Composability of Markov Models for Processing Sensor Data

Sander Evers

December 20, 2007

Abstract

We show that it is possible to apply the 'divide-and-conquer' principle in constructing a Markov model for sensor data from available sensor logs. The state space can be partitioned into clusters, for which the required transition counts or probabilities can be acquired locally. The combination of these local parameters into a global model takes the form of a system of linear equations with a confined solution space. Expected advantages of this approach lie for example in reduced (wireless) communication costs.

1 Introduction

In sensor data processing, a probabilistic model of the observed phenomenon is often useful. It can help to eliminate measurement noise, fill in missing values, detect faulty sensors or other abnormal conditions, or predict future behaviour. Other uses include inferring unobservable high-level variables and data compression.

This research investigates how to apply the 'divide-and-conquer' principle to the construction of a probabilistic model: as is often the case in computer science, we expect that dividing the task into local subtasks (smaller, but similar in nature) and then combining their results, rather than tackling the entire task as a whole, will have advantages in many situations. These advantages could consist of reducing communication costs or allowing for identity changes of observed objects—we detail these two examples below.

We restrict ourselves to a simple model with discrete time and state space, namely the *first-order Markov model*. The parameters to be acquired when constructing such a model consist of one value for each possible transition between two states; this value expresses the probability that when the process is currently in the source state, the corresponding transition will be the next one.

There are several ways to acquire such probabilities. One is to estimate them from domain knowledge, as is sometimes done in medical knowledge systems. However, in sensor data processing it is customary to deduce them from available data, namely logs of sensor readings. In the basic case, these readings map directly to states in the model of the process. One can then keep a count of how often every transition has occurred within a certain period (of sufficient length); from these counts, the probabilities are deduced. Thus, acquiring probabilities reduces to acquiring a count for all transitions. Therefore,

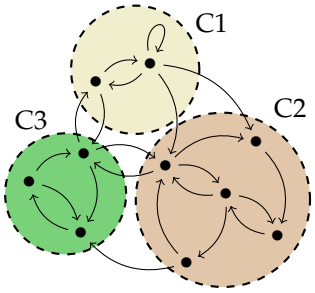


Figure 1: A discrete state space, partitioned into three clusters. Arrows represent the possible transitions between states.

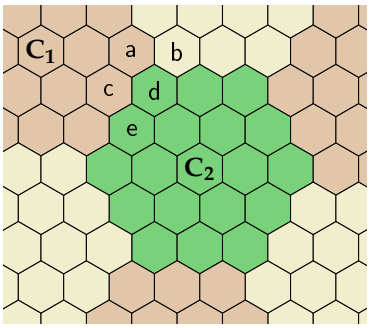


Figure 2: A 2D area as a discrete state space for a moving object's location. The states (a, b, ...) are called granules. Again, the state space is partitioned into clusters (C_1, C_2, ...). Transitions (not shown) are possible between adjacent granules. Self-transitions are also possible.

most of the article deals with the problem in this simpler form: to combine local transition counts into global transition counts. (In section 2, we show that it is also possible to start with local probabilities.)

Now, what does it mean to acquire parameters *locally*? Our assumption is that the state space is partitioned into *clusters*, as shown in Fig. 1, and that a model can be constructed for each of these partitions, based only on observations of the states in that partition. This model would contain values (counts or probabilities) for all transitions *within* this cluster, as well as for transitions entering or leaving the cluster, albeit with all external states aggregated into one. This away state can be thought of as 'out of local sensor reach' and, as a consequence, transitions between one internal state and several external states cannot be distinguished from each other. The main research question, then, becomes: can these local models be combined into an accurate global model?

To illustrate the situation in a somewhat more concrete way, we will hereafter consider the state space to be the location of a moving object in a two-dimensional area. This area is split up into discrete *granules*, with clusters formed by adjacent granules; an example is shown in Fig. 2. As a matter of fact, two-dimensional location is very suitable for our technique of combining local models: the graph of transitions between granules from different clusters—which we term the *inter-cluster transition graph*—is then always planar. Hence it is sparse, and as we will show later, this improves the accuracy of our technique.

We have opted for hexagonal granules arranged in a triangular grid, rather than squares in a square one, partially because its inter-cluster transition graph provides a better illustration of the problem, but also because it is an economical way of placing sensors on a plane: if every sensor can observe objects within the same distance around it, this is the optimal way to cover the whole area.

As we mentioned above, we expect that acquiring transition counts locally has advantages. We briefly explore two example scenarios with different advantages:

- *Avoid communication over cluster borders in a multi-hop wireless network.* Each of the granules in Fig. 2 is observed by one sensor (e.g. a camera) that can detect when the moving object is present. Transitions are not observed explicitly; they are deduced by comparing logs from two adjacent sensors, and aggregated into transition counts. The sensors are battery-powered and communicate wirelessly with their neighbours, forming a multi-hop network. At the center of each cluster, a wired hub collects the transition counts every once in a while. Due to the high communication costs, it is important to do the aggregation from logs into counts close to the source sensors.
  Acquiring the counts globally can mean two things: either the border sensors of the different clusters have to communicate with *each other* to compare logs, or they have to send their logs to their *respective hubs*, so that they can be compared using the wired network. The former option entails some additional wireless communication, and technical difficulties if clusters use incompatible communication standards; the latter option means a lot of additional communication from border nodes to hub. Alternatively, acquiring only *local* counts means that one registers at the border when the object disappears or appears in this cluster (by comparison of the logs of the border sensors and their neighbours within the cluster); it is represented as a transition to or from the away state, aggregated into the count, and sent to the hub at low cost. Afterwards, the data from all hubs can be combined (at low cost) using the wired network, in order to deduce the counts of the border transitions using the technique we present in the following sections.

- *Allow for identity changes of objects when crossing cluster borders.* There can also be a more fundamental reason for not being able to distinguish inter-cluster transitions: objects might have a different identity in different clusters. The reasons for this can be technical (one cluster uses fixed Bluetooth receivers as sensors, while the other uses cameras) or privacy-related (all clusters use Bluetooth, but users adopt a different identity in each cluster to remain anonymous). In both cases, it is not possible to create a global model of one specific object, but a goal can be to create a model of the "average object". Still, we cannot directly infer transition counts by comparing logs from c and d (Fig. 2): the objects appearing in d at time t might not be the objects leaving from c at $t-1$; they could also have come from a.

Note that in both examples, the difference between obtaining the transition counts locally and globally only manifests itself at the cluster borders: in the global case, the transition counts crossing a border are directly obtained, while in the local case we only obtain an aggregate at both sides of the border.

