Hamiltonian Paths in Graphs

Atul Mehta
India

May 28, 2018

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 exists is now shown to be equivalent to solving the halting problem. We investigate applications of the halting problem to problems in number theory. 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. A classical problem in number theory is tackled in Section 4. We explore ways of constructing a newer class of computing machines in Section 5. In Section 6 we indicate lines for future research.

2 Graph Transformation

We need to formalize the process of converting a graph $G = (V, E)$ [1] to its equivalent Turing machine counterpart $M =< Q, \Gamma, b, \Sigma, \delta, q_0, F >$[2]. 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.

[1] For the sake of simplicity we assume that there is at most 1 direct edge connecting a pair of vertices. So if $(v_i, v_j) \in E$ then $(v_j, v_i) \notin E$.

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

- Γ is a finite, non-empty set of tape alphabet symbols. Every tuple in E is represented by a unique (non-zero) symbol.

- $b \in \Gamma$ is the blank symbol. We denote that by 0.

- $\Sigma \subseteq \Gamma \backslash b$ is the set of input symbols.

- $q_0 \in Q$ 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* states. ***This is same as*** v_0[3].

- $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is our transition function[4]. 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 in the graph 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 $v_j \times \alpha \to v_i \times \alpha \times R$.

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.

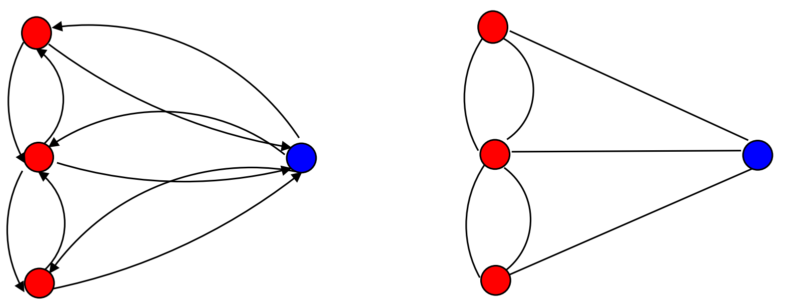

Figure 1: A Turing Machine modelling the bridges of Königsberg

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.

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 after exactly 6 steps. The remaining two do indeed stop for input 1345 and 237651 respectively. *If T_g is the Turing representation of a graph G and we know that G contains a hamiltonian path, then it is obvious that it halts on atleast one input of size $|V|$ with no repeating alphabets.*

[3]To keep machines from spinning after the computation is over, we can assume that they are rigged to self-destruct if *stop* is reached even once.

[4]Without any loss of generality, we assume our machine has read-only input and a separate output tape

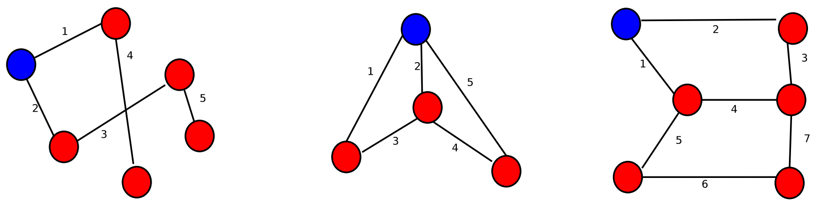

Figure 2: Turing machines constructed from various graphs

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 for solving the problem is to run $(V-1)!$ copies [5] of T_g each with one unique input of size $|V|$. We draw our input from a set of all possible paths and hence this set also has a cardinality of $(V-1)!$. 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. If even one machine has completed computation at exactly $t = N$, 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 all the infinities 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. But this tampering of the input is akin to feeding a program only valid input so that we circumvent the halting problem. This reliance on manually *renormalizing* the input rules it out as a proper partial halting solver.

[5] We always assume that the first vertex is our v_0, hence it is $(V-1)!$ machines

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 define $\mathbf{H}$ as a computable function and assume 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.

Theorem 3.2 *Even with a the function* $\mathbf{H}$ *as defined above, we cannot compute whether a given graph has a hamiltonian path or not.*

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 $\mathbf{H}$.

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

Algorithm 1 Total Function HAM

```
1: procedure HAM(P)
2:     if $|F| \neq 1$ and $q_0 \neq F$ then
3:         return 0
4:     for all $(v_i, \alpha) \in \delta$ do
5:         if $v_i \times \alpha \to v_j \times \alpha \times R \in \delta$ and $v_j \times \alpha \to v_i \times \alpha \times R \notin \delta$ then
6:             return 0
7:     return $\mathbf{H}(Q, \text{distinct}(v_i, v_j \in \delta))$
```

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 [1]. So our assumption of HAM being computable is wrong.

