axioms

MDPI

Article

Bakry–Émery Curvature Sharpness and Curvature Flow in Finite Weighted Graphs: Implementation

David Cushing [1], Supanat Kamtue [2], Shiping Liu [3], Florentin Münch [4], Norbert Peyerimhoff [5,*] and Ben Snodgrass [5]

1 Department of Mathematics, The University of Manchester, Manchester M13 9PL, UK; david.cushing@manchester.ac.uk
2 Yau Mathematical Sciences Center, Tsinghua University, Beijing 100190, China; skamtue@tsinghua.edu.cn
3 School of Mathematical Sciences and CAS Wu Wen-Tsun Key Laboratory of Mathematics, University of Science and Technology of China, Hefei 230026, China; spliu@ustc.edu.cn
4 Max Planck Institute for Mathematics in the Sciences, 04103 Leipzig, Germany; muench@mis.mpg.de
5 Department of Mathematical Sciences, Durham University, Durham DH1 3LE, UK; ben-snodgrass@outlook.com
* Correspondence: norbert.peyerimhoff@durham.ac.uk

Abstract: In this paper, we discuss the implementation of a curvature flow on weighted graphs based on the Bakry–Émery calculus. This flow can be adapted to preserve the Markovian property and its limits as time goes to infinity turn out to be curvature sharp weighted graphs. After reviewing some of the main results of the corresponding paper concerned with the theoretical aspects, we present various examples (random graphs, paths, cycles, complete graphs, wedge sums and Cartesian products of complete graphs, and hypercubes) and exhibit various properties of this flow. One particular aspect of our investigations is asymptotic stability and instability of curvature flow equilibria. The paper ends with a description of the Python functions and routines freely available in an ancillary file on arXiv or via github. We hope that the explanations of the Python implementation via examples will help users to carry out their own curvature flow experiments.

Keywords: curvature flow; Bakry-Émery calculus; weighted graphs; stability

MSC: 05C21; 05C50; 51K10; 53C21; 53E20

Citation: Cushing, D.; Kamtue, S.; Liu, S.; Münch, F.; Peyerimhoff, N.; Snodgrass, B. Bakry–Émery Curvature Sharpness and Curvature Flow in Finite Weighted Graphs: Implementation. *Axioms* **2023**, *12*, 577. https://doi.org/10.3390/axioms12060577

Academic Editors: Emil Saucan and David Xianfeng Gu

Received: 19 April 2023
Revised: 3 June 2023
Accepted: 7 June 2023
Published: 11 June 2023

1. Introduction

This paper is concerned with computational aspects of a curvature flow on weighted graphs based on the Bakry–Émery calculus. This curvature flow was originally introduced in our paper [1]. The current paper focuses on properties for various concrete examples of graphs as well as stability investigations of the flow limits. Many observations in this paper are gained with the help of our curvature flow software, which is freely available as an ancillary file of [2] or via github at https://github.com/georgestagg/graph-curvature-server (accessed on 6 June 2023). A summary of our findings is presented in Section 4.

A *weighted graph* in this paper is a finite simple mixed combinatorial graph $G = (V, E)$ with vertex set V and edge set $E = E^1 \cup E^2$ of one- and two-sided edges, together with a weighting scheme of transition rates $p_{xy} \geq 0$ for $x, y \in V$ (which can be represented by a generally non-symmetric matrix P after an enumeration of the vertices). Transition rates p_{xy} can only be positive if $x = y$ or if there is a one- or two-sided edge from x to y. One-sided edges are denoted by ordered pairs $(x, y) \in E^1 \subset V^2$, and two-sided edges are denoted by sets $\{x, y\} \in E^2$. One- or two-sided edges $(x, y) \in E^1$ or $\{x, y\} \in E^2$ with vanishing transition rates $p_{xy} = 0$ are called *degenerate* and a weighted graph (G, P) is called *non-degenerate* if it does not have degenerate edges. A weighted graph (G, P) is called *Markovian*, if we have $\sum_{y \in V} p_{xy} = 1$ for all $x \in V$. In this case, P is a stochastic matrix and

we can view the transition rates p_{xy} as transition probabilities of a lazy random walk (with non-zero laziness if there exists a vertex $x \in V$ with $p_{xx} > 0$). Our curvature flow does not affect the underlying combinatorial graph G, but it changes the weighting scheme. We will focus on a version of our flow that preserves the Markovian property. In other words, starting with an initial Markovian weighted graph (G, P_0), this flow will provide a family of Markovian weighting schemes $\{P(t)\}_{t\geq 0}$ with $P(0) = P_0$, depending on the continuous time parameter t.

Before we introduce our curvature flow, we need to briefly discuss the relevant Bakry–Émery curvature background. This curvature notion is based on the weighted Laplacian $\Delta = \Delta_P$, acting on functions $f : V \to \mathbb{R}$ as follows:

$$\Delta_P f(x) = \sum_{y \in V} p_{xy}(f(y) - f(x)).$$

The Laplacian gives rise to the following symmetric bilinear "carré du champ operators" Γ and Γ_2:

$$\begin{aligned} 2\Gamma(f,g) &= \Delta(fg) - f\Delta g - g\Delta f, \\ 2\Gamma_2(f,g) &= \Delta\Gamma(f,g) - \Gamma(f,\Delta g) - \Gamma(g,\Delta f). \end{aligned}$$

1.1. Bakry–Émery Curvature and Curvature Sharpness

Bakry–Émery curvature depends on a dimension parameter N and is well-defined at every vertex $x \in V$ that is *not isolated*, that is, there exists another vertex $y \in V$ with $p_{xy} > 0$. For isolated vertices, there is some ambiguity on how to define their curvature, and we decided to assign to such a vertex the curvature value 0. (Another natural choice of curvature for an isolated vertex $x \in V$ would be $K_N(x) = \infty$ for all $N \in (0, \infty]$. An argument for that choice is that an isolated vertex can be viewed as a discrete analogue of a limit of round spheres with radii shrinking to 0, whose curvatures would diverge to infinity.) The definition of Bakry–Émery curvature reads as follows:

Definition 1 (Bakry–Émery curvature)**.** *The Bakry–Émery curvature of a non-isolated vertex $x \in V$ for a fixed dimension $N \in (0, \infty]$ is the supremum of all values $K \in \mathbb{R}$, satisfying the curvature-dimension inequality*

$$\Gamma_2(f)(x) \geq \frac{1}{N}(\Delta f(x))^2 + K\Gamma(f)(x) \tag{1}$$

for all functions $f : V \to \mathbb{R}$. We use the simplified notation $\Gamma(f) = \Gamma(f,f)$ and $\Gamma_2(f) = \Gamma_2(f,f)$. We denote the curvature at $x \in V$ by $K_N(x) = K_{P,N}(x)$. If $x \in V$ is isolated, that is, we have $p_{xy} = 0$ for all $y \in V \setminus \{x\}$, we set $K_N(x) = K_{P,N}(x) = 0$ for all $N \in (0, \infty]$.

This curvature notion is motivated by Bochner's identity (see, e.g., [3] (Prop. 4.15)), a fundamental pointwise formula in the smooth setting of n-dimensional Riemannian manifolds involving gradients, Laplacians, Hessians, and Ricci curvature. It was introduced for the smooth setting in [4]. The curvature was then reintroduced several times in the setting of graphs, see [5–7]. For further research about Bakry–Émery curvature on finite graphs, see, e.g., [8–19].

By the definition, the inequality

$$\Gamma_2(f)(x) \geq \frac{1}{N}(\Delta f(x))^2 + K_N(x)\,\Gamma(f)(x) \tag{2}$$

holds for every function f, and therefore, also holds for the combinatorial distance function $d(x,\cdot)$. Here, $d(x,y)$ is the length of a shortest directed path from x to y (if there is no such path, we set $d(x,y) = \infty$). If we have equality at x in (2) for this particular function $f = d(x,\cdot)$, we say that the vertex $x \in V$ is N-curvature sharp. Curvature sharpness

will be particularly important in our considerations. Curvature sharpness was originally introduced in [20] (Definition 1.4). The (equivalent) definition given in this paper is inspired by [21] (Proof of Theorem 1.2). For more details about relations between different curvature sharpness definitions see Section 3 of [1].

Definition 2 (Curvature sharpness)**.** *Let (G, P) be a weighted graph and $N \in (0, \infty]$. A vertex $x \in V$ is called N-curvature sharp if we have*

$$\Gamma_2(f)(x) = \frac{1}{N}(\Delta f(x))^2 + K_N(x)\,\Gamma(f)(x) \tag{3}$$

for the distance function $f = d(x, \cdot)$. Moreover, a vertex $x \in V$ is curvature sharp if it is curvature sharp for some dimension $N \in (0, \infty]$. A weighted graph (G, P) is called curvature sharp, if every vertex of G is curvature sharp.

Note that each function $f : V \to \mathbb{R}$ with $\Gamma(f)(x) \neq 0$ gives rise to an upper curvature bound $K^f_{P,N}(x)$ via the inequality (2). Namely, we have

$$K_N(x) \leq K^f_{P,N}(x) := \frac{1}{\Gamma(f)(x)}\left(\Gamma_2(f)(x) - \frac{1}{N}(\Delta f(x))^2\right). \tag{4}$$

A vertex $x \in V$ is therefore N-curvature sharp if its Bakry–Émery curvature $K_N(x)$ agrees with the specific upper curvature bound $K^{d(x,\cdot)}_{P,N}(x)$. We would also like to mention the following monotonicity property of curvature sharpness: If $x \in V$ is N-curvature sharp that this vertex is also curvature sharp for any dimension $\leq N$ (see [1] (Prop. 3.1)).

In the next subsection, we present an important reformulation of Bakry–Émery curvature at $x \in V$ using a specific matrix $Q(x)$, which will be important in the definition of the curvature flow.

1.2. Reformulation of Curvature via a Schur Complement

The combinatorial distance function allows us to define distance spheres and distance balls,

$$\begin{aligned} S_r(x) &= \{z \in V : d(x,z) = r\}, \\ B_r(x) &= \{z \in V : d(x,z) \leq r\}. \end{aligned}$$

Let $x \in V$ be a non-isolated vertex. It turns out that the Bakry–Émery curvature $K_N(x)$ is determined locally, that is, can be derived solely from the information about the 2-ball

$$B_2(x) = \{x\} \cup S_1(x) \cup S_2(x).$$

More precisely, denoting $S_1(x) = \{y_1, \ldots, y_m\}$ and $S_2(x) = \{z_1, \ldots, z_n\}$, there exist a column vector $\Delta(x)$ and a symmetric matrix $\Gamma(x)$ of size m and a symmetric matrix $\Gamma_2(x)$ of size $m + n$ such that, for functions $f, g : V \to \mathbb{R}$ with $f(x) = g(x) = 0$,

$$\begin{aligned} \Delta f(x) &= \Delta(x)^\top \vec{f}_m, \\ \Gamma(f,g)(x) &= \vec{f}_m^\top \Gamma(x) \vec{g}_m, \\ \Gamma_2(f,g)(x) &= \vec{f}_{m+n}^\top \Gamma_2(x) \vec{g}_{m+n}, \end{aligned}$$

where $\vec{f}_m = (f(y_1), \ldots, f(y_m))^\top$ and

$$\vec{f}_{m+n} = (f(y_1), \ldots, f(y_m), f(z_1), \ldots, f(z_n))^\top,$$

and $\vec{g}_n, \vec{g}_{m+n}$, accordingly. Using the $(n+m)$-block decomposition

$$\Gamma_2(x) = \begin{pmatrix} \Gamma_2(x)_{S_1} & \Gamma_2(x)_{S_1,S_2} \\ \Gamma_2(x)_{S_2,S_1} & \Gamma_2(x)_{S_2} \end{pmatrix}$$

and employing the Schur complement

$$Q(x) = \Gamma_2(x)_{S_1} - \Gamma_2(x)_{S_1,S_2}\Gamma_2(x)^{\dagger}_{S_2}\Gamma_2(x)_{S_2,S_1}$$

for matrices $\Gamma_2(x)$ with positive semidefinite $\Gamma_2(x)_{S_2}$-blocks with $A^{\dagger}$ the pseudoinverse of A (for a given matrix $A \in \mathbb{R}^{N\times M}$, its pseudoinverse $A^{\dagger} \in \mathbb{R}^{M\times N}$ is defined by the following conditions: $AA^{\dagger}A = A$, $A^{\dagger}AA^{\dagger} = A^{\dagger}$, and $AA^{\dagger} \in \mathbb{R}^{N\times N}$ and $AA^{\dagger} \in \mathbb{R}^{M\times M}$ are both symmetric matrices), we can reformulate Bakry–Émery curvature at $x \in V$ for dimension $N \in (0,\infty]$ as follows (see Section 2.4 in [1]):

Let $x \in V$ be a non-isolated vertex. $K_N(x)$ is then the maximum of all $K \in \mathbb{R}$ such that

$$Q(x) - \frac{1}{N}\Delta(x)\Delta(x)^{\top} - K\Gamma(x) \succeq 0,$$

where $A \succeq B$ means that $A - B$ is positive semidefinite.

This curvature translation was motivated originally by the aim to reformulate the computation of Bakry–Émery curvature as an eigenvalue problem (see [22–24]).

The symmetric matrix $Q(x)$ of a non-isolated vertex $x \in V$ is of size m and is—in the non-degenerate case—closely related to another symmetric matrix $A_\infty(x)$, which, in turn, can be viewed as a discrete counterpart of the Ricci curvature tensor at a point $x \in M$ of a Riemannian manifold (M, g) (see formula (1.2) and Section 7 in [24]). In the case of a Markovian weighted graph, curvature sharpness at a vertex $x \in V$ can also be alternatively expressed with the help of the matrix $Q(x)$ as follows.

Theorem 1 (see Theorem 1.3 in [1]). *Let (G, P) be a Markovian weighted graph and $x \in V$ be a non-isolated vertex with $S_1(x) = \{y_1, \ldots, y_m\}$. Then, the following statements are equivalent:*

(1) x is curvature sharp;
(2) x is curvature sharp for dimension $N = 2$;
(3) We have

$$Q(x)\mathbf{1}_m = \frac{1}{2}K_{P,\infty}^{d(x,\cdot)}(x)\mathbf{p}_x, \tag{5}$$

where $\mathbf{p}_x = (p_{xy_1}, \ldots, p_{xy_m})^{\top}$ and $\mathbf{1}_m$ is the all-one column vector of size m.

Note that the term $K_{\infty}^{d(x,\cdot)}(x)$ in (5) is the upper curvature bound introduced in (4) (in the special case $N = \infty$), that is,

$$K_{P,\infty}^{d(x,\cdot)}(x) = \frac{\Gamma_2(d(x,\cdot))(x)}{\Gamma(d(x,\cdot))(x)}.$$

1.3. Curvature Flow

Let (G, P_0) be a fixed initial weighted graph with $N = |V|$. For every non-isolated vertex $x \in V$, the size of the corresponding symmetric matrix $Q(x)$ agrees with the degree of the vertex x, that is, the number of one- and two-sided edges emanating from x, and the entries of $Q(x)$ are determined by the transition rates of edges of the 2-ball $B_2(x)$. Our curvature flow associates to this initial data a smooth matrix-valued function $P : [0,\infty) \to \mathbb{R}^{N\times N}$ with $P(0) = P_0$. The corresponding symmetric Q-matrices at time $t \in [0,\infty)$ depend on the weighting schemes $P(t)$, and we denote them henceforth by $Q_x(t)$ for all $x \in V$. Our curvature flow is now defined as follows.

Definition 3 (Curvature flow)**.** *Let (G, P_0) be a finite weighted graph. The associated* curvature flow *is given by the following differential equations for all non-isolated vertices $x \in V$ and all $t \geq 0$:*

$$p'_{xx}(t) = 0, \tag{6}$$
$$\mathbf{p}'_x(t) = -4Q_x(t)\mathbf{1}_m + 2C_x(t)\mathbf{p}_x(t), \tag{7}$$

where $S_1(x) = \{y_1, \ldots, y_m\}$ and

$$\mathbf{p}_x(t) = (p_{xy_1}(t), \ldots, p_{xy_m}(t))^\top.$$

In the case of an isolated vertex $x \in V$, its curvature flow is given by the simple equation $p'_{xx}(t) = 0$, that is $p_{xx}(t)$ is a constant function in t.

