Graph Algorithms for Dynamical Systems

Michael Dellnitz[1], Mirko Hessel-von Molo[1], Philipp Metzner[2], Robert Preis[1], and Christof Schütte[2]

[1] Institute for Mathematics, University of Paderborn, 33095 Paderborn. dellnitz@uni-paderborn.de, mirkoh@uni-paderborn.de, robsy@uni-paderborn.de

[2] Department of Mathematics, Freie Universität Berlin, Arnimallee 14, 14195 Berlin. metzner@math.fu-berlin.de, schuette@math.fu-berlin.de

1 Introduction

This article is concerned with the numerical analysis of dynamical systems using methods that are based on a discretized description of the system as a graph. The graph-based description provides a unifying framework to approach a wide and diverse variety of dynamical systems, from time-discrete maps via ordinary differential equations to stochastic differential equations describing e. g. diffusion in a potential landscape.

Within this variety, this article focusses on those dynamical systems that can possess a 'multiscale structure' in the sense that they exhibit interesting dynamical behavior on more than one timescale. We will explain what we mean with this phrase by means of some examples. Consider in Fig. 1 one trajectory of Chua's circuit, that is described by the well-known three-dimensional ordinary differential equation

$$\begin{aligned}\dot{x} &= \alpha(y - m_0 x - \frac{1}{3} m_1 x^3)\\ \dot{y} &= x - y + z\\ \dot{z} &= -\beta y\end{aligned}$$

(see e. g. [HP*96]). It is clearly visible that relatively long parts of the whole trajectory are contained in two 'leaves' within which the trajectory shows a spiralling motion, with only some quick 'jumps' between the two leaves.

Similar phenomena can be observed in systems of quite different mathematical type. As an example, consider a stochastic process in $\mathbf{R}^n$ describing diffusion within a potential given by a function $V : \mathbf{R}^n \to \mathbf{R}$. The system is given by the Smoluchovski equation

$$\dot{X}(t) = -\nabla V(X(t)) + \varepsilon \dot{W}(t)$$

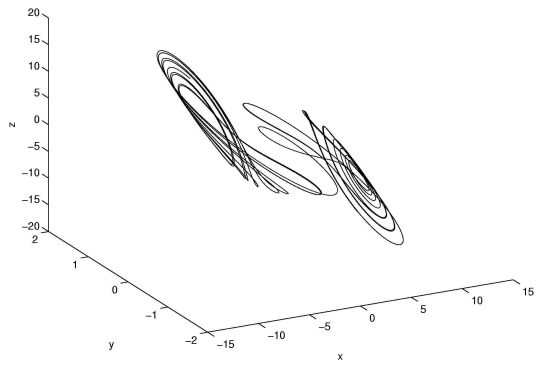

Fig. 1.1. A trajectory of Chua's circuit that switches relatively rarely between two almost invariant sets.

with W_t being a standard n-dimensional Wiener process and ε a small parameter. (A variation of this example is described in more detail in Sect. 4.) If we assume $V(x) \to \infty$ for $\|x\| \to \infty$ (in order to avoid sample paths drifting off to infinity), then any sample path will spend most of the time in the vicinity of the local minima of V, with transitions between the minima being rare events.

The common feature of the Chua circuit example and of the diffusion example is the existence of subsets of the state space that are, although not being invariant under the dynamics considered, nevertheless *almost* invariant in the sense that on a short timescale, a change of a trajectory between the sets is an event rarely encountered. This suggests to analyze such systems on the *short* timescale as if those almost invariant sets were indeed invariant, concentrating on features of the dynamics within the sets, and neglecting outside interactions. On the *long* timescale, on the contrary, the dynamics of such systems can be considered as some kind of 'flipping process' between several almost invariant 'superstates'. In this view, the first step of an analysis that separates different timescales is the identification of almost invariant sets in the dynamics, which forms the prime motivation for the work presented in this article.

As was already alluded to in the beginning, we choose the approximation of continuous dynamics through discrete Markov chains as the unifying approach to various kinds of dynamical systems. Reading a transition matrix as the adjacency matrix of a graph naturally transforms the situation into a graph theoretic framework. The problem of identifying almost invariant sets thus becomes the problem of finding partitions of a graph that are optimal with respect to certain cost functions, for which a plethora of solution or approximation methods is at hand.

The remainder of this article is organized as follows. In Sect. 2, together with some notation we introduce the basic concepts used in this work, in particular the concept of almost invariant sets that is central for the contents of this article. We formulate the problem of identifying almost invariant partitions and then reformulate it first as a discrete optimization problem and then as a graph-theoretic problem. Sect. 3 takes up the last formulation and introduces algorithmic possibilities graph theory offers for the solution of the problem. We pay particular attention to the concept of the congestion of a graph with its connection to dynamical systems concepts. Sect. 4 illustrates the use of shortest-path-algorithms for the detection of dynamically meaningful transition paths between almost invariant sets. Here the crucial point is the appropriate choice of edge weights in the graph, for which two particular examples are presented.

2 Numerical Analysis of Dynamical Systems

In this section we introduce the concept of almost invariant sets of a dynamical system, and we describe a standard framework for their numerical analysis using hierarchical set oriented methods.

2.1 Dynamical Systems and Invariant Measures

A map $f : X \to X$ on a compact subset $X \subset \mathbb{R}^n$ defines a discrete-time dynamical system with state space X. Trajectories of the system are sequences of points in X of the form

$$x_{k+1} = f(x_k),\ k = 0, 1, \dots\ .$$

A particularly important class of such maps f is that of time-T maps of an ordinary differential equation. In this case, under mild assumptions on the ODE (local existence and uniqueness) f is even a diffeomorphism; in the following we will assume this to be the case. Note that the state space X need not be the maximal domain of the map f. In many cases it is more appropriate to consider the dynamical system on some (invariant) subset of the maximal domain, e. g. an attractor, an ergodic component, the set of non-wandering points or the chain recurrent set.

In this work, we are interested in questions about the *global* dynamical behavior of the dynamical system $f : X \to X$. A powerful approach to these questions is to use the *transfer operator* (or *Perron-Frobenius operator*) associated with f, which, instead of generating single trajectories of points in X, describes the evolution of sets or, more generally, of (signed) measures on X. More precisely, the transfer operator associated with f is the linear operator $P : \mathcal{M} \to \mathcal{M}$,

$$(P\nu)(S) = \nu(f^{-1}(S)), \quad S \text{ measurable},$$

on the space $\mathcal{M}$ of signed measures on the Borel σ-algebra over X.

In the following we will assume that μ is an invariant measure for f, that is, the probability measure μ satisfies

$$\mu(S) = \mu(f^{-1}(S)) = (P\mu)(S) \text{ for all measurable } S \subset X ,$$

and thus is a fixed point of the transfer operator. Moreover we assume that μ is a unique so-called SRB-measure (*Sinai-Ruelle-Bowen*) in the sense that this is the only invariant measure which is robust under small random perturbations, in other words the only physically relevant invariant measure for the dynamical system f.

2.2 Almost Invariant Sets

For two measurable sets $S_1, S_2 \subset X$ we define the *transition probability* ρ from S_1 to S_2 as

$$\rho(S_1, S_2) := \frac{\mu(S_1 \cap f^{-1}(S_2))}{\mu(S_1)},$$

whenever $\mu(S_1) \neq 0$. The transition probability $\rho(S) := \rho(S, S)$ from a set $S \subset X$ into itself is called the *invariance ratio* of S. If for a number $\delta \in [0, 1]$ the relation

$$\rho(S) \geq \delta$$

holds, S is called an δ-*almost invariant set.* In practice, we will be interested in numbers $\delta = 1 - \varepsilon$ with $0 < \varepsilon << 1$. When no precise bound δ on the invariance ratio is important, we will also simply speak of almost invariant sets.

The following observation will be crucial for the rest of this article. Let S be an δ-almost invariant set, with $\delta = 1 - \varepsilon$. From $\mu(S) = \mu(f^{-1}(S))$ one has on the one hand that

$$\mu(S) = \mu(f^{-1}(S)) = \mu(S \cap f^{-}1(S)) + \mu(X \setminus S \cap f^{-1}(S)). \tag{2.1}$$

On the other hand,

$$\mu(X \setminus S) = \mu(X \setminus S \cap f^{-1}(S)) + \mu(X \setminus S \cap f^{-1}(X \setminus S)). \tag{2.2}$$

As S is δ-almost invariant, (2.1) means that

$$\mu(X \setminus S \cap f^{-1}(S)) \leq \varepsilon\mu(S) \tag{2.3}$$

which together with (2.2) implies that

$$\begin{aligned} \mu(X \setminus S \cap f^{-1}(X \setminus S)) &\geq \mu(X \setminus S) - \varepsilon\mu(S) \\ &= \left(1 - \varepsilon\frac{\mu(S)}{\mu(X \setminus S)}\right) \cdot \mu(X \setminus S), \end{aligned}$$

and thus

$$\rho(X \setminus S) \geq \left(1 - \varepsilon \frac{\mu(S)}{\mu(X \setminus S)}\right).$$

In short, this means that the complement of an almost invariant set is also almost invariant, with the respective invariance ratios being the more similar the closer the ratio $\frac{\mu(S)}{\mu(X \setminus S)}$ is to one.

