A Metropolis Sampler for Polygonal Image Reconstruction

Peter Clifford and Geoff Nicholls

Department of Statistics
Oxford University
Oxford OX1 3TG, UK
clifford@stats.ox.ac.uk

22/12/1994

Abstract

We show how a stochastic model of polygonal objects can provide a Bayesian framework for the interpretation of colouring data in the plane. We describe a particular model and give a Markov Chain Monte Carlo (MCMC) algorithm for simulating the posterior distribution of the polygonal pattern.

Two important observations arise from our implementation of the algorithm. First, it is computationally feasible to use MCMC to simulate the posterior distribution of a polygonal process for moderately large problems (*ie*, 10000 data points, with polygonal patterns involving around 120 edges). Our implementation, which we would describe as careful, but unsophisticated, produces satisfactory approximations to the mode of the posterior in about 5 minutes on a SUN Sparc 2. Independent samples from the posterior take a few seconds to generate.

The second observation is that the Arak process, a particular type of polygonal process, makes a wonderful debugging tool for testing shape simulation software. This is no mundane observation. Given the usually very complicated nature of such software it is essential to demonstrate that samples genuinely do come from whichever shape distribution is proposed. The Arak process has the advantage that many of its properties can be expressed in precise analytic terms. Computer programmes designed to sample mosaics formed from polygonal shapes can thereby be tested on this process by comparing computer output with exact theoretical expectations.

Contents

1 Introduction

Contextual knowledge can be applied to the analysis of data most simply if the context can be specified in terms of identifiable features of a mathematical model. For example, when we look at the idealised image, Figure 1, of a *block-world* scene made up of rectangular prisms, the straight line graph of edges and edge junctions that we observe satisfies certain topological and geometrical constraints. These constraints impose structure on noisy pictures of the block world, since the set of plausible globally consistent interpretations is small.

When we describe patterns in terms of edges, vertices and coloured regions, the number of elements is unlikely to be predetermined. Thus the state space of a random polygonal configuration is a union of spaces of different dimension. It is this fact, along with the hard geometrical and topological constraints on vertex type, that is the source of most of the problems.

The first step in constructing a probability distribution on the state space is to specify a base measure. The next step is to suggest a density with respect to the measure and then to confirm that it is normalisable. This has been demonstrated for certain polygonal models [2], a remarkable feat of mathematics, when we consider the complexity of the state space. One would like to build a sampler for polygonal random fields. Again, this has been accomplished in special cases, by using the so called *dynamic description*, though the sampler is not able to sample the kind of the posterior distribution which arises when data are incorporated.

In Section 3 we construct a Metropolis sampler for a general class of polygonal processes which includes the posterior processes which arise in Bayesian analysis. When we use MCMC it is necessary to check that samples are distributed as intended. Inconsistencies can arise either because the approach to equilibrium is slow or because there are errors in the computer code. Convergence may be checked by taking long runs, with distinct and carefully chosen starting configurations, looking for significant temporal inhomogeneities in key statistics. Such checks, or convergence diagnostics as they are sometimes called, only provide tests of self-consistency, they do not compare the stationary measure with the intended measure. Clearly, it would be more satisfactory to compare estimated properties of the simulations with theoretical values. However, in models of two-dimensional shape, analytically calculable expectations are not in general available. Of course, this is part of the reason we are using MCMC in the first place. It is because the probability measure is hard to normalise in closed form.

The same problems arise in simulating the grand canonical ensemble in physical chemistry [3, 4]. The Metropolis algorithm must deal with changes in the dimension of the sample space, using moves involving the destruction of molecules and the creation of new ones inserted with the correct energy configuration. Unfortunately, it is not easy to extract useful simulation guidelines from the chemistry literature since the lack of realistic and analytically tractable test sytems has made it difficult to provide convincing demonstrations of the effectiveness of particular implementations. This, coupled, in many cases, with inadequate descriptions of the computer algorithms which have been used, renders the pioneering work of physical chemists on complex Metropolis samplers of limited use in other statistical areas.

Surprisingly, the expected values of several summary statistics can be calculated exactly for consistent polygonal models and hence for the Arak process. Indeed, the normalising constant or partition function is given explicitly. Furthermore, there is a very fast method of simulating these processes using the dynamic description, which enables the expected values of more complex statistics to be estimated efficiently. No algorithm of comparable speed is known for simulating the posterior distribution of a polygonal process. It seems inevitable that any algorithm will be cumbersome and that implementations will be vulnerable to errors in computer code. The Arak process provides an ideal test system for debugging these programmes. A great deal of insight into the performance and correctness of an implementation can be gained by using the same computer code to simulate samples from the Arak process where one can compare estimate values obtained by simulation with exact analytic results. We explain how we checked our work in Section 4.2.