We note that the curvature flow Equation (6) guarantees that the diagonal entries of the weighting scheme do not change. The functions $C_x(t)$ in the curvature flow Equation (7) play the role of a normalization since, for the choice $C_x \equiv 0$, various transition rates will be unbounded as the time parameter $t \geq 0$ increases. Note that in the smooth case of closed Riemannian manifolds (M, g_0), a suitable normalization leads to volume perseverance of (M, g_t) under the Ricci curvature flow. Our aim is to preserve the Markovian property, and it was shown in our first paper that the curvature flow preserves this property if we choose the normalization functions

$$C_x(t) = K^{d(x,\cdot)}_{P(t),\infty}(x). \tag{8}$$

Let us give the explicit formulas for the curvature flow Equation (7) for this particular choice of $C_x(t)$, where y, y', y'' always represent vertices in $S_1(x)$ (see [1] (Formula (66)):

$$p'_{xy}(t) = p_{xy}(t)\left(-4p_{yx}(t) - 2\sum_{y'\neq y} p_{yy'}(t) + \frac{4}{D_x}\sum_{y'} p_{xy'}(t)p_{y'x}(t) + \frac{1}{D_x}\sum_{y',y''} p_{xy'}(t)p_{y'y''}(t) - p_{yy}(t)\right) + \underbrace{\sum_{y'\neq y} p_{xy'}(t)p_{y'y}(t)}_{(*)}. \tag{9}$$

Here, we use $D_x = \sum_{y'} p_{xy}(t) = 1 - p_{xx}(t)$. Note that (6) guarantees that $D_x \leq 1$ is independent of the time parameter t.

The following theorem collects some fundamental properties of the normalized curvature flow.

Theorem 2 (see Theorem 1.5 and Prop. 1.6 in [1])**.** *Let (G, P_0) be a Markovian weighted graph. Then, the curvature flow $(G, P(t))_{t\geq 0}$ associated with (G, P_0) with normalization (8) is well defined for all $t \geq 0$ and preserves the Markovian property. If (G, P_0) is non-degenerate, then $(G, P(t))$ is also non-degenerate for all $t \geq 0$. Moreover, if the flow converges for $t \to \infty$ to $P^\infty = \lim_{t\to\infty} P(t)$, then the weighted graph (G, P^∞) is curvature sharp.*

We would like to emphasize that, even in the Markovian case, a flow limit $P^\infty = \lim_{t\to\infty} P(t)$ of a non-degenerate weighted graph (G, P_0) is in most cases no longer non-degenerate, despite the fact that all weighting schemes $P(t)$ for finite $t \geq 0$ are non-degenerate. In other words, some transition probabilities converge to zero under our normalized curvature flow, as time tends to infinity.

1.4. Other Discrete Curvature Flows

We presented already other curvature flows in [1] and briefly recall them here for the readers' convenience.

To our knowledge, the curvature flow on discrete Markov chains in this paper is the first one that is based on Bakry–Émery curvature. Other curvature flow investigations for discrete Ricci-type curvature are the following:

- Combinatorial Ricci flows on surfaces [25];
- Ollivier Ricci flow ([26,27], Problem N) and [28,29];
- Forman-Ricci curvature flow [30,31];
- Stochastic Ricci flow for balanced Forman curvature [32,33];
- Entropic super Ricci flow [34,35];
- Resistance curvature flow [36].

2. Curvature Flow Examples

In this section, we investigate normalized curvature flows on some unmixed combinatorial graphs $G = (V, E)$. By *unmixed* we mean that G does not have one-sided edges. We assume in all examples and statements in this section that our graphs $G = (V, E)$ are finite, simple, unmixed, and connected and that our initial weighting schemes $P_0 = P(0) = (p_{xy}(0))_{x,y \in V}$ are non-degenerate Markovian without laziness (even if we do not mention this). Curvature flow limits (G, P^∞) with $P^\infty = \lim_{t \to \infty} P(t)$ are necessarily curvature sharp by Theorem 2 above. We do not know of any initial Markovian weighted graph that does not converge as $t \to \infty$. Moreover, we know that every finite connected graph with at least two vertices admits many curvature sharp Markovian weighting schemes without laziness (see [1] (Theorem 1.10)). These facts give rise to the following conjecture.

Conjecture 1 (see Conjecture 1.7 in [1])**.** *The curvature flow* $(G, P(t))_{t \geq 0}$ *with normalization* (8) *converges for any initial condition* (G, P_0) *as* $t \to \infty$.

Let us now address some practical aspects of the curvature flow implementation. The Python code is freely available as an ancillary file of [2] or via github at https://github.com/georgestagg/graph-curvature-server (accessed on 6 June 2023). The solution of the curvature flow is computed numerically by the Runge–Kutta (RK4) method, which is based on a time discretization with time increments $dt > 0$. In the following examples, we choose the step sizes $dt = 0.1$ and $dt = 0.3$. In order to distinguish between the theoretical curvature flow and its implementation, we refer to the latter as the *numerical curvature flow*. Since a numerical curvature flow cannot run forever, a suitable numerical convergence criterion needs to be introduced. Our convergence criterion is based on the parameter $\lim_{\text{tolerance}} > 0$. We say that a numerical curvature flow solution $(P(t))_{t \geq 0}$ has converged numerically at time t (with respect to the parameter $\lim_{\text{tolerance}}$) if all the entries of $P(t)$ differ from the corresponding entries of $P(t + 10)$ and $P(t + 20)$ by less than $\lim_{\text{tolerance}}$. The numerical flow limit is then defined to be $P(t)$. In all the following examples, we set $\lim_{\text{tolerance}} = 0.001$.

In this section, we are particularly interested in numerical flow limits that are not *numerically totally degenerate*. An unmixed weighted graph (G, P) is called *numerically totally degenerate* if there are no edges with numerical non-zero transition rates in both directions, where we consider a transition rate p_{xy} as numerically non-zero (with respect to a parameter threshold > 0), if and only if $p_{xy} \geq$ threshold. In all the following examples we set threshold $= 0.001$.

2.1. Random Graphs

In this subsection, we investigate the numerical curvature flow for random weighted graphs (G, P_0) with vertex set V. The relevance of this example is that it provides insights into flow limits for general graphs without any special symmetries. At the same time, readers are familiarized with our software tool, enabling them to run examples of their own interest. The edge set E of the random graph G is generated by an Erdös-Rényi process [37], that is, any pair of vertices is independently and randomly connected by a two-sided edge with a probability $p \in [0, 1]$. Similarly, we choose a random initial weighting scheme P_0

with the property that all non-zero transition rates $p_{vv'}(0)$ lie in the interval $[\text{threshold}, 1]$, for some positive parameter threshold > 0. We are interested in the properties of the numerical flow limits of these graphs. If the parameter $p > 0$ in the Erdös-Rényi process is too small, these flow limits are always numerically totally degenerate. A reasonable choice to obtain not numerically totally degenerate flow limits for such random graphs with, say, 10 vertices in roughly half of the cases, is $p = 0.7$.

Example 1 (A random graph with 10 vertices)**.** *Let (G, P_0) be the unmixed weighted Markovian graph with vertex set $V = \{v_0, \dots, v_9\}$ and*

$$P_0 = \begin{pmatrix} 0 & 0.1 & 0.08 & 0.17 & 0 & 0.28 & 0.21 & 0.08 & 0 & 0.08 \\ 0.08 & 0 & 0 & 0.16 & 0 & 0.2 & 0.07 & 0.3 & 0.04 & 0.15 \\ 0.27 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0.43 \\ 0.02 & 0.19 & 0 & 0 & 0.17 & 0.17 & 0.11 & 0.34 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.04 & 0.21 & 0 & 0.41 & 0 & 0 & 0.34 & 0 & 0 & 0 \\ 0.06 & 0.29 & 0.14 & 0.12 & 0 & 0.3 & 0 & 0.09 & 0 & 0 \\ 0.08 & 0.31 & 0 & 0.19 & 0 & 0 & 0.23 & 0 & 0.19 & 0 \\ 0 & 0.13 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0.62 \\ 0.1 & 0.33 & 0.38 & 0 & 0 & 0 & 0 & 0 & 0.19 & 0 \end{pmatrix}$$

This randomly generated initial graph is illustrated on the left-hand side of Figure 1. The numerical curvature flow of (G, P_0) has numerical convergence time $t_{\max} = 20.7$ (with respect to $\lim_{\text{tolerance}} = 0.001$). The numerical flow limit $(G, P(t_{\max}))$ is presented on the right hand side of Figure 1. Let us briefly explain the illustration of the edges of this flow limit: Edges with numerical non-zero transition rates in both directions are displayed in green. Edges with only one-sided numerical non-zero transition rates are displayed as dashed red lines with arrows. Edges whose transition rates shrink in both directions numerically to zero under the curvature flow are displayed as dotted black lines. The corresponding non-zero transition rates are written along these edges. The vertices of the green edges of the flow limit in Figure 1 are given by

$$W = \{v_0, v_1, v_3, v_5, v_6, v_7\}.$$

Since there are no numerical non-zero transition rates from these vertices to the other vertices $v_2, v_4, v_8, v_9 \in V \setminus W$, the vertex set W together with the green edges and their transition rates represent a highly connected non-degenerate Markovian weighted subgraph (G_W, P_W). The combinatorial graph G_W with vertex set W can be viewed as a double cone over the complete graph K_4 of the four vertices v_0, v_1, v_3, v_6 with the vertices v_5, v_7 as its cone tips. The transition rates of the weighting scheme P_W towards all vertices in K_4 are $0.25 = 1/4$, all transition rates towards v_5 are $x = 0.05$ and all transition rates towards v_7 are $y = 0.2$. Note that such a weighted double cone over K_4 with these transition rates for any choice of $x, y > 0$ satisfying $x + y = 1/4$ is curvature sharp (this follows readily from the geometric criterion in [1] (Theorem 3.15), since the weighting scheme is volume homogeneous in all vertices and reversible with $\pi(v_5) = 4x/5$, $\pi(v_7) = 4y/5$ and $\pi|_{K_4} \equiv 1/5$). Therefore, not only is the flow limit itself curvature sharp in Theorem 2 but also the highly connected non-degenerate weighted subgraph (G_W, P_W).

Figure 2 presents the transition rates of the vertices v_3, v_7 under the curvature flow as functions over the interval $[0, t_{\max}]$. While most transition rates converge to strictly positive limits, the transition rates $p_{v_3v_4}(t)$ and $p_{v_8v_9}(t)$ shrink to zero. Consequently, the corresponding edges on the right-hand side of Figure 1 are represented by a dashed red line and a black dotted line, respectively.

Let us finally consider the curvatures $t \mapsto K_{P(t),N}(v_j)$ of the vertices v_j under the curvature flow. We focus primarily on the vertices v_1 and v_9 and the dimension parameter $N = \infty$. Figure 3 presents the ∞-curvature (in blue) and upper curvature bound $K^{d(v,\cdot)}_{P(t),\infty}(v)$ (in orange) of $v \in \{v_1, v_9\}$ as functions over the interval $[0, t_{\max}]$. Note that the absence of laziness implies that the

upper curvature bounds $K^{d(v,\cdot)}_{P(t),\infty}(v)$ *are all* ≥ 0 *(see [1] (19)). For both vertices, the curvature and upper curvature bound functions are numerically asymptotic as* $t \to t_{\max}$*, indicating that* v_1 *and* v_9 *of the flow limit are* ∞*-curvature sharp. (In fact, all vertices in this example are* ∞*-curvature sharp with respect to the flow limit.) This is not always the case, but Theorem 1 confirms that all vertices of a flow limit* (G, P^∞) *are at least* N*-curvature sharp for dimension* $N = 2$*. Moreover, the initial and final* ∞*-curvatures of all vertices under the curvature flow are presented in Table 1. The final curvatures of all vertices in* K_4 *assume the highest values* 0.875*, followed by the final curvature values* 0.773 *and* 0.476 *of the vertices* v_7 *and* v_5*, respectively. All other vertices in* $V \setminus V_0$ *have much lower final curvatures with values* ≤ 0.125*.*

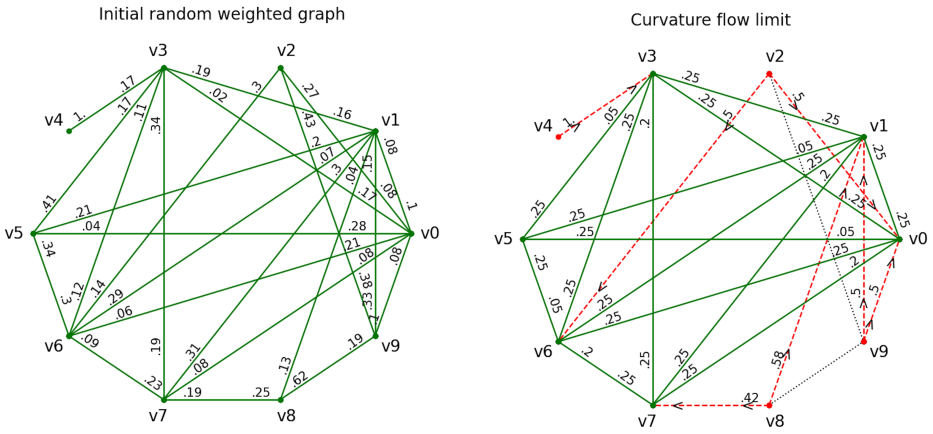

Figure 1. Curvature flow of a random graph with 10 vertices with initial weighting scheme P_0 (**left**-hand side) and final weighting scheme $P(t_{\max})$ (**right**-hand side).

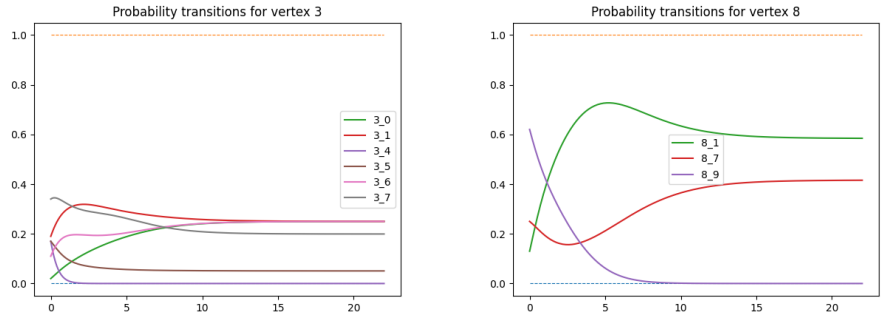

Figure 2. Transition rates of vertices v_3 and v_8 of a random graph with 10 vertices under the curvature flow.

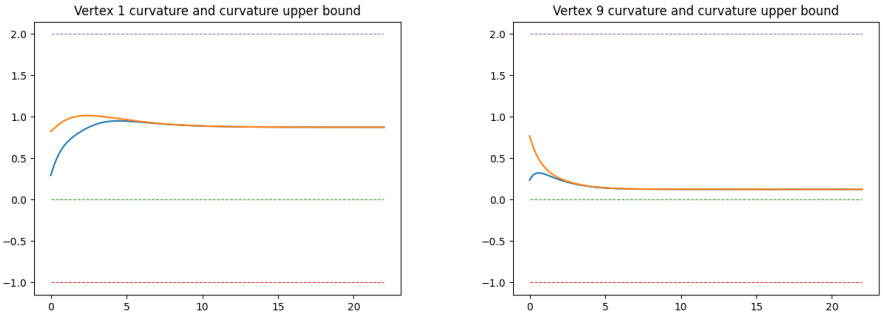

Figure 3. Curvatures (blue) and upper curvature bounds (orange) of vertices v_1 and v_9 of a random graph with 10 vertices for the dimension ∞ under the curvature flow.