This observation naturally motivates the problem of determining a *partition* of X consisting of almost invariant sets of roughly equal weight. For the rest of this article we will be concerned with this problem.

Although it may seem obvious, it is important to note that unlike e. g. the decomposition into ergodic components, which is unique for any given system, the decomposition of the state space into almost invariant sets will in general not be unique. In fact, any small (with respect to e. g. Lebesgue measure) variation of an almost invariant set will also be an almost invariant set, probably with a slightly different invariance ratio.

We now formally define the problem of finding a partition of almost invariant sets in the spatially continuous setting we have been considering so far. It will be reformulated twice in the course of this article, first for a spatially discretized setting and later in the language of graph theory.

Problem 2.1. For some fixed $p \in \mathbb{N}^+$ find a collection of pairwise disjoint sets $\mathcal{S} = \{S_1, \ldots, S_p\}$ with $\bigcup_{1 \leq i \leq p} S_i = X$ and $\mu(S_i) > 0$, $1 \leq i \leq p$, such that

$$\rho(\mathcal{S}) := \frac{1}{p} \sum_{i=1}^{p} \rho(S_i) \to \max .$$

2.3 Discretization of the Transfer Operator

For the detection and approximation of almost invariant sets we need to explicitly deal with the transfer operator. Since an analytical expression for it will only be derivable for none but the most simple systems, we need to derive a finite-dimensional approximation to it. The following description is based on results from e.g. [DH*97, DJ99, DFJ01, DJ02].

The basic idea for the discretization is to construct a sufficiently fine covering of the state space of the system consisting of *boxes*, i. e. generalized rectangles, by means of a *subdivision algorithm* as described in [DH97]. The basic principle of the subdivision algorithm is as follows. One starts with a box $Q \supset X$ containing the state space. Setting $\mathcal{B}_0 = \{Q\}$, a sequence $(\mathcal{B}_n)_{n \in \mathbf{N}}$ is iteratively constructed, with each iteration step $i \to i + 1$ of the iteration consisting of two parts. In the first part, from the collection $\mathcal{B}_i$ a new collection $\tilde{\mathcal{B}}_{i+1}$ is constructed by subdividing each box $B \in \mathcal{B}_i$ along a prescribed coordinate axis into two new boxes. In the second part, $\mathcal{B}_{i+1}$ is constructed as the collection of those boxes that do intersect with X, i. e.

$$\mathcal{B}_{i+1} = \left\{ B \in \tilde{\mathcal{B}}_{i+1} \mid B \cap X \neq \emptyset \right\}$$

There are several modifications of this scheme, in particular regarding the choice of boxes to be subdivided. While in the simple scheme every box is subdivided in every step, one can reduce the numerical effort by introducing an additional selection criterion that decides which boxes to subdivide and which ones to leave at the present level. More detailed expositions of the subdivision scheme can be found e. g. in [DH97, DJ99, Jun01].

Of course, in practice one cannot infinitly go on with the construction of an arbitrarily fine box covering, but will have to stop the process at some level, which results in a covering of the state space X by a finite collection $\mathcal{B} = \{B_1, \ldots, B_b\}$ of boxes, i. e.

$$X \subset \bigcup_{i=1}^{b} B_i \quad \text{with} \quad m(B_i \cap B_j) = 0 \quad \text{for } i \neq j \ .$$

Here m denotes Lebesgue measure.

To discretize the transfer operator, we replace the space $\mathcal{M}$ of signed measures over the Borel σ-algebra by the finite-dimensional space $\mathcal{M}_\mathcal{B}$ of signed measures on the σ-algebra that is given by the set of arbitrary unions of boxes in $\mathcal{B}$. The standard basis for this vector space is given by those measures that assign the weight 1 to precisely one box in $\mathcal{B}$ and 0 to all other boxes.

With respect to this basis, the discretized transfer operator $P_\mathcal{B} : \mathcal{M}_\mathcal{B} \to \mathcal{M}_\mathcal{B}$ is represented by the matrix of transition probabilities

$$P_\mathcal{B} = (p_{ij}), \quad \text{where} \quad p_{ij} = \frac{m(f^{-1}(B_i) \cap B_j)}{m(B_j)}, \quad 1 \leq i, j \leq b. \tag{2.4}$$

In the compution of the transition probabilities p_{ij}, the denominator poses no problem, as the boxes B_i are generalized rectangles. For the computation of $m(f^{-1}(B_i) \cap B_j)$, that is, the measure of the subset of B_j that is mapped into B_i, there are several possibilities described e. g. in [DFJ01]. A method that is often used is the Monte Carlo approach as described in [Hun94]:

$$m(f^{-1}(B_i) \cap B_j) \approx \frac{1}{K} \sum_{k=1}^{K} \chi_{B_i}(f(x_k)),$$

where the x_k's are selected at random in B_j from a uniform distribution. Evaluation of $\chi_{B_i}(f(x_k))$ only means that we have to check whether or not the point $f(x_k)$ is contained in B_i. There are efficient ways to perform this check based on a hierarchical construction and storage of the collection $\mathcal{B}$ (see [DH97, DH*97]).

Note that once we have computed an approximation $P_\mathcal{B}$ of the transfer operator we can obtain a discretized version of the natural invariant measure μ of the box covering $\mathcal{B}$ of A as the eigenvector to the eigenvalue 1 of $P_\mathcal{B}$.

As described in the beginning of this section, a region will be of interest if it is almost invariant in the sense that typical points are mapped into the region

itself with high probability. Evidently the infinite dimensional optimization problem 2.1 needs to be discretized in order to be treated numerically. To this end we again restrict ourselves to subsets that are unions of elements of the partition $\mathcal{B}$. Consider the *transition matrix* $P_{\mathcal{B}}$ from (2.4). Then, our goal in the discretized setting is to solve the following problem.

Problem 2.2 (Boxes). For some $p \in \mathbb{N}^+$ find a collection of pairwise disjoint sets $\mathcal{S} = \{S_1, \ldots, S_p\}$ with $\bigcup_{1 \le i \le p} S_i = \mathcal{B}$ and $\mu(S_k) > 0$, $1 \le i \le p$, such that

$$\rho(\mathcal{S}) = \frac{1}{p} \sum_{k=1}^{p} \rho(S_k) = \frac{1}{p} \sum_{k=1}^{p} \frac{\sum_{B_i, B_j \subset S_k} p_{ij} \cdot \mu(B_j)}{\sum_{B_j \subset S_k} \mu(B_j)} \to \max \ .$$

2.4 Graph Formulation

In this section, we go one step further with reformulating the problem of finding almost invariant sets of a dynamical system. As it turns out, the optimization problem 2.2 can be translated into the problem of finding an optimal cut in a graph. To see this, we first show how the matrix describing the discretized transfer operator can also be understood as a matrix describing a directed graph, and then show that the quantity $\rho(\mathcal{S})$ can be naturally expressed in terms of edge and vertex weights of the graph.

As in the previous section, let $\mathcal{B}$ be a box covering of X. Let $G = (V, E)$ be a graph with vertex set $V = \mathcal{B}$ and directed edge set

$$E = E(\mathcal{B}) = \{(B_1, B_2) \in \mathcal{B} \times \mathcal{B} : f(B_1) \cap B_2 \neq \emptyset\} \ .$$

The function $vw : V \to \mathbb{R}$ with $vw(B_i) = \mu(B_i)$ assigns a weight to the vertices and the function $ew : E \to \mathbb{R}$ with $ew((B_i, B_j)) = \mu(B_i)p_{ji}$ assigns a weight to the edges. Furthermore, let

$$\bar{E} = \bar{E}(\mathcal{B}) = \{\{B_1, B_2\} \subset \mathcal{B} : (f(B_1) \cap B_2) \cup (f(B_2) \cap B_1) \neq \emptyset\} \ .$$

This defines an undirected graph $\bar{G} = (V, \bar{E})$ with a weight function $e\bar{w} : \bar{E} \to \mathbb{R}$ with $e\bar{w}(\{B_i, B_j\}) = \mu(B_j)p_{ij} + \mu(B_i)p_{ji}$ on the edges. The difference between the graphs G and $\bar{G}$ is that in $\bar{G}$ the edge weight between two vertices is the sum of the edge weights of the two directed edges between the same vertices in G. Thus, the total edge weights of both graphs are identical.