Combining the local models into a global model therefore amounts to deducing these specific transition counts from the aggregates. To illustrate the nature of this problem, consider the count of the transition from c to d, written $\#(\mathsf{c},\mathsf{d})$. In cluster C_1, this transition is observed as an exit transition $(\mathsf{c},\mathsf{away}_1)$, and in cluster C_2 it is an entrance transition $(\mathsf{away}_2,\mathsf{d})$. Nevertheless, from the

arrangement of the granules it follows that:

$$\#(\mathsf{c}, \mathsf{away}_1) = \#(\mathsf{c}, \mathsf{d}) + \#(\mathsf{c}, \mathsf{e})$$
$$\#(\mathsf{away}_2, \mathsf{d}) = \#(\mathsf{c}, \mathsf{d}) + \#(\mathsf{a}, \mathsf{d}) + \#(\mathsf{b}, \mathsf{d}).$$

Together with the other transition counts, this forms a system of linear equations, whose solutions can be close enough together to enable a good approximation of $\#(\mathsf{c}, \mathsf{d})$. Previously[1], we have shown that under quite restrictive conditions on the inter-cluster transition graph, there is one exact solution. Currently, by also considering approximations, we present a method that broadens these conditions.

2 Traces, counts, frequencies and probabilities

In this section, we formalize the problem. Although the main goal remains constructing a probabilistic model, it is easier to deal with the problem in terms of transition counts, or more generally, transition *frequencies*. Hence, we will first state the problem definition in terms of counts and frequencies, and then outline its relationship to the Markov models.

We assume that the *structure* of the state space, consisting of the possible *states* and *transitions*, is known. This structure is expressed as a directed graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$, of which the vertices $V^{\rightarrow}$ correspond to states and the edges $E^{\rightarrow} \subseteq V^{\rightarrow} \times V^{\rightarrow}$ to transitions. We call this graph the *global transition graph*.

Furthermore, the states are partitioned in n local clusters by a partition function $p : V^{\rightarrow} \rightarrow \{1, \ldots, n\}$. In cluster i ($1 \leq i \leq n$), a local state v with $p(v) = i$ can be observed as such, while the other states are all 'out of local sensor reach', and together form one state away_i. We express this using an observation function $o_i : V^{\rightarrow} \rightarrow V_i^{\rightarrow}$:

$$o_i(v) \stackrel{\text{def}}{=} \begin{cases} v & \text{if } p(v) = i \\ \mathsf{away}_i & \text{if } p(v) \neq i \end{cases}$$

This induces a *local transition graph* $C_i^{\rightarrow} = (V_i^{\rightarrow}, E_i^{\rightarrow})$ for cluster i. Its vertices $V_i^{\rightarrow}$ are those states that o_i maps to, and its edges are derived from the global transitions by mapping their source and target state into their local observations. Formally:

$$V_i^{\rightarrow} = \{o_i(v) | v \in V^{\rightarrow}\}$$
$$E_i^{\rightarrow} = \{(o_i(v), o_i(w)) | (v, w) \in E^{\rightarrow}\}$$

An example is shown in Fig. 3 for a small part of the location state space in Fig. 2. The three local transition graphs in Fig. 3a are obtained from the global transition graph in Fig. 3b using partition function $p(\mathsf{a}) = p(\mathsf{c}) = 1$, $p(\mathsf{d}) = p(\mathsf{e}) = 2$, $p(\mathsf{b}) = 3$. The colors serve as a visual aid for this partition function.

A *trace* T is a sequence of consecutive states, where $T(\tau) \in V$ represents the state at time τ. A trace is consistent with the possible transitions, i.e. if $T(\tau) = s$ and $T(\tau + 1) = t$ then $(s, t) \in E^{\rightarrow}$. Traces arise as observations of the modelled process in action. Our main assumption is that it is impossible to directly obtain a *global trace* of the system; however it is possible to obtain *local traces*. A

local trace is a sequence of local observations of consecutive (global) states. The local trace observed in cluster i, derived from an unobservable global trace T, is written $o_i[T]$. It is defined by applying o_i pointwise to each state in T:

$$o_i[T](\tau) \stackrel{\text{def}}{=} o_i(T(\tau))$$

Local trace $o_i[T]$ contains states in $V_i^{\rightarrow}$ and is consistent with the transitions in $E_i^{\rightarrow}$.

If local traces $o_i[T]$ were available for all i, it would be trivial to reconstruct T (and derive global transition counts or probabilites from it). Our assumption is that, instead of local traces, only their corresponding transition counts are available. Formally, a *transition count* $\#_T(s,t)$ is the number of times that the transition (s,t) occurs in trace T (i.e. the number of distinct times τ for which $T(\tau) = s \wedge T(\tau+1) = t$).

Thus, we arrive at a first formulation of the problem statement, in terms of transition counts:

- Assume we have, for each cluster i, a count $\#_{o_i[T]}(s,t)$ of all local transitions $(s,t) \in E_i^{\rightarrow}$. All counts are derived from the same (unobservable) trace T. Is it possible to deduce or approximate $\#_T(s,t)$ for all global transitions $(s,t) \in E^{\rightarrow}$?

Note that, in this formulation, all transition counts should be acquired over the same period. We can generalize the problem to a broader setting in which this is not required. For this we need to make two extra assumptions, namely that the "average behaviour" of the process does not change, and that the observed traces are long enough to exhibit this average behaviour. Formally, in terms of Markov models, the first assumption is that the process is homogeneous (not time dependent) and ergodic (keeps returning to all states). The second assumption means that the ratios of observed transition counts (see below) approximate the model's probabilities.

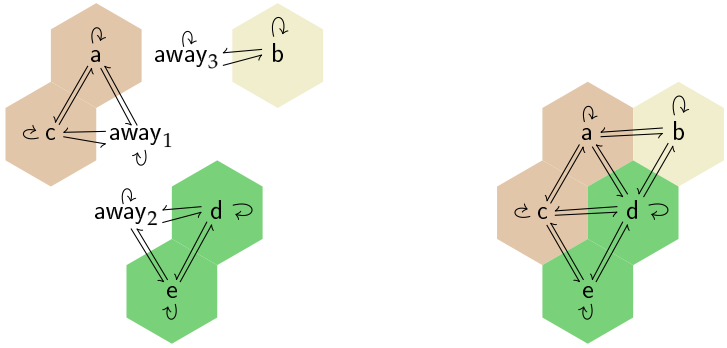
When these assumptions are satisfied, the local observations can originate from different global traces, that is, they can be performed at different times. The length of these traces may also vary; in order to compare transition counts regardless, we normalize them w.r.t. the trace length. These normalized counts are termed *transition frequencies* and are defined:

$$F_T(s,t) \stackrel{\text{def}}{=} \frac{\#_T(s,t)}{\sum_{x,y} \#_T(x,y)}$$

From now on, our formulation of the problem will be in terms of these more general transition frequencies:

- Assume we have, for each cluster i, a frequency $F_{o_i[T_i]}(s,t)$ of all local transitions $(s,t) \in E_i$. These frequencies may now originate from different (unobservable) traces T_i, but the observed process should be homogeneous and ergodic, and the traces should be sufficiently long (see above). Is it possible to deduce or approximate $F_T(s,t)$ (with T an arbitrary, sufficiently long trace) for all global transitions $(s,t) \in E$?