Table 1. Vertex curvatures for the dimension ∞ of a random graph with 10 vertices at the beginning and the end of the curvature flow.

j	$K_{P(0),\infty}(v_j)$	$K_{P(t_{\max}),\infty}(v_j)$
0	0.406	0.875
1	0.293	0.875
2	0.168	0.125
3	0.346	0.875
4	0.34	0
5	0.527	0.476
6	0.404	0.875
7	0.202	0.773
8	0.246	0.11
9	0.236	0.125

Example 1 and its illustrations were generated by the following code.

```
1  dt = 0.1                    # time increment in the Runge-Kutta (RK4) algorithm
2  stoch_corr = False          # automatic stochastic correction in the RK4 algorithm
3  norm_tolerance = 0.001      # threshold to apply stochastic correction
4  threshold = 0.001           # threshold to consider a number numerally as zero
5  lim_tolerance = 0.001       # threshold to define flow convergence
6  t_lim = 10000               # maximal flow time
7  p = 0.7                     # Erdoes-Renyi probability parameter
8  k = 1                       # time multiplier for consecutive curvature computations
9  is_Markov = True            # curvatures are w.r.t. a Markovian weighting scheme
10 N = inf                     # dimension parameter for curvature computations
11 laziness = False            # Boolean in the generation of random weighting schemes
12
13 A = rand_adj_mat(10, 0.7, False)
14 P_0 = randomizer(A, threshold, laziness)
15
16 limit = norm_curv_flow_lim(A, P_0, dt, stoch_corr, norm_tolerance, lim_tolerance,
       t_lim)
17
18 t_max=limit[1]              # limit[1] is the convergence time
19                             # as calculated by norm_curv_flow_lim
20
21 flow = norm_curv_flow(A, P_0, t_max, dt, stoch_corr, norm_tolerance)
22
23 display_weighted_graph(A, P_0, "Initial random weighted graph", threshold)
24 display_weighted_graph(A, limit[0], "Curvature flow limit", threshold)
25 display_trans_rates(A, flow, dt, [3, 8])
26
27 curvs = calc_curvatures(A, flow, N, k)
28 curv_bound = calc_curv_upper_bound(A, flow, N, k)
29 display_curvatures(curvs, dt, is_Markov, N, k, curv_bound, [1, 9])
```

Let us explain the commands of this program in detail. After initializing various parameters in lines 1–11, a random graph $G = (V, E)$ is generated in line 13 via an Erdös-Renyi process with probability $p = 0.7$. A corresponding random non-degenerate Markovian weighting scheme without laziness is generated in line 14, with each non-zero transition rate satisfying $p_{vv'}(0) \in [\text{threshold}, 1]$. The numerical convergence time $t_{\max} \geq 0$ is determined in lines 16–18. In most cases, the convergence time does not exceed 100. (If convergence is not achieved by $t_{\lim} = 10000$, the curvature flow computation stops at that time and notifies the user.) The numerical curvature flow on the interval $[0, t_{\max}]$ is solved again in line 21. Initial and final weighting schemes are displayed by the commands in

lines 23 and 24, respectively, providing the illustrations given in Figure 1. The transition rates of the vertices v_3 and v_8 are displayed by the command in line 25, providing the illustrations given in Figure 2. The curvatures and upper curvature bounds at time steps $j \cdot \frac{k \cdot t_{\max}}{dt}$, $j = 0, 1, \ldots$, are computed in lines 27 and 28, respectively, with the choice $k = 1$. They are displayed for the vertices v_1 and v_9 via the command in line 29, providing the illustrations given in Figure 3.

When running this program, users may be faced with the following message:

```
'norm_tolerance' has been exceeded at one or more vertices, at time
t = ...  Would you like to:
A = Stop calculation and return list of P-matrices so far
B = Apply manual normalization now, and apply it again when necessary without
asking (you will still be notified when it is applied)
C = Apply manual normalization now, and ask again before reapplying it
Please enter A, B or C here:
```

The reason behind this message is the following. The computation of the numerical curvature flow is based on a time discretization. Therefore, the solution will increasingly depart from the Markovian property after each time increment $dt = 0.1$. If the sum of entries of one of the rows of $P(t)$ at time t differs from one by more than $\text{norm}_{\text{tolerance}} = 0.001$, the program informs the user that a normalization of the weighting scheme is needed for the continuation of the flow calculations. After choosing the option `'B'`, the program will continue with its flow calculations without further interruptions, and the user is simply notified about the times at which the program applies further artificial normalizations of the transition rates. The user can suppress this message entirely by changing line 2 of the program into "`stoch_corr = True`", in which case the program applies stochastic corrections automatically, each time with the message

```
Transition rates have been artificially normalized at time t = ...
```

The numerical observations of Example 1 suggest that similar properties may also hold in general in the theoretical setting. Firstly, we call an edge $\{x, y\} \in E$ in an unmixed weighted graph (G, P) *non-degenerate* if $p_{xy}, p_{yx} > 0$. An unmixed weighted graph (G, P) is called *totally degenerate* if it does not have any non-degenerate edges. Secondly, we denote the vertices of all non-degenerate edges of a flow limit (G, P^∞) by $W \subset V$, and G_W denotes the subgraph of G consisting of the vertices W and all non-degenerate edges of (G, P^∞). Moreover, P_W denotes the restriction of the weighting scheme P^∞ to the vertex set W. We call G_W the non-degenerate subgraph of (G, P^∞) and we conjecture the following:

Conjecture 2. *If the normalized curvature flow of a nondegenerate unmixed Markovian weighted graph (G, P_0) converges to a not totally degenerate limit (G, P^∞), then the non-degenerate subgraph G_W coincides with the induced subgraph (of G) of the subset $W \subset V$ and all transition rates from W to $V \setminus W$ are zero. (G_W, P_W) is a non-degenerate Markovian weighted graph, which is itself curvature sharp.*

Figuratively speaking, the curvature flow converges towards the non-degenerate Markovian subgraph (G_W, P_W) consisting of highly connected components. Moreover, each vertex of $V \setminus W$ is usually connected to the set W by a sequence of edges with one-sided non-zero transition rates pointing towards the set W, and we generally expect that the ∞-curvature values of the vertex set W in (G, P^∞) are significantly larger than the ∞-curvature values of the set $V \setminus W$ in (G, P^∞).

In Example 1, the weighted Markovian subgraph (G_W, P_W) has only one connected component, but we will see in the next subsection in the case of paths and cycles that (G_W, P_W) may be composed of more than just one connected component.

2.2. Paths and Cycles

Paths and cycles are the easiest examples of graph families and are therefore natural examples to study. We will see that both families have flow limits of very similar types. Let $G = (V, E)$ be a path of length $N \geq 2$, that is, $V = \{v_0, \ldots, v_{N-1}\}$ with a two-sided edge between v_i and v_j if and only if $|i - j| = 1$. If $N = 2$, G is a trivial case of a star graph and any Markovian weighting scheme satisfying $p_{01} = p_{21} = 1$ and $p_{10} + p_{12} = 1$ is curvature sharp (see [1], Example 4.3). For that reason, we consider only paths of lengths $N \geq 3$. A cycle of length 3 is the complete graph K_3, and a full list of all curvature sharp weighting schemes was given in [1], Prop. 1.9, so we consider only cycles of length $N \geq 4$. The following example provides some insights into some features of curvature flow limits of weighted paths and cycles.

Example 2 (A path and a cycle of length 12). *Figure 4 presents numerical curvature flow limits of a Markovian weighted path with vertices* 12 *vertices (left-hand side) and of a weighted cycle with* 12 *vertices (right-hand side). The non-zero transition rates of the initial weighting scheme* $P_0 = (p_{ji})_{0 \leq i,j \leq 11}$ *for the path limit in Figure 4 were chosen as follows:*

$p_{0,1}$	$p_{1,2}$	$p_{2,3}$	$p_{3,4}$	$p_{4,5}$	$p_{5,6}$	$p_{6,7}$	$p_{7,8}$	$p_{8,9}$	$p_{9,10}$	$p_{10,11}$
1	0.25	0.72	0.46	0.23	019	0.84	0.71	0.62	0.9	0.55
$p_{1,0}$	$p_{2,1}$	$p_{3,2}$	$p_{4,3}$	$p_{5,4}$	$p_{6,5}$	$p_{7,6}$	$p_{8,7}$	$p_{9,8}$	$p_{10,9}$	$p_{11,10}$
0.75	0.28	0.54	0.77	0.81	0.16	0.29	0.38	0.1	0.45	1

The non-degenerate subgraph G_W *of the path limit consists of* $W = \{v_1, v_2, v_3, v_9, v_{10}, v_{11}\}$ *as its vertex set together with the four green edges. Moreover,* G_W *has two paths of length* 2 *as its connected components. For each vertex* $v \in V \backslash W$*, there exists a directed path to one of the components of* G_W *via a sequence of one-sided transition rates. For example, the vertices* v_5, v_6, v_7, v_8 *are connected to the vertex* $v_9 \in W$ *via such one-sided paths, and* v_4, v_5 *are connected to the vertex* v_3 *via such one-sided paths.*

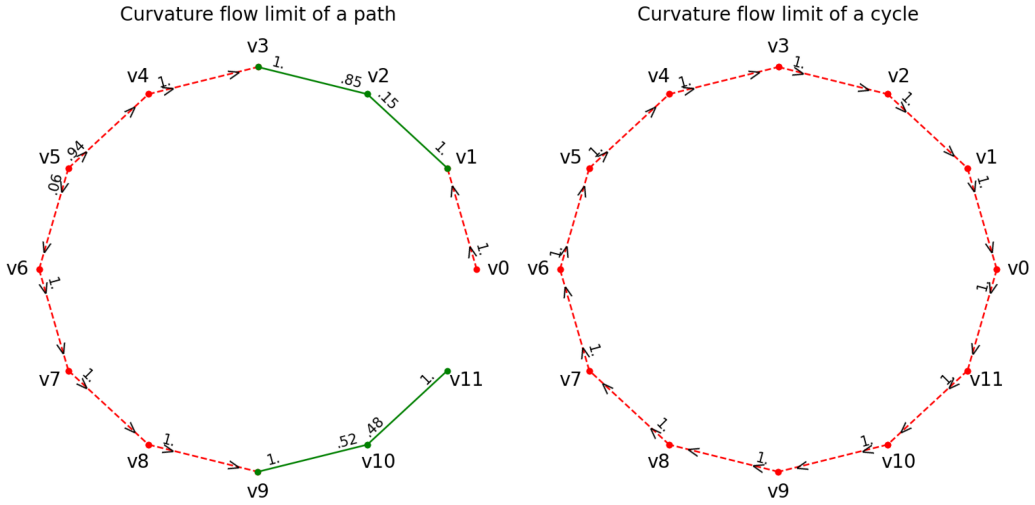

Figure 4. Examplesof numerical curvature flow limits of a path and of a cycle of length 12.

The cycle limit on the right-hand side of Figure 4 is totally degenerate with no green edges, and all one-sided non-zero transition rates are oriented in a clockwise direction. The experiments show that Figure 4 exhibits generic limit properties: flow limits of weighted paths are never totally degenerate and their non-degenerate subgraphs G_W *consist of disjoint paths of length* ≤ 2*. Such types of limits also appear in the case of weighted cycles. However, in contrast to the path case, sometimes a cycle limit is totally degenerate with all its one-sided transition rates oriented either clockwise or anti-clockwise. The code for running the curvature flow for paths and cycles of length n reads as follows:*

```
N = 12
A = path(N)
# A = cycle(N)
P = randomizer(A)
limit = norm_curv_flow_lim(A, P)[0]
display_weighted_graph(A, P, "Initial weighted graph")
display_weighted_graph(A, limit, "Curvature flow limit")
```

Before we provide the following result about path limits, let us first introduce the notion of a two-sided degenerate edge of weighted graphs (G, P): An edge $\{x, y\} \in E$ of G is called *two-sided degenerate* if its transition rates vanish in both directions, that is, $p_{xy} = p_{yx} = 0$. Similarly, an edge is called *numerically two-sided degenerate*, if its transition rates in both directions are $<$ threshold. Such edges are displayed in the routine `display_weighted_graph` as dotted black lines (see, e.g., the edges $\{v_2, v_9\}$ and $\{v_8, v_9\}$ on the right hand side of Figure 1).

Proposition 1 (Flow limits of paths of length ≥ 3)**.** *Let (G, P_0) be a weighted path of length $N \geq 3$ with consecutive vertices $v_0, \ldots, v_{N-1}$. Let $P(t)$ be its corresponding curvature flow converging to a limit $P^\infty = \lim_{t\to\infty} P(t)$, such that (G, P^∞) does not have two-sided degenerate edges. Then, this limit is neither totally degenerate nor is it not non-degenerate (that is, it contains both green and dashed red edges). Moreover, the components of the non-degenerate subgraph G_W are paths of lengths ≤ 2, and they are separated from each other by at least two degenerate edges. If a component of G_W is a path of length 1, that is, just one edge $\{v_j, v_{j+1}\}$, then we have $p^\infty_{j,j+1} = p^\infty_{j+1,j} = 1$.*

Proof. By the last statement of Theorem 2, we only need to prove these statements for curvature sharp weighting schemes $P = (p_{ij})_{0\leq i,j\leq N}$ of paths of length $N \geq 3$. Such weighting schemes can never be non-degenerate by [1] (Prop. 1.11).

For the proof that curvature sharp weighting schemes can never be totally degenerate, we note that we cannot have two consecutive degenerate edges $\{v_j, v_{j+1}\}, \{v_{j+1}, v_{j+2}\} \in E$ with $p_{j,j+1} = 1 = p_{j+2,j+1}$ (since this would imply $p_{j+1,j} = p_{j+1,j+2} = 0$, contradicting to $p_{j+1,j} + p_{j+1,j+2} = 1$). Therefore, since $p_{01} = 1$, any totally degenerate weighting scheme would require $p_{j,j+1} = 1$ for any $j \geq 0$, in particular, $p_{N-2,N-1} = 1$, but this contradicts the fact that we also have $p_{N-1,N-2} = 1$ and that the edge $\{v_{N-1}, v_{N-2}\}$ needs to be degenerate.

A curvature sharp weighting scheme cannot have more than two consecutive non-degenerate edges. This can be seen as follows: At any vertex v_j with $p_{j,j-1}, p_{j,j+1} > 0$ we must have $p_{j-1,j} = p_{j+1,j}$. This follows from the arguments in the proof of [1] (Lemma 4.1) (namely, since the vertex v_j is not contained in any triangle, we have $p_{j-1,j} = p_{y+1,j} = p_{j,j-1}p_{j-1,j} + p_{j,j+1}p_{j+1,j}$.) If $0 < p_{j-1,j} = p_{j+1,j} < 1$, we could iterate this argument backward and forward and would end up with the fact that all entries of the matrix P above the diagonal and below the diagonal would lie in $(0, 1)$, which is a contradiction to $p_{01} = 1$. So, we must have $p_{j-1,j} = p_{j+1,j} \in \{0, 1\}$. Therefore, we cannot have two consecutive indices $j \in \{0, \ldots, N-1\}$ with $0 < p_{j-1,j} = p_{j+1,j} < 1$, which would exist in the case of three consecutive non-degenerate edges. So, the components of G_W are paths of length ≤ 2.

Moreover, any gap between two consecutive non-degenerate edges must be at least two edges: this follows from the fact that if a non-degenerate edge $\{v_j, v_{j+1}\}$ is followed by a degenerate edge $\{v_{j+1}, v_{j+2}\}$, then we must have $p_{j+1,j+2} = 0$: if we had $p_{j+1,j+2} > 0$, then we had $p_{j+1,j}, p_{j+1,j+2} > 0$ and, therefore, $0 < p_{j+1,j} = p_{j+2,j+1}$ and $\{v_{j+1}, v_{j+2}\}$ would be non-degenerate, which is a contradiction. Similarly, if a degenerate edge $\{v_k, v_{k+1}\}$ is followed by a non-degenerate edge $\{v_{k+1}, v_{k+2}\}$, we must have $p_{k+1,k} = 0$. Combining both facts implies that there cannot be a single degenerate edge separating two components of G_W. These arguments also show that components of G_W that are single edges $\{v_j, v_{j+1}\}$

must satisfy $p_{j,j+1} = p_{j+1,j} = 1$ since the adjacent degenerate edges have one-sided transition rates pointing towards this component. □

Similar arguments to the above can be used to prove that for cycles, the components of any flow limit of a weighted cycle are again paths of length ≤ 2, unless the limit is non-degenerate. Such non-degenerate limits exist for cycles, namely the simple random walks, but experiments show that simple random walks are very unstable stationary solutions of the curvature flow. Small perturbations of simple random walks do not converge back to the simple random walk (unless our cycle is K_3) but converge usually to a degenerate limit. Finally, if a totally degenerate cycle limit does not have two-sided degenerate edges, then its transition rates must all be either oriented clockwise or anti-clockwise; otherwise, we would necessarily have a vertex with transition rates of both incident degenerate edges pointing towards this vertex. This would fail to satisfy the Markovian condition at this vertex.

Paths and cycles of length $N \geq 3$ have the property that no edge is contained in a triangle. We would like to finish the section with a general statement about the curvature flow for edges not contained in triangles.

Proposition 2. *Let (G, P_0) be a weighted Markovian graph without laziness and $(P(t))_{t\geq 0}$ be its associated normalized curvature flow. If we have, for some $t_0 \geq 0$ and an edge $e = \{x, y\} \in E$, $p_{xy}(t_0) = 0$ and e is not contained in a triangle of G, then we have*

$$p_{xy}(t) = 0 \quad \textit{for all } t \geq t_0.$$

Proof. This proposition is an easy consequence of the flow Equation (9). Since we assume no laziness, we have $p_{yy}(t) = 0$, and since e is not contained in a triangle, the last term of (9), denoted by $(*)$, is zero and the statement follows now from the uniqueness of the solution satisfying $p_{xy}(t_0) = 0$. □

2.3. Complete Graphs

Complete graphs are the natural choice for the study of the behavior of curvature flows of highly connected graphs. For these graphs, the simple random walks turn out to be the flow limits of non-degenerate initial weighting schemes. The simple random walk on any complete graph is a non-degenerate curvature sharp Markovian weighting scheme. Our experiments show that any non-degenerate initial weighting scheme P_0 on K_n converges to the simple random walk, that is $p_{jk}^{\infty} = \frac{1}{n-1}$. The following example shows that convergence to the simple random walk appears even if the initial weighting scheme is degenerate. This is not in contradiction to Proposition 2 since, in a complete graph K_n, $n \geq 3$, every edge is contained in a triangle. Example 3 is the only exception in this section where we allow an initial weighting scheme to have degenerate edges.