To formulate the problem of partitioning the state space into almost invariant sets, we will define two cost functions that describe how much weight remains within a certain set on the one hand, and how much weight changes the set of a partition on the other hand. In order to so, we first need some more notation and write $\mu(S) = \sum_{i \in S} \mu(B_i)$ for $S \subset V$, and with $\bar{S} = V \setminus S$ we further denote

$$E_{S,S} = \sum_{i,j \in S} \mu(B_i)p_{ij} \quad \text{and} \quad E_{S,\bar{S}} = \frac{1}{2} \sum_{i \in S, j \in \bar{S}} \mu(B_i)p_{ij} + \mu(B_j)p_{ji}.$$

Definition 2.3. *For a set $S \subset V$ we define*

$$C_{int}(S) = \frac{E_{S,S}}{\mu(S)}$$

as the ***internal cost*** *of S, and*

$$C_{ext}(S) = \frac{E_{S,\bar{S}}}{\mu(S) \cdot \mu(\bar{S})}$$

as the ***external cost*** *of S.*

Note that both cost functions are independent from the choice between the directed graph G or the undirected graph $\bar{G}$. We can therefore choose the simpler undirected graph, and we will do that in the following.

Definition 2.4. *For a partition $\mathcal{S} = \{S_1, \ldots, S_p\}$ of V we define*

$$C_{int}(\mathcal{S}) = \frac{1}{p} \sum_{i=1}^{p} C_{int}(S_i) \tag{2.5}$$

as the ***internal cost*** *of $\mathcal{S}$, and*

$$C_{ext}(\mathcal{S}) = \frac{\sum_{1 \le i < j \le p} E_{S_i,S_j}}{\prod_{i=1}^{p} \mu(S_i)} \tag{2.6}$$

as the ***external cost*** *of $\mathcal{S}$.*

Intuitively, optimal almost invariant partitions have maximal internal cost and minimal external cost. Therefore, both the internal and external costs are useful cost functions for the problem of computing almost invariant sets. However, the minimization of the external cost is not equivalent to the maximization of the internal cost. In fact, the maximization of the internal cost favors in general parts that are on average very loosely coupled to the rest of the system. However, the size of these parts can in principle become very small. On the other hand the minimization of the external cost favors balanced weighting of the components.

It is an easy task to check that $\rho(\mathcal{S}) = C_{\text{int}}(\mathcal{S})$. Thus, the optimization of problem 2.2 is identical to the optimization of the internal costs of the partition $\mathcal{S}$ in equation (2.5) written in graph notation. Therefore, we have established the following graph partitioning problem.

Problem 2.5 (Graph). For some fixed $p \in \mathbb{N}^+$ find a collection of pairwise disjoint sets $\mathcal{S} = \{S_1, \ldots, S_p\}$ with $\bigcup_{1 \le i \le p} S_i = V$ and $vw(S_i) > 0$, $1 \le i \le p$, such that

$$C_{\text{int}}(\mathcal{S}) \to \max \ . \tag{2.7}$$

One can also consider the analogous problem for the external cost function.

Problem 2.6 (Graph). For some fixed $p \in \mathbb{N}^+$ find a collection of pairwise disjoint sets $\mathcal{S} = \{S_1, \ldots, S_p\}$ with $\bigcup_{1 \le i \le p} S_i = V$ and $vw(S_i) > 0$, $1 \le i \le p$, such that

$$C_{\text{ext}}(\mathcal{S}) \to \min \ . \tag{2.8}$$

3 Computation of Almost Invariant Sets as a Graph Partitioning Problem

In this section we show how existing graph partitioning methods and tools can be applied to compute almost invariant sets. We first describe some state of the art graph partitioning heuristics. Then, we introduce the notion of *congestion* in a graph and its use in the analysis of dynamical systems, in particular in view of the problem of finding a partition of almost invariant sets. Finally, we explain how the congestion can be used as a criterion to decide on the number of almost invariant sets.

3.1 Graph Partitioning Heuristics

In this section we briefly review existing approaches and algorithms for partitioning the vertex set of a graph. As most variations of the partitioning problem – including those we are interested in in this article – are **NP**-complete, the algorithms we present are approximation algorithms that are often based on some heuristic for obtaining good partitionings.

The existing methods and tools for graph partitioning do not exactly optimize the cost functions we introduced in the previous section. We therefore give an overview of the most successful graph partitioning methods and implementations and point out the necessary modifications to such tools.

For the remainder of this section we assume that we aim to partition a the vertex set of a graph into a known number of p parts. In Sect. 3.3 we will present a way to identify this number.

We want to calculate a partition of the vertex set V of a graph $G = (V, E)$ into p parts $V = S_1 \cup \cdots \cup S_p$ such that one of our cost functions of equations (2.5) or (2.6) is optimized. However, the calculation of an optimal solution of both cost functions is **NP**-complete. Another widely discussed partitioning problem is to minimize the cut size $cut(\pi) = \sum_{1 \le i < j \le p} E_{S_i,S_j}$ of the partition π under the constraint that all parts have an equal (or almost equal) number of vertices. This problem is sometimes called *Balanced Partitioning Problem* and is **NP**-complete, even in the simplest case when a graph with constant vertex and edge weights is to be partitioned into two parts [GJ79].

Efficient graph partitioning strategies have been developed for a number of different applications. Efficiency and generalizations of graph partitioning methods strongly depend on specific implementations. There are several software libraries, each of which provides a range of different methods. Examples are CHACO [HL94], JOSTLE [Wal00], METIS [KK98a], SCOTCH [Pel96] or PARTY [Pre98]. The goal of the libraries is to both provide efficient implementations and to offer a flexible and universal graph partitioning interface to applications. These libraries are designed to create solutions to the balanced partitioning problem.

The tool PARTY has been developed by one of the authors and we have used it for partitioning the graphs in this paper. PARTY, like other graph par-

titioning tools, follows the Multilevel Paradigm. The multilevel graph partitioning strategies have been proven to be very powerful approaches to efficient graph-partitioning [Bou98, Gup97, HL95, KK98b, KK99, MPD00, PM*94, Pre00]. The efficiency of this paradigm is dominated by two parts: graph coarsening and local improvement.

The graph is coarsened down in several levels until a graph with a sufficiently small number of vertices is constructed. A single coarsening step between two levels can be performed by the use of graph matchings. A matching of a graph is a subset of the edges such that each vertex is incident to at most one matching edge. A matching of the graph is calculated and the vertices incident to a matching edge are contracted to a super-vertex. Experimental results reveal that it is important to contract those vertices which are connected via an edge of a high weight, because it is very likely that this edge does not cross between parts in a partition with a low weight of crossing edges. PARTY uses a fast approximation algorithm which is able to calculate a good matching in linear time [Pre00].

PARTY stops the coarsening process when the number of vertices is equal to the desired number of parts. Thus, each vertex of the coarse graph is one part of the partition. However, it is also possible to stop the coarsening process as soon as the number of vertices is sufficiently small. Then, any standard graph partitioning method can be used to calculate a partition of the coarse graph.

Finally, the partition of the smallest graph is projected back level-by-level to the initial graph. The partition is locally refined on each level. Standard methods for local improvement are Kernighan/Lin [KL70] type of algorithms with improvement ideas from Fiduccia/Mattheyses [FM82]. An alternative local improvement heuristic is the Helpful-Set method [DMP95] which is derived from a constructive proof of upper bounds on the bisection width of regular graphs [HM92, MD97, MP01].

As mentioned above, the tools are designed for solving the balanced graph partitioning problem. Thus, the optimization criterion is different from our cost functions of Sect. 2.4. The coarsening step of the multilevel approach does not consider the balancing of the weights of the super-vertices. It is the local refinement step which not only improves the partition locally but also balances the weights of the parts. Thus, we have to modify the local improvement part of the multilevel approach. We therefore modified the Kernighan/Lin implementation in PARTY such that it optimizes the cost-function C_{int}. Overall, we use the algorithm of Fig. 3.1 to calculate almost invariant sets.

As an example to illustrate the partitioning we consider a graph that was obtained as the discretization of the dynamics of a pentane molecule that is considered in detail in [DH*00]. This molecule has two dihedral angles which are used as state space coordinates. The left plot of Fig. 3.2 shows the box collection and all transitions between boxes. As we will see in sect. 3.3, it is adequate to partition the graph into five or seven parts. These are shown in the center and right plots of Fig. 3.2.

Partition graph $G_0 = (V_0, E_0)$ into p parts

$i = 0$;
WHILE ($|V_i| > p$)
  calculate a graph matching $M_i \subset E_i$;
  use M_i to coarse graph $G_i = (V_i, E_i)$ to a graph $G_{i+1} = (V_{i+1}, E_{i+1})$;
  $i := i + 1$;
END WHILE
let π_i be a p-partition of G_i such that each vertex is one part;
WHILE ($i > 0$)
  $i := i - 1$;
  use M_i to project π_{i+1} to a p-partition π_i of G_i;
  modify the partition π_i on G_i locally to optimize $C_{\text{int}}(\pi_i)$;
END WHILE
output π_0.