However, the solution method presented in the following sections also applies to the first problem statement, which is more restrictive in that only simultaneous observations can be combined, but more permissive w.r.t. the nature of the process and the length of the observations.



(a) For each of the three colored clusters (C_1, C_2, C_3), we know the frequencies (not shown) on the edges of the *local transition graph*.

(b) We know the structure of the *global transition graph*, and want to obtain the frequencies on the edges. Intra-cluster frequencies like $F(\mathsf{c}, \mathsf{a})$ can be directly copied; inter-cluster frequencies like $F(\mathsf{c}, \mathsf{d})$ cannot.

Figure 3: Problem statement.

The problem statement is illustrated in Fig. 3. We know the frequencies on the local transition graphs, and the goal is to find the frequencies on the global transition graph. Some of them can be directly copied, namely those between states of the same cluster. For the others, which we term *inter-cluster* frequencies, the structure of the global transition graph defines a system of linear equations. For example, the relevant equations for the (c, d) transition are:

$$F_{o_1[T_1]}(\mathsf{c}, \mathsf{away}_1) = F_T(\mathsf{c}, \mathsf{d}) + F_T(\mathsf{c}, \mathsf{e})$$
$$F_{o_2[T_2]}(\mathsf{away}_2, \mathsf{d}) = F_T(\mathsf{c}, \mathsf{d}) + F_T(\mathsf{a}, \mathsf{d}) + F_T(\mathsf{b}, \mathsf{d})$$

In the next section, we will show how to solve the system of equations.

Closely related to transition counts and frequencies are the transition probabilities that form the parameters of a first-order Markov model. When we make the assumption that the observed process can be described by such a model, the observed transition counts or frequencies provide the so-called *maximum likelihood estimate* of these parameters:

$$\mathsf{P}(X_{\tau+1} = t | X_\tau = s) = \frac{\#(s,t)}{\sum_y \#(s,y)} = \frac{F(s,t)}{\sum_y F(s,y)}$$

The basic use case for applying our technique with regard to Markov models is as follows. We observe local transition frequencies in the clusters, use these to derive the global transition frequencies, and use the latter to determine the parameters of a global Markov model (using the above formula). In other words, the problem is in terms of frequencies, and its solution is only translated to probabilites afterwards.

However, it is also possible that, instead of starting with local transition frequencies, we want to start with local Markov models on the cluster state spaces C_i. For example, the parameters of these Markov models could have been

estimated from noisy observations using expectation maximization (a *hidden* Markov model setting). Under certain restrictions, it is possible to:

1. transform these local Markov models into local frequencies of long-term average behaviour;
2. then, use our technique to combine them into global frequencies of long-term average behaviour;
3. finally, use the above formula to calculate the maximum likelihood estimate for the global Markov model.

To perform step 1, we can use the formula

$$F(s,t) = \pi_{C_i}(s)\mathsf{P}(X_{\tau+1} = t | X_\tau = s).$$

In this formula, a crucial role is played by the *stationary distribution* π_{C_i}, which can be derived from the collective probabilities in the local Markov model on C_i. For a further discussion on this transformation and its applicability, we refer to our earlier article[1].

3 Solving a flow with known vertex values

In the previous section, we formulated the problem of finding global transition frequencies. In this section, we analyse this problem using a combination of linear algebra and graph theory. The frequencies form a *flow* on the global transition graph; we are looking for the unknown values of this flow, which are constrained by known sums.

3.1 Problem definition

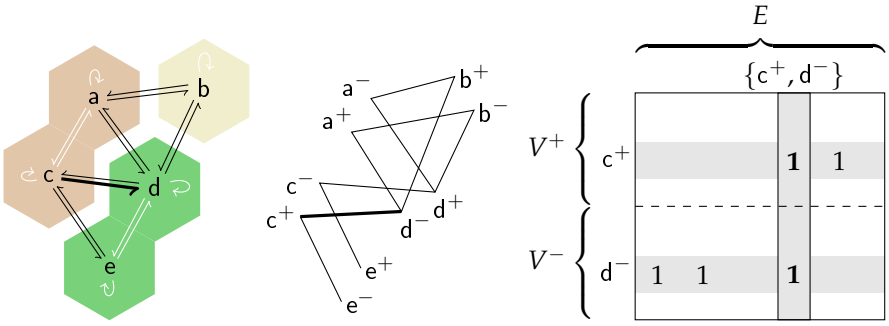
Given a directed graph $G^\rightarrow = (V^\rightarrow, E^\rightarrow)$, a *flow* is a function $f^\rightarrow : E^\rightarrow \to \mathbb{R}$ assigning a value to each edge. A given flow $f^\rightarrow$ establishes, for each vertex $v \in V^\rightarrow$, its inflow $f^-(v)$ and outflow $f^+(v)$:

$$\begin{aligned} f^-(v) &= \sum_{u|(u,v)\in E^\rightarrow} f^\rightarrow(u,v) \\ f^+(v) &= \sum_{w|(v,w)\in E^\rightarrow} f^\rightarrow(v,w) \end{aligned} \qquad \text{(IN-OUT-FLOW)}$$

Our problem statement is described by exactly these equations, but with the values of $f^\rightarrow$ as unknown variables, and a known inflow and outflow for each vertex.

As we stated earlier, when considering the global transition graph, some values of $f^\rightarrow$ *are* already known, namely the frequencies on transitions within a cluster. For convenience, we disregard these in our further analysis of the problem. So, for every transition (s,t) within partition i (so $p(s) = p(t) = i$):

- we leave its edge out of the graph altogether, so that the remaining graph $G^\rightarrow = (V^\rightarrow, E^\rightarrow)$ consists only of inter-cluster transitions $E^\rightarrow$ and the corresponding vertices $V^\rightarrow$ (border granules)—see Fig. 4a; so, we also leave out vertices in which no transition starts or ends anymore;



(a) Inter-cluster transition graph $G^{\rightarrow}$ (also shown, in white, are the left out intra-cluster transitions)

(b) G, an undirected representation of $G^{\rightarrow}$ ('uncoiled graph')

(c) $\mathbf{N}$, the incidence matrix of G. Values in the white part are not shown; the grey part has 0s except where 1s are shown.