Consistent polygonal models have several other special features. First of all, they have a Markov property in the continuum. This is handy for Monte Carlo simulation. In the Metropolis algorithm we have repeatedly to evaluate ratios of probability densities for consecutive states in the chain. The Markov property ensures that these ratios are only affected by contributions from those parts of the patterns that differ. Our algorithms are thereby, at least potentially, parallelisable. Secondly, the polygonal fields are consistent. This means that edge effects are avoided in the specification of the prior distribution. The more traditional pixel-based Markov random fields which have been used in image processing suffer from an irritating boundary influence, particularly when the clique functions which appear in the Gibbs potential involve large neighbourhoods. In practice, the effect can be minimised by introducing a band of artificial 'guard-pixels' around the image. Consistent polygonal fields, while allowing long range linear structure to be described, essentially have a boundary effect of zero width since the Markov dependence is summarised by the values of the process on an arbitrarily thin boundary set. Finally, there are natural isotropic versions of these models so that artificial directional effects introduced by pixel grids can be eliminated completley in the specification of colouring patterns.

1.1 Background

For a comprehensive introduction to the statistical analysis of spatial patterns, see Grenander [5]. For a useful reference to the measure theory we need, see [6]. There have been several recent papers [7, 8, 9], in which stochastic models of shapes and shape patterns have been constructed, and analysed by MCMC methods. In [7] complex spatial processes are simulated by a jump-diffusions. In [8] planar patterns of identical objects are simulated by a spatial birth and death process [10, 11]. To use MCMC methods based on jump diffusions and spatial birth and death processes, it is necessary to normalise the transition density over the set of destination states at any given time. If this can be done efficiently it may well be an effective approach. A disadvantage of this approach is that new applications need case by case treatment.

The Metropolis-Hastings MCMC algorithm is generally straightforward to apply when a probability density is specified over a complex state space. The algorithm leads to simple and efficient implementations, which have been shown to coverge more rapidly than spatial birth and death samplers for some simple trial problems [12, 13]. Our longterm aim is to develop MCMC algorithms for more elaborate shape models than those described in this paper. We expect to be able to do this, using the prescription we outline below, in a fairly programmatic way, checking our software by reference to the family of polygonal models. The objective is to produce relatively simple, easily verified software for shape simulation, even when the models get complex.

Storvik [9] has described a Bayesian approach to contour modeling. He has shown [14] how to enlarge the usual class of Metropolis-Hastings MCMC algorithms to include a certain class of coupled non-reversible processes, which have tidy implementations. We would like to apply these processes to our simulation problem, as it is likely that there would be a gain in efficiency. However, the models he describes are lattice based, and there is just one polygon in each realisation. Also, although the dimension of the random state varies through the MCMC, the measure is purely atomic. These are just the restrictions we set out to remove.

2 The Model

2.1 The Arak Process

The Arak process [2] was the first of the now fully enumerated consistent polygonal models [1] to be discovered. A typical realisation is shown in Figure 2. It is a model of polygons scattered in the plane. They are non-overlapping, though they need not be simply connected. No vertices coincide. Polygons are coloured uniformly black or white. The process produces a colouring χ of the plane, with an associated graph γ consisting of the polygon boundaries, ie, the discontinuity set of χ.

The process is spatially homogeneous and isotropic, with realisations made up of regions bounded by straight line edges. Although the process is defined on the whole of $\Re^2$ it will typically be observed only through a window $\mathcal{D}$, which we will assume to be an open convex subset of $\Re^2$. The edges of γ meet at V vertices only, except at the boundary $\partial\mathcal{D}$, where edges terminate at I vertices. Any graph fitting this description is called legal for $\mathcal{D}$. The set of such graphs is denoted by $\Gamma_\mathcal{D}$ and the set of polygonal colourings is denoted by $C_\mathcal{D}$. We will sketch how a probability measure can be constructed on each $C_\mathcal{D}$ in such a way that the measures form a consistent family.