Fig. 3.1. Computing a graph partitioning with the multilevel approach.

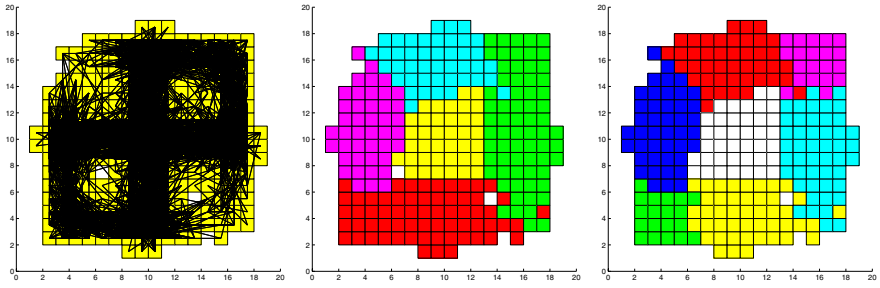

Fig. 3.2. Partition of graph describing the dynamics Pentane300. Left: The graph with all edges. Center: Partition consisting of five parts with $C_{\text{int}} = 0.980$. Right: Partition consisting of seven parts with $C_{\text{int}} = 0.963$. (See page 695 for a colored version of the figure.)

3.2 Congestion

Standard graph partitioning methods partition the graph into a predefined number of parts. However, if we do not know the resulting number of almost invariant sets a priori, we need to find mechanisms which help us to decide on a natural number of parts. As we will see, such a mechanism can be devised on the basis of the concept of *congestion* of a graph.

Intuitively, the congestion of a graph is a quantity that can be used to identify 'bottlenecks' in the graph, i. e. edges that connect subgraphs which have relatively many internal and relatively few external edges. This description already explains the relevance of the congestion for the problem of finding almost invariant sets of a dynamical system. The concept is based on the idea of so-called multi-commodity flows on the graph (see e.g. [Lei92, Sin93]).

In the following, we first formally define the congestion. As it is often not feasibly to precisely compute this quantity for a given graph, we will then shortly discuss heuristics for an approximation of the congestion. These heuristics will produce upper bounds on the congestion, which we can use in a lower bound on the external cost of a bisection of a graph that is discussed immediately afterwards.

The first concept we need for the definition of the congestion is that of a *single-commodity flow* in a graph. Such a flow may be imagined as a way of describing the transport of a certain quantity c of some good from a source s to a target t through a network of roads that is given by the graph.

Definition 3.1. *Let $G = (V, E)$ be an undirected graph with the vertex set $V = \{1, \dots, d\}$. Let $s, t \in V$ be vertices of G with $s \neq t$, let $c \in \mathbf{R}$. A* ***single-commodity flow*** *f of the commodity c from s to t on G is a function $f : V \times V \to \mathbf{R}$ such that*

i. $f(v, w) = 0$ *for all* $\{v, w\} \notin E$ *(flow on edges only),*
ii. $f(v, w) = -f(w, v)$ *for all* $v, w \in V$ *(symmetry),*
iii. $\sum_{w \in V} f(v, w) = 0$ *for all* $v \in V \backslash \{s, t\}$ *(flow conservation) and*
iv. $\sum_{w \in V} f(s, w) = \sum_{w \in V} f(w, t) = c$.

As the next step, we will define *multi-commodity flows* which generalize single-commodity flows. Intuitively, a multi-commodity flow describes the transport of certain commodities from every vertex to every vertex of the graph. The formal definition is as follows.

Definition 3.2. *Let $c_{s,t} \in \mathbf{R}$ for $1 \leq s, t \leq d$. A* **multi-commodity flow** *F of the commodities $c_{s,t}$ on G is a function $F : V \times V \times V \times V \to \mathbf{R}$ such that for each pair $(s, t) \in V \times V$, the function $F(s, t, \cdot, \cdot)$ is a single-commodity flow of the commodity $c_{s,t}$ from s to t on G.*

With these concepts, we are in a position that allows us to introduce the *congestion* of a graph as we are using it in this work.

Definition 3.3. *Let $G = (V, E)$ be an undirected graph with the vertex set $V = \{1, \dots, d\}$, vertex weights μ_i^d for $i \in V$, and edge weights A_{ij} for $\{i, j\} \in E$. For $s, t \in V$, let $c_{s,t} = \mu_s^d \cdot \mu_t^d$. Denote by $\mathcal{F}$ the set of all multi-commodity flows of the commodities $c_{s,t}$ on G. For an edge $\{v, w\} \in E$ and $F \in \mathcal{F}$, the* ***edge congestion of*** *$\{v, w\}$* ***in*** *F is*

$$\operatorname{cong}(\{v, w\}, F) = \frac{\sum_{1 \leq s,t \leq d} |F(s, t, v, w)|}{A_{vw}} \,. \tag{3.1}$$

The **flow congestion of** *F on G is*

$$\operatorname{cong}(F) = \max_{\{v,w\} \in E} \operatorname{cong}(\{v, w\}, F) \,, \tag{3.2}$$

and finally the **congestion** *of the graph G is*

$$\mathrm{cong}(G) = \min_{F \in \mathcal{F}} \mathrm{cong}(F) \ . \tag{3.3}$$

In principle, other choices of commodities $c_{s,t}$ than those used in this definition are also possible. However, the choice we made here seems the most appropriate for the use we will make of the congestion in this paper, as will become clear in the following section.

Approximation of the congestion

The computation of the congestion $\mathrm{cong}(G)$ of a graph can be costly and is often infeasible. We will see in the next section that the congestion can be used to bound the external cost of a partition. However, that bound holds for the congestion $\mathrm{cong}(F)$ of any multi-commodity flow F. Thus, a sub-optimal flow produces a sub-optimal, but still valid bound. In this section we discuss heuristics for calculating a flow with a small flow congestion.

There are some hints of how to construct a flow with a small flow congestion. Clearly, cycles in the flow should be avoided. Furthermore, it is easy to imagine that a low-congestion flow should - at least primarily - go along shortest paths between the pairs of vertices. Here, the length of a path is the sum of the reciprocal values of the edge weights along the path.

A straightforward method is to send the flow along shortest paths only. If more than one shortest paths exist, the flow value can be split among them. If the edge weights are constant, all shortest paths can be calculated in time $O(|V| \cdot |E|)$. This can be done by $|V|$ independent Breath-First searches in time $O(|E|)$ each. If the edge weights are non-negative, all shortest paths can be calculated in time $O(|V| \cdot (|V| \cdot \log |V| + |E|))$, e.g. with $|V|$ runs of the single-source shortest path Dijkstra algorithm using Fibonacci heaps. We refer to [CLR90] for a deeper discussion of shortest paths algorithms.

A different method is to consider n commodities at a time. For each vertex v_s, $1 \leq s \leq n$, consider the commodities $c_{s,t}$, $1 \leq t \leq n$, i.e. all commodities with v_s as the source. For each source v_s we calculate a flow F_s which transports all commodities from v_s to all other vertices, i.e. it replaces n single-commodity flows such that $F_s(v,w) = \sum_{t=1}^{n} F(s,t,v,w)$. F_s is a single-source, multiple-destination commodity flow. Definitions (i.) and (ii.) for the single-commodity flow (Definition 3.1) remain unchanged whereas the definitions of (iii.) and (iv.) are replaced by

v. $\sum_{w \in V} F_s(v_t, w) = -c_{s,t}$ for all $t \neq s$ (from source s to target t) and

vi. $\sum_{w \in V} F_s(v_s, w) = \sum_{t=1}^{n} c_{s,t} - c_{s,s}$ (from source s to all targets t except to source s itself).

It is left to show how we calculate a single-source flow F_s. We use algorithms from diffusion load balancing on distributed processor networks for

this task, see e.g. [DFM99, EF*99, EMP00]. Here, the problem is to balance the work load in a distributed processor network such that the volume of data movement is as low as possible. We use these algorithms in the setting that the vertex v_s models a processor with load $\sum_{t=1}^{n} c_{s,t}$ and all other vertices model a processor with no load. Furthermore, the processors are heterogeneous with a capacity of $c_{s,t}$ for processor v_t [EMP00]. The diffusion algorithms calculate a balancing flow such that each vertex/processor v_t gets a load of $c_{s,t}$. That is exactly what we need in our context. The resulting balancing flow has a nice property: it is minimal in the l_2-norm, i.e. the diffusion algorithms minimize the value $\sqrt{\sum_{1\leq v,w\leq n} |F_s(v,w)|}$. This ensures that there are no cycles in any flow F_s. Furthermore, the flows are not restricted along shortest paths and can avoid high traffic along shortest paths. However, the flows are still favored to be along reasonably short paths. Thus, it is expected that the overall edge congestion of the resulting flow is reasonably small and that the flow congestion is close to the congestion of the graph.