Figure 4: Three representations of the inter-cluster transition graph. The transition from c to d is highlighted everywhere.

- we subtract its known frequency $F_{o_i[T_i]}(s,t)$ from $f^+(s)$ and $f^-(t)$; after doing this for all said transitions, only inter-cluster frequencies remain, and they sum up to the known $F_{o_i[T_i]}(s, \mathsf{away}_i)$ and $F_{o_i[T_i]}(\mathsf{away}_i, t)$; so, for every vertex v that is left over, the known in- and outflow are:

$$f^-(v) = F_{o_{p(v)}[T_{p(v)}]}(\mathsf{away}_{p(v)}, v)$$
$$f^+(v) = F_{o_{p(v)}[T_{p(v)}]}(v, \mathsf{away}_{p(v)})$$

We term the graph that is left over the *inter-cluster transition graph*. The problem statement can then be summarized as follows: given inter-cluster transition graph $G^{\rightarrow}$ and the above values of f^+ and f^-, we are looking for solutions $f^{\rightarrow}$ that satisfy (IN-OUT-FLOW). Essentially, this amounts to solving a system of linear equations in which all coefficients are 1 or 0. Each edge corresponds to a variable, and each vertex contributes the two equations that constitute (IN-OUT-FLOW).

However, to analyze the system using graph-theoretic concepts, it is more convenient to have a one-to-one correspondence between vertices and equations. With this goal in mind, we introduce a different representation of the graph $G^{\rightarrow}$ and the corresponding equations. We term this representation the *uncoiled* system.[1]

Definition 1. Given the directed graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$ with n vertices labelled $\{v_1, v_2, \ldots, v_n\}$, we define its *uncoiled graph* $G = (V, E)$. This graph G is a bipartite undirected graph with partitions $V = V^+ + V^-$, where $V^+ = \{v_1^+, v_2^+, \ldots, v_n^+\}$ and $V^- = \{v_1^-, v_2^-, \ldots, v_n^-\}$. Each vertex v_i from the original graph is represented twice in V: as a source v_i^+ and as a target v_i^-. The edge set E contains an undirected edge $\{v_i^+, v_j^-\}$ iff $E^{\rightarrow}$ contains a directed edge (v_i, v_j).

[1]The name *uncoiled* is chosen after the effect it has on loops in the graph. In our application, however, we do not encounter loops, because a loop is a transition within a cluster, and has been filtered out.

For an example, see Fig. 4b. A flow $f^{\rightarrow}$ on the edges of the directed graph is represented by a flow f on the corresponding edges of G: $f\{v_i^+, v_j^-\} = f^{\rightarrow}(v_i, v_j)$. Again, this flow is assumed to be unknown, while the values of certain sums are known. In the original directed system, these values were represented by two different functions f^+ and f^- on each vertex; in the uncoiled system this role is played by one function $f^{\pm} : V \to \mathbb{R}$ on the two different types of vertex. The two representations are related as follows:

$$f^{\pm}(v_i^+) = f^+(v_i)$$
$$f^{\pm}(v_i^-) = f^-(v_i)$$

The problem statement can now be formulated in terms of the uncoiled system: given the uncoiled representation G of inter-cluster graph $G^{\rightarrow}$, and the vertex sums $f^{\pm}$ above, find a flow $f : E \to \mathbb{R}$ that satisfies, for all v,

$$f^{\pm}(v) = \sum_{w|\{v,w\}\in E} f\{v, w\}. \qquad \text{(VERTEX-FLOW)}$$

As in the above equation in this article we will often refer to the vertices in V regardless of whether they are in partition V^+ or V^-. In these cases we also use the variable v_i, imposing a certain ordering: $V = \{v_1, v_2, \ldots, v_{2n}\}$. Likewise, we order the edges as $\{e_1, e_2, \ldots, e_m\}$; we keep the convention that $|V| = 2n$ and $|E| = m$.

With such orderings in place, the uncoiled system can easily be written in the matrix-vector form conventional in linear algebra. The matrix of coefficients corresponds to G's *incidence matrix* $\mathbf{N}$ (Fig. 4c):

$$\mathbf{N}_{ij} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{if } v_i \notin e_j \end{cases}$$

Using the same orderings, the functions $f^{\pm}$ and f are represented as vectors:

$$\mathbf{f}^{\pm} \stackrel{\text{def}}{=} (f^{\pm}(v_1), f^{\pm}(v_2), \ldots, f^{\pm}(v_{2n}))^{\mathsf{T}}$$
$$\mathbf{f} \stackrel{\text{def}}{=} (f(e_1), f(e_2), \ldots, f(e_m))^{\mathsf{T}}$$

The matrix-vector formulation of our problem statement is then as follows: given $\mathbf{N}$ and $\mathbf{f}^{\pm}$, find a solution $\mathbf{f}$ of the system

$$\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}. \qquad \text{(VECTOR-FLOW)}$$

In principle, this system can be solved using basic textbook techniques such as Gauss-Jordan elimination. Instead, we exploit the special structure of $\mathbf{N}$—the equations are partitioned into two sets, and each variable participates in exactly one equation of each set—and solve parts using graph theory. This provides us with:

- insight in the structure of the solutions: the existence and dimension of the solution space are related to components and cycles in the graph;
- a fast method of solving the system.

3.2 Solution (summary)

A basic result in linear algebra is that every solution $\mathbf{f}$ of $\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}$ can be written in the form

$$\mathbf{f} = \mathbf{p} + \mathbf{k}$$

where $\mathbf{p}$ is any fixed particular solution of the system, and $\mathbf{k}$ is a solution of $\mathbf{N}\mathbf{k} = \mathbf{0}$. All possible solutions of this last equation define $\mathbf{N}$'s *null space* or *kernel*:

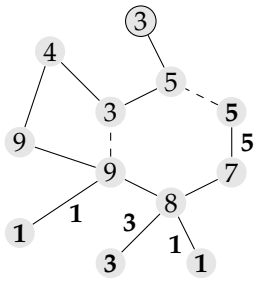
$$\operatorname{Ker}\mathbf{N} \stackrel{\text{def}}{=} \{\mathbf{k} | \mathbf{N}\mathbf{k} = \mathbf{0}\}$$

Conversely, all vectors $\mathbf{f}$ that can be written as $\mathbf{f} = \mathbf{p} + \mathbf{k}$ are solutions. Hence, to specify all the solutions of the system, it is enough to give one particular solution $\mathbf{p}$ and a specification of $\operatorname{Ker}\mathbf{N}$. This specification has the form of a *basis*, a minimal set of vectors of which every $\mathbf{k}$ is a linear combination.

