Maple Computation of the Ollivier-Ricci Curvature of Graphs

Kaylee Weatherspoon

Experimental Math 2025

1 Background

Ricci curvature, originally defined in the setting of differential geometry, has recently attracted significant attention in the areas of graph theory, network science, and machine learning[4]. Departing from the original setting of Ricci Curvature, the *Ollivier-Ricci Curvature on Graphs* adapts the concept to discrete spaces, enabling the analysis of geometric and structural properties of networks.

Whereas the original definition of Ricci curvature refers to the inherent geometric properties of a Riemannian manifold, the Ollivier-Ricci Curvature on Graphs assigns a probability distribution to each vertex and takes the *Wasserstein/Earth Mover's metric* as its notion of distance ([3]). Put simply, the probability distribution assigned to a vertex v encodes the likelihood of moving to any one of the neighbors of v. In the context of "earth-moving," it represents the fraction of the total mass of the neighbors of v which is assigned to each of the individual neighbors. (*Note:* From this point forward, *Ollivier-Ricci Curvature* refers to the Ollivier-Ricci Curvature on Graphs.) In addition to this, we permit edge weights, which often represent cost or distance. Following [2], the Ollivier-Ricci curvature of an edge xy is

$$\kappa(x, y) := 1 - \frac{W_1(\mu_x, \mu_y)}{d(x, y)},$$

where $W_1(\mu_x, \mu_y)$ is the Wasserstein-1 metric taking the probability distributions on μ_x and μ_y as its inputs and $d(x, y)$ is the weight of the edge xy.

1.1 Examples:

1.1.1 Cycles:

The Ollivier-Ricci curvature of C_5: Assuming each vertex is assigned a uniform distribution, it suffices to compute the Ollivier-Ricci Curvature of an arbitrary edge, say $\{v_1, v_2\}$.

1. **Establishing the Distributions:** We chose to use a uniform distribution. Each vertex of C_5 has exactly two neighbors, so the distribution vector assigned to each vertex is $(1/2, 1/2)$.

2. **Cost Matrix:** The cost matrix represents the graph distance from the neighbors of v_1, which are v_2 and v_5, to the neighbors of v_2, which are v_1 and v_3.

	to v_1	to v_3
from v_2	1	1
from v_5	1	2

Table 1: Cost Matrix for the edge $\{v_1, v_2\}$ in C_5

3. **Finding the Wasserstein Distance:** This amounts to solving an LP. We want to move mass from μ_1, the distribution vector of v_1, to μ_2, the distribution vector of v_2.

	to v_1	to v_3	Total Supply
from v_2	a	b	1/2
from v_5	c	d	1/2
Total Demand	1/2	1/2	

Table 2: Transport Plan for $\{v_1, v_2\}$ in C_5

   We seek to minimize $C = 1 \cdot a + 1 \cdot b + 1 \cdot c + 2 \cdot d$, under the constraints that $a + b = 1/2$, $c + d = 1/2$, $a + c = 1/2$, $b + d = 1/2$. The solution to this LP is $a = 0, b = 1/2, c = 1/2, d = 0$, so the minimal cost is $C = 1$.

4. **Computing Edge Curvature:**

$$\kappa(v_1, v_2) = 1 - \frac{W_1(\mu_1, \mu_2)}{d(v_1, v_2)} = 1 - 1 = 0$$

The edge curvatures for all edges is 0, so the Ollivier-Ricci curvature of C_n, which is the average over all edge curvatures, is 0. This is an opportune time for the interested reader to note how a weighted edge ($d(v_1, v_2) \neq 1$) would affect the curvature.

1.1.2 Complete Graphs:

Given uniform probability distributions at each vertex, the Ollivier-Ricci Curvature of a complete graph is easy to compute because of its substantial vertex transitivity. With this in mind, we compute the Ollivier-Ricci Curvature for a general complete graph, rather than a complete graph of fixed order, below.

1. **Establishing the Distributions:** In K_n, the uniform distribution vector has length $n-1$ and entries all equal to $(n-1)^{-1}$.

2. **Conceptualizing the Cost Matrix:** The neighborhoods of any two vertices u, v in K_n have substantial overlap. The only non-overlap in the neighborhoods is the two endvertices themselves: the only neighbor of u which is not also a neighbor of v is v itself. The Wasserstein distance between u and v is therefore $1/(n-1)$.