The PARTY library includes efficient code of a variety of diffusion algorithms. We use them to calculate the single-source, multiple-destination flows for each source s and then add up the values to get the multi-commodity flow. Numerical experiments indicate that the resulting flow is indeed very small.

An Example: Pentane

To illustrate the meaning of the congestion in the context of dynamical systems, we again consider as example the graph describing the dynamics of a pentane molecule from [DH*00]. The graph corresponding to this dynamical system is shown in the left part of Fig. 3.3. The middle and the right part of Fig. 3.3 show the edges with low and with high congestion, respectively. The coloring of the boxes indicates the partition into seven almost invariant sets.

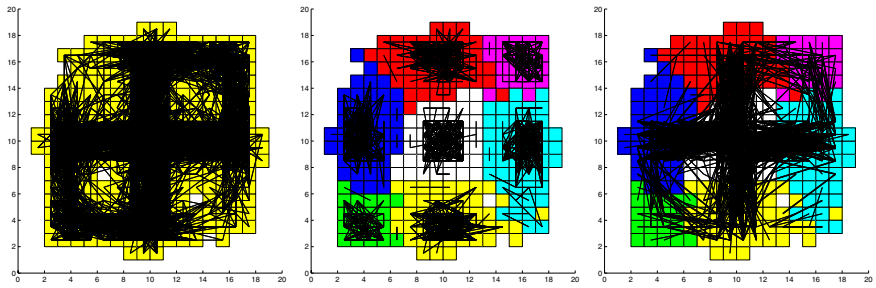

Fig. 3.3. Congestion of the Pentane. Left: all transitions. Center: only transitions with a low congestion. Right: only transitions with a high congestion. (See page 695 for a colored version of the figure.)

It can be observed that – as expected – edges with low congestion can mainly be found inside the almost invariant sets. On the other hand edges between different almost invariant sets have a large congestion. Thus, a high congestion indicates that there are at least two regions in the phase space which are only loosely coupled. As we will see in Sect. 3.3, this observation is the basis for using the congestion as an identifier for the number of almost invariant sets which have to be approximated.

The congestion bound on C_{ext}

We will now see how the concept of congestion of a graph can be used for the analysis of dynamical systems, in particular for the problem of finding a partition into almost invariant sets.

The congestion can be used to derive a lower bound on $C_{ext}(S)$ for any $S \subset I$. As before, we use multi-commodities $c_{s,t} = vw(v_s) \cdot vw(v_t)$ for each source $v_s \in V$ and each destination $v_t \in V$. On the one hand side, for any multi-commodity flow with commodities $c_{s,t}$ as given above, at least $\mu(S) \cdot \mu(\bar{S})$ 'units' have to cross the cut between S and $\bar{S}$, and as many in the opposite direction. On the other hand, with $E_{S,\bar{S}}$ being the sum of edge weights of edges crossing the cut, by definition of the congestion, at most $\mathrm{cong}(G) \cdot E_{S,\bar{S}}$ units can cross the cut. Therefore we have $2 \cdot \mu(S) \cdot \mu(\bar{S}) \leq \mathrm{cong}(G) \cdot E_{S,\bar{S}}$, which at once gives the important inequality

$$C_{\mathrm{ext}}(S) = \frac{E_{S,\bar{S}}}{\mu(S) \cdot \mu(\bar{S})} \geq \frac{2}{\mathrm{cong}(G)}. \tag{3.4}$$

Obviously, a high and tight lower bound can only be achieved with a small congestion. Although the congestion can be computed in polynomial time, it remains to be very costly. Nevertheless, the congestion can be approximated by the congestion of any flow. Heuristics for calculating a small congestion were discussed above. Further discussion of lower bounds based on different variations of multi-commodity flows can be found in [Sen01].

3.3 Identification of the Number of Almost Invariant Sets

We now discuss the problem of identifying an appropriate number of almost invariant sets a given space should be partitioned into.

Informally, we want to determine a number $p \in \mathbf{N}$ such that there is a partition of V consisting of p parts $V = S_1 \cup \cdots \cup S_p$ with a high internal cost. As the internal cost is monotonically decreasing for the optimal partitions with an increasing number of parts p, we are looking for a number p such that an (almost) optimal partition into $p-1$ parts has an only slightly higher internal cost than an (almost) optimal partition consisting of p parts while (almost) optimal partitions into $p+1$ parts have a substantially lower internal cost. The idea is that if we try to split a *compact set* (one with a small congestion), the internal cost will drop substantially. Thus, our strategy is to start with

the whole vertex set as the initial set and keep on bisecting the sets until they become compact sets. This leads us to a strategy of how to determine the number of compact parts. It can be phrased as a general method:

Recursively bisect the vertices of the graph until all parts are compact.

Recursive bisection is a widely used technique in graph partitioning. Although there are many partitioning methods which directly partition the vertices of a graph into a number of parts, we cannot apply them here, because we do not know the number of parts a priora priori. Aditionally, the graph bisection methods are often much more efficient than their generalized counterparts.

We have seen in the previous section that the congestion of a graph can be used to derive a lower bound on the external cost of a set bisection, i.e. a large congestion indicates a large external cost and, therefore, also a small internal cost. We use the congestion in order to decide whether a set is compact or not. In our experiments we use a threshhold of 5 and say that if the congestion is larger than 5 than the set has at least one bottleneck and is not compact. Thus, our strategy is to subdivide the parts until all parts have a congestion of at most 5.

One needs to solve two tasks in order to follow the recursive bisection strategy and we described both in the previous sections. We use the methods described in Sect. 3.1 to recursively calculate bisections of a graph. Furthermore, we use the congestion in order to indicate whether a part is compact or not.

Figure 3.4 illustrates the recursive bisection process in the partitioning of the graph in the pentane molecule example from [DH*00] which we already used before. From the top left to the bottom right picture, the levels of the recursive procedure are shown rowwise. As we can observe in the first picture, the first bisection results in one part of 43 boxes and a very low congestion of 0.88. However, the other part consisting of 212 boxes has a high congestion of 168.82. We continue to bisect parts with a congestion value higher than 5. Thus, after a total of 4 bisection levels we get a partition into 7 parts and the highest congestion of any part is 3.67 .

4 Short Paths

Broadly speaking, the previous section has been concerned with the use of graph partitioning algorithms to obtain information about almost invariant sets of a dynamical system. In this section, we will consider the use of another class of graph algorithms, namely that of shortest-path algorithms, in the context of dynamical systems. We will see that such algorithms can be used to compute discrete approximations to transition paths of a dynamical system. The crucial question for this undertaking is the choice of a weight function that defines the *length* of an edge. We will present two such functions for different

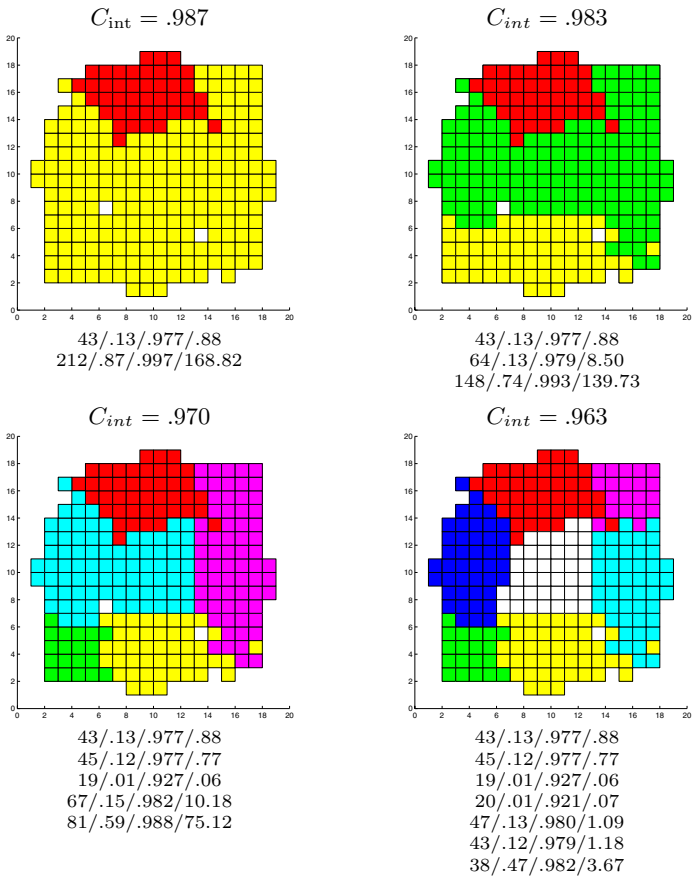

Fig. 3.4. Recursive bisection of the graph for the pentane example. The values indicate: number of boxes / invariant measure / internal cost C_{int} / congestion of subgraph. The graph has 255 vertices and a congestion of 139.67. (See page 696 for a colored version of the figure.)

types of dynamics. Both have a natural motivation, and we will compare the results of both approaches.