Example 3 (A degenerate weighted complete graph with six vertices)**.** *Let (G, P_0) be the complete weighted Markovian graph with vertex set $V = \{v_0, \ldots, v_5\}$ and*

$$P_0 = \begin{pmatrix} 0 & 0.2 & 0.1 & 0.2 & 0 & 0.5 \\ 0.1 & 0 & 0.3 & 0.25 & 0.25 & 0.1 \\ 0.2 & 0 & 0 & 0.3 & 0.15 & 0.35 \\ 0.3 & 0.5 & 0.1 & 0 & 0.1 & 0 \\ 0.2 & 0.3 & 0.3 & 0.2 & 0 & 0 \\ 0.6 & 0.1 & 0.1 & 0.2 & 0 & 0 \end{pmatrix},$$

as illustrated in Figure 5. Note that the edges $\{v_0, v_4\}, \{v_1, v_2\}, \{v_3, v_5\}$ and $\{v_4, v_5\}$ of this initial weighting scheme are degenerate, with the latter being two-sided degenerate. The numerical curvature flow has numerical convergence time $t_{\max} = 18.5$ (with respect to $\lim_{\text{tolerance}} = 0.001$) with the simple random walk as its numerical flow limit. Figure 6 presents the transition rates of the

vertices v_2 and v_5. Of particular interesting are the functions $p_{2,1}(t)$ and $p_{5,4}(t)$ for $t \in [0, t_{\max}]$, since their initial values are zero.

Based on our experiments, we conjecture the following, which is a strengthening of [1] (Conjecture 1.8).

Conjecture 3 (Curvature flow of complete graphs)**.** *Let P_0 be a Markovian weighting scheme without laziness on a complete graph $K_n = (V, E)$ with $n \geq 2$ such that, for every proper subset $W \subset V$, there exist $x, x' \in W$ and $y, y' \in\in V \setminus W$ with $p_{xy} > 0$ and $p_{y'x'} > 0$. Then, the curvature flow has a limit P^∞, which is the simple random walk.*

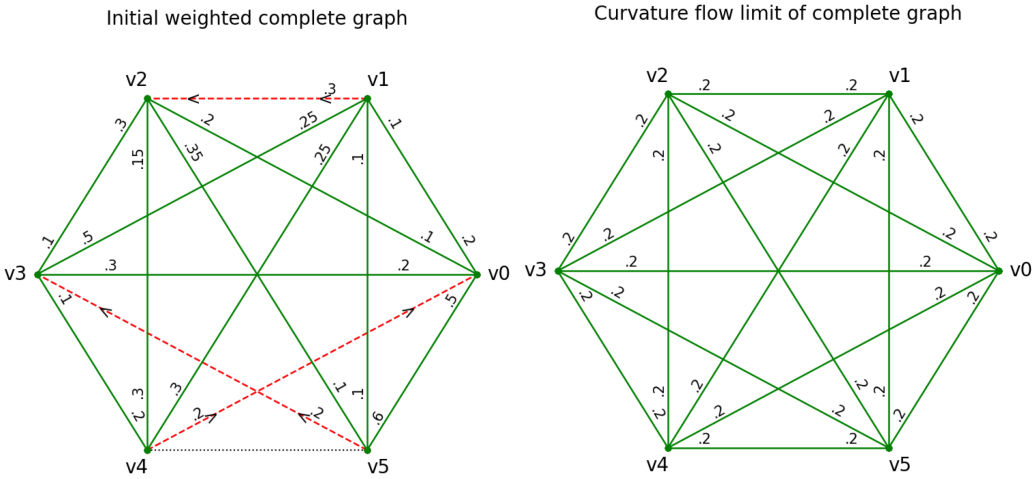

Figure 5. Curvature flow of a complete graph with six vertices with degenerate initial weighting scheme P_0 (**left**-hand side) and numerical flow limit $P(t_{\max})$, the simple random walk (**right**-hand side).

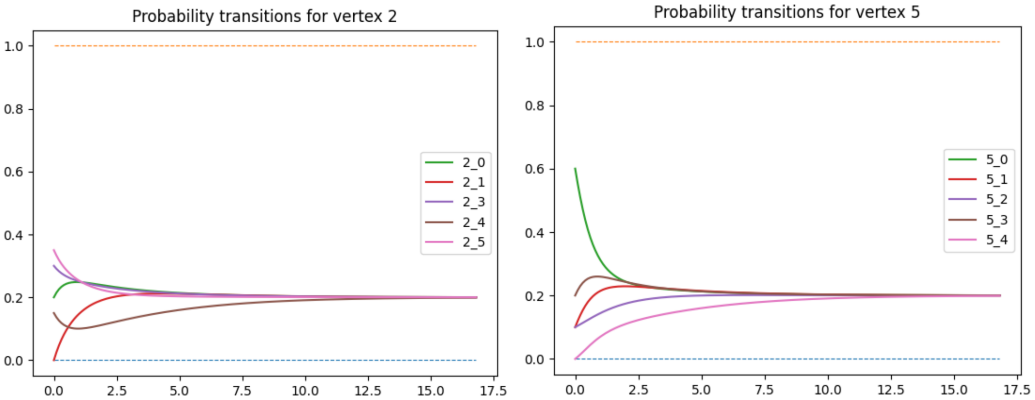

Figure 6. Transition rates of vertices v_2 and v_5 of a complete graph with six vertices under the curvature flow.

2.4. Wedge Sums of Complete Graphs

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two combinatorial graphs and $x_1 \in V_1$ and $x_2 \in V_2$. By merging the vertices x_1 and x_2 into one new vertex x, which inherits the incident edges of both vertices x_1 and x_2, we obtain a new combinatorial graph in which we denote the wedge sum of G_1 and G_2. In this subsection, we consider the flow limits of wedge sums of complete graphs. The study of these graphs leads to an interesting dynamical aspect of our curvature flow, namely that its limits concentrate on one of the components of these wedge sums.

Example 4 (A wedge sum of a K_4, K_5, K_2, and K_3)**.** *We consider the curvature flow on the wedge sum $G = (V, E)$ of complete graphs presented in Figure 7 with random initial weighting schemes $P = P_0$. The adjacency matrix A of this graph is generated with our code in the following way:*

```
A1 = wedge_sum(complete(4), complete(5), 2, 1)
A2 = wedge_sum(A1, complete(2), 6, 0)
A = wedge_sum(A2, complete(3), 8, 0)
```

Our experiments show that, depending on the initial weighting scheme P_0, the curvature flow converges to a limit that is concentrated in one of the complete graphs. More precisely, the limit weighting scheme P^∞ represents a simple random walk on one of the complete graphs while, from all other vertices, there is a directed path of $\{0,1\}$ transition rates towards this particular complete graph. Figures 8–10 show the numerical curvature flow limits of various random initial weighting schemes. We carried out several runs of 100,000 numerical curvature flows with random initial weighting schemes to describe the limit behavior of these flows quantitatively. The results are presented in Table 2. While more than 80% of limits concentrate on the largest clique K_5, it is somewhat surprising that more limits concentrate on K_3 than on the larger subgraph K_4. Not a single flow limit ended up concentrating on K_2. The mean numerical convergence time is shortest for K_3, followed by K_4 and K_5 (with respect to $\lim_{\text{tolerance}} = 0.001$). While most convergence times are below 100, there were maximal numerical convergence times well above 500.

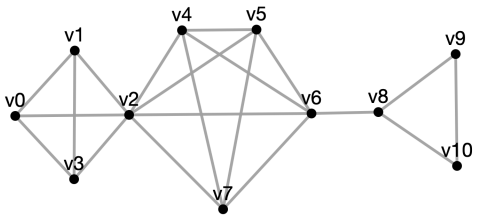

Figure 7. A wedge sum of a K_4, K_5, K_2 and a K_3.

$$P_0 = \begin{pmatrix} 0 & 0.4 & 0.5 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0.4 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0 & 0.3 & 0.1 & 0.1 & 0.1 & 0.2 & 0 & 0 & 0 \\ 0.2 & 0.5 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0.1 & 0.3 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.1 & 0 & 0.4 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.3 & 0.1 & 0 & 0.1 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.1 & 0.3 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0.8 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.3 & 0 \end{pmatrix}$$

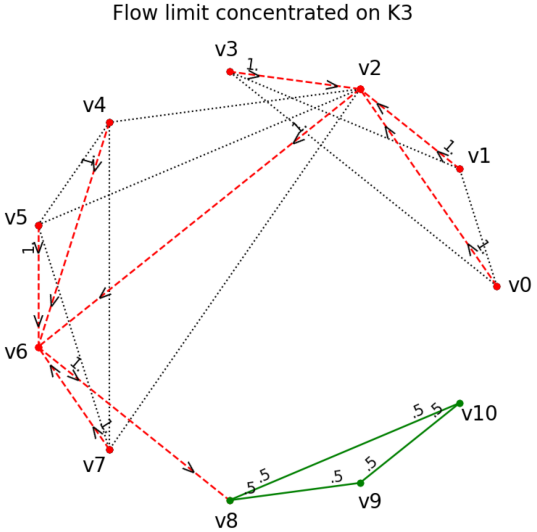

Figure 8. Initial weighting scheme (**left**) and numerical flow limit a simple random walk on K_3 (**right**).

$$P_0 = \begin{pmatrix} 0 & 0.3 & 0.3 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0.4 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0.2 & 0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0 & 0 & 0 \\ 0.4 & 0.4 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0.1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.2 & 0 & 0.5 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 & 0.1 & 0.1 & 0 & 0.3 & 0.1 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0.1 & 0.1 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.4 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 & 0 \end{pmatrix}$$

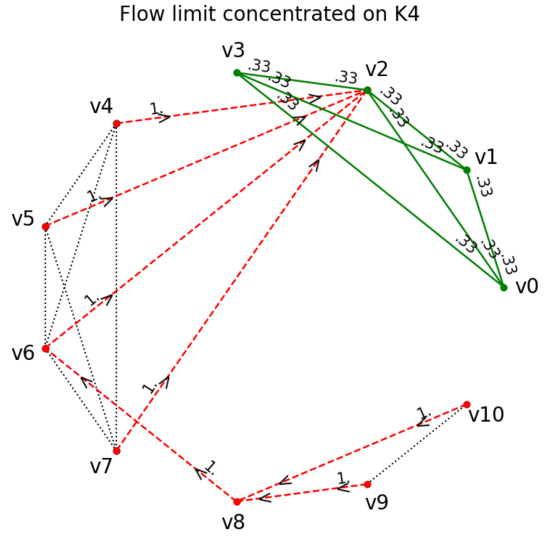

Figure 9. Initial weighting scheme (**left**) and numerical flow limit a simple random walk on K_4 (**right**).

$$P_0 = \begin{pmatrix} 0 & 0.2 & 0.2 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0.3 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0 & 0.2 & 0.1 & 0.1 & 0.2 & 0.2 & 0 & 0 & 0 \\ 0.3 & 0.3 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0.2 & 0.3 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.4 & 0 & 0.2 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0.2 & 0.1 & 0 & 0.2 & 0.2 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.1 & 0.3 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0.2 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.7 & 0 \end{pmatrix}$$

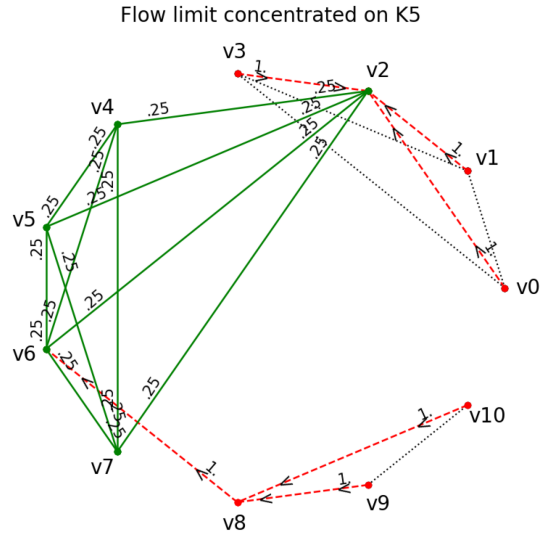

Figure 10. Initial weighting scheme (**left**) and numerical flow limit a simple random walk on K_5 (**right**).

Table 2. Statistics of flow limit concentrations on the complete subgraphs K_4, K_5, K_2 and K_3 of their wedge sum, together with mean convergence times.

	K_4	K_5	K_2	K_3
concentration of flow limits	8.7%	80.5%	0%	10.8%
mean convergence time	18.9	24.7	–	17.7

The limit behavior described in the above example seems to be common for many wedge sums of complete graphs. It is, however, not always true that flow limits concentrate on just one of the constituents of a wedge sum. A path of length ≥ 2 can be viewed as a wedge sum of consecutive K_2's, and we have seen in Section 2.2 that flow limits will concentrate on more than only one of these K_2's (see left-hand side of Figure 4). Another special case of a wedge sum is a dumbbell which is our next example.

Example 5 (A symmetric weighted dumbbell)**.** *The weighted graph (G, P_0) in this example is a wedge sum of a K_5, K_2 and another K_5, together with a simple random walk as initial weighting scheme (see line 4 in the code). This situation can be set up by the following code:*

```
A1 = complete(5)
A2 = complete(5)
A = bridge_at(A1,A2,0,0)
P_0 = srw(A)
```

The numerical convergence time is 79 *and the limit of the numerical curvature flow concentrates on the "bridge" K_2 between the two K_5's, as illustrated in Figure 11. (To obtain the flow limit illustrated at the right-hand side of this figure, users should choose* $\lim_{\text{tolerance}} = 0.0001$.*)*

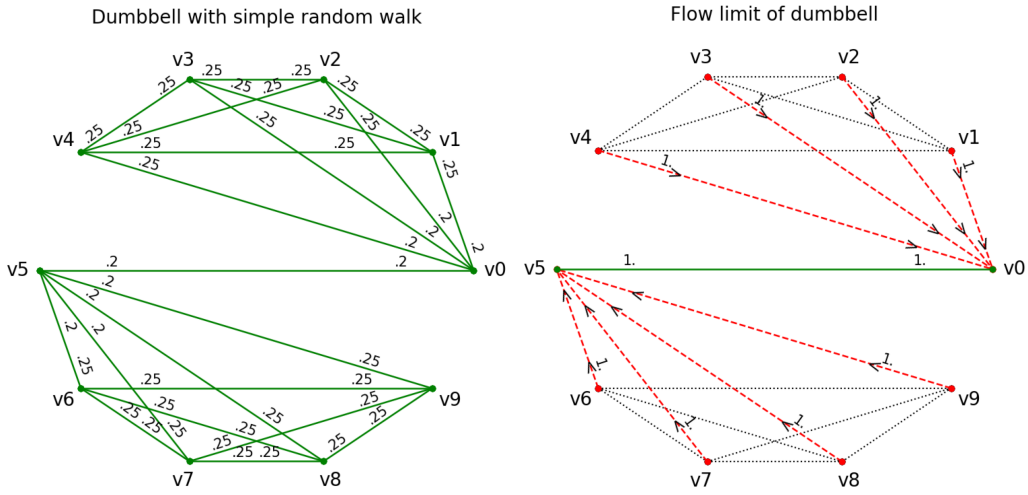

Figure 11. Curvature flow of a dumbbell as a wedge sum of a K_5, K_2 and another K_5 with simple random walk initial weighting scheme P_0 (**left**-hand side) and numerical flow limit concentrated on K_2 (**right**-hand side).

This limit could have been predicted assuming that the initial weighted graph symmetry across the "bridge" is preserved under the curvature flow and that the limit concentrates on only one of the complete graphs K_5, K_2, K_5. If instead of the simple random walk, a random initial weighting scheme would have been chosen on the dumbbell G, the limit would have usually concentrated on one of the two K_5's.

2.5. Cartesian Products of Complete Graphs

It is tempting to assume that simple random walks are always the preferred curvature flow limits in relation to complete graphs. However, this is not always the case as the following example shows. Namely, we have the following result for Cartesian products of complete graphs:

Theorem 3. *Let G be the Cartesian product of two complete graphs K_{n+1} and K_{m+1} with the non-lazy simple random walk P as the initial weighting scheme. Then, the curvature flow $P(t)$ converges to a limit as $t \to \infty$ with limit transition rates*

$$\begin{aligned} a^{\infty} &= \frac{m+3}{2nm+3n+3m}, \\ b^{\infty} &= \frac{n+3}{2nm+3n+3m}, \end{aligned}$$

where a^{∞} are the transition rates along edges between (x,x') and (y,x') with $x,y \in K_{n+1}$, $x \sim y$ and $x' \in K_{m+1}$ ("horizontal edges"), and b^{∞} are the transition rates along edges between (x,x') and (x,y') with $x \in K_{n+1}$ and $x',y' \in K_{m+1}$, $x' \sim y'$ ("vertical edges").