We build our base measure from the product of two Poisson point processes of unit intensity. One process acts on $\mathcal{D}$, the other on the boundary $\partial\mathcal{D}$. For notational convenience we omit the exponential normalising term which appears in the Poisson density. Let dv be the element of length when v is a point on the boundary; if v is interior dv is the element of area. Then the set of points $\mathbf{v} = \{v_1, v_2 \dots v_n\}$ in $\mathcal{D}$ has elementary Poisson measure $d\mathbf{v} = dv_1 dv_2 \dots dv_n$. The elementary base measure for the set of legal graphs is then given simply by

$$\nu(d\gamma) = d\mathbf{v}(\gamma), \tag{1}$$

where $\mathbf{v}(\gamma)$ is the set of vertices of γ. In other words a simple counting measure is put on the set of the legal graphs which can be constructed using a given set of vertices. Similarly, the base measure for $C_\mathcal{D}$ is the product of ν and counting measure on the (two) possible colourings of $\mathcal{D}$ once the discontinuity set is given. Since the base measure for the colourings is a trivial modification of that of the graphs, we will concentrate on building a measure on the graphs which has desirable properties: it will be spatially Markovian and consistent.

Let $V_i(\gamma)$ be the vertex points of γ in $\mathcal{D}$, let $V_b(\gamma)$ be the vertex points on the boundary $\partial\mathcal{D}$. let $V(\gamma) = V_i(\gamma) \cup V_b(\gamma)$ and let $E(\gamma)$ be the set of edges. We denote the length of an edge e by $|e|$. Let ϕ_v be the smaller angle between adjacent edges if $v \in V_i$ and the smaller angle between the edge and the tangent to ∂D if $v \in V_b$. We are now in a position to give the fundamental measure for graphs associated with consistent polygonal fields. We define λ on $\Gamma_{\mathcal{D}}$ by

$$\lambda(d\gamma) = p^{n_i + n_b/2} \prod_{e \in E(\gamma)} |e|^{-1} \prod_{v \in V(\gamma)} \sin(\phi_v)\nu(d\gamma), \tag{2}$$

where p is a free parameter greater than zero and n_i and n_b are the number of vertices in $\mathcal{D}$ and $\partial\mathcal{D}$ respectively.

Before defining the full prior probability measure, we will stop to make sense of the fundamental measure in Equation (2). It is possible to formulate polygonal models in a number of different ways. The approach used here is a point-based formulation. An earlier formulation of the same family of distributions was line based. In that case, rather than starting with a Poisson point process, the Poisson line process is used. One colours in the convex polygons outlined by line segments, using a certain set of rules which reproduce $\Gamma_{\mathcal{D}}$. If one looks at a graph created in this way, sharp and flat angles are uncommon compared with approximate right angles; at least, they are uncommon when compared with a graph chosen at random from the legal graphs on the points of a Poisson point process in $\mathcal{D}$. The factor of $\sin(\phi_v)$ for each vertex in γ reproduces in the point-based measure the distributional properties of the line-based measure. The factors of $1/|e|$ have a similar origin. One cannot cancel these terms by a choice of density without leaving the safe and consistent world of polygonal models. All the useful analytical results described in the next section depend on the detailed form of the measure.

The final step in constructing a prior distribution on $\Gamma_{\mathcal{D}}$ and hence on the space of polygonal colourings of $\mathcal{D}$ is to introduce a density on the fundamental measure. A flexible class of prior measures Q on $C_{\mathcal{D}}$ is then given by:

$$Q(d\chi) \propto e^{-F(\chi)}\lambda(d\gamma), \tag{3}$$

where γ is the graph associated with the colouring χ and $F(\chi)$ is an arbitrary additive function of χ called the Gibbs potential. The potential can be used, for example, to promote convexity in polygons of a particular colour or to discourage lengthy boundary edges. A function $F(\chi : \mathcal{D})$ of the colouring on $\mathcal{D}$ is additive if, for any two disjoint subsets S and U of $\mathcal{D}$, $F(\chi : S) + F(\chi : U) = F(\chi : S \cup U)$. Additivity is needed to preserve the spatial Markov property in the following sense. Pick a convex subset S of $\mathcal{D}$. Given local information about the colouring χ in the neighbourhood of ∂S, the distribution of the colouring χ observed within S is independent of the colouring of $\mathcal{D}$ outside S. This is a continuum version of the Markov property on graphs and lattices.

To obtain the Arak process we use a particular form for $F(\chi)$. The Gibbs potential of the Arak prior is simply

$$F_{\mathrm{Arak}}(\chi) \equiv 2p \sum_{e \in E(\gamma)} |e|. \tag{4}$$

The summation is the total edge length of γ in $\mathcal{D}$, and is clearly an additive function. A sample from the Arak prior with $p = 1$ is given in Figure 2.

2.2 Properties of the Arak Process

We have closed form expressions for a number of non-trivial expected values for the Arak process. First of all, we have the following interpretation [2] of the parameter p appearing in Equations (2) and (4). Any line l in $\Re^2$ intersects some edges of γ. The intersection points form a marked point process on l. This process is equivalent to a two-dimensional Poisson process on $l \times (0, \pi)$, with intensity element $p\sin(\phi)$, where ϕ is the angle between l and an edge of γ intersecting l. Thus we should observe an average of $2p$ points on a line segment of length one, with the same variance.