Definition 4.1. *Let $G = (V, E)$ be any graph with edge weights given by a function $ew : E \to \mathbf{R}$. A sequence $[(v_1, v_2), (v_2, v_3), \ldots (v_i, v_{i+1})]$ of edges $(v_j, v_{j+1}) \in E$, $1 \le j \le i$, is called a **path** from vertex v_1 to vertex v_{i+1} of size i and of length $l = \sum_{j=1}^{i} ew((v_j, v_{j+1}))$. A* **shortest path** *from a vertex v_s to a vertex v_d is a path of minimum length from v_s to v_d. The* **distance** $\text{dist}(v_s, v_d)$ *from v_s to v_d is the length of a shortest path from v_s to v_d.*

The standard algorithm used for computing shortest paths in graphs is the Dijkstra algorithm. It solves the so called *Single Source Shortest Path Problem*

where the shortest paths from one source vertex $v_s \in V$ to all other vertices $v \in V$ have to be determined. The *Single Source, Single Destination Shortest Path Problem* is a special case in which only one path from v_s to a designated destination vertex v_d has to be determined. In both cases the runtime of the Dijkstra algorithm is $O(|V| \log(|V|) + |E|)$. For a profound discussion of this standard algorithm we refer to e. g. [CLR90], in the following we will only roughly sketch its basic principle.

Given a vertex v_s as starting vertex, the algorithm maintains a list of distances to v_s assigned to every other vertex that is initialised with the value ∞ and in the end contains the lengths of the shortest paths from v_s to any vertex. In the first step, the distances of all neighbors of v_s are set to the weight of the edge connecting them to v_s. These vertices form the initial *halo set*, i.e. they are the vertices for which one path from v_s is known but it is not known whether this path is a shortest path. In the main loop of the algorithm, it removes a vertex v_{min} with the minimum known distance from the halo set, and considers all neighbors of v_{min}. If a neighbor is also in the halo set, the algorithm checks whether a path through v_{min} would result in a distance from v_s less than the current known distance. If a neighbor is not yet in the halo set, it is added to it, with its distance value being the sum of the distance of v_{min} and the length of the edge connecting the neighbor to v_{min}. The algorithm terminates when a prescribed target vertex is reached or when the halo set becomes empty.

By two slight modifications, the Dijkstra algorithm can be generalized to find a shortest path from any vertex of a source set $V_s \subset V$ to any vertex of a destination set $V_d \subset V$. The first modification is that in the initialization step all vertices of V_s are assigned the distance value 0, and that all neighbors of vertices from V_s that do not themselves belong to V_s form the initial halo set. The second modification is that in the main loop, every time a vertex v is removed from the halo set, it is checked whether $v \in V_d$.

4.1 Several Short Paths

For the purposes of this article, the purely graph theoretic consideration of shortest paths we have seen until now has to be extended by some ideas related to the specialized setting of graphs describing (temporal and/or spatial) discretizations of continuous dynamical systems. In particular we have in mind the fact that the numerical realizations of these graphs necessarily come with a discretization error which makes it doubtful whether the notion of *the* shortest path between two vertices v_s and v_d is really a meaningful quantity in our applications – even leaving out the possible existence of several shortest paths. Therefore, we are not only interested in one (or all) precisely shortest paths, but we are also interested in all paths which are only slightly longer than a path with the shortest length.

For this reason, we want to calculate all paths from a vertex v_s to a vertex v_d which have a length of at most $(1 + \epsilon)\,\mathrm{dist}(v_s, v_d)$. In order to do so, we

need to apply the Dijkstra algorithm only two times. Firstly, we calculate all distances from v_s to all other vertices and denote these distances by $\mathrm{dist}_1(v)$ for all vertices $v \in V$. Among all distances this also includes the distance between v_s and v_d. Secondly, we consider a new graph $G_r = (V, F)$ where F consists of the edges in E with direction reversed. Then, we calculate all distances from v_d to all other vertices in G_r, and denote these distances by $\mathrm{dist}_2(v)$ for all vertices $v \in V$. Note that $dist_2(v)$ is also the distance from v to v_d in G for any vertex $v \in V$.

It is now simple to decide whether or not an edge (v_i, v_j) lies on a path between v_s and v_d of length at most $\mathrm{dist}(v_s, v_d)(1+\epsilon)$. Such a path has to consist of three parts: a path from v_s to v_i, the edge (v_i, v_j) itself and a path from v_j to v_d. The shortest length for the first part is $\mathrm{dist}_1(v_i)$ and the shortest length of the last part is $\mathrm{dist}_2(v_j)$. Thus, an edge (v_i, v_j) lies on a path between v_s and v_d of length at most $(1+\epsilon)\,\mathrm{dist}(v_s, v_d)$ if and only if

$$\mathrm{dist}_1(v_i) + ew((v_i, v_j)) + \mathrm{dist}_2(v_j) \leq (1+\epsilon)\,\mathrm{dist}(v_s, v_d)\ .$$

The result is a subset $E_{sp} \subset E$ of edges belonging to the short paths.

4.2 Choices of edge weights

Until now, we have considered graphs with edge weights $ew((v_i, v_j)) = \mu_i^d P_{ji}$ that where introduced in Sect. 2.4. While this weighting is appropriate for graph partitioning algorithms which aim to minimize the internal cost of a partition, it is less useful for shortest path algorithms.

Instead, we want to use an edge weight such that the length of a path $((v_1, v_2), (v_2, v_3), \ldots, (v_i, v_{i+1}))$ from a vertex v_1 to a vertex v_{i+1} reflects the product of the probabilities to choose the next edge along the path, i.e. $\prod_{j=1}^{i} P_{j+1,j}$. Thus, a high probability to go along this path should be reflected by a short path and vice versa.

We can do this by using shortest paths algorithms on the graph with edge weights

$$ew(v_i, v_j) := \frac{1}{\log(P_{ji})} = -\log(P_{ji})\ .$$

Then the length of a path $((v_1, v_2), (v_2, v_3), \ldots, (v_i, v_{i+1}))$ is

$$l = \sum_{j=1}^{i} ew((v_j, v_{j+1})) = -\log(\prod_{j=1}^{i} P_{j+1,j})\ .$$

Note that the product of probabilities on the right hand side of this equation is the probability that the Markov chain described by the matrix P produces the considered sample path when started in v_1.

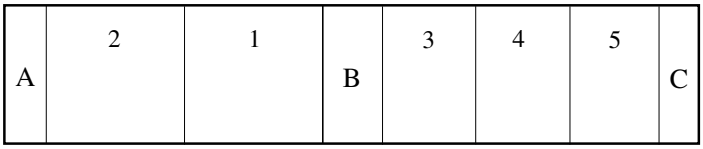

Fig. 4.1. Schematic representation of the example. The rectangular domain of the pure diffusion with a reflecting boundary is discretized into 8 boxes.

Motivational example

As an example for a type of dynamics for which the edge weight introduced in the previous section seems inappropriate, we consider diffusion in a flat potential landscape (i. e. with $V \equiv 0$, see below). We choose a rectangular domain and apply reflecting boundary conditions. In Fig. 4.1 we give a schematic picture of the situation. Suppose, we start the process in box B. From the symmetry of the domain and the nature of diffusion, it is clear that the probability to end up in box A is the same as to reach the box C, namely 0.5. But the particular decomposition of the domain, with the boxes 3, 4 and 5 having only two thirds of the width of the boxes 1 and2, implies that the transition probabilities between boxes 1 and 2 on the one hand side and between boxes 3 and 4, and 4 and 5 on the other hand are all equal. This means that the discrete path $(B, 3, 4, 5, C)$ is less probable than the path $(B, 1, 2, A)$, in contradiction to the continuous picture.