3. **Computing Edge Curvature:** Assuming the edges are all unweighted,

$$\kappa(u, v) = 1 - \frac{(n-1)^{-1}}{d(u, v)} = 1 - \frac{1}{n-1}.$$

4. **Graph Curvature:** Since all the edge curvatures are the same, the Ollivier-Ricci Curvature of the graph is simply

$$1 - \frac{1}{n-1}.$$

   However, if one of the edges had weight 100, for example, the curvature of that edge would be $1-(100n-100)^{-1}$. In this case, the Ollivier-Ricci curvature of the graph would be

$$\frac{\left(\binom{n}{2} - 1\right)\left(1 - \frac{1}{n-1} + \left[1 - \frac{1}{100n-100}\right]\right)}{\binom{n}{2}}.$$

1.1.3 A Negatively Curved Edge

Consider the graph G with 5 vertices and edge set

$$\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_5\}, \{v_5, v_4\}\}.$$

We compute the curvature of the edge $\{1, 3\}$ by hand and leave the remaining edges as an exercise to the interested reader.

1. **Determining the Distribution:** Assuming a uniform distribution, the distribution vector at v_3 is $(1/3, 1/3, 1/3)$ corresponding to the three neighbors of v_3. The distribution vector at v_5 is $(1/2, 1/2)$.

2. **Constructing the Cost Matrix:**

	to v_3	to v_4
from v_1	1	3
from v_2	1	3
from v_5	1	1

Table 3: Cost Matrix for the edge $\{v_3, v_5\}$ in G

3. **Finding the Wasserstein Distance:**

	to v_3	to v_4	Total Supply
from v_1	a	b	1/3
from v_2	c	d	1/3
from v_5	e	f	1/3
Total Demand	1/2	1/2	

Table 4: Transport Plan for edge $\{v_3, v_5\}$ in G

   We seek to minimize $C = 1a + 3b + 1c + 3d + 1e + 1f$, subject to the following constraints: $a + b = 1/3, c + d = 1/3, e + f = 1/3, a + c + e = 1/2, b + d + f = 1/2$. The solution to this LP is $a = 1/3, b = 0, c = 1/6, d = 1/6, e = 0, f = 1/3$, which corresponds to $C = 4/3$.

4. **Computing the Curvature:** The curvature of the unweighted edge v_3, v_5 in G is therefore $1 - 4/3 = -1/3$.

Note: In the accompanying Maple file, you can verify this example by running `EdgeCurvsList(G)` where

$$\texttt{G=Graph}([1,2,3,4,5],\{\{1,2\},\{1,3\},\{2,3\},\{3,5\},\{4,5\}\}).$$

1.2 Community Detection

We can, of course, compute the Ollivier-Ricci curvature of graphs much larger and/or denser than those shown above. In practice, graphs of interest are quite large; for example, a protein-protein interaction network can have thousands of nodes, as could the network representing researchers and their coauthors. Methods relying on Ollivier-Ricci curvature have generated substantial interest in the area of community detection, where these methods often outperform standard connectivity- and eigenvalue-based techniques (see [4]).

The connection between Ollivier-Ricci Curvature and Community Detection is as follows: edges with negative Ollivier-Ricci Curvature represent bridges between communities, and edges with positive curvature tend to hold a community together. The community detection algorithm given in [4] iteratively removes the negatively curved edge of greatest magnitude until no negatively curved edges remain, leaving a set of likely communities. We implement this algorithm at the end of the following section.

2 Maple Code

There are three key pieces of information involved in the computation of Ricci Curvature: a graph, edge weights, and the distributions associated with each vertex. In the code that follows, we handle each of these components, while giving increasing flexibility to the user in terms of input parameters.