More fundamentally, the process is spatially homogeneous, consistent, and exists on the whole of $\Re^2$. If we were to simulate the process on a larger set $\mathcal{D}^* \supset \mathcal{D}$ but continue to take measurements just from the piece of the process in $\mathcal{D}$, we should observe the same statistical properties as we would in simulating in $\mathcal{D}$ alone. The transparency of the border is achieved by the particular form of the base measure. If $\mathcal{D}$ happens to be a square window, its sides are perfectly representative edges in $\Re^2$, and so the point process of boundary vertices and angles on each side is Poisson with intensity $p\sin(\phi)$. Moreover, multiplying p by a positive real r is the same as scaling the window side by r. It follows that if we double p we ought to see on average four times the number of vertices in the interior. We can do better than this though.

It can also be shown that the expected number of vertices has a density $4\pi p^2$ with respect to Lebesgue measure in $\mathcal{D}$, so the expected number of vertices in a window of area one is

$$< |V_i(\gamma)| >= 4\pi p^2. \tag{5}$$

The number of edges doesn't scale in this simple quadratic way, because they cross the boundary, and we have to avoid double counting when we consider the process in the enlarged window. The correct formula for the expected number of edges in a square window of area one is

$$< |E_i(\gamma)| >= 4p + 4\pi p^2, \tag{6}$$

and if the window side changes by a factor r then $p \to rp$ in Equations (5) and (6). These two results are new. Taken together, the properties described in this section are just what we need to test software.

2.3 Bayesian analysis

We wish to draw inferences about the colouring process χ on $\mathcal{D}$ from observations y_i taken at points t_i for $i = 1, \ldots n$. The posterior probability measure $Q(d\chi|y)$ for χ given y is

$$Q(d\chi|y) \propto g(y|\chi)Q(d\chi), \tag{7}$$

where $g(y|\chi)$ is the density of $y = \{y_1, \ldots, y_n\}$ given χ. In our experiments the y_i are normally distributed, with standard deviation σ, about $\chi(t_i)$ a numerical value associated with the colouring at t_i. The posterior Gibbs potential $F(\chi|y)$ becomes

$$F(\chi|y) = F_{\text{Arak}}(\chi) + \sum_{i=1}^{n}(\chi(t_i) - y_i)^2/2\sigma^2. \tag{8}$$

This is a very simple observation model. A more useful model of point data with binary marks might be as follows. We observe the colours y exactly, but the recorded positions t vary about unknown true values $\theta = \{\theta_1, \theta_2 \ldots \theta_n\}$ according to some probability density $\rho(t|\theta)$. This is a situation which is common in archaeological data where artefacts can migrate from their original sites as a result of ploughing and other agricultural activity. The posterior distribution of χ in this case has a complicated form. However, we can use a trick characteristic of Bayesian MCMC. Anticipating the next few sections, we remark that the joint posterior distribution of χ and θ may be simulated via a Markov chain with a stationary measure equal to the required distribution. Both χ and θ are updated, at different steps, so that the margin of the stationary measure is the full posterior in χ. We thereby simultaneously recover the colouring χ and the observation positions θ.

3 The Sampling Algorithm

3.1 Markov Chain Monte Carlo for the Arak Process

As we have already indicated, our objective is to construct a Metropolis sampler for polygonal random fields, in particular, for the inhomogeneous fields which arise in Bayesian analyses. We do this by constructing a Markov chain with state space $C_{\mathcal{D}}$, and a unique stationary measure (USM) equal to the probability measure of interest. Note that since we are concentrating on two-colour processes, it is only necessary to keep track of the graphs associated with the colourings. Starting from an arbitrary configuration, any other polygonal colouring of $\mathcal{D}$ can be obtained by successively modifying the graph of the discontinuity set. We will take advantage of this fact and build a discrete time Markov chain $\{X_n, n = 1, 2, \ldots\}$ on the set of polygonal graphs $\Gamma_{\mathcal{D}}$ rather than the colourings. We suppose that the USM of interest is given by $Q(d\gamma)$. Note that this will depend on colouring information too, but we will not indicate this explicitly to keep the notation simple. The Markov chain is specified by a transition measure