Free Energy

Another important quantity to characterize the transition behavior of a dynamics in a complex system is the free energy barrier which the dynamics has to overcome on its way between two almost invariant sets. Suppose we consider the Smoluchowsky dynamics generated by the stochastic differential equation

$$\dot{X}(t) = -\nabla V(X(t)) + \sqrt{2\beta^{-1}}\dot{W}(t) \tag{4.1}$$

where $X(t) \in \mathbf{R}^n$, W is a standard Browian motion, $V : \mathbf{R}^n \to \mathbf{R}$ is a potential and β is a parameter that is referred to as the inverse temperature. The probability to find the equilibrated system in a certain region, say $C \subset \mathbf{R}^n$, is given by

$$\mu(C) = Z^{-1} \int_C \exp(-\beta V(x))\mathrm{d}x \tag{4.2}$$

where Z is the normalization factor. The traditional way to define the free energy is by means of the marginal density with respect to a given reaction coordinate $\xi : \mathbf{R}^n \mapsto \mathbf{R}$

$$Z(q) = Z^{-1} \int_{\mathbf{R}^n} \exp(-\beta V(x))\delta(\xi(x) - q)\mathrm{d}x. \tag{4.3}$$

Then the free energy is given by the logarithm of the partition sum $Z(q)$:

$$F(q) = -\beta^{-1} \ln Z(q). \tag{4.4}$$

Discrete free energy

Now consider a reversible Markov process on a finite state space $S = \{s_1, \ldots, s_n\}$ and let $\pi = (\pi_1, \ldots, \pi_n)$ its unique stationary distribution. Analogously to the continuous case, we define the free energy in terms of a probability distribution

$$F(i) = -\ln \pi_i > 0, \quad i \in S. \tag{4.5}$$

New weight

Given two disjoint sets $A, B \subset S$, we are interested in the state space path which crosses the lowest free energy barriers on its way from A to B. To this end, we introduce the new edge weights

$$w(i,j) = |F_j - F_i|. \tag{4.6}$$

Let $p = (i_1, \ldots, i_s)$ be a path such that

$$F_{i_j} \leq F_{i_{j+1}} \Leftrightarrow \pi_{i_j} \geq \pi_{i_{j+1}}, \quad j = 1, \ldots, s-1 \tag{4.7}$$

then the length of the path is

$$l(p) = \sum_{j=1}^{s-1} w_{i_j, i_{j+1}} = F_{i_s} - F_{i_1}. \tag{4.8}$$

This means that the weight of such a path is simply given by the free energy difference between the last and the first state of the path. Moreover, if we fix the states i_1 and i_s, then all paths connecting these two states and satisfying (4.7), have the same length. Next consider a path $p = (i_1, \ldots, i_n)$ which can be decomposed into two parts $p_1 = (i_1, \ldots, i_s)$ and $p_2 = (i_s, \ldots, i_n)$ such that

$$\begin{cases} F_{i_j} \leq F_{i_{j+1}}, & j = 1, \ldots, s-1 \\ F_{i_j} \geq F_{i_{j+1}}, & j = s, \ldots, n-1 \end{cases}. \tag{4.9}$$

One immediately verifies that the length of such a path is given by

$$l(p) = 2F_{i_s} - (F_{i_1} + F_{i_n}) \geq 0. \tag{4.10}$$

Again, the length of the path depends only on free energy differences, namely the barriers $F_{i_s} - F_{i_1}$ and $F_{i_s} - F_{i_n}$. Consequently, if we fix the states i_1 and i_n then the shortest path between i_1 and i_n w.r.t. to the weights (4.6) is the one which crosses the lowest free energy barriers.

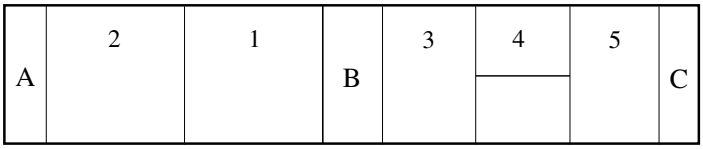

Fig. 4.2. Schematic representation of the modified example.

Interpretation

The first observation is that the new weight allows to find barriers between invariant sets. Furthermore, it is more insensitive with respect to the underlying discretization. To explain this issue, let us go back to example in 4.2. The probabilities to find the equilibrated dynamics in the boxes 1 and 2 are equal, and so are the probabilities of the boxes 3, 4 and 5. That means that there are no free energy barriers for the dynamics on its way to the box A or C, respectively, conditioned on starting in the box B and thus the paths are equal. But this exactly results from the new weight: The lengths of both paths are equal,

$$l(B, 1, 2, A) = F_A - F_B = F_C - F_B = l(B, 3, 4, 5, C) = const.$$

In the previous example the volumes of the boxes are equal. What happens if the volumes of the boxes differ? Suppose we decompose the box 4 into two boxes with equal volume. In Fig. 4.2 we give a schematic representation of the modified example. For this discretization both weights would tell that the path $(B, 1, 2, A)$ is the preferred one since both the transition probabilities w.r.t. the box 4 and its stationary distribution decrease. But nevertheless, the new weight is more insensitive to the underlying discretization because the length of a path does not depend on the *entire* path but only on the barriers which the path overcomes.

4.3 Modified update step in the Dijkstra algorithm

The twofold contribution of a barrier can be seen as reflecting the reversibility of the process. If it is necessary to know the value of the sum of barriers only in one direction then this can be done by modifying the update step in the Dijkstra algorithm. Let v be the current node in the main loop of the Dijkstra algorithm and let k be a neighbor which has to be updated. Instead of using the weight $w(v, k)$ we propose to use the weight $\tilde{w}(v, k) = \max\{0, F_k - F_v\}$ for updating the distance of the node k. Doing so, the bidirectional Dijkstra algorithm with the modified update-step computes the same paths as the unmodified one, but the length of a path only depends on the barrier in one direction.

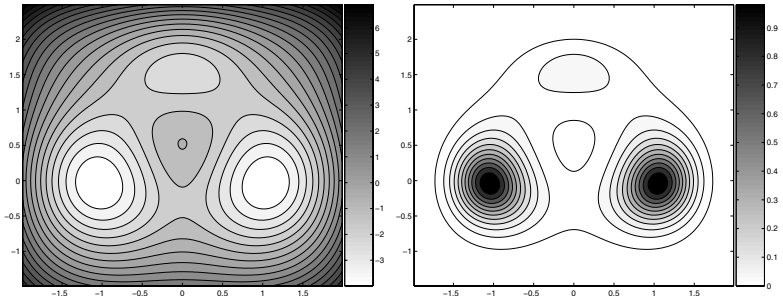

Fig. 4.3. Left :Contour plot of the three-hole potential. Right: Equilibrium distribution at $\beta = 1.67$.

4.4 Illustrative example: diffusion in a potential landscape

In the following example we study the behavior of the bidirectional Dijkstra-Algorithm in the presents of two possible transition channels. We compare the families of transition paths resulting from the probability weight and the free energy weight. For this purpose we choose the three-well potential

$$V(x,y) = 3e^{-x^2-(y-\frac{1}{3})^2} - 3e^{-x^2-(y-\frac{5}{3})^2} \\ -5e^{-(x-1)^2-y^2} - 5e^{-(x+1)^2-y^2}$$

which already has been investigated in [PS*03]. As one can see in the left picture of Fig. 4.3 the two deep minima at $(-1,0)$ and $(1,0)$ are connected by an upper and a lower channel. We choose the inverse temperature $\beta = 1.67$ such that despite the dominance of the two deep minima there is still a little probability to find the dynamics in the shallow minimum around $(0, \frac{5}{3})$. The dynamical bottlenecks in the upper channel are two saddle points with equal potential energy whereas the dynamics in the lower channel only has to overcome one saddle point with potential energy higher than that of the upper ones.

The following experiments are based on a discrete realization of the dynamics given in (4.1) for the inverse temperature $\beta = 1.67$. To be more precise, we use the Euler-Maruyama-scheme

$$x_{n+1} = x_n - \nabla V(x_n)\tau + \sqrt{2\beta^{-1}\tau}\ \eta_n, \quad n = 0, \ldots, N-1 \tag{4.11}$$

to discretize the SDE (4.1) in time, where $x_n \in \mathbf{R}^2$, τ is the time step and η_n denotes a realization of a gaussian random variable with mean zero and variance one. We choose the time step $\tau = 10^{-3}$ and generate a trajectory of total length τN with $N = 10^6$.

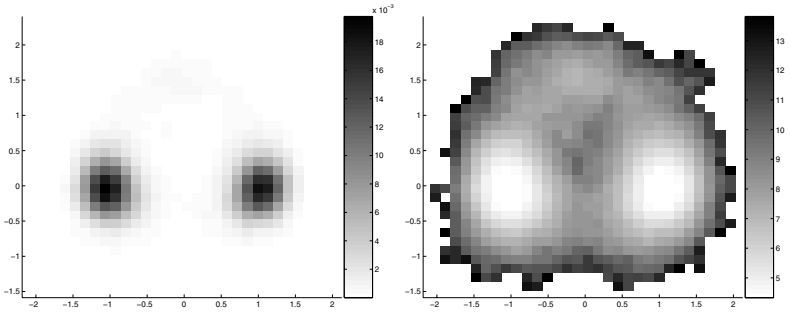

Fig. 4.4. Left: Stationary distribution of the transition matrix. Right: The free energy of the boxes.

Equidistant discretization

Next we decompose the rectangular domain into 30×30 equidistantly spaced boxes. In the left picture of Fig. 4.4 we depict the stationary distribution of the reversible transition matrix and in the right picture we plot the corresponding free energy.

In Fig. 4.5 we illustrate the results of bidirectional Dijkstra for both weights, the weights which are based on the transition probability and the weights (4.6) incorporating the free energy. For all computations we use the same sets A and B which consists of boxes covering the two deep minima, respectively. In the two columns of Fig. 4.5 we draw the edges which belong to the family of shortest paths between the sets A and B. From top to bottom we increase the parameter ϵ which results in increasing number of edges. The left column shows the edges of the most probable paths, whereas in the right column we draw the edges of paths which crosses the lowest free energy barriers.