4 Applications to Number Theory

Turing machines are a fundamental concept in mathematics and consequenly a reduction to the halting problem can prove helpful in other areas besides theoretical computer science. We explore possible applications in number theory. More specifically we look at the problem of whether all even numbers greater than 4 can be expressed as the sum of two odd prime numbers. So for example 12 can be expressed as $5 + 7$, 8 can be expressed as $5 + 3$ and so on. It is not known whether *all* even numbers greater than 4 can be decomposed as a sum of two primes.

Given any even number, we can create an equivalent Turing machine to test out the *Goldbach Conjecture*. Figure 3 illustrates our approach. The blue colored node denotes the *start* state while the green node represents the final *stop* state. The numbers besides the edges mark the various inputs.

Before we formalize our Turing machine, we need to define a few terms. n denotes a even integer greater than 4. The set I is a finite set defined as: $\{i/n \in I$: iff i is prime

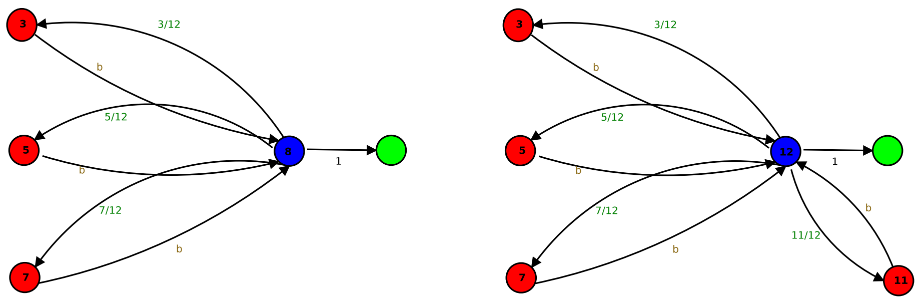

Figure 3: Turing machines modeling the Goldbach Conjecture for various even numbers

and $2 < i < n\}$[6]. P is a finite set of nodes representing a collection of primes such that $\{p \in P : \text{iff } p \text{ is prime and } 2 < p < n\}$. We denote our initial start state by q_i and the final state by s_f. We will also need a filler symbol in our input which we denote by $\boxtimes$. Now we can define our Turing machine $T_n =< Q, \Gamma, b, \Sigma, \delta, q_i, F >$ as follows.

- Q is the finite, non-empty set of states and is the set $P \cup \{q_i, s_f\}$.
- Γ is a finite, non-empty set of tape alphabet symbols. It is defined as $I \cup \{n/n, 0, \boxtimes\}$
- $b \in \Gamma$ is the blank symbol. We denote that by 0.
- $\Sigma \subseteq \Gamma \backslash b$ is the set of input symbols.
- $q_i \in Q$ is the initial state.
- $F \subseteq Q$ is the set of *final* or *stopping* states. This is just the set $\{s_f\}$
- $\delta :: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is our transition function[7]. This can be derived from the sets P and I. Since the cardinality is the same for both the sets a one to one match between the two can be established. Each tuple (p, i) where $p \in P$ and $i \in I$ will contribute to our δ with 2 rows: $q_i \times i \rightarrow p \times i \times R$ and $p \times \boxtimes \rightarrow q_i \times i \times R$.

  Finally the row $q_i \times n/n \rightarrow s_f \times n/n \times R$ is also included in δ.

There are an infinite number of even numbers and consquently we can build an infinite number of our custom Turing machines.

Lemma 4.1 *If the Goldbach conjecture is true, then our custom Turing machine is guaranteed to halt for at least one input*

Proof This is easy to see. If α and β are in I, then our machine should stop for the input $\alpha \boxtimes \beta \boxtimes (\alpha + \beta)$. Because there will be one pair of primes which add up to n, so consquently there should be one input such that $(\alpha + \beta) = 1$

[6]We dont compute the fractions, but leave it in the form i/p itself.

[7]Without any loss of generality we assume our machine has read-only input and a separate output tape

But this will contradict the halting problem since we cannot determine *apriori* if an arbitrary Turing machine halts on a particular input. A negative proof to the effect that the conjecture holds true only for certain values or fails for all n above a certain value, will also similarly contradict the halting problem. We are forced to conclude that ***algorithmically*** the Goldbach conjecture is undecidable. Or in other words, if in the future the conjecture is settled *axiomatically*, we still cannot find an algorithm to split an arbitrary even number into two primes.

5 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.

6 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 5). Possibly more open questions in other areas of mathematics can benefit from a reduction to the halting problem. It is an open question as to whether non-Turing computation models 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.

References

[1] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.