$$\Pr\{\gamma \to d\gamma'\} \equiv \Pr\{X_{n+1} \in d\gamma' | X_n = \gamma\}$$

on $\Gamma_{\mathcal{D}}$. When the state space is discrete and finite, one proves the existence of a USM by a chain of reasoning leading from irreducibility through recurrence to ergodicity. For a state space formed from the union of a countable number of continuous subspaces, we must show that $\{X_n\}$ is Harris recurrent. From a theorem of Harris [15] we know that in this case a USM exists. If so, it remains to show that this USM is just $Q(d\gamma)$. The usual strategy is to set up a transition measure satisfying detailed balance with respect to $Q(d\gamma)$ in $\Gamma_{\mathcal{D}}$. This is a sufficient condition for Q to be a stationary measure of $\{X_n\}$, and by recurrence it is unique. However, sufficiency restricts us to considering reversible Markov processes.

We use the Metropolis-Hastings prescription [16, 17] to get detailed balance. When we apply this prescription to a Markov Chain which jumps between spaces of different dimension, that is, when the state of the Markov process has randomly variable dimension, the equations of detailed balance involve the ratios of probability measures on spaces of unequal dimension. Provided there is *reciprocity* and *compatibility* (see Section 3.2) the ratios are well defined and detailed balance holds.

In Section 3.3 we define a transition probability measure, in terms of a set of moves types, in such a way that detailed balance is transparent. Finally, in Section 3.4 we show that the Markov process we have defined is Harris recurrent on $\Gamma_{\mathcal{D}}$.

3.2 Detailed Balance

We want to build a transition probability measure satisfying the detailed balance equations

$$\Pr\{\gamma \to d\gamma'\} \times Q(d\gamma) = \Pr\{\gamma' \to d\gamma\} \times Q(d\gamma'), \tag{9}$$

since Q is then a stationary measure. We use the Metropolis-Hastings algorithm.

Suppose the chain is in state γ at time n. The state at time $n+1$ is determined by combining a proposal step with an acceptance step. We make a small change, or *update*, to γ, generating a candidate state γ'. The update is chosen from a small set of possible updates according to some easily sampled probability measure $q(\gamma \to d\gamma')$. This is the proposal step. In the acceptance step we let the state at time $n+1$ be γ' with probability $\alpha(\gamma \to \gamma')$, otherwise it remains unchanged at γ. If γ' is not in $\Gamma_{\mathcal{D}}$ then $\alpha(\gamma \to \gamma') \equiv 0$.

Following Metropolis *et al* [16], Hastings [17] showed how to define a large class of transition probability measures satisfying Equation (9). If we set

$$\begin{aligned} \Pr\{\gamma \to d\gamma'\} &= q(\gamma \to d\gamma') \times \alpha(\gamma \to \gamma') & (10) \\ \alpha(\gamma \to \gamma') &\equiv \min\left(1, \quad \frac{Q(d\gamma')}{Q(d\gamma)} \times \frac{q(\gamma' \to d\gamma)}{q(\gamma \to d\gamma')}\right) & (11) \end{aligned}$$

then $\Pr\{\gamma \to d\gamma'\}$ satisfies Equation (9) subject to certain conditions on $q(\gamma \to d\gamma')$. Because it is still unusual to define a discrete time Markov chain with a state of randomly variable dimension we spell these conditions out. Following work [18] by the physical chemists, [12, 19] have shown how detailed balance can hold in problems of this kind. We are obliged to consider a relatively complex family of updates, and so we gather the conditions on $q(\gamma \to d\gamma')$ into a pair of easily demonstrated conditions, necessary and sufficient for detailed balance.

Assertion 3.1 *The prescription in Equations (10) and (11) will give detailed balance provided there is:*

1. Reciprocity, *ie,* $q(\gamma' \to d\gamma) > 0 \Leftrightarrow q(\gamma \to d\gamma') > 0$, *and*
2. Compatibility, *ie, the following ratio of elementary measures*

$$r(\gamma \to \gamma') \equiv \frac{q(\gamma' \to d\gamma)}{q(\gamma \to d\gamma')} \times \frac{\nu(d\gamma')}{\nu(d\gamma)} \tag{12}$$

is a positive function of γ *and* γ'.

3.3 Choosing the Update-Generating Probability Measure

In this section we define $q(\gamma \to d\gamma')$. This defines the Markov chain, since the transition probability measure $\Pr\{\gamma \to d\gamma'\}$ is given by Equation (10). There are many ways to do this. Our main concerns were as follows. First, the generation step should respect the innate length scale $1/p$ of the process. Second, Harris recurrence should be evident, and, beyond that, we would like to be able to move from one pattern to another without disturbing parts of the pattern which already agree with the target pattern. These features help us get convergence on time intervals of reasonably short