As one can see, both methods detect for small ϵ the lower transition channel as the preferred one. But with increasing ϵ the families of transition paths differ. The family of transition paths resulting from transition probability weight for $\epsilon = 0.6$ includes paths which overcome the big barrier in the middle of the potential. Since the probability that the dynamics leaves the basin of attraction scales exponentially with the barrier which has to be overcame, the paths over the big barrier make no sense. Although the dynamics could get trapped in the upper shallow minima, the two lower saddle points rather should allow the dynamics to make transition than to go over the big barrier in the middle. This behavior is reflected by the free energy weight as can be seen in the last picture of the right column.

Acknowlegdments. This work has been supported by the DFG Priority Program 1095 "Analysis, Modeling and Simulation of Multiscale Problems" under grants number De 448/8 and Schu 1368/2.

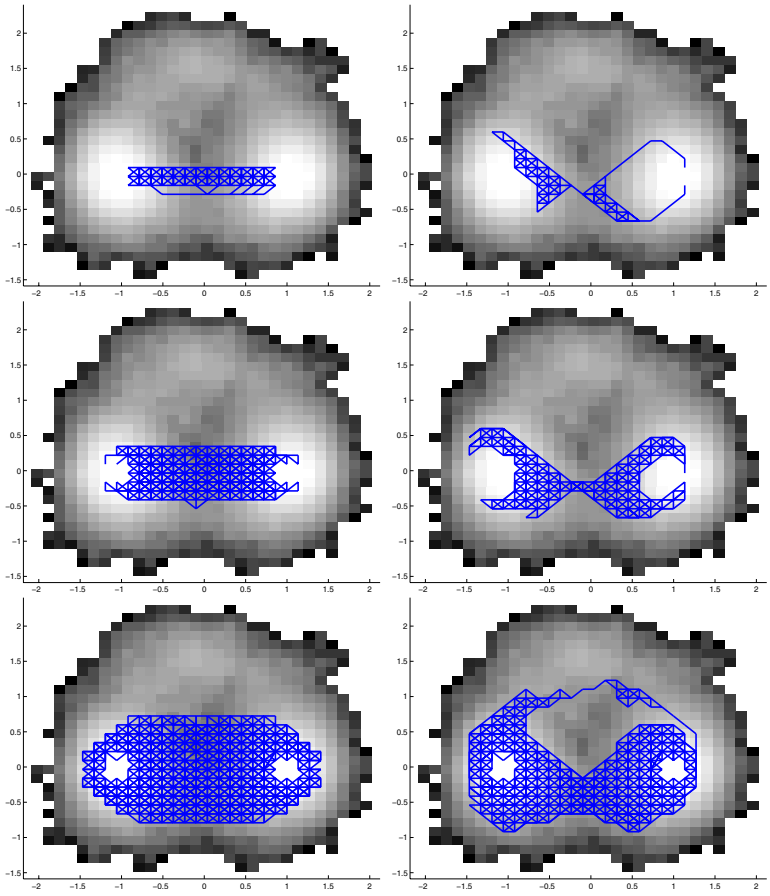

Fig. 4.5. Left column: Family of most probable paths between the set A (left minimum) and the set B (right minimum). From the top to the bottom we choose $\epsilon = 0.1, \epsilon = 0.3$ and $\epsilon = 0.6$. Right column: Family of paths which crosses the lowest free energy barriers. From the top to the bottom we choose $\epsilon = 0.01, \epsilon = 0.05$ and $\epsilon = 0.13$. (See page 697 for a colored version of the figure.)

References

[Bou98] N. Bouhmala. Impact of different graph coarsening schemes on the quality of the partitions. Technical Report RT98/05-01, University of Neuchatel, Department of Computer Science, 1998.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[DFJ01] M. Dellnitz, G. Froyland, and O. Junge. The algorithms behind GAIO – set oriented numerical methods for dynamical systems. In *Ergodic Theory, Analysis, and Efficient Simulation of Dynamical Systems*, pages 145–174, 2001.

[DFM99] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7), 789–812, 1999.

[DH97] M. Dellnitz and A. Hohmann. A subdivision algorithm for the computation of unstable manifolds and global attractors. *Numerische Mathematik*, 75, 293–317, 1997.

[DH*00] P. Deuflhard, W. Huisinga, A. Fischer, and C. Schütte. Identification of almost invariant aggregates in reversible nearly uncoupled Markov chains. *Lin. Alg. Appl.*, 315, 39–59, 2000.

[DH*97] M. Dellnitz, A. Hohmann, O. Junge, and M. Rumpf. Exploring invariant sets and invariant measures. *Chaos*, 7(2), 221–228, 1997.

[DJ99] M. Dellnitz and O. Junge. On the approximation of complicated dynamical behavior. *SIAM J. Numer. Anal.*, 36(2), 491–515, 1999.

[DJ02] M. Dellnitz and O. Junge. Set oriented numerical methods for dynamical systems. In B. Fiedler, G. Iooss, and N. Kopell, editors, *Handbook of Dynamical Systems II: Towards Applications*, pages 221–264. World Scientific, 2002.

[DMP95] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In D. Hsu, A. Rosenberg, and D. Sotteau, editors, *Interconnection Networks and Mapping and Scheduling Parallel Computations*, volume 21 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 57–73. AMS, 1995.

[EF*99] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. A. et al., editor, *Euro-Par'99 Parallel Processing*, LNCS 1685, pages 280–290, 1999.

[EMP00] R. Elsässer, B. Monien, and R. Preis. Diffusive load balancing schemes on heterogeneous networks. In *12th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 30–38, 2000.

[FM82] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. IEEE Design Automation Conf.*, pages 175–181, 1982.

[GJ79] M. Garey and D. Johnson. *COMPUTERS AND INTRACTABILITY - A Guide to the Theory of NP-Completeness*. Freemann, 1979.

[Gup97] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM J. of Research and Development*, 41, 171–183, 1997.

[HL94] B. Hendrickson and R. Leland. The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.

[HL95] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*. ACM, 1995.

[HM92] J. Hromkovič and B. Monien. The bisection problem for graphs of degree 4 (configuring transputer systems). In Buchmann, Ganzinger, and Paul, editors, *Festschrift zum 60. Geburtstag von Günter Hotz*, pages 215–234. Teubner, 1992.

[HP*96] A. Huang, L. Pivka, C. Wu, and M. Franz. Chua's equation with cubic nonlinearity. *Int. J. Bif. Chaos*, 6, 1996.

[Hun94] F. Hunt. A monte carlo approach to the approximation of invariant measures. *Random Comput. Dynam.*, 2, 111–133, 1994.

[Jun01] O. Junge. An adaptive subdivision technique for the approximation of attractors and invariant measures: Proof of convergence. *Dynamical Systems*, 16(3), 213–222, 2001.

[KK98a] G. Karypis and V. Kumar. *METIS Manual, Version 4.0*. University of Minnesota, Department of Computer Science, 1998.

[KK98b] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. of Parallel and Distributed Computing*, 48, 96–129, 1998.

[KK99] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1), 1999.

[KL70] B. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell Systems Technical J.*, pages 291–307, 1970.

[Lei92] F. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1992.

[MD97] B. Monien and R. Diekmann. A local graph partitioning heuristic meeting bisection bounds. In *8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[MP01] B. Monien and R. Preis. Bisection width of 3- and 4-regular graphs. In *26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, LNCS 2136, pages 524–536, 2001.

[MPD00] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Parallel Computing*, 26(12), 1609–1634, 2000.

[Pel96] F. Pellegrini. SCOTCH 3.1 user's guide. Technical Report 1137-96, LaBRI, University of Bordeaux, 1996.

[PM*94] R. Ponnusamy, N. Mansour, A. Choudhary, and G. Fox. Graph contraction for mapping data on parallel computers: A quality-cost tradeoff. *Scientific Programming*, 3, 73–82, 1994.

[Pre98] R. Preis. *The PARTY Graphpartitioning-Library, User Manual - Version 1.99*. Universität Paderborn, Germany, 1998.

[Pre00] R. Preis. *Analyses and Design of Efficient Graph Partitioning Methods*. Heinz Nixdorf Institut Verlagsschriftenreihe, 2000. Dissertation, Universität Paderborn, Germany.

[PS*03] S. Park, M. Sener, D. Lu, and K. Schulten. Reaction paths based on mean first-passage times. *J. Chem. Phys.*, 2003.

[Sen01] N. Sensen. Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows. In *Proc. European Symposium on Algorithms*, 2001.

[Sin93] A. Sinclair. *Algorithms for Random Generation & Counting: A Markov Chain Approach*. Progress in Theoretical Computer Science. Birkhäuser, 1993.

[Wal00] C. Walshaw. *The Jostle user manual: Version 2.2*. University of Greenwich, 2000.