonian path m then $\mathbf{H}(V, E)$ returns 1. For all other cases $\mathbf{H}$ returns 0. We further assume $\mathbf{H}$ is computable[3] and that the input has *not* been *renormalized*. An undirected graph can either have a Hamiltonian path or none exists. So we are guaranteed that $\mathbf{H}$ will always return a value.

[2]If we always assume that the first vertex is our v_0, then it is $(V-1)!$ machines

[3]Such a function will in theory allow us to *a priori* determine whether an arbitrary Turing machines does stop for some input. But we disregard this issue right now

Theorem 3.2 *The general method* **H** *defined above can be used to construct a* total *function and hence cannot exist.*

Proof We construct a program HAM based on the function H. HAM tries to extract the undirected graph from a given program p. If it is not able to do that then it returns 0, else it returns the value of **H**.

The pseudo code for the same is given in Algorithm 1. We use the Hopcroft/Ullmann representation to extract the graph information from p.

Algorithm 1 Total Function HAM

```
1: procedure HAM(P)
2:     if |F| ≠ 1 and q_0 ≠ F then
3:         return 0
4:     for all (v_i, α) ∈ δ do
5:         if v_i × α → v_j × α × R ∈ δ and v_j × α → v_i × α × L ∉ δ then
6:             return 0
7:         if v_i × α → v_j × α × L ∈ δ and v_j × α → v_i × α × R ∉ δ then
8:             return 0
9:     return H(Q,distinct(v_i, v_j ∈ δ))
```

For every input program p, HAM is guaranteed to produce a result. Or in other words, HAM halts on all inputs. But this is a contradiction. If we've a total function then clearly, we've solved the Universal halting problem. So our assumption of **H** being computable is wrong.

4 Hyper Turing Machines

We present an improvement over current brute force approaches to solving problems like Hamiltonian path problem. Consider a general graph $G = (V, E)$. Our task is to determine if a Hamiltonian path exists in linear time. The total number of vertices in the graph is denoted by N.

Assumptions We assume that the processing starts at node v_0. Each node in the graph knows the nodes directly connected to it and can pass messages to its neighbours. Nodes can delete messages, copy messages and update messages. All nodes also share a common clock. A message in our system is just a listing of the vertices in the graph along with a binary flag attached to each vertex. The flag can be updated only once for each vertex. In one time interval, nodes collect messages, process messages and send out updates to its neighbors.

When a node v receives a message in our scheme, the actions performed are the following:

- If the flag(v) in the message is already set to 1, then discard the message.
- if the flag(v) in the message is set to 0, then set it to 1. Copy and send out the updated message to all its neighbors.

We start our processing at node v_0 at time $t = 0$. It sends a message to its neighbors with flag(v_0) set to 1 and all other entries set to 0. At time $t = N$ if a node receives a message, then we know that a Hamiltonian path exists.

While our scheme works in linear time, it is inefficient in terms of space consumption. The number of messages can be polynomial but will still be less than $N!$ in most cases since we only investigate actual paths. This is still an improvement over current brute force approaches and also presents a different approach in parallel computation.

5 Conclusions

We proved that a general algorithm to solve problems like Hamiltonian path problem do not exist. Lines of future research involve investigating non-Turing models (as in Section 4). It is an open question as to whether such schemes can solve problems like the Hamiltonian path efficiently for graphs with low sparsity. Also if we limit processing on some nodes picked arbitrarily, then are we still guaranteed an answer? If so what is the space and time complexity involved.