Information encoded in the number of closed paths through a graph vertex

Grady Ward

November 20, 2015

In searching for a polynomial time-complexity solution to the graph isomorphism problem, many researchers have focused on, *vertex invariants*, numerical properties calculated over a vertex of a graph that can be deterministically calculated, regardless of how the graph is represented or labeled. This article will aim to describe the descriptive power and information density of the $Paths(p, v)$ function, a vertex invariant that counts the number of closed paths of length p through a vertex v.

Questions to Answer

1. What is the connection between the *Paths* Function and the *Polygon* Function? Are there limitations on it?

2. How does the *Paths* Function compare against other vertex or graph invariants? Is it more or less descriptive than the chromatic polynomial of the graph?

3. Does the *Paths* Function have an upper bound for usefulness/descriptive ability? Is there some point at which we can predict its value in terms of previous values?

4. Does the *Paths* Function give enough information to deterministically recreate a graph which could generate its values? If so how many values of p and v would we need? (Note this is not the same question as the Graph Isomorphism problem: to answer the broader question of $GI? = P$ we would have to show that no two non-isomorphic graphs could lead to the same *Paths* function)

5. Does the *Paths* function allow us to place $GI \in P$?

Definitions

First, lets establish a shared understanding of the classes of graphs we are focusing on. We will be examining *undirected* graphs that do not contain multi-edges nor loops. Any graph under these three constraints can be represented as a symmetric adjacency matrix, with zeros along the diagonal, and ones or zeroes in the remaining locations denoting an edge or disjoint vertices respectively. We will regularly reference this matrix as A with no additional annotation.

The vertex invariant we are going to examine ($Paths(p, v)$) is the number of closed paths of a length p that contain the vertex v. A path is a sequence of vertices such that each vertex shares an edge with the next one. Paths can be notated by the sequence of the vertices that are visited (e.g. ABCDE or BACAB). Paths can be *closed* if the first and last vertex in their traversal sequence are the same. Note that though $ABCDE$ and $EDCBA$ describe the same edges, they are distinct paths. Even though an undirected graph doesn't have directional edges, paths maintain the order of the edges they traverse.

For this article, when we describe a path as a geometric polygon (such as a triangle, quadrilateral, pentagon, etc.), we will define to be a closed path such that the vertices of the path are non-repeating (aside from the first and last, as we assume that it is closed). Thus, $ABABA$ does not form a quadrilateral, even though it is a closed path of length four. Note that this path could equally well be described as $BABAB$, as closed paths are inherently cyclic, and do not demand a starting vertex. Additionally, we will assume that unlike paths, polygons directionality does not matter (thus the polygon ABCA is the same as the polygon ACBA).

Computability of the Paths Function

The $Paths(p, v)$ function can be computed using A. Just as the entries of A^1 represent the existence of paths of length 1 between two vertices (edges), the entries of A^p represent the number of paths of length p between any two vertices (by examining the row and column corresponding to two vertices). Thus, to find the number of closed paths of length p that contain a given vertex v, we simply need to calculate

$$Paths(p, v) = A^p[v, v]$$

Where v here is being used interchangeably with its represented position within the adjacency matrix. Thus, calculating a specific value of $Paths(p, v)$

must occur in the time it takes to exponentiate A to the power p. Though it is a well known result that matrix multiplication can be done in faster than $O(n^3)$ time, we will be using the nave assumption that matrix multiplication runs in that time in order to make the computational complexity calculations more accessible. Under that assumption, it is clear that calculating $Paths(p, v)$ will occur in $O(pv^3)$ time. This is well within polynomial time, so long as we request a polynomial number of values from $Paths(p, v)$. Thus we can consider this invariant one in the traditional vein of looking for polynomial invariants with the broad aim of solving GI in polynomial time.