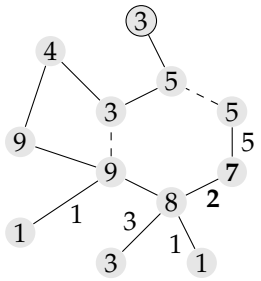
We show that both can be found by considering an arbitrary spanning tree of G:

- A particular solution can be found by solving the system with only the edges of the tree, and assigning $f(e) = 0$ for each left out edge e.
- The basis vectors correspond to the fundamental cycles w.r.t. this tree: cycles that are obtained by putting one of the left-out edges back in.

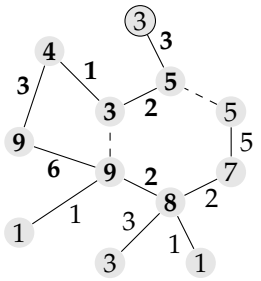
Informally, these points are illustrated in Fig. 5 and Fig. 6, respectively. In the next section, we give a more formal treatment of the approach, and prove its correctness. Although the formal treatment is complete when read independently, we advise the reader to study the figures for a quick insight in the linear system and its solutions.



(a) Partial solution after the traversal of one-node subtrees. These nodes connect only to one edge, so the value on this edge should equal the node value.

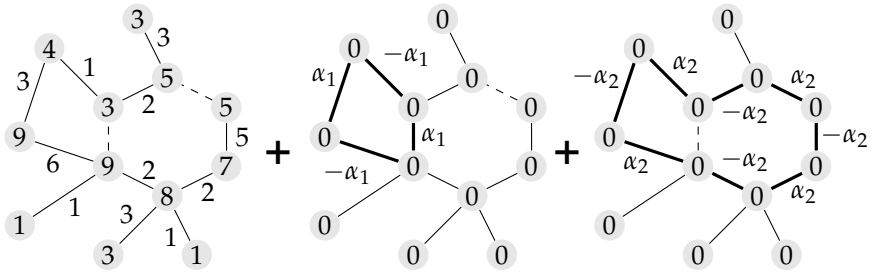


(b) Now, there is only one empty edge left for the 7 node, so we can deduce its value:
$7 - 5 = \mathbf{2}$.



(c) Likewise, we continue up the tree. After having filled in the last edge, we check whether the sum in the root node (= 3) is correct. It is, so we have found a solution.

Figure 5: Finding a particular solution. The goal is to find f values at the edges that sum up to the given $f^{\pm}$ values in the nodes. This is done by a traversal of a spanning tree, here rooted in the topmost node. After traversal of each subtree, the value of the connecting edge can be deduced. Edges not in the tree (dashed) are given the value 0.



(a) Particular solution. **(b)** Fundamental cycle 1. **(c)** Fundamental cycle 2.

Figure 6: Each fundamental cycle leads to solutions of $\mathbf{Nf} = \mathbf{0}$. By adding them to the particular solution of the original system $\mathbf{Nf} = \mathbf{f}^{\pm}$, we find other solutions of that system. For example, the edges of the top-left node get the values $3 + \alpha_1 - \alpha_2$ and $1 - \alpha_1 + \alpha_2$, respectively. Together, these still add up to 4.

3.3 Solution (formalization and proof)

A key part in our solution method is played by a spanning tree of G. For completeness, we take into account that G might not be fully connected; we then use multiple trees that span the components of G.

Definition 2. A *component* $C = (V_C, E_C)$ is a subgraph of G, where $V_C \subseteq V$ is a maximal set of transitively connected vertices, and $E_C = \{\{v, w\} \in E | v, w \in V_C\}$ the corresponding edges. In other words, a component consists of all the vertices and edges reachable from a certain vertex. We also define V_C^+ and V_C^-, the classes of the bipartition within C:

$$V_C^+ \stackrel{\text{def}}{=} V_C \cap V^+$$
$$V_C^- \stackrel{\text{def}}{=} V_C \cap V^-$$

Definition 3. A *spanning forest* S is a subgraph of G consisting of trees, each of which *spans* a component C of G. This means that the vertices of C also form the vertices of the tree, and all the edges in the tree occur in C, but not necessarily vice versa.

First, we show how to find a particular solution $\mathbf{p}$ under the condition that G contains no cycles: the spanning forest is G itself. We do this using Algorithm 1 (which has already been shortly demonstrated in Fig. 5). For each component of G, it picks an arbitrary root and does a depth-first traversal of the tree, incrementally building a solution f. (Note that the algorithm also has a provision for cycles, which we do not use yet.)

We give a necessary and sufficient condition for the existence of a solution that depends on the values of $f^{\pm}$ for each component of G. We prove that when it exists, it is unique and Algorithm 1 will find it.

Theorem 1. *A system $(G, f^{\pm})$ with G a bipartite forest has a solution iff for each component tree C of G the following equation holds:*

$$\sum_{v \in V_C^+} f^{\pm}(v) = \sum_{v \in V_C^-} f^{\pm}(v)$$

Input: bipartite graph $G = (V, E)$ and vertex summations $f^{\pm} : V \to \mathbb{R}$
Output: solution f that matches the summations as in (VERTEX-FLOW)

$f \leftarrow$ the empty (partial) function on $E \to \mathbb{R}$
$visited \leftarrow \emptyset$

while $visited \neq V$ **do**
  $root \leftarrow$ an arbitrary element of $(V - visited)$
  SolveSubtree($root, \emptyset$)
  if $f^{\pm}(root) \neq \sum_w f\{root, w\}$ **then error** 'no solution exists'
end

procedure SolveSubtree(*root*,*maybeparent*)
  // *maybeparent* records the node we came from, to prevent going back
  if $root \in visited$ **then**
    // cycle detected
    $f(\{root\} \cup maybeparent) \leftarrow 0$
  else
    $visited \leftarrow visited \cup \{root\}$
    foreach $v \in (V - maybeparent)$ **such that** $\{root, v\} \in E$ **do**
      SolveSubtree($v, \{root\}$)
      $f\{root, v\} \leftarrow f^{\pm}(v) - \sum_w f\{v, w\}$
    end
  end
end

Algorithm 1: A depth-first traversal of each component of the graph, recording a particular solution in f (and implicitly constructing a spanning tree).

If this is the case, Algorithm 1 will find the unique solution f; if not, Algorithm 1 will halt with an error.

Proof. We first prove, by induction on the tree structure of a component, that the algorithm finds a unique solution that satisfies all $f^{\pm}$ equations, except those for the roots of the components. The induction hypothesis is as follows:

> After a call SolveSubtree(*root*, *maybeparent*), the unique f values for all edges in the subtree are filled in, satisfying the $f^{\pm}$ equations for all vertices in the subtree except its root.