We begin with the case of Ollivier-Ricci Curvature for unweighted graphs with uniform distributions assigned to each vertex. The following is a simple procedure to create the uniform distribution vector and list of Neighbors for a vertex x in a graph G. Graphs are to be given as a standard Maple graph object unless otherwise noted.

```
distributionx:=proc(G,x) local Neis, n, distvec, i:
    Neis:=Neighbors(G,x):
    n:=nops(Neis):
    distvec:= Vector(n):
```

```
    distvec:=Vector([seq(1/n, i=1..n)]):
return([Neis, distvec]):
end:
```

Given two vertices x, y, we observe that the associated distributions μ_x, μ_y are supported on the set of neighbors of x and y, respectively. We denote these sets by $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$, respectively. Let $d(x_i, y_i)$ denote the graph shortest path distance between x_i and y_i. The cost matrix procedure in OlliRicci.txt inputs G and two vertices x, y and outputs the $n \times m$ matrix C such that $C(i, j) = d(x_i, y_j)$. We refer to this matrix in the procedure `wasserstein`, in which we compute the Wasserstein/Earth Mover's distance using the following LP (see [1]):

$$\textbf{minimize} \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} z_{ij} :$$

$$\textbf{subject to} \sum_{j=1}^{m} z_{ij} = \text{mass at } i \text{ in } \mu_x$$

$$\sum_{i=1}^{n} z_{ij} = \text{mass at } j \text{ in} \mu_x$$

$$z_{ij} \geq 0 \text{ for all i, j.}$$

The implementation is as follows:

```
wasserstein:=proc(x, y, G) local n,m, obj, distx, disty,
Neisx, Neisy, C, flovars, constr,
i, j, flovarslist, LPsol, wassersteindist:
distx:=distributionx(G, x):
disty:=distributionx(G, y):

Neisx:=distx[1]:
Neisy:=disty[1]:
n:=nops(Neisx):
m:=nops(Neisy):
```

```
    C:=costmatrix(G, x, y):

    if n=0 or m=0 then return("ERROR: isolated vertex"):
    fi:

    flovars:=Matrix(n,m,(i,j) -> Z[i,j]):
    obj:=add(add(C[i,j]*flovars[i,j], j=1..m),i=1..n):
    constr:={}:

 #outflow contstraints
    for i from 1 to n do
    constr:=constr union {add(flovars[i,j], j=1..m)=distx[2][i]}:
    od:
 #inflow constraints
    for j from 1 to m do
    constr:=constr union {add(flovars[i,j], i=1..n)=disty[2][j]}:
    od:
#nonnegativity
    for i from 1 to n do
        for j from 1 to m do
        constr:=constr union {flovars[i,j]>=0}:
        od:
    od:
#formatting for LP solver
    flovarslist:=[seq(seq(flovars[i,j], j=1..m), i=1..n)]:

    LPsol:=LPSolve(obj, constr):
    wassersteindist:=LPsol[1]:
    RETURN(wassersteindist);

end:
```

We also include in OlliRicci.txt the procedure `wassermanualdist`, whose inputs are `(x, distx, y, disty, G)`, where `x` and `y` are vertices, `G` is a graph, and, distributions `distx` and `disty` as lists.

Synthesizing the aforementioned procedures, `OlliRicciEdge` computes the Ollivier-Ricci curvature of an edge. This procedure allows the user to input the weight of the edge, allowing for use with weighted graphs. Extend-

ing this, `OlliRicciEdgeD` calls `wassermanualdist`, accommodating graphs with non-uniform distribution vectors.

Finally, the collection of procedures `OlliRicci`, `OlliRicciW`, and `OlliRicciWD`, compute the Ollivier-Ricci Curvature of unweighted graphs with uniform distributions, weighted graphs with uniform distributions, and weighted graphs with non-uniform manually entered distributions. In the case of weighted graphs, we depart from the standard Maple graph notation and use the format $G = E$, where each $e \in E$ is of the form `{endvertex1, endvertex2, weight}`. For convenience, `OlliRicciWD` is included below:

```
OlliRicciWD:=proc(G1, D) local G, verts, wedges, edgecurvs,
    edges,i, f, e:
    verts:=[seq(i, i=1..G1[1])]:
    wedges:=G1[2]:
    edgecurvs:=[]:
#need to process G
    edges:={}:
    for f in wedges do
        edges:= edges union {{f[1], f[2]}}:
    od:
    G:=Graph(verts, edges):
#build list of weighted curvatures
    for e in wedges do
        dist(e[1]):=[Neighbors(G, e[1]), Vector(D[e[1]])]:
        print(e, e[1], dist(e[1]));
        dist(e[2]):=[Neighbors(G, e[2]), Vector(D[e[2]])]:
        print(e, e[2], dist(e[2]));
        edgecurvs:=[op(edgecurvs), OlliRicciEdgeMD(G, e[1],
                            dist(e[1]), e[2], dist(e[2]), e[3])]:
    od:
    return(add(edgecurvs)/nops(wedges)):
end:
```