1 The Paths and Polygon Functions

The $Paths(p, v)$ function has a very convenient geometric interpretation for small values of p. This interpretation relies on the *Polygon* Function: $Polygon(p, v)$ = *Number of distinct Polygons of size p that contain v as a vertex.*

For small values of p, we can express $Polygon(p, v)$ in terms of $Paths(p, v)$:

- Both invariants are uniform for all vertices for $p = 0$ and $p = 1$, as every vertex is trivially on a path to itself (so a path of size zero travels through it) and we have specified that there are no loops in the graph, thus there cannot be any edges that connect two of the same vertex. Thus for all v, $Paths(0, v) = Polygon(0, v) = 1$ and $Paths(1, v) = Polygon(1, v) = 0$.

- For $p = 2$, we know that $Paths(2, v)$ is equivalent to the number of nodes that v shares an edge with. This number is called the degree of the node, and is often referred to as the "base invariant". We also know that this must equal $Polygon(2, v)$, as a closed path of length two in a graph without loops must contain two distinct vertices.

- For $p = 3$ we know that the only way to have a closed path of length three on the type of graph that we have described is to have three different edges (and three different vertices) form a triangle. Thus, we know that $p = 3$ is related to the number of triangles that a vertex is a part of. However, since we are counting all possible combinatorial possibilities when we are describing our $Paths(p, v)$ function, we need to take orientation into account. That means that the path $ABCA$ is fundamentally distinct from the counter directional path that visits the same nodes in the reverse order: $ACBA$.

Though for $p < 3, Paths(p, v) = Polygon(p, v)$, we know that for $p = 3, \frac{Paths(3,v)}{2} = Polygon(3, v)$.

- The interpretation for $p = 4$ is less intuitive, but some simple examination yields that it is the number of quadrilaterals that contain v when one subtracts off the possibility of repeating vertices. Unlike $p \leq 3$ and below, calculating $Polygon(4, v)$ simply using values of $Paths(x, v)$ is not possible unless you have the degrees of adjacent nodes to v (information not directly encoded into the paths function). Using external information you could calculate the number of quadrilaterals that v is a part of. Here we see the beginning of a divergence, of a differentiation between the polygon driven interpretation of $Paths(p, v)$, and the more "combinatorial" reality.

Examination beyond $p = 4$ yields a similar picture: at each point in time, you need additional information (an increasing amount of it, too) in order to successfully calculate $Polygons(p, v)$, even given unrestricted access to $Paths(p, v$ for all p and v. A critical reader will have already come to the conclusion of this section, rooted in complexity theory. If our graph is of size V, we know that the calculation of $Polygon(V, v)$ is the same for all v in the graph, as each vertex must be included exactly once on a closed path of length V. This type of path is also called a *Hamiltonian Cycle*, and its computation is known to be in NP-Complete. If we hope that $Paths$ offers easily computable access to lots of information about our graph, its divergence from $Polygon(p, v)$ and $Paths(p, v)$ is critical, as it thankfully does not imply that computing values of $Paths$ is in the same complexity class as apparently intractable problems.

2 Information of Paths and the Chromatic Polynomial

One of the most discriminatory polynomial time algorithms for ruling out graph isomorphism is the Chromatic Polynomial of the graph's adjacency matrix A. It is a well proven result that shuffling the order of the vertices in the representation of a given graph as an Adjacency Matrix does not change the eigenvalues of the matrix (thus the eigenvalues, as a multiset, form a graph invariant), and thus the chromatic polynomial is a graph invariant. In practice, it is a highly discriminating graph invariant, as a very small proportion of graphs are "cospectral" (having the same chromatic polynomial) and not isomorphic.

In seeking out a relationship between the chromatic polynomial and the $Paths$ function, it helps to remember the helpful property of eigenvalues:

$$E(A) = \{\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_V\} \rightarrow E(A^P) = \{\lambda_1^P, \lambda_2^P, \lambda_3^P, \ldots, \lambda_V^P\}$$

Additionally, since we know that our original matrix A is diagonalizable, positive, and symmetric, we know that all of the eigenvalues are real and non-negative. Finally, we know that the sum of the diagonal of any matrix is the sum of its eigenvectors. Thus, using the $Paths$ function to determine the entries of the diagonal, we know that for any adjacency matrix A,

$$\lambda_1 + \lambda_2 + \lambda_3 + \cdots + \lambda_V = \sum_{i=0}^{V} Paths(1, i)$$