For subtrees of one vertex, this holds trivially. To satisfy the hypothesis for the next level, we consider a tree consisting of a root *Root* with edges to n subtrees for which the hypothesis holds. Each subtree has a root $root_i$ whose summation $f^{\pm}(root_i)$ has to be satisfied after we fill in the $f\{Root, root_i\}$ value. Because all the other edges $\{root_i, w\}$ have already been filled in (by the induction hypothesis; they are in the subtree), there is only one option for this value: $f\{Root, root_i\}$ must equal $f^{\pm}(root_i) - \sum_w f\{root_i, w\}$. Thus, we find a unique solution for each $f\{Root, root_i\}$ value, and satisfy all the $f^{\pm}(root_i)$ equations: the induction hypothesis now also holds for the tree under consideration.

Thus, after the SolveSubtree call on a component's root, the constructed f satisfies all the component's equations but one. When we group together the

vertices with the same distance $\mathrm{d}(v)$ to the root vertex, these equations imply for $n \geq 1$:

$$\sum_{\mathrm{d}(v)=n} f^{\pm}(v) = \sum_{\substack{\mathrm{d}(u)=n-1 \\ \mathrm{d}(v)=n}} f\{u,v\} + \sum_{\substack{\mathrm{d}(v)=n \\ \mathrm{d}(w)=n+1}} f\{v,w\}$$

and hence

$$\sum_{\substack{\mathrm{d}(u)=n-1 \\ \mathrm{d}(v)=n}} f\{u,v\} = \sum_{\mathrm{d}(v)=n} f^{\pm}(v) - \sum_{\substack{\mathrm{d}(v)=n \\ \mathrm{d}(w)=n+1}} f\{v,w\}.$$

We first fill in $n = 1$, and recursively substitute the negated term at the end by using the above equation for $n = 2, 3, \ldots$:

$$\sum_{\substack{\mathrm{d}(u)=0 \\ \mathrm{d}(v)=1}} f\{u,v\} = \sum_{\mathrm{d}(v)=1} f^{\pm}(v) - \sum_{\mathrm{d}(v)=2} f^{\pm}(v) + \sum_{\mathrm{d}(v)=3} f^{\pm}(v) - \sum_{\mathrm{d}(v)=4} f^{\pm}(v) + \ldots$$

Notice that all the vertices at odd distance from the root appear in a positive term, and all the vertices at even distance and appear in a negative term, so:

$$\sum_{\substack{\mathrm{d}(u)=0 \\ \mathrm{d}(v)=1}} f\{u,v\} = \sum_{\mathrm{d}(v)\text{ odd}} f^{\pm}(v) - \sum_{\mathrm{d}(v)>0\text{ and even}} f^{\pm}(v).$$

Now, the constructed f will satisfy *all* the $f^{\pm}$ equations iff the equation $f^{\pm}(\mathit{root}) = \sum_w f\{\mathit{root}, w\}$ holds. Its right-hand side is equivalent to the left-hand side of the above equation, so we can substitute:

$$f^{\pm}(\mathit{root}) = \sum_{\mathrm{d}(v)\text{ odd}} f^{\pm}(v) - \sum_{\mathrm{d}(v)>0\text{ and even}} f^{\pm}(v).$$

Suppose, without loss of generality, that *root* is in V_C^+. Then all vertices at odd distance are in V_C^- and all vertices at even distance are in V_C^+. Moving the negated term to the left, the equation rewrites to

$$\sum_{v \in V_C^+} f^{\pm}(v) = \sum_{v \in V_C^-} f^{\pm}(v).$$

Iff for each component this equation is satisfied, f is a solution for the system $(G, f^{\pm})$. □

As a corollary, note that forests with $\mathbf{f}^{\pm} = \mathbf{0}$ have the unique solution $\mathbf{f} = \mathbf{0}$. This may be directly clear from the algorithm; it also follows from the fact that $\mathbf{N0} = \mathbf{0}$ is true for any $\mathbf{N}$, combined with the uniqueness property from the theorem.

We now extend our method to systems in which G can contain cycles. Basically, we apply the same algorithm, only on a spanning forest of G, and we fix the other edges at 0. In fact, Algorithm 1 already does all this; depth-first traversal of a graph is a well-known way of creating a spanning tree. When the algorithm encounters an already visited node, it considers the edge by which it is reached as not belonging to the tree, and it backtracks.

Assume, without loss of generality, that the left out edges $E(G) - E(S)$ are last in the order e_i. That means that G's incidence matrix $\mathbf{N}$ can be split in two parts:

$$\mathbf{N} = [\mathbf{N}_S, \mathbf{N}_L]$$

Algorithm 1 may find a solution $\mathbf{f}_S$ for $\mathbf{N}_S\mathbf{f}_S = \mathbf{f}^{\pm}$. When this is the case, there is also a solution for $\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}$, namely $\mathbf{f} = (\mathbf{f}_S; \mathbf{0})$. The question is if the implication also works the other way around: if $(G, f^{\pm})$ has a solution, does $(S, f^{\pm})$ also have one? The answer is positive; we can construct this solution using G's fundamental cycles.

Definition 4. Let G be a graph with m edges $\{e_1, e_2, \ldots, e_m\}$, of which the first s edges $\{e_1, e_2, \ldots, e_s\}$ form a spanning tree S. For each q with $s < q \leq m$, the *fundamental cycle* $\mathbf{f}_q$ consists of e_q and the edges in S that form a cycle with it. We identify $\mathbf{f}_q$ not only with a set of edges, but also with a vector of length m with alternating 1 and -1 on the positions corresponding to cycle edges:

$$\begin{aligned}
\mathbf{f}_q(q) &\stackrel{\text{def}}{=} 1 \\
\mathbf{f}_q(i) &\stackrel{\text{def}}{=} 1, && \text{for } e_i \text{ on the cycle at an even distance of } e_q \\
\mathbf{f}_q(j) &\stackrel{\text{def}}{=} -1, && \text{for } e_j \text{ on the cycle at an odd distance of } e_q \\
\mathbf{f}_q(k) &\stackrel{\text{def}}{=} 0, && \text{for } e_k \text{ not on the cycle}
\end{aligned}$$

See also the examples in Fig. 6, where f_q has been multiplied by α_1 and α_2, respectively. Note that the even and odd distances only make sense because the cycles in our graph are always of even length (because the graph is bipartite). Vector $\mathbf{f}_q$ forms a solution of $\mathbf{N}\mathbf{f} = \mathbf{0}$, because in the vertex sums $f^{\pm}$, every vertex on the cycle has two terms that cancel each other out; all the other terms, also for the other vertices, are zero.