As discussed in 1 community detection is an application of Ricci Curvature that has garnered recent interest. The authors of [4] supply an algorithm for detecting communities in graphs using Ricci Curvature. Their implementation scales well (much better than what we show below) and has yielded novel insights for real-world problems. In the following, we implement their algorithm in Maple.

The first step involves "pruning" the given graph by removing all edges with negative Ricci curvatures. We recall that negative curvatures tend to indicate FILL IN. It is worth noting that this is the slowest step in our implementation.

```
Prune:=proc(G) local Gcopy, ecurvs, x, y, edge,
                minEdge, minCurv, aff, u,v, Neis, e:
Gcopy:=CopyGraph(G):
ecurvs:=table():
for edge in Edges(Gcopy) do
    x:=edge[1]:
    y:=edge[2]:
    ecurvs[[x,y]]:=OlliRicciEdge(Gcopy, x, y, 1):
od:

while true do
    minEdge:=NULL:
    minCurv:=0:

    for edge in Edges(Gcopy) do
        x:=edge[1]:
        y:=edge[2]:
        if assigned(ecurvs[[x,y]]) and ecurvs[[x,y]]< minCurv then
            minCurv:=ecurvs[[x,y]]:
            minEdge:=[x,y]:
        fi:
    od:

    if minEdge =NULL then break: fi:

    x:=minEdge[1]:
    y:=minEdge[2]:
    RemoveEdge(Gcopy, x, y):

    ecurvs[[x,y]]:='undef':

    #optimizing: recomputing curvatures only for
    #affected edges, not for all except removed
```

```
    for u in [x,y] do
        Neis:=Neighbors(Gcopy, u):
        for v in Neis do
            if v<u then
                e:=[v,u]:
            else
                e:=[u,v]:
            fi:
            ecurvs[[op(e)]]:=OlliRicciEdge(Gcopy, e[1], e[2], 1):
        od:
    od:
od:
RETURN(Gcopy):
end:
```

After the graph is "pruned" we consider each of the remaining connected components as communities. In the Maple package accompanying this paper, the community detection procedure `RicciCommDetect` takes two arguments, `(G,s)`, where `s` is the smallest community size we consider.

3 Future Work

As with any computer program, there is room for optimization. Particularly, the shortest path computations in the cost matrix could be optimized for large graphs potentially using a breadth-first search or other algorithms. The current computation is slow for dense graphs. In the table below, we show the computing time on a standard laptop for the unweighted Ollivier-Ricci Curvature of each of the complete graphs on 2 to 20 vertices:

2	3	4	5	6	7	8	9	10
0.001	0.003	0.005	0.012	0.025	0.044	0.081	0.238	0.241

11	12	13	14	15	16	17	18	19	20
1.41	1.77	3.56	5.57	8.63	13.0	19.1	25.3	41.5	52.4

Table 5: Computation times for OlliRicci for Complete Graphs

Additionally, for any graph with negative edge curvatures, the `Prune` step is very slow. Optimizing that step would make the community detection

algorithm more viable.

References

[1] Nazanin Azarhooshang, Prithviraj Sengupta, and Bhaskar DasGupta. "A review of and some results for Ollivier–Ricci network curvature". In: *Mathematics* 8.9 (2020), p. 1416.

[2] Yann Ollivier. "Ricci curvature of Markov chains on metric spaces". In: *Journal of Functional Analysis* 256.3 (2009), pp. 810–864.

[3] Yann Ollivier. "Ricci curvature of metric spaces". In: *Comptes Rendus Mathematique* 345.11 (2007), pp. 643–646.

[4] Jayson Sia, Edmond Jonckheere, and Paul Bogdan. "Ollivier-ricci curvature-based method to community detection in complex networks". In: *Scientific reports* 9.1 (2019), p. 9800.