Or more generally (leveraging our knowledge about the eigenvalues of A^P:

$$\sum_{i=0}^{V} \lambda_i^P = \sum_{i=0}^{V} Paths(P, i)$$

This means that we can create V nonparallel (as not divisible in the ring $R[x]$) equations, each containing λ_1 to λ_V as distinct variables. Additionally, since we have the guarantee that our eigenvalues are all non-negative, we know that we can assuredly solve any given equation in terms of one of the λ's. If we then use these substitutions, and substitute out all but one of the variables, what we will get is a continuous function whose zeros are well defined and positive.

Given the equation, each one of the zeros is an eigenvalue, assign that to a particular eigenvalue, and reduce our equations using that assumed value. This kind of procedure is necessary because our system of equations has variables whose relationships are inherently non-distinct. In other words, we cannot make any differentiation between $lambda_1$ and $lambda_2$ based on the relationships of the equations, but if we are given a choice for either, it will lead to a valid conclusion. This is by virtue of the fact that our substitutions are "condition-less", and we don't need to worry about any domains of substitution.

From a physical perspective, these equations can be described as surfaces in V dimensions, whose intersection we are interested in isolating for. Graphing the equation $x^3 + y^3 + z^3 = \kappa_3$ yields an interesting quasi planar sheet which becomes highly regular at large scales. The equation $x^2 + y^2 + z^2 = \kappa_2$ describes a sphere with the radius $\sqrt{\kappa_2}$, and the equation $x + y + z = \kappa_1$ describes a plane. Each of these functions is axis symmetric, so their intersections are also axis symmetric. The intersection of all of the equations yields a

set of points (or none), which represent the potential values for our lambda's, which are axis invariant. *** HOW DO WE KNOW THAT THESE ARE UNIQUE?!?! WHY NOT MULTIPLE LAMBDA POSSIBILITIES?!?! HOW DO WE GO ABOUT PROVING THIS? THIS IS A SIMILAR QUESTION TO GI GENERALLY... ***

Thus, given the function $Paths(p, v)$, we could calculate the eigenvalues of the matrix A using $Paths(p, v)$ restricted to $1 <= p <= V$. This is superb, as it shows that the $Paths$ function encodes at least as much information as the chromatic polynomial, and that any graphs which produce the same $Paths$ function up to (p == V) are cospectral. A quick verification of the base non-isomorphic cospectral graphs (the butterfly and the box) show that they produce *different* values of the paths function, showing that the $Paths$ function carries *more* information than graph spectrum does, even though both are computed in polynomial time.

3 The Information Bounds of the Paths Function

4 Reconstruction With Fixed $Paths$ Values

If we are given full access to the $Paths$ function, could we create a graph that would produce that function? Note that this question is fundamentally distinct from the one which asks if we could reconstruct the *only* graph that could generate that Graph Function. That is a separate question, equivalent to the question of if $GI = P$ but is not the focus of this section.

The answer to this question, fortunately, is yes. Over the next few paragraphs, I am going to detail a method of constructing such a graph by first constructing constituent integer-valued equations which describe the interrelationships between edges, then transform the integer edge equations into equally specific boolean sentences, which by their definition will always hold. The final step is then to transform each of these sentences into CNF (Codd Normal Form), and use a satisfiability solver to find a solution to them.

Before we begin, lets examine three important points:

- Our reconstruction algorithm is going to run in exponential time, and that is okay. We have already demonstrated that our invariant, the paths function, is calculable in polynomial time. The fact that we can reconstruct a valid graph from the paths function is about beginning to establish the invertible nature of the paths function, and the time it

takes to perform an inverse of our primary operation tells us nothing about the computational complexity of that operation.

- The equations in this section for all but the most trivial of graphs take up an enormous amount of space, and as such, it is not recommended that this reconstruction technique be used in practice. Realistically, a brute force search would likely yield better solutions (if one were trying to find a graph fitting the paths function), but the reader should try to convince themselves that this kind of problem would terminate, and would yield a valid solution, given a valid paths function.
-

We know that the diagonals of A^p yield the path function. We also know that A is comprised of $(v * (v-1))/2$ boolean variables, and all entries of A^p must be some combination of the variables in the original adjacency matrix. We will refer to these variables as the x_i's. Generally, we will arrange them in the pattern shown in the graph below, and will number them starting at 1.

The x_i's also have the helpful property that $\forall k \geq 1$, $x_i^k = x_i$. This stems from the fact that each one of the x_i's is either zero or one, the two solutions to the aforementioned equation. This allows us to reduce any polynomial degree in our resulting equations down to one.

I have been discussing these "equations" quite a bit; lets formally define them. We assume that we are given the $Paths$ function for a graph, and we will call $Paths(p, v) = k_{p,v}$ to simplify our work. In each equation, we will set the paths function for a specific v and p equal to the symbolic representation of the exponentiated adjacency matrix (which will solely be in terms expressed by the x_i's). We will refer to this specific equation as Equation p.v for v in the range $[1, V]$ and p in the range $[1, \text{inf})$. For each v and p in their respective ranges, our equation p.v is:

$$k_{p,v} = A^p[v, v]$$

Some example Equations are shown below for a five node graph (2.1, 3.1, 4.1). Note the rapid expansion in the number and complexity of the terms. Also note that we don't need any polynomials over one variable, so we have collapsed them.

$$k_{2,1} = x_1 + x_2 + x_3 + x_4$$

$$k_{3,1} = 2x_1x_2x_5 + 2x_1x_3x_6 + 2x_1x_4x_7 + 2x_2x_3x_8 + 2x_2x_4x_9 + 2x_3x_4x_{10}$$

$$k_{4,1} = x_1 + x_2 + x_3 + x_4 + 2x_1x_2 + 2x_1x_3 + 2x_1x_4 + 2x_2x_3 + x_1x_5 + 2x_2x_4 + x_1x_6 + x_2x_5 \\ +2x_3x_4 + x_1x_7 + x_3x_6 + x_2x_8 + x_2x_9 + x_3x_8 + x_4x_7 + x_3x_{10} + x_4x_9 + x_4x_{10} \\ +2x_2x_3x_5x_6 + 2x_1x_2x_6x_8 + 2x_1x_3x_5x_8 + 2x_2x_4x_5x_7 + 2x_1x_2x_7x_9 + 2x_1x_4x_5x_9 \\ +2x_3x_4x_6x_7 + 2x_1x_3x_7x_{10} + 2x_1x_4x_6x_{10} + 2x_2x_3x_9x_{10} + 2x_2x_4x_8x_{10} + 2x_3x_4x_8x_9 \tag{1}$$

An interesting aspect of these equations actually has a cool natural cause. Notice that in equation 2.1 and equation 4.1, both have linear terms of four variables. (Those happen to be the four variables in the row that we chose, row 1, but if we chose row 4, we would have gotten the variables in that row). A nice property that arises out of these linear variables is that if we were to solve for the variable x_1 from equation 4.1, we would get an expression with the following denominator:

$$2x_2+2x_3+2x_4+x_5+x_6+x_7+2x_2x_6x_8+2x_3x_5x_8+2x_2x_7x_9+2x_4x_5x_9+2x_3x_7x_{10}+2x_4x_6x_{10}+1$$

This is of particular interest, because we know that each one of the terms in this statement has to be positive, as $\forall i \in v, x_i \in \{0, 1\}$. Thus, there is no possible valid input of x_i's which results in this denominator being zero, and our substitution is thus universally valid. That is really important because it means that we might be able to do the same thing for other variables, and potentially come up with a system of equations that fully describes the interactions of the vertices within our graph.