Now, suppose that $(G, f^{\pm})$ has a solution $\mathbf{g}$. Then we can subtract a multiple of every fundamental cycle from it, to obtain a vector $\mathbf{f}$ that is zero on $e_{s+1}, \ldots, e_m$:

$$\mathbf{f} \stackrel{\text{def}}{=} \mathbf{g} - \sum_{s<q\leq m} \mathbf{g}(q)\mathbf{f}_q$$

Then $\mathbf{N}\mathbf{f} = \mathbf{N}\mathbf{g} - \sum_{s<q\leq m} \mathbf{g}(q)\mathbf{N}\mathbf{f}_q = \mathbf{f}^{\pm} - \mathbf{0}$, so $\mathbf{f}$ is a solution to $(G, f^{\pm})$. This implies that $\mathbf{f}$ without the trailing zeros is a solution to $(S, f^{\pm})$. So, the problem of finding a particular solution to a general $(G, f^{\pm})$ system can be reduced to finding a solution to a spanning graph. We can now generalize Theorem 1 by extending it from forests to general graphs.

Theorem 2. *A system $(G, f^{\pm})$ with G a bipartite graph has a solution iff for each component C of G the following equation holds:*

$$\sum_{v\in V_C^+} f^{\pm}(v) = \sum_{v\in V_C^-} f^{\pm}(v)$$

If this is the case, Algorithm 1 will find a solution f; if not, Algorithm 1 will halt with an error.

Proof. As we have argued above, there exists a solution of $(G, f^{\pm})$ iff there exists a solution for $(S, f^{\pm})$, with S an arbitrary spanning forest of G. By Theorem 1, the latter is the case if the equation holds for the V_C^+ and V_C^- sets of the tree components; these are the same vertex sets that make up G's components.

Algorithm 1 creates this spanning forest by depth-first graph traversal from an arbitrary root vertex of each component C, and finds its solution if it exists (Theorem 1). It fills in 0 at the left-out edges, which turns it into a solution of $(G, f^{\pm})$. □

Now we have found a particular solution $\mathbf{p}$, the next step is to find the solutions for $\mathbf{Nf} = \mathbf{0}$. We can specify these by specifying a *basis* for $\operatorname{Ker} \mathbf{N}$, i.e. a set of vectors $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_p\}$ which:

1. are in the kernel: $K \subseteq \operatorname{Ker} \mathbf{N}$
2. are linearly independent: each $\mathbf{k}_j \in K$ can not be written as a linear combination $\sum_{i \neq j} \alpha_i \mathbf{k}_i$ of the other vectors
3. span the kernel: $\mathbf{k} \in \operatorname{Ker} \mathbf{N} \implies \exists \alpha_1, \ldots, \alpha_p.\ \mathbf{k} = \sum_i \alpha_i \mathbf{k}_i$

We already saw that the fundamental cycles are solutions for $\mathbf{Nf} = \mathbf{0}$; now we prove that they form a basis.

Theorem 3. *Let G be a bipartite graph with $2n \times m$ incidence matrix $\mathbf{N}$, and an arbitrary spanning forest S of G, consisting of the edges $\{e_1, \ldots, e_s\}$ and giving rise to fundamental cycles $F = \{\mathbf{f}_{s+1}, \ldots, \mathbf{f}_m\}$. Then F is a basis of* $\operatorname{Ker} \mathbf{N}$.

Proof. We prove each of the requirements for a basis in turn.

1. As we argued at the definition of fundamental cycles, $\mathbf{Nf}_q = \mathbf{0}$ for each $\mathbf{f}_q$.
2. Vector $\mathbf{f}_q$ is the only fundamental cycle with e_q in it; $\mathbf{f}_q(q) = 1$, while all the other $\mathbf{f}_i$ have $\mathbf{f}_i(q) = 0$. Hence, $\mathbf{f}_q$ cannot be written as a linear combination of the other $\mathbf{f}_i$.
3. Take an arbitrary $\mathbf{g} \in \operatorname{Ker} \mathbf{N}$. Again, we define

$$\mathbf{f} \stackrel{\text{def}}{=} \mathbf{g} - \sum_{s<q\leq m} \mathbf{g}(q)\mathbf{f}_q,$$

   with $\mathbf{f}(q) = 0$ for $s < q \leq m$. We write $\mathbf{f} = (\mathbf{f}_S; \mathbf{0})$. Like above, we argue that $\mathbf{Nf} = \mathbf{0}$; but then also $\mathbf{N}_S \mathbf{f}_S = \mathbf{0}$, and because S is a forest, it has a unique solution $\mathbf{f}_S = \mathbf{0}$; so $\mathbf{f} = \mathbf{0}$. Therefore, $\mathbf{g}$ is a linear combination of the fundamental cycles:

$$\mathbf{g} = \sum_{s<q\leq m} \mathbf{g}(q)\mathbf{f}_q$$

□

We can now give a succint characterisation of all solutions of our problem statement (VECTOR-FLOW). Given a particular solution $\mathbf{p}$ and fundamental cycles $\mathbf{f}_q$ found using Algorithm 1, we construct a $m \times (m - s)$ matrix $\mathbf{B}$ containing the basis:

$$\mathbf{B} \stackrel{\text{def}}{=} [\mathbf{f}_{s+1}, \ldots, \mathbf{f}_m]$$

Now, the solutions $\mathbf{f}$ of $\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}$ consist of

$$\{\mathbf{p} + \mathbf{B}\alpha | \alpha \in \mathbb{R}^{m-s}\}$$

This concludes our analysis of the problem defined in section 3.1. However, the original transition frequency problem has some extra constraints, which we discuss in the next section.

3.4 Inequalities

Until now, we have disregarded the fact that a meaningful solution to our original problem can only consist of positive frequencies. So, in addition to satisfying (IN-OUT-FLOW), the flow $f^{\rightarrow}$ should have $f^{\rightarrow}(u, v) \geq 0$ for every edge (u, v). This requirement can drastically reduce the space of possible solutions, which we can put to good use.

We write the m inequalities as follows ($\mathbf{e}_i$ is a vector of length m with $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0$ for $j \neq i$)):

$$\mathbf{e}_i(\mathbf{p} + \mathbf{B}\alpha) \geq 0 \quad \text{for all } 1 \leq i \leq m$$

This can be rewritten to

$$-\mathbf{B}_{i\star}\alpha \leq \mathbf{p}(i)$$

in which $\mathbf{B}_{i\star}$ denotes the ith row of matrix $\mathbf{B}$. This row contains the coefficients (either 0, 1 or -1) of all fundamental cycles on edge e_i. For example, the top-left edge in Fig. 6 yields the inequality $-(1, -1) \cdot (\alpha_1, \alpha_2)^{\mathsf{T}} \leq 3$.