Proof. By symmetry of the configuration, we have only two types $a(t)$ and $b(t)$ of transition rates for the curvature flow at time $t \geq 0$ (for horizontal and vertical edges, respectively) with

$$a(0) = b(0) = \frac{1}{n+m}$$

and

$$na(t) + mb(t) = 1,$$

due to the Markovian property. This implies that $b(t) = \frac{1-na(t)}{m}$ and $b'(t) = -\frac{-na'(t)}{m}$, and we only need to consider an ordinary differential equation for a with initial condition $a(0) = \frac{1}{n+m}$. We can also assume without loss of generality that $n \leq m$. We derive from the explicit description (9) of the curvature flow that

$$\begin{aligned} a'(t) = & \\ & a(t)\Big(-4a(t) - 2(n-1)a(t) + 4(na^2(t) + mb^2(t)) + n(n-1)a^2(t) + m(m-1)b^2(t)\Big) \\ & + (n-1)a^2(t) = n(n+3)a^3(t) + m(m+3)a(t)\left(\frac{1-na(t)}{m}\right)^2 - (n+3)a^2(t) = \\ & a(t)\left(n\left(2n+3+\frac{3n}{m}\right)a^2(t) - \left(3n+3+\frac{6n}{m}\right)a(t) + 1 + \frac{3}{m}\right). \end{aligned} \quad (10)$$

If this differential equation converges as $t \to \infty$, its limit a^∞ must satisfy

$$a^\infty \in \left\{0, \frac{1}{n}, \frac{m+3}{2nm+3n+3m}\right\},$$

that is

$$(a^\infty, b^\infty) \in \left\{\left(0, \frac{1}{m}\right), \left(\frac{1}{n}, 0\right), \left(\frac{m+3}{2nm+3n+3m}, \frac{n+3}{2nm+3n+3m}\right)\right\},$$

with

$$b^\infty = \lim_{t\to\infty} b(t) = \lim_{t\to\infty} \frac{1-na(t)}{m} = \frac{1-na^\infty}{m}.$$

Our assumption $n \leq m$ implies that we have

$$\frac{1}{n+m} \leq \frac{m+3}{2nm+3n+3m} < \frac{1}{n}$$

and that the right hand side of (10) is strictly positive for $a(t)$ in the interval

$$\left[\frac{1}{n+m}, \frac{m+3}{2nm+3n+3m}\right),$$

zero at $a(t) = \frac{m+3}{2nm+3n+3m}$, and strictly negative for $a(t)$ on the interval

$$\left(\frac{m+3}{2nm+3n+3m}, \frac{1}{n}\right).$$

These monotonicity properties force the function $a(t)$ to converge to the limit $a^\infty = \frac{m+3}{2nm+3n+3m}$. □

Example 6 (Flow limit of $K_3 \times K_4$ with simple random walk)**.** *The following code computes the numerical flow limit for $K_3 \times K_4$ with the simple random walk as initial weighting scheme.*

```
n,m = 2,3
A = cart_prod(complete(n+1),complete(m+1))
P = srw(A)
limit = norm_curv_flow_lim(A, P)[0]
display_weighted_graph(A, P, "Initial weighting scheme of K3 x K4")
display_weighted_graph(A, limit, title="Curvature flow limit")
```

The initial transition rates are all equal to $1/5 = 0.2$, and the transition rates of the numerical limit are $0.22\cdots \approx 2/9$ and $0.19\cdots \approx 5/27$, as predicted by the above theorem.

2.6. Totally Degenerate Flow Limits

As mentioned before, many curvature limits are totally degenerate. Therefore, it is worth investigating the properties of those particular limits.

Example 7 (The octahedron with a totally degenerate flow limit)**.** *Let $G = (V, E)$ with $V = \{v_0, \ldots, v_5\}$ be the unmixed graph representing the octahedron, as illustrated in Figure 12 (left hand side). While the simple random walk without laziness is a stationary solution of the normalized curvature flow, any small perturbation of this initial weighting scheme leads to another curvature sharp limit, which is totally degenerate. For example, the initial weighting scheme*

$$P_0 = \begin{pmatrix} 0 & 0.26 & 0 & 0.24 & 0.25 & 0.25 \\ 0.25 & 0 & 0.25 & 0 & 0.25 & 0.25 \\ 0 & 0.25 & 0 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0 & 0.25 & 0 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0 & 0 \end{pmatrix} \tag{11}$$

converges to the totally degenerate limit illustrated in Figure 12 (right-hand side) under the numerical curvature flow.

The following considerations show that the limit in Example 7 is essentially the only totally degenerate curvature sharp weighting scheme without two-sided degenerate edges for the octahedron (see Proposition 3 below). We start with a general unmixed combinatorial graph $G = (V, E)$ without isolated vertices. A totally degenerate weighting scheme P assigns to each edge $\{x, y\} \in E$ either a direction ($x \to y$ if $p_{xy} > 0$ and $y \to x$ if $p_{yx} > 0$, illustrated by a red dashed line with an arrow) or the edge is two-sided degenerate (that is $p_{xy} = p_{yx} = 0$, illustrated by a black dotted line). A first observation is that none of the vertices $x \in V$ can be a sink: the Markovian property requires that at least one edge incident to x must be outward directed. The following lemma presents a useful property of triangles.

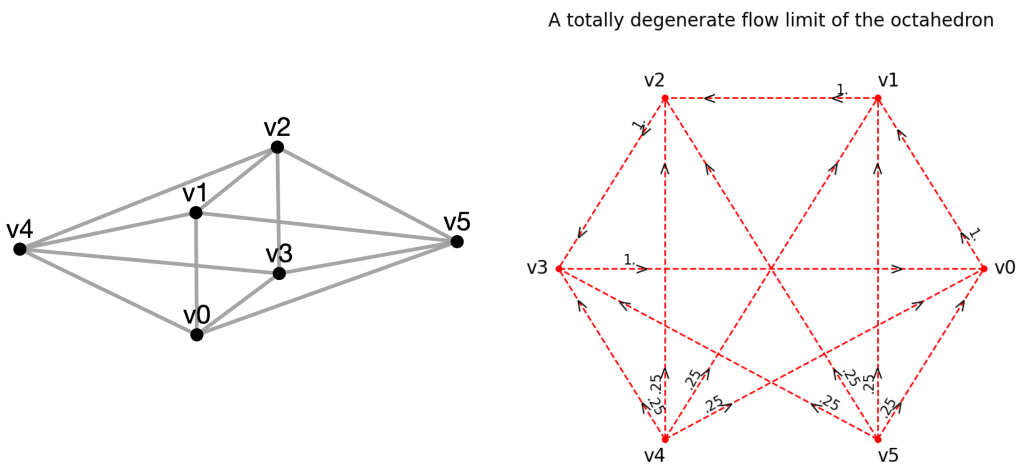

Figure 12. Left-hand side: An octahedron with vertices $v_0, \ldots, v_5$. Right-hand side: A numerical flow limit of the octahedron with a small perturbation of simple random walk as initial weighing scheme.

Lemma 1. *Let (G, P) be a totally degenerate curvature sharp Markovian weighted graph without laziness. Then, the directions of a triangle $T = \{x, y, z\} \subset V$ without two-sided degenerate edges cannot be oriented, that is, its edges cannot have the orientations $x \to y \to z \to x$ or $x \to z \to y \to x$.*

Proof. Assume that a totally degenerate curvature sharp Markovian weighting scheme without laziness contains a triangle $T = \{x, y, z\}$ with $p_{xz}, p_{zy}, p_{yx} > 0$, that is, we have an orientation $x \to z \to y \to x$. This means, in particular, that $p_{xy} = 0$. We can read off the flow equation (9) that curvature sharpness of a totally degenerate Markovian weighting scheme means

$$p_{xy}(-2 \sum_{y' \neq y} p_{yy'} + \sum_{y',y''} p_{xy'} p_{y'y''}) + \sum_{y' \neq y} p_{xy'} p_{y'y} = 0. \tag{12}$$

Since we have $p_{xy} = 0$, this means

$$0 \leq p_{xz} p_{zy} \leq \sum_{y' \neq y} p_{xy'} p_{y'y} = 0,$$

in contradiction to $p_{xz}, p_{zy} > 0$. The orientation $x \to y \to z \to x$ can be ruled out similarly. □

The main tool in the proof of the following proposition can be found in [38] (Exercise 25.14). Assume a tessellation of the two-dimensional sphere carries an orientation along all its edges. For every vertex v of the tessellation, let $\mathrm{ind}(v) = 1 - c(v)/2$, where $c(v)$ is the number of changes in the orientation of edges adjacent to v (in cyclic order). For a face f, let $\mathrm{ind}(f) = 1 - c(f)/2$, where $c(f)$ is the number of changes in the orientation (clockwise vs. anti-clockwise) of edges of f. Then, we have

$$\sum_v \mathrm{ind}(v) + \sum_f \mathrm{ind}(f) = 2.$$

Proposition 3. *Let $G = (V, E)$ be the octahedron, as illustrated in Figure 12 (left-hand side). Then, G has essentially only one totally degenerate non-lazy curvature sharp Markovian weighting scheme without two-sided degenerate edges, namely, we have (up to a permutation of the vertices corresponding to a graph automorphism)*

$$p_{v_0,v_1} = p_{v_1,v_2} = p_{v_2,v_3} = p_{v_3,v_0} = 1$$

and

$$p_{v_4,v_0} = p_{v_4,v_1} = p_{v_4,v_2} = p_{v_4,v_3} = 1/4, \quad p_{v_5,v_0} = p_{v_5,v_1} = p_{v_5,v_2} = p_{v_5,v_3} = 1/4.$$

Proof. We can think of the octahedron as a tessellation of the sphere by eight triangles, all of them not oriented, by Lemma 1. This means that $\mathrm{ind}(f) = 0$ for all faces of the octahedron. Therefore, we must have

$$\sum_v \mathrm{ind}(v) = 2,$$

that is, at least two vertices of the octahedron must have $\mathrm{ind}(v) = 1$, which means that each of them must be a source or a sink. The Markovian property rules out sinks, and two sources cannot be adjacent (otherwise they would be connected by a non-degenerate edge). Therefore, the octahedron must have two sources at distance 2, which we denote by v_4, v_5. Their edges are all directed towards a cycle of length 4. The directions of the edges in

this cycle must be directed, for otherwise there would be a sink in this cycle, which is not possible. We denote this oriented cycle by $v_0 \to v_1 \to v_2 \to v_3 \to v_0$, and we must have

$$p_{01} = p_{12} = p_{2,3} = p_{3,0} = 1,$$

where we used the notation $p_{ij} = p_{v_i,v_j}$ for simplicity. Applying formula (12) to $(x,y) = (v_4, v_0)$ yields

$$p_{40}(-2(p_{01} + p_{03}) + \sum_{i=0}^{3} p_{4i}) + p_{41}p_{10} + p_{43}p_{30} = p_{40}(-2+1) + p_{43} = 0,$$

that is $p_{40} = p_{43}$. Similarly, we can show that all transition rates p_{4i} must coincide and, therefore, $p_{40} = p_{41} = p_{42} = p_{43} = 1/4$. The same arguments apply to the vertex v_5, finishing the proof of the proposition. □

Conjecture 4 (Flow limits of the octahedron)**.** *The normalized curvature flow on the octahedron for any non-degenerate initial weighting scheme without laziness different from the simple random walk always converges to a limit described in Proposition 3.*

3. Asymptotically Stable and Unstable Curvature Sharp Markovian Weighting Schemes

Our curvature flow can be viewed as a continuous dynamical system with its limits as equilibrium states. This leads naturally to stability questions at these equilibria, which is the concern of this section. Further information about the relevance and the background theory of stability theory for dynamical systems can be found, for example, in [39] (Chapter 9).

Since every curvature sharp Markovian weighting scheme on a given combinatorial graph is a stationary solution of the normalized curvature flow, the question arises whether such a stationary solution P^s is *asymptotically stable*, that is, whether any closeby Markovian weigthing scheme P converges back to this equilibrium P^s as $t \to \infty$. This can be decided via the linearization of the curvature flow equations around such an equilibrium. In the next subsection, we will describe this linearization in full detail before we consider various examples in the following subsection.

3.1. Linearization of the Curvature Flow Equations at Equilibria

Let P^s be a curvature sharp Markovian weighting scheme of a finite simple mixed combinatorial graph $G = (V, E)$. We consider Markovian weighting schemes P near P^s with the same laziness, that is, $p_{xx} = p^s_{xx}$ for all $x \in V$. Let $E^{\mathrm{dir}} = \{(x,y) \in V \times V : d(x,y) = 1\}$. The linearization of F at the equilibrium P^s of the normalized curvature flow is given for each component function F_{xy}, $(x,y) \in E^{\mathrm{dir}}$, by

$$\begin{aligned} &DF_{xy}(P^s)(q_{uv})_{(u,v)\in E^{\mathrm{dir}}}) = \\ &\qquad \left(-4p^s_{yx} - 2\sum_{y'\neq y} p^s_{yy'} + \frac{4}{D_x}\sum_{y'} p^s_{xy'}p^s_{y'x} + \frac{1}{D_x}\sum_{y',y''} p^s_{xy'}p^s_{y'y''} - p^s_{yy}\right) q_{xy} \\ &+ p^s_{xy}\left(-4q_{yx} - 2\sum_{y'\neq y} q_{yy'} + \frac{4}{D_x}\sum_{y'}\left(p^s_{xy'}q_{y'x} + p^s_{y'x}q_{xy'}\right) + \frac{1}{D_x}\sum_{y',y''}\left(p^s_{xy'}q_{y'y''} + p^s_{y'y''}q_{xy'}\right)\right) \\ &\qquad = \sum_{y'\in S_1(x)} B_{xy}(xy')q_{xy'} + \sum_{y'\in S_1(x)}\sum_{z\in B_1(x)} B_{xy}(y'z)q_{y'z}. \end{aligned}$$

Here, y', y'' are vertices in $S_1(x)$ and the potentially non-zero B-coefficients are given by

$$\begin{aligned}
B_{xy}(xy) &= -4p^s_{yx} - p^s_{yy} - 2\sum_{y'\neq y} p^s_{yy'} \\
&\quad + \frac{1}{D_x}\left(4p^s_{xy}p^s_{yx} + 4\sum_{y'} p^s_{xy'}p^s_{y'x} + p^s_{xy}\sum_{y'} p^s_{yy'} + \sum_{y',y''} p^s_{xy'}p^s_{y'y''}\right), && (13)\\
B_{xy}(xy') &= \frac{4}{D_x}p^s_{xy}p^s_{y'x} + \frac{1}{D_x}p^s_{xy}\sum_{y''} p^s_{y'y''} + p^s_{y'y} && \text{if } y'\neq y, \quad (14)\\
B_{xy}(yx) &= 4p^s_{xy}\left(\frac{1}{D_x}p^s_{xy} - 1\right), && (15)\\
B_{xy}(y'x) &= \frac{4}{D_x}p^s_{xy}p^s_{xy'} && \text{if } y'\neq y, \quad (16)\\
B_{xy}(yy') &= p^s_{xy}\left(\frac{1}{D_x}p^s_{xy} - 2\right) && \text{if } y'\neq y, \quad (17)\\
B_{xy}(y'y) &= p^s_{xy'}\left(\frac{1}{D_x}p^s_{xy} + 1\right) && \text{if } y'\neq y, \quad (18)\\
B_{xy}(y'y'') &= \frac{1}{D_x}p^s_{xy}p^s_{xy'} && \text{if } y'\neq y \text{ and } y''\neq y, y'. \quad (19)
\end{aligned}$$

All other B-coefficients are chosen to be zero. Note, however, that the transition probabilities $(p_{xy})_{y\in S_1(x)}$ are not independent, and therefore, the choice of the B-coefficients is not unique, as explained in the following remark.

Remark 1. *Since we have $\sum_{v\in S_1(u)} q_{uv} = 0$ for all $u \in V$, there is a degree of freedom in the choice of the B-coefficients. For example, in the case that G is an unmixed complete graph, we can replace $B_{xy}(u,v)$ by $B'_{xy}(u,v) = C_u + B_{xy}(u,v)$ with arbitrary constants C_u. This allows us to modify the B-coefficients $B_{xy}(yy')$ and $B_{xy}(y'y'')$ in (17) and (19) to vanish, and we can use instead*

$$DF_{xy}(P^s)((q_{uv})_{(u,v)\in E^{\mathrm{dir}}}) = \sum_{y'\neq x} B'_{xy}(xy')q_{xy'} + \sum_{y'\neq x} B'_{xy}(y'x)q_{y'x} + \sum_{y'\neq x,y} B'_{xy}(y'y)q_{y'y}$$

with

$$\begin{aligned}
B'_{xy}(xy) &= -4p^s_{yx} - p^s_{yy} - 2\sum_{y'\neq y} p^s_{yy'} \\
&\quad + \frac{1}{D_x}\left(4p^s_{xy}p^s_{yx} + 4\sum_{y'} p^s_{xy'}p^s_{y'x} + p^s_{xy}\sum_{y'} p^s_{yy'} + \sum_{y',y''} p^s_{xy'}p^s_{y'y''}\right),\\
B'_{xy}(xy') &= \frac{4}{D_x}p^s_{xy}p^s_{y'x} + \frac{1}{D_x}p^s_{xy}\sum_{y''} p^s_{y'y''} + p^s_{y'y} && \text{if } y'\neq x,y,\\
B'_{xy}(yx) &= p^s_{xy}\left(\frac{3}{D_x}p^s_{xy} - 2\right),\\
B'_{xy}(y'x) &= \frac{3}{D_x}p^s_{xy}p^s_{xy'} && \text{if } y'\neq x,y,\\
B'_{xy}(y'y) &= p^s_{xy'} && \text{if } y'\neq x,y.
\end{aligned}$$

Note that in the case of the unmixed complete graph, we have $S_1(x) = V\setminus\{x\}$ and, in the formulas for the B'-coefficients, $y'y''$ represent all vertices different from x, as before.

To end up with a uniquely defined Jacobi matrix of F, we need to restrict transition probabilities that are independent. For that, we introduce the subset $E^{\mathrm{ess}} \subset E^{\mathrm{dir}}$ of "essential" transition probabilities by removing, for each $x \in V$ with outgoing di-

rected edges, that is $S_1(x) \neq \emptyset$, one pair (x, y) from E^{dir}. The cardinality of E^{ess} is $M := |E^1| + 2|E^2| - |V_0|$, where $V_0 \subset V$ is the subset of vertices $x \in V$ for which we have $S_1(x) \neq \emptyset$. Any choice $(p_{uv})_{(u,v)\in E^{\mathrm{ess}}}$ determines then a weighting scheme P by setting $p_{uv} = D_u - \sum_{v' \in S_1(u)\setminus\{v\}} p_{uv'}$ for the directed edge $(u, v) \notin E^{\mathrm{ess}}$. Similarly, any choice $(q_{uv})_{(u,v)\in E^{\mathrm{ess}}}$ also determines the parameters q_{uv} with $(u, v) \notin E^{\mathrm{ess}}$ by setting $q_{uv} = -\sum_{v' \in S_1(u)\setminus\{v\}} q_{uv'}$. Then, $DF((p^s_{uv})_{(u,v)\in E^{\mathrm{ess}}})$ is a square matrix of size M, and the weighting scheme P^s corresponding to $(p^s_{uv})_{(u,v)\in E^{\mathrm{ess}}}$ is *asymptotically stable* if and only if the real parts of all eigenvalues of this square matrix are negative, and the weighting scheme P^s is *unstable* if and only if at least one of the real parts of these eigenvalues is positive.

Let us reformulate this restriction in terms of matrix multiplications. We start by enumerating the vertices of the graph $G = (V, E)$: $V = \{v_0, \ldots, v_N\}$. We also introduce the following enumeration on the directed edges in E^{dir}: Let $1 = j_0$ and $a_{j_0}, \ldots, a_{k_0}$ be the edges of the type $(v_0, *)$ (where second vertices are chosen with increasing indices) in E^{dir}, $j_1 = k_0 + 1$ and $a_{j_1}, \ldots, a_{k_1}$ be the edges of the type $(v_1, *)$ in E^{dir}, and so on. For all vertices $v_l \in V$ with $S_1(v_l) = \emptyset$, we set $j_l = k_{l-1} + 1$ and $k_l = k_{l-1}$. We remove the edges $a_{k_0}, a_{k_1}, \ldots, a_{k_N}$ from E^{dir} to obtain E^{ess}. For simplicity, we use the notation p^s_j and q_j for $p^s_{a_j}$ and q_{a_j}, and we can write for all $a_j \in E^{\mathrm{ess}}$,

$$DF_{a_j}((p^s_k)_{a_k\in E^{\mathrm{ess}}})((q_k)_{a_k\in E^{\mathrm{ess}}}) = \sum_{l=0}^{N}\sum_{k=j_l}^{k_l} B_{a_j}(a_k)q_k = \sum_{l=0}^{N}\sum_{k=j_l}^{k_l-1} (B_{a_j}(a_k) - B_{a_j}(a_{k_l}))q_k,$$

where the last expression involves only parameters q_k corresponding to essential directed edges $a_k \in E^{\mathrm{ess}}$. Consequently, the Jacobi matrix $DF((p^s_k)_{a_k\in E^{\mathrm{ess}}})((q_k)_{a_k\in E^{\mathrm{ess}}})$ can be written as

$$DF((p^s_k)_{a_k\in E^{\mathrm{ess}}}) = P_1 B P_2 \tag{20}$$

where B is the square matrix of size k_N with $B_{jk} = B_{a_j}(a_k)$ for $a_j, a_k \in E^{\mathrm{dir}}$, P_1 is obtained from the identity matrix I_{k_N} by removing the rows $k_0, k_1, \ldots, k_N$, and $P_2 = P_1^\top - P_3$ with P_3 a $k_N \times M$ matrix whose first $k_0 - 1$ columns are all the standard basis vector e_{k_0}, the next $k_1 - j_1$ columns are all the standard basis vector e_{k_1}, and so on.

3.2. Examples of Asymptotically Stable and Unstable Equilibria

In this subsection, we investigate curvature sharp weighting schemes of various examples of unmixed combinatorial graphs.

Example 8 (Curvature sharp weighting schemes on a cycle)**.** *Let $C_N = (V, E)$ be a cycle of length $N \geq 4$, that is, $V = \{v_0, v_1, \ldots, v_{N-1}\}$ and $v_i \sim v_{i+1}$ with indices i modulo N. For simplicity, we refer to vertex v_i henceforth as i. We assume P^s to be a non-lazy curvature sharp weighting scheme on C_N, and we remove the directed edges $(i, i+1) \in E^{\mathrm{dir}}$ to obtain E^{ess}, and the only coefficients $B_{a_j}(a_k)$ with $a_j \in E^{\mathrm{ess}}$ and $a_k \in E^{\mathrm{dir}}$, which may be potentially non-zero are (see (13)–(16)),*

$$\begin{aligned}
B_{i,i-1}(i, i-1) &= -4p^s_{i,i-1} + 8p^s_{i,i-1}p^s_{i-1,i} + 4p^s_{i,i+1}p^s_{i+1,i},\\
B_{i,i-1}(i, i+1) &= 4p^s_{i,i-1}p^s_{i+1,i},\\
B_{i,i-1}(i-1, i) &= 4(p^s_{i,i-1})^2 - 4p^s_{i,i-1},\\
B_{i,i-1}(i+1, i) &= 4p^s_{i,i-1}p^s_{i,i+1}.
\end{aligned}$$

This implies

$$\begin{aligned}
&DF_{i,i-1}((p^s_k)_{a_k\in E^{\mathrm{ess}}})((q_k)_{a_k\in E^{\mathrm{ess}}})\\
&= (B_{i,i-1}(i, i-1) - B_{i,i-1}(i, i+1))q_{i,i-1} + B_{i,i-1}(i+1, i)q_{i+1,i} - B_{i,i-1}(i-1, i)q_{i-1,i-2}\\
&= 4((p^s_{i,i-1})^2 - p^s_{i,i-1})q_{i-1,i-2} + 4(-p^s_{i,i-1} + 2p^s_{i,i-1}p^s_{i-1,i} + p^s_{i,i+1}p^s_{i+1,i} - p^s_{i,i-1}p^s_{i+1,i})q_{i,i-1}\\
&\qquad + 4p^s_{i,i-1}p^s_{i,i+1}q_{i+1,i}.
\end{aligned}$$

In the case of the simple random walk $p^s_{i,i-1} = p^s_{i,i+1} = 1/2$, $i \in \{0, \dots, N-1\}$, *this simplifies to*

$$DF_{i,i-1}((p^s_k)_{a_k \in E^{\text{ess}}})((q_k)_{a_k \in E^{\text{ess}}}) = q_{i-1,i-2} + q_{i+1,i},$$

and the corresponding matrix $DF((p^s_k)_{a_k \in E^{\text{ess}}})$ *coincides with the adjacency matrix of* C_N *whose largest eigenvalue is* 2. *Therefore, the simple random walk on* C_N, $N \geq 4$, *is an unstable equilibrium. This is in contrast to the simple random walk on* $C_3 = K_3$, *which is asymptotically stable, as we will see in Example 10.*

In the case of the totally degenerate clockwise weighting scheme $p_{i,i-1} = 1$ *and* $p_{i,i+1} = 0$ *(see right-hand side of Figure 4), we have*

$$DF_{i,i-1}((p^s_k)_{a_k \in E^{\text{ess}}})((q_k)_{a_k \in E^{\text{ess}}}) = -4q_{i,i-1},$$

that is, $DF((p^s_k)_{a_k \in E^{\text{ess}}}) = -4\text{Id}_N$, *and this curvature sharp Markovian weighting scheme is asymptotically stable. This agrees with the fact that many initial weighting schemes end up in this limit under the curvature flow. The same holds true for the corresponding totally degenerate anti-clockwise weighting scheme with* $p_{i,i+1} = 1$ *and* $p_{i,i-1} = 0$.

Example 9 (Flow limits of the octahedron). *We know from Example 7 that stationary solutions of the normalized curvature flow on the octahedron are the simple random walk without laziness as well as the totally degenerate weighting scheme given as matrix* P^s *in the following code (see also the right-hand side of Figure 12). The linearization* $DF(P^s)$ *is analyzed in line 10 of the program, and the function* `equilibrium_type` *with the parameters chosen in lines 6 and 7 returns one of the values* $-1, 0, 1$ *(corresponding to "asymptotically stable", "undecided", "unstable", respectively), followed by a list of its eigenvalues. That is, after execution of line 10,* `result[0]` *is one of the values* $-1, 0, 1$ *and* `result[1]` *is a list of the 18 eigenvalues* λ_j *of* $DF(P^s)$.

```
1  A = [[0,1,0,1,1,1],[1,0,1,0,1,1],[0,1,0,1,1,1],
2       [1,0,1,0,1,1],[1,1,1,1,0,0],[1,1,1,1,0,0]]
3  # Ps = srw(A)
4  Ps = [[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],
5        [1,0,0,0,0,0],[1/4,1/4,1/4,1/4,0,0],[1/4,1/4,1/4,1/4,0,0]]
6  eigenvalues = True
7  jacobi_matrix = False
8  threshold = 0.001
9  norm_tolerance = 0.001
10 result = equilibrium_type(A,Ps,eigenvalues,jacobi_matrix,norm_tolerance,threshold)
11 print("Flow dynamics eigenvalues:")
12 for j in range(18):
13     print(np.around(result[1][j],3))
```

The program provides us with the following list of complex eigenvalues:

λ_j	-1	$-1+i$	$-1-i$	-2	$-2+i$	$-2-i$	-3	-4
multiplicity	2	2	2	2	2	2	2	4

This shows that the totally degenerate curvature sharp weighting scheme P^s *on the octahedron is asymptotically stable. This is expected since the initial weighting scheme* P_0 *in (11) of Example 7 converges to this limit under the normalized numerical curvature flow.*

Running the same code for the simple random walk without laziness instead (by uncommenting line 3 and commenting out lines 4 and 5 in the above code) shows that this second curvature sharp weighting scheme is unstable. The eigenvalues, in this case, are all real-valued, one of them 0.5, *and given as follows:*

λ_j	0.5	0	-0.75	-1	-1.5
multiplicity	3	3	2	6	4

Example 10 (Simple random walk on a complete graph). *Let $K_{n+1} = (V, E)$ be the complete unmixed graph with $n+1 \geq 3$ vertices. Instead of the B-coefficients, we make use of the B'-coefficients introduced in Remark 1. The simple random walk $p_{xy} = \frac{1}{n}$ for $x \neq y$ is a curvature sharp Markovian weighting scheme, and the non-zero B'-coefficients are then given by*

$$B'_{xy}(xy) = \frac{(n+1)(3-n)}{n^2},\ B'_{xy}(xy') = \frac{2n+3}{n^2},\ B'_{xy}(yx) = \frac{3-2n}{n^2},\ B'_{xy}(y'x) = \frac{3}{n^2},\ B'_{xy}(y'y) = \frac{1}{n},$$

where $y' \in V$ is an arbitrary vertex different from x, y. Let $\{v_0, v_1, \ldots, v_n\}$ be the vertex set of K_{n+1} and, as in the previous example, we refer to vertex v_i as i, for simplicity.

Let us now consider the case $n = 2$. The process of removing edges from E^{dir} described earlier leads to the following remaining edges in E^{ess}:

$$a_1 = (0,1), \quad a_3 = (1,0), \quad a_5 = (2,0).$$

Choosing the simple random walk $p^s_{01} = p^s_{10} = p^s_{20} = 1/2$, we have

$$DF(p^s_{01}, p^s_{10}, p^s_{20}) = \begin{pmatrix} e_1^\top \\ e_3^\top \\ e_5^\top \end{pmatrix} B' \begin{pmatrix} e_1 - e_2 & e_3 - e_4 & e_5 - e_6 \end{pmatrix}$$
$$= \begin{pmatrix} B'_{01}(01) - B'_{01}(02) & B'_{01}(10) - B'_{01}(12) & B'_{01}(20) - B'_{01}(21) \\ B'_{10}(01) - B'_{10}(02) & B'_{10}(10) - B'_{10}(12) & B'_{10}(20) - B'_{10}(21) \\ B'_{20}(01) - B'_{20}(02) & B'_{20}(10) - B'_{20}(12) & B'_{20}(20) - B'_{20}(21) \end{pmatrix} = \begin{pmatrix} -1 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -1 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -1 \end{pmatrix}, \quad (21)$$

which is a negative definite matrix with eigenvalues $-\frac{1}{2}, -\frac{5}{4}, -\frac{5}{4}$, showing that the simple random walk on K_3 is an asymptotically stable equilibrium. This result can be numerically verified via the following code:

```
1  n = 2
2  A = complete(n+1)
3  P = srw(A)
4  result = equilibrium_type(A, P, True, True)
5  print()
6  print("Eigenvalues:")
7  print(np.around(result[1], 3))
8  print()
9  print("Linearised flow matrix at equilibrium:")
10 print(np.around(result[2], 3))
```

The program returns the eigenvalues of the linearized flow matrix of K_3 at the simple random walk. Note, however, that the computed matrix is based here on the B-coefficients instead of the B'-coefficients, so this matrix is slightly different from the one given above, while the eigenvalues are the same.

In the case $n = 3$, we have $p^s_{01} = p^s_{02} = p^s_{10} = p^s_{12} = p^s_{20} = p^s_{21} = p^s_{30} = p^s_{31} = 1/3$ and the returned matrix by the program (after changing `n=2` *into* `n=3` *in line 1 of the code) is as follows:*

$$DF(p^s_{01}, p^s_{02}, p^s_{10}, p^s_{12}, p^s_{20}, p^s_{21}, p^s_{30}, p^s_{31}) = \begin{pmatrix} -1 & 0 & -1/3 & 0 & 1/3 & 1/3 & 1/3 & 1/3 \\ 0 & -1 & 1/3 & 1/3 & -1/3 & 0 & 0 & -1/3 \\ -1/3 & 0 & -1 & 0 & 1/3 & 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & -1 & 0 & -1/3 & -1/3 & 0 \\ 0 & -1/3 & 1/3 & 1/3 & -1 & 0 & 0 & -1/3 \\ 1/3 & 1/3 & 0 & -1/3 & 0 & -1 & -1/3 & 0 \\ 1/3 & 1/3 & 0 & -1/3 & 0 & -1/3 & -1 & 0 \\ 0 & -1/3 & 1/3 & 1/3 & -1/3 & 0 & 0 & -1 \end{pmatrix}.$$

Its eigenvalues are $-\frac{2}{3}$ with multiplicity 6 and -2 with multiplicity 2. This shows that the simple random walk on K_4 is again an asymptotically stable equilibrium.

Since numerical experiments show that all non-degenerate Markovian weighting schemes without laziness on K_{n+1} converge to the simple random walk, we expect that the simple random walk on K_{n+1} is an asymptotically stable equilibrium for all $n \geq 2$ (see also our Conjecture 3, which is an even stronger statement). To this end, our code provides the following numerical results: The eigenvalues of $DF(P^s)$ for the simple random walk without laziness on the complete graph K_{n+1} are real-valued and given by:

- $-\frac{7}{16}$ with multiplicity 4, $-\frac{3}{4}$ with multiplicity 6, and $-\frac{7}{4}$ with multiplicity 5 for $n = 4$;
- $-\frac{8}{25}$ with multiplicity 5, $-\frac{4}{5}$ with multiplicity 10, and $-\frac{8}{5}$ with multiplicity 9 for $n = 5$;
- $-\frac{1}{4}$ with multiplicity 6, $-\frac{5}{6}$ with multiplicity 15, and $-\frac{3}{2}$ with multiplicity 14 for $n = 6$;
- $-\frac{10}{49}$ with multiplicity 7, $-\frac{6}{7}$ with multiplicity 21, and $-\frac{10}{7}$ with multiplicity 20 for $n = 7$.

These results give rise to the following conjecture:

Conjecture 5. *The non-lazy simple random walk P^s on the unmixed complete graph K_{n+1}, $n \geq 2$, is asymptotically stable and the eigenvalues of $DF(P^s)$ are given by*

$$\begin{aligned} &-\frac{n-1}{n} && \text{with multiplicity } \binom{n}{2}, \\ &-\frac{n+3}{n^2} && \text{with multiplicity } n, \\ &-\frac{n+3}{n} && \text{with multiplicity } \binom{n}{2} - 1. \end{aligned}$$

As $n \to \infty$, we have $-\frac{n-1}{n} \to -1$, $-\frac{n+3}{n^2} \approx -\frac{1}{n} \to 0$ and $-\frac{n+3}{n} \to -1$.

Example 11 (Simple random walks on hypercubes)**.** *We know from [1] (Corollary 1.14) that the simple random walk without laziness on a hypercube $Q^d = (K_2)^d$ is curvature sharp, and it is the only non-degenerate curvature sharp weighting schemes without laziness if and only if d is odd. For $d = 2$, a non-degenerate curvature sharp weighting scheme on Q^2 different from the simple random walk is given in line 4 of the following code:*

```
1 A = hypercube(2)
2 p = rand()
3 q = 1-p
4 P = [[0,p,q,0],[p,0,0,q],[p,0,0,q],[0,p,q,0]]
5 result = equilibrium_type(A,P,True)
6 print("flow dynamics eigenvalues:", result[1])
```

All eigenvalues λ_j of the corresponding linearized flow matrix provided by this code via line 6 seem to be real. Two of them are numerically zero and the other two are given by $\pm\lambda$ with a non-zero real λ. So this equilibrium is unstable.

Our experiments for arbitrary weighting schemes on Q^d, $d \geq 2$, close to the simple random walk show that the numerical flow converges usually to a degenerate limit. This agrees with our observation that the linearized flow matrix at the simple random walk always seems to have real eigenvalues with some of them being positive. These eigenvalues can be obtained via the following code:

```
d = 3
A = hypercube(d)
P = srw(A)
result = equilibrium_type(A,P,True)
print("Flow dynamics eigenvalues at the simple random walk:")
for j in range((d-1)*(2**d)):
    print(np.around(result[1][j],4))
```

Let us finish this section with a conjecture that we verified numerically for $d = 2, 3, \ldots, 9$:

Conjecture 6. *The non-lazy simple random walk P^s on the hypercube Q^d, $d \geq 2$, is unstable and the eigenvalues of the corresponding linearized curvature flow matrix are all real with the largest eigenvalue $\lambda_{\max}$ equals*

$$\lambda_{\max} = \frac{4}{d}.$$

4. Conclusions and Open Questions

In Section 2, we investigated curvature flows of various concrete examples. Our findings and possible directions for further research are as follows:

(a) *Random graphs:* We observed an edge density threshold for Erdös-Rényi graphs. For values below this threshold, the flow usually converges to a totally degenerate limit, while for values above this threshold, the flow limit concentrates on a non-degenerate cluster (see Conjecture 2). Further research could encompass curvature flows of random regular graphs with simple random walks as their initial weighting schemes.
(b) *Paths and cycles:* Our observed limits for these graphs are usually directed graphs with isolated clusters of three vertices (see Figure 4 and Proposition 1).
(c) *Complete graphs and their wedge sums and Cartesian products:* In the case of complete graphs, non-degenerate initial weighting schemes seem to converge always to simple random walks (see Conjecture 3). In wedge sums of complete graphs, curvature flow limits usually concentrate totally on one of its components, which is most often the largest component (see Table 2). This observation might motivate further research on the largest clusters of general highly connected graphs. We also describe the flow limit of Cartesian products of complete graphs with the simple random walk as its initial weighting scheme (see Theorem 3). A natural question here is the limit behavior of Cartesian products of graphs with curvature sharp initial weighting schemes.
(d) *Octahedron:* In the case of the octahedron, there is a unique totally degenerate flow limit (up to symmetries), which has two sources and a directed 4-cycle (see Figure 12 and Proposition 3). The shape of totally degenerate flow limits for general graphs is obscure and would benefit from further investigations.

As mentioned already in [1], curvature flow limits are always curvature sharp. A rough interpretation of curvature sharpness at a vertex is that its Ricci curvature is constant in all directions (see [1] Equation (8)). Section 3 was concerned with a dynamical aspect of these curvature flow limits, namely their stability properties. Our findings are the following:

(e) *Cycles:* Simple random walks on cycles C_n for $n \geq 4$ are unstable, and totally degenerate directed weighting schemes are asymptotically stable (see Example 8).
(f) *Complete graphs:* We have strong numerical evidence that simple random walks on complete graphs K_n are asymptotically stable (numerically verified for $n = 3, \ldots, 8$); see Example 10. Moreover, our numerics provide explicit information about all eigenvalues of the linearized curvature flow as stated in Conjecture 5.
(g) *Hypercubes:* Simple random walks of hypercubes Q^d for $d \geq 2$ seem to be unstable and our experiments give rise to a conjecture about the largest eigenvalue of the linearized curvature flow (see Conjecture 6).
(h) *Octahedron:* The totally degenerate flow limit of the octahedron is asymptotically stable (see Example 9).
(i) *Edge-transitive graphs:* In view of (e), (f), and (g), an interesting research direction would be to investigate the stability of the simple random walk for general edge-transitive graphs.

Let us end this section with some final reflections: After having introduced a discrete curvature flow based on the Bakry–Émery calculus in [1], the main concern of this paper

was to gain further insights into this flow from a dynamical viewpoint using a specially developed Python software. The most pressing and still open question is whether our curvature flow is always convergent as time tends to infinity (see Conjecture 1), or whether other limiting behavior may occur such as periodic orbits. Moreover, this could be just the starting point of various further questions, for example, whether every curvature flow allows for asymptotically stable limits—a natural technique here might be to use particular Lyapunov functions (see, e.g., [39] (Section 9.3)). The dynamical viewpoint also gives rise to other questions such as the existence of heteroclinic orbits between equilibria. All these questions may lead to further research in the future.

5. Implementation of the Curvature Flow and Useful Tools

This section provides a short description of a program, written in Python, which allows users to carry out their own curvature flow experiments. This program is provided as an ancillary file of [2] or via github at https://github.com/georgestagg/graph-curvature-server (accessed on 6 June 2023) and can be used in combination with the "Graph Curvature Calculator", which can be freely accessed at http://www.mas.ncl.ac.uk/graph-curvature (accessed on 6 June 2023) .

The Graph Curvature Calculator is a powerful but easy-to-use interactive Web tool to draw graphs and to compute various types of curvatures such as Bakry–Émery curvature on its vertices or Ollivier Ricci curvature on its edges (for details see [40]). This Web tool allows users to obtain the adjacency matrix of the graph under consideration, which can then be used as input for the curvature flow code.

The curvature flow code provides functions and routines, which can be divided into six categories:

1. Functions related to graphs and their adjacency matrices;
2. Functions related to weighting schemes;
3. Functions testing combinatorial and weighted graphs;
4. Curvature flow computation routines;
5. Curvature computation routines;
6. Display routines.

Each category is discussed in one of the following subsections.

5.1. Functions Related to Graphs and Their Adjacency Matrices

Recall that the topology of a given Markov chain (G, P) is contained in the combinatorial graph $G = (V, E)$. This information is provided by the adjacency matrix $A = A_G$ of the graph. Firstly, we go through some useful functions performing operations with these adjacency matrices. Every graph generated by one of the functions below is returned as such an adjacency matrix; in particular, as a NumPy array. The names of these functions are as follows:

- `rand_adj_mat(n, p, connected=False);`
- `complete(n);`
- `path(n);`
- `cycle(n);`
- `wedge_sum(A, B, i, j);`
- `bridge_at(A, B, i, j);`
- `hypercube(n);`
- `cart_prod(A, B);`
- `onespheres(A).`

`rand_adjmat(n, p)` returns a random adjacency matrix with `n` vertices, where `p` is the probability that an edge exists between any two vertices. Therefore, higher values of `p` usually lead to better-connected graphs. Choosing `connected=True` instead guarantees that the returned adjacency matrix provides a connected combinatorial graph.

The functions `complete(n)`, `path(n)` and `cycle(n)` return the adjacency matrix of a complete graph with `n` vertices, a path of length `n` and a cycle of length `n`, respectively.

`wedge_sum(A, B, i, j)` produces an adjacency matrix of a graph that is the wedge sum of two pointed graphs represented by the adjacency matrices `A` and `B` and the `i`th vertex of `A` and the `j`th vertex of `B`, that is, the new graph is obtained from the disjoint union of these two graphs by only identifying these two vertices and keeping all other edges and vertices disjoint.

There is also `bridge_at(A, B, i, j)`, which returns an adjacency matrix of the graph formed by connecting the graphs represented by `A` and `B` by a single edge at the `i`th vertex of `A` and the `j`th vertex of `B`.

The function `hypercube(n)` returns the adjacency matrix of the `n`-dimensional hypercube.

The Cartesian product of two graphs represented by adjacency matrices `A` and `B` is returned by the function `cart_prod(A, B)`. (The `n`-dimensional hypercube can also be alternatively generated as the Cartesian product of n complete graphs K_2.)

Finally, for a graph $G = (V, E)$ given by the adjacency matrix `A` and the vertex set $V = \{0, 1, \ldots, n-1\}$, the function `onespheres(A)` returns a list of lists whose i-th entry is a list of all neighbors of vertex $i \in V$, and whose n-th entry is a list of the combinatorial degrees of the vertices in V. This function is mainly used in the execution of other functions.

5.2. Functions Related to Weighting Schemes

The weighting scheme of a Markov chain (G, P) is provided via the weighted matrix $P = P_G$. The functions in this category are the following:

- `randomizer(A, threshold=0.001, laziness=False);`
- `srw(A, laziness=False);`
- `cart_prod_prob(P, Q, p, q).`

`randomizer(A)` and `srw(A)` are two useful functions that can be used to give weighted matrices from an adjacency matrix.

The function `randomizer(A)` returns a weighting scheme for the graph G with random, numerically non-degenerate transition rates, that is, no transition rate is chosen to be below the parameter `threshold`. Usually, the returned weighting schemes are without laziness, but choosing `laziness=True` returns weighting schemes with all vertices having laziness $\geq$ `threshold`.

`srw(A)` returns the weighting scheme corresponding to the non-lazy simple random walk on the graph $G = (V, E)$ represented by the adjacency matrix `A`. Choosing `laziness=True`, the transition rates of a vertex $v \in V$ with degree n to its neighbors are chosen to be $\frac{1}{n+1}$ and its laziness is also chosen to be $\frac{1}{n+1}$.

The function `cart_prod_prob(P, Q, p, q)` is a "weighted" analog of the function `cart_prod` from the previous subsection for the Cartesian product of two weighting schemes `P`, `Q` with weights `p`, `q`, with `p + q` $= 1$. If `P` and `Q` are of size n and m, respectively, this function returns the matrix `p`$P \otimes I_n +$ `q`$I_m \otimes$ `Q` of size nm, where I_n is the identity matrix of size n and $A \otimes B$ is the Kronecker product of A and B.

5.3. Functions Testing Combinational and Weighted Graphs

For combinatorial graphs given by their adjacency matrices `A` and weighted graphs given additionally by their weighting scheme `P`, we have the following test functions:

- `is_connected(A);`
- `is_weakly_connected(A, threshold=0.001);`
- `is_totally_degenerate(A, P, threshold=0.001);`
- `is_markovian(P, norm_tolerance=0.001);`
- `is_curvature_sharp(A, P, norm_tolerance=0.001, threshold=0.001);`
- `equilibrium_type(A, P, eigenvalues=False, jacobi_matrix=False, norm_tolerance=0.001, threshold=0.001).`

The function `is_connected(A)` returns `True` if and only if the adjacency matrix `A` represents a connected graph.

The function `is_weakly_connected(P)` is a "weighted" analog, which returns `True` if and only if the weighted matrix `P` represents a weakly connected graph. It does this by forming an adjacency matrix with a `1` in the `(i,j)`th entry if and only if `P[i, j] > threshold` or `P[j, i] > threshold` and then testing for connectedness.

Recall that a weighted graph (G, P) is called *numerically totally degenerate* if there are no two-sided edges with numerical non-zero transition rates in both directions and no one-sided edges with numerical non-zero transition rate, where we consider a transition rate p_{xy} as numerically non-zero if and only if $p_{xy} \geq$ `threshold`. The function `is_totally_degenerate(A,P)` tests this property.

The function `is_markovian(P)` tests whether the entries of each of the columns of P add up numerically to 1 up to an error $\leq$ `norm_tolerance`.

Numerical curvature sharpness (up to an error $\leq$ `threshold`) of Markovian weighted graphs given by `(A,P)` is tested by `is_curvature_sharp(A,P)`. If `(A,P)` fails to be Markovian (with respect to `norm_tolerance`), this function returns `NONE` and gives notice to the user by an error message.

For dynamical investigations of curvature flow equilibria, we have the function `equilibrium_type(A,P)`, which always returns a list of length three. The function checks first whether `(A,P)` satisfies the Markovian property and is numerically curvature sharp. If this is not the case, it returns a list of three `NONE` values. Otherwise, the function investigates the real parts of the eigenvalues λ_j of the linearized curvature flow matrix at the equilibrium `P`. The first entry of the return list is $-1, 0$ or 1 depending on the maximum $\max_j \mathrm{Re}(\lambda_j)$. If this maximum is $\geq$ `threshold`, the return value is 1 (for "unstable") and if this maximum is $\leq -$`threshold`, the return value is -1 (for "asymptotically stable"). Otherwise, the dynamical nature of the equilibrium cannot be numerically decided and the function returns the value 0. The following two entries of the return list are usually `NONE` unless the user made the choices `eigenvalues=True` or `jacobi_matrix=True`. In the first case, the second entry of the return list is a list of all eigenvalues of the linearized curvature flow matrix, and in the second case the third entry of the return is the linearized curvature flow matrix itself.

5.4. Curvature Flow Computation Routines

At the heart of the program are the curvature flow routines solving the initial value ordinary differential Equations (6) and (7). The relevant routines are the following:

- `curv_flow(A, P, t_max, dt=0.3, C=zeroes)`,
- `norm_curv_flow(A, P, t_max, dt=0.3, stoch_corr=True, norm_tolerance=0.001)`,
- `norm_curv_flow_lim(A, P, dt=0.3, stoch_corr=True, norm_tolerance=0.001, lim_tolerance=0.001, t_lim=10000)`.

The initial Markov chain (G, P_0) with $G = (V, E)$ is entered by the adjacency matrix `A` describing the topology of the graph G and the weighting scheme `P` containing the initial probability transitions $p_{xy}(0)$.

The first routine `curv_flow(A,P)` computes the non-normalized numerical curvature flow with coefficients $C_x(t) = 0$ in (7). If users decide to choose other coefficient functions, they need to modify the input parameter `C=zeroes`. Note that `zeroes(A,P)` is a function returning simply a list of zeroes of length $|V|$. Users can investigate modifications of the curvature flow by choosing their own coefficient functions with input values `(A,P)` and returning a list of length $|V|$ of real values. Using the discretization parameter `dt=0.3` for the discrete time steps starting at `t=0`, the curvature flow routine creates a list `P_list` of weighting schemes (represented by NumPy arrays of size $|V| \times |V|$) at each time increment using the Runge–Kutta algorithms RK4. There are two internal subroutines involved, which we would like to mention briefly. `Pvecs_to_P` translates a weighting scheme given by a list of lists (where each inner list contains the transition rates of the corresponding vertex) into the corresponding NumPy array. `Pvecs_prime` computes, for a given weighting

scheme `P`, the right-hand side of the ordinary differential equations describing the curvature flow. The representation of this right-hand side is again a list of lists, as described before. The computation stops just before the discrete time steps exceed the limit time `t_max` and the routine returns `P_list`. Note that in this general setting, transition rates can assume arbitrary values and even negative ones or diverge to infinity in finite time which may lead to system error messages. Users need to be aware of this possibility.

The normalized curvature flow, using the coefficients $C_x(t) = K^{d(x,\cdot)}_{P(t),\infty}(x)$ (see (8)), is numerically computed by the routine `norm_curv_flow(A,P)`. This special flow is the main focus of this paper and could also be mimicked by choosing `C=K_inf` in `curv_flow`. The function `K_inf(A,P)` returns a list of upper curvature bounds for all vertices of the Markovian weighted graph represented by `(A,P)` (see [1] (Formula (66)) for an explicit expression of this function in terms of transition rates). During the numerical computations of subsequent time steps, the Markovian property of the corresponding weighting schemes may be slightly violated. If this violation exceeds the threshold `norm_tolerance`, the routine prepares for a potential correction according to the Boolean variable `stoch_corr`. If `stoch_corr=False`, the program stops with a message to the user as discussed in Example 1. Otherwise, the program carries out the following automatic Markovian renormalization of the currently considered weighting scheme: while the diagonal entries (the laziness values) are unchanged, the off-diagonal entries in each row are rescaled by a suitable factor to guarantee that the resulting matrix becomes stochastic again. As before, this routine returns a list `P_list` of consecutive weighting schemes up to the time limit `t_max`.

While the user needs to specify the time limit `t_max` in the above two routines, the third routine `norm_curv_flow_lim(A,P)` continues computing the normalized numerical curvature flow until a numerical flow limit is reached. This limit is determined by the parameter `lim_tolerance`. The details for this numerical limit are explained in the introductory part of Section 2. This routine returns a list of length two: the limiting weighting scheme as a NumPy array followed by the numerical convergence time. Since it may happen that a normalized numerical flow does not converge at all (even though we are not aware of any such example), the parameter `t_lim` provides an upper time limit beyond which the routine will not continue. The parameters `stoch_corr` and `norm_tolerance` play the same role as in the routine `norm_curv_flow`.

5.5. Curvature Computation Routines

The functions in this section calculate Bakry–Émery curvatures and curvature upper bounds of graphs with given weighting schemes at all vertices.

- `curvatures(A, P, N=inf, onesps=[], q=None);`
- `calc_curvatures(A, P_list, N=inf, k=1);`
- `K_inf(A, P);`
- `calc_curv_upper_bound(A, P_list, N=inf, k=1).`

The routine `curvatures(A,P)` computes, for a weighted graph (G, P) with $G = (V, E)$ and represented by `(A,P)`, the curvatures of all vertices for dimension $\mathtt{N} = \infty$ and returns them as a list of length $|V|$. If users are interested in curvatures for other dimensions, they need to change the parameter `N=inf`. There are two other inputs that can speed up the curvature calculations: if the number `q` of vertices in V is given, it can be specified to avoid its repeated recalculation, for example, during a curvature flow process. Similarly, if `onespheres(A)` has already been calculated earlier, this information can be communicated to the routine via the input variable `onesps`.

After a curvature flow computation with corresponding list `P_list` of consecutive weighting schemes, the routine `calc_curvatures(A,P_list)` computes the corresponding evolution of vertex curvatures by calling `curvatures(A,P_list[j])` and returns it as a list of lists. Here, the j-th inner list contains the curvature evolution of the j-th vertex of the graph $G = (V, E)$ represented by `A`. The dimension parameter `N=inf` plays the same

role as before. `calc_curvatures` computes curvatures only of each `k`-th weighting scheme provided by `P_list`. Where appropriate, this can help to reduce computation time.

The routine `K_inf(A,P)` was already discussed in the previous subsection and provides a list of upper curvature bounds for all vertices of the weighted graph (G, P) represented by `(A,P)`.

`calc_curv_upper_bound` is completely analogous to `calc_curvatures`, but it calls `K_inf` instead of `curvatures`.

5.6. Display Routines

The main display routines for users are the evolution of curvatures at various vertices during the curvature flow, the evolution of transition rates of edges emanating from vertices, and the display of individual weighted graphs with vertices arranged in a circle. The relevant routines are the following:

- `display_curvatures(curv, dt=0.3, is_Markovian=True, N=inf, k=1, curv_bound=[], vertex_list=[]);`
- `display_trans_rates(A, P_list, dt=0.3, vertex_list=[]);`
- `display_weighted_graph(A, P, title=None, threshold=10**(-3), display_options=[10, True, 2, []], laziness=False).`

Given the evolution of vertex curvatures during a curvature flow process via a list of lists, where the j-th inner list is the curvature evolution of the j-th vertex, `display_curvatures(curv)` displays the curvature evolution for each consecutive vertex separately, as illustated, for example, in Figure 3. If this information should be only given for specific vertices, this can be specified by the input parameter `vertex_list`. The time step `dt=0.3` and the value of `k` together determine the labeling of the horizontal time axis. For the role of `k`, we refer readers to our explanation about the routine `calc_curvatures`. Upper curvature bounds can be inserted into the displays by the input parameter `curv_bound`, which needs to be given in the same format as the vertex curvatures. Constant lower and upper curvature bounds -1 and 2 are plotted alongside if the Boolean `is_Markovian` is chosen to be `True` and if the dimension parameter `N` is ≥ 2. These bounds appear, for example, in the illustrations given in Figure 3.

Given the evolution of weighting schemes during a curvature flow process on a graph with adjacency matrix `A` by a list `P_list` of NumPy arrays, `display_trans_rates(A,P_list)` displays the evolution of transition rates of emanating edges for each consecutive vertex separately, as illustrated, for example, in Figure 2. The input parameters `dt` and `vertex_list` play the same role as in the previous routine.

Finally, there is the routine `display_graph(A,P)` with input parameters `A` and `P` representing a weighted graph (G, P). This routine produces a MatPlotLib plot of this weighted graph, with the vertices arranged counter-clockwise in a circle. The plot uses the following convention to illustrate different types of edges:

- Green, solid lines represent numerically non-degenerate edges, that is $\{x, y\} \in E$ with both $p_{xy}, p_{yx} \geq$ `threshold`;
- Red, dashed lines with an arrow represent numerically degenerate edges, that is, $\{x, y\} \in E$ with exactly one of p_{xy} and p_{yx} strictly less than `threshold`;
- Black, dotted lines represent edges with numerically vanishing transition rates in both directions, that is $\{x, y\} \in E$ with both p_{xy}, p_{yx} strictly less than `threshold`.

Users can add a title to the display by specifying the input parameter `title`. For Markovian weighted graphs with non-vanishing laziness, the option `laziness=True` labels each vertex with its corresponding laziness.

The input parameter `display_options` of the `display_graph` routine remains to be discussed. This parameter is a list of four entries. The first entry determines the size of the plot. Usually, the transition rates are printed above the edges, but if the second entry is chosen to be `False`, this information about the transition rates is omitted. Otherwise, the transition rates are given to a number of decimal places determined by the third entry.

The default positions of these transition rates are $\frac{1}{6}$ of the way along the edges, with the number closest to the vertex x in the edge (x, y) being p_{xy}, but this can be altered manually to avoid overlapping by specification in the fourth entry of `display_options`. For example, if one wishes the p_{45} label to be moved to a position $\frac{1}{4}$ of the way from vertex 4 to vertex 5 and the p_{62} label to be moved to a position $\frac{1}{5}$ of the way from vertex 6 to vertex 2, this fourth entry should be chosen to be `[[4, 5, 1/4, 1/6], [6, 2, 1/5, 1/6]]`.

This completes the description of the functions and routines in the accompanying Python program to this article.

Author Contributions: All authors of this paper contributed equally to the project, with the exception of the Python code, which was generated by B.S. funded by an LMS Undergraduate Research Bursary. All authors have read and agreed to the published version of the manuscript.

Funding: Shiping Liu is supported by the National Key R and D Program of China 2020YFA0713100 and the National Natural Science Foundation of China (No. 12031017). We would like to thank the London Mathematical Society for their support of Ben Snodgrass via the Undergraduate Research Bursary URB-2021-02, during which the curvature flow was implemented and which lead to many of the research results presented in this paper. David Cushing is supported by the Leverhulme Trust Research Project Grant number RPG-2021-080. Supanat Kamtue is supported by Shuimu Scholar Program of Tsinghua University No. 2022660219.

Data Availability Statement: All numerical results in this paper can be reproduced via the freely accessible Python programme. We did not store this data but generated it on the fly.

Acknowledgments: Florentin Münch expresses his gratitude for Durham University and the University of Newcastle for their hospitality and support during his UK research visit. We would also like to express our gratitude for various useful and constructive suggestions by the referees which improved the presentation of our paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cushing, D.; Kamtue, S.; Liu, S.; Münch, F.; Peyerimhoff, N.; Snodgrass, H.B. Bakry-Émery curvature sharpness and curvature flow in finite weighted graphs. I. Theory. *arXiv* **2022**, arXiv:2204.10064.
2. Cushing, D.; Kamtue, S.; Liu, S.; Münch, F.; Peyerimhoff, N.; Snodgrass, H.B. Bakry-Émery curvature sharpness and curvature flow in finite weighted graphs. II. Implementation. *arXiv* **2022**, arXiv:arxiv.2212.12401.
3. Gallot, S.; Hulin, D.; Lafontaine, J. *Riemannian Geometry*, 3rd ed.; Springer: Berlin/Heidelberg, Germany, 2004.
4. Bakry, D.; Émery, M. Hypercontractivité de semi-groupes de diffusion. *C. R. Acad. Sci. Paris Sér. I Math.* **1984**, *299*, 775–778.
5. Elworthy, K.D. Manifolds and graphs with mostly positive curvatures. In *Stochastic Analysis and Applications (Lisbon, 1989)*; Progress in Probability; Birkhäuser Boston: Boston, MA, USA, 1991; Volume 26, pp. 96–110.
6. Schmuckenschläger, M. Curvature of nonlocal Markov generators. In *Convex Geometric Analysis (Berkeley, CA, 1996)*; Mathematical Sciences Research Institute Publications; Cambridge University Press: Cambridge, MA, USA, 1999; Volume 34, pp. 189–197.
7. Lin, Y.; Yau, S.T. Ricci curvature and eigenvalue estimate on locally finite graphs. *Math. Res. Lett.* **2010**, *17*, 343–356. [CrossRef]
8. Bauer, F.; Chung, F.; Lin, Y.; Liu, Y. Curvature aspects of graphs. *Proc. Am. Math. Soc.* **2017**, *145*, 2033–2042. [CrossRef]
9. Chung, F.; Lin, Y.; Yau, S.T. Harnack inequalities for graphs with non-negative Ricci curvature. *J. Math. Anal. Appl.* **2014**, *415*, 25–32. [CrossRef]
10. Fathi, Z.; Lakzian, S. Bakry-Émery Ricci curvature bounds for doubly warped products of weighted spaces. *J. Geom. Anal.* **2022**, *32*, 75. [CrossRef] [PubMed]
11. Fathi, M.; Shu, Y. Curvature and transport inequalities for Markov chains in discrete spaces. *Bernoulli* **2018**, *24*, 672–698. [CrossRef]
12. Hua, B.; Lin, Y. Graphs with large girth and nonnegative curvature dimension condition. *Comm. Anal. Geom.* **2019**, *27*, 619–638. [CrossRef]
13. Ma, L. Bochner formula and Bernstein type estimates on locally finite graphs. *arXiv* **2013**, arXiv:1304.0290.
14. Man, S. Logarithmic Harnack inequalities for general graphs with positive Ricci curvature. *Differ. Geom. Appl.* **2015**, *38*, 33–40. [CrossRef]
15. Pouryahya, M.; Elkin, R.; Sandhu, R.; Tannenbaum, S.; Georgiou, T.; Tannenbaum, A. Bakry-Émery Ricci curvature on weighted graphs with applications to biological networks. In Proceedings of the International Symposium on Mathematical Theory of Networks and Systems, Bayreuth, Germany, 12–15 July 2016; Volume 22, p. 52.
16. Robertson, S.J. Harnack Inequality for Magnetic Graphs. *arXiv* **2019**, arXiv:1910.04019.
17. Salez, J. Sparse expanders have negative curvature. *arXiv* **2021**, arXiv:2101.08242.
18. Salez, J. Cutoff for non-negatively curved Markov chains. *arXiv* **2021**, arXiv:2102.05597.

19. Shi, Y.; Yu, C. Comparisons of Dirichlet, Neumann and Laplacian eigenvalues on graphs and applications. *arXiv* **2020**, arXiv:2011.04160.
20. Cushing, D.; Liu, S.; Peyerimhoff, N. Bakry-Émery curvature functions on graphs. *Can. J. Math.* **2020**, *72*, 89–143. [CrossRef]
21. Klartag, B.; Kozma, G.; Ralli, P.; Tetali, P. Discrete curvature and abelian groups. *Can. J. Math.* **2016**, *68*, 655–674. [CrossRef]
22. Siconolfi, V. Coxeter groups, graphs and Ricci curvature. *Sém. Lothar. Combin.* **2020**, *84B*, 67.
23. Siconolfi, V. Ricci curvature, graphs and eigenvalues. *Linear Algebra Appl.* **2021**, *620*, 242–267. [CrossRef]
24. Cushing, D.; Kamtue, S.; Liu, S.; Peyerimhoff, N. Bakry-Émery curvature on graphs as an eigenvalue problem. *Calc. Var. Partial Differential Equations* **2022**, *61*, 62. [CrossRef]
25. Chow, B.; Luo, F. Combinatorial Ricci flows on surfaces. *J. Differ. Geom.* **2003**, *63*, 97–129. [CrossRef]
26. Ollivier, Y. Ricci curvature of Markov chains on metric spaces. *J. Funct. Anal.* **2009**, *256*, 810–864. [CrossRef]
27. Ollivier, Y. A survey of Ricci curvature for metric spaces and Markov chains. In *Probabilistic Approach to Geometry*; Advanced Studies in Pure Mathematics; Mathematical Society of Japan: Tokyo, Japan, 2010; Volume 57, pp. 343–381. [CrossRef]
28. Ni, C.C.; Lin, Y.Y.; Luo, F.; Gao, J. Community detection on networks with Ricci flow. *Sci. Rep.* **2019**, *9*, 9984. [CrossRef]
29. Bai, S.; Lin, Y.; Lu, L.; Wang, Z.; Yau, S.T. Ollivier Ricci-flow on weighted graphs. *arXiv* **2020**, arXiv:2010.01802.
30. Weber, M.; Saucan, E.; Jost, J. Characterizing complex networks with Forman-Ricci curvature and associated geometric flows. *J. Complex Netw.* **2017**, *5*, 527–550. [CrossRef]
31. Forman, R. Bochner's method for cell complexes and combinatorial Ricci curvature. *Discret. Comput. Geom.* **2003**, *29*, 323–374. [CrossRef]
32. Topping, J.; Di Giovanni, F.; Chamberlain, B.P.; Dong, X.; Bronstein, M.M. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv* **2021**, arXiv:2111.14522.
33. Bober, J.; Monod, A.; Saucan, E.; Webster, K.N. Rewiring Networks for Graph Neural Network Training Using Discrete Geometry. *arXiv* **2022**, arXiv:2207.08026.
34. Erbar, M.; Maas, J. Ricci curvature of finite Markov chains via convexity of the entropy. *Arch. Ration. Mech. Anal.* **2012**, *206*, 997–1038. [CrossRef]
35. Erbar, M.; Kopfer, E. Super Ricci flows for weighted graphs. *J. Funct. Anal.* **2020**, *279*, 108607. [CrossRef]
36. Devriendt, K.; Lambiotte, R. Discrete curvature on graphs from the effective resistance. *arXiv* **2022**, arXiv:2201.06385.
37. Erdős, P.; Rényi, A. On random graphs. I. *Publ. Math. Debrecen* **1959**, *6*, 290–297. [CrossRef]
38. Pak, I. Lectures on Discrete and Polyhedral Geometry. 2010. Available online: https://www.math.ucla.edu/~pak/geompol8.pdf (accessed on 6 June 2023).
39. Hirsch, M.W.; Smale, S. *Differential Equations, Dynamical Systems, and Linear Algebra*; Pure and Applied Mathematics; Academic Press: New York, NY, USA; Harcourt Brace Jovanovich, Publishers: London, UK, 1974; Volume 60.
40. Cushing, D.; Kangaslampi, R.; Lipiäinen, V.; Liu, S.; Stagg, G.W. The Graph Curvature Calculator and the Curvatures of Cubic Graphs. *Exp. Math.* **2022**, *31*, 583–595. [CrossRef]