Lets explore this notion further. Lets say that we have an equation that describes the interactions of the binary relations in such a way that we can express the equality for of the variable x_i solely in terms of the other variables such that

$$x_i = \frac{N}{D + z}$$

Where $z \geq 1$ and N and D can be any number of positive expressions that are comprised of terms of the addition and multiplication of variables in the set $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{(v(v-1))/2}\}$. Since we know that $x_i \in 0, 1$, we know that any positive term comprised of the multiplication and addition of these variables must be greater than or equal to zero. Thus, we know the denominator of the equation for x_i must be greater than or equal to one ($D + z \geq 1$. This is a helpful result, as it means that our substitution is valid, and we know that the equality holds, because our division of both sides by $D + z$ does not risk dividing by zero. We will call this a *valid substitution* for x_i over the equation for which it is generated (e.g. 2.5).

It turns out that there is a clever way to use a *valid substitution* to construct a graph which would generate the paths functions that generated

the equations which generated the valid substitutions. Since we know that the denominator of a valid substitution is non-zero, we know the substitution is valid. We also happen to know that the variable we are solving for is either equal to zero or one. Thus, if the numerator (N) of our substitution is equal to zero, then we know that x_i is equal to zero. Otherwise, we know that x_i must be equal to one (for all other values are not in its domain). If we allow ourselves to do an informal conversion to predicate logic, we can transform an equation of the form:

$$x_i = \frac{N}{D + z}$$

$$(x_i = 0) \text{ iff } (N = 0)$$

$$\neg(x_i) \text{ iff } (N = 0)$$

$$x_i \leftrightarrow \neg(N = 0)$$

Expressing that N, a summation of terms over the x_i's, is equal to zero, is actually quite a simple construct. We imagine that each of the terms in N has a positive, integer valued weight that corresponds to its coefficient. If N contains any integer valued negatives (which it does in real applications), we will call this the *target*. Given this (and the terms and their weights as corresponding lists), we can generate a simple procedure for generating a boolean statement that contains the same information as our equation $N = 0$.

```
exactlyKTrue(clauses, weights, target):
        if (target == 0) then
                return negatedConjuction(clauses)
        clause = clauses.pop()
        weight = weights.pop()
        caseF = exactlyKTrue(clauses, weights, target)
        newTarget = target - weight
        caseT = exactlyKTrue(clauses, weights, newTarget)
        return (clause & caseT) | (!clause & caseF))
```

Though this code doesn't totally cover all edge cases, it should convince the reader that there is an appropriate transform between N and a boolean expression of the x_i's which

5 Paths, Uniqueness, and GI

If we restrict our graph for a moment to a specific type of graph,

6 Cardinality

It should be immediately clear to someone who thus far understands the paths function that its cardinality can be used to prove that it solves the Graph Isomorphism problem. First lets notice that the number of distinct values for the path function over V and P is bounded by the number of non-isomorphic graphs of size V. Why is that true? Intuitively, the paths function cannot differ over isomorphic graphs, so long as we provide some kind of systematic way to compare if two paths functions are the same (say by sorting their component vertex vectors), there cannot be more paths functions than there are distinct graphs.

If this bound wasn't simply an inequality, but a equality, stating that the number of paths functions over a certain number of vertices is the same as the number of non-isomorphic graphs over the same number of vertices, we would have proved that the Graph Isomorphism problem is in P, by presenting a polynomial time algorithm (the generation and vector sorting of the paths Values) which would allow complete vertex discrimination, and when paired with a simple deduction algorithm, could deterministically generate isomorphisms (if they exist) in polynomial time.

Before pursuing the topic of this paper in detail, I wanted to make sure it was a possibility that it is correct. I wrote a simple program in Java which generates and stores the results of the Paths function for all possible adjacency matrices of a certain size, and counts the number of non-isomorphic graphs for graphs over a certain number of vertices (a fixed property of the program). What I found was that the cardinality of the paths functions over V vertices matched up exactly with the sequence of Non-isomorphic graphs up to eight vertices (given by A000088 in the On-Line Encyclopedia of Integer Sequences (OEIS)): 1, 1, 2, 4, 11, 34, 156, 1044.

Computationally exploring higher dimensions would require signif