4 Experiment

As we described in section 3.4, the solution space can be quite small due to the requirement that every frequency should be positive. In this section, we describe an experiment to test whether the space is indeed so small that an arbitrary solution from this space will be close enough to the goal values.

4.1 Picking a solution

Given a certain inter-cluster transition graph, we carry out the following experiment: we generate a certain flow $\mathbf{f}_0$ on the graph, calculate $\mathbf{f}^{\pm}$, and solve the system $\{\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}, \mathbf{f} \geq \mathbf{0}\}$ using the method described above. From the solution space constrained by the inequalities, instead of picking a vector at random, we pick the α' that produces the $\mathbf{f}$ that is closest to a certain *optimal vector* $\mathbf{b}$:

$$\alpha' = \arg\min_{\alpha} |\mathbf{p} + \mathbf{B}\alpha - \mathbf{b}|$$
$$\mathbf{f} = \mathbf{p} + \mathbf{B}\alpha'$$

We define $\mathbf{b}$ as the following vector: for each i, we divide both the $f^{\pm}(v_i^+)$ and the $f^{\pm}(v_i^-)$ sums evenly over the incident edges; of the two values that each edge is assigned in this way, we take the average. However, in this process we ignore every edge e_j that is not on any cycle, because it takes the same

value $f(e_j) = \mathbf{p}(j)$ in every solution. So, before dividing the $f^{\pm}$ sums, we first subtract these fixed values from them.

The reason that we pick the vector closest to $\mathbf{b}$, rather than a random vector, is that we actually found this to be easier. Picking a random vector is not as easy as it may seem, because we have to find one that satisfies all the inequalities. We could do this by using a linear optimization algorithm (optimizing an arbitrary linear function of the vector), but for some reason MATLAB's linprog algorithms often take a long time and do not find an answer at all.

The distance optimization is a quadratic problem, for which we use MATLAB's library function quadprog, which does find a solution where the linear algorithms fail.

4.2 Results

We have performed the experiment on multiple inter-cluster transition graphs. As the global state space, we have always taken a 70×70 hexagonal granule grid, on which we generate a random *circulation* (a flow in which, for each vertex, the outflow equals the inflow). We then divide the granules into hexagonal clusters of the same *radius* (the number of steps from the cluster's center to its edge); see Fig. 2 for an example of a grid with clusters of radius 2. We have used 10 test runs in which we generate a new random circulation, and within each test run we let the cluster radius vary from 5 to 15.

After having picked $\mathbf{f}$ in the way described above, we evaluated the difference between $\mathbf{f}$ and $\mathbf{f}_0$. We used the following measure:

$$\frac{\text{median}\{\text{abs}(\mathbf{f}(i) - \mathbf{f}_0(i)) | e_i \text{ on a cycle}\}}{\text{avg } \mathbf{f}_0}$$

In other words, we take the median error of all edges that are on a cycle; to compare the different experiments, we normalize this error by dividing it by the average edge value. The results are shown in Fig. 7. As expected, the solution becomes more accurate the larger the clusters. This can be explained by the fact that there are more inequalities (edges) per fundamental cycle. The figure also shows that the variance in accuracy (among test runs) becomes larger; this is because the total number of edges is becoming smaller, so the influence of single random $\mathbf{f}_0$ values becomes bigger.

In Fig. 8, we show the running time of the quadprog function per cycle edge. Although it seems that larger cluster sizes imply shorter running times, recall that we have kept the total granule area equal, so that with larger clusters, the number of clusters is smaller. Most probably, the running time is exponential in the number of clusters. As an indication, the largest problem (cluster radius 5) has 3418 cycle edges (inequalities), 295 fundamental cycles (dimensions), and a running time of 50–250 seconds.

5 Future work

For our technique to scale to large numbers of clusters, using MATLAB's quadprog algorithm is not an option because it is too slow. We need to find an algorithm that does not scale exponentially. It probably does not have to find an

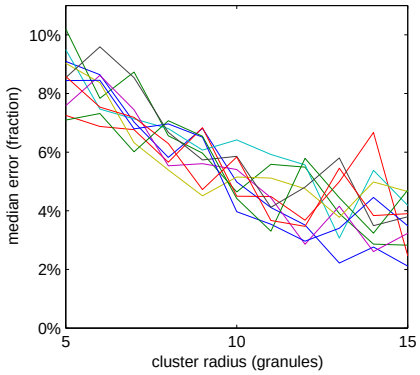


Figure 7: Median error on cycle edges as a fraction of the average edge value, for varying cluster sizes. Shown are 10 test runs.

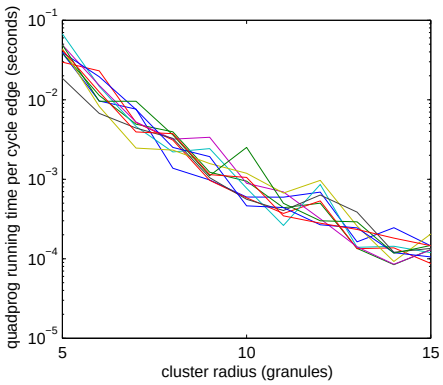


Figure 8: Running time of quadprog function, per cycle edge, for varying cluster sizes. Shown are 10 test runs.

optimal solution like quadprog; *any* solution could be good enough. If even that is out of reach, we could settle for an approximate solution.

A related research question is whether it is possible to solve the combination of cluster frequencies *incrementally* per cluster. The idea is to start with one cluster and a large away state. Then, we position a new cluster in this away state (still leaving room for more) and solve the problem. We continue like this for the other clusters. In our localization example, this would correspond to a situation in which we gradually add sensor clusters, thereby enlarging the observed area.

Finally, we would like to have a sound method to deal with inconsistencies between the several local frequencies. In this article, we have assumed that the local frequencies are consistent with each other—either because they result from the exact same global trace, or because the different traces are all so long that the frequencies have converged to the transition probabilities. In both cases, the result is that $\sum_{v \in V_C^+} f^{\pm}(v) = \sum_{v \in V_C^-} f^{\pm}(v)$ for each component, and the system has an exact solution. In practice, we do not expect this to be the case, and it would be useful to be able to give the best estimate for the solution (maybe using maximum likelihood or Bayesian techniques).

References

[1] S. Evers, M. M. Fokkinga, and P. M. G. Apers. Composable Markov Building Blocks. In H. Prade and V. S. Subrahmanian, editors, *Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM 2007), Washington DC, USA*, volume 4772 of *LNCS*, pages 131–142, Berlin, October 2007. Springer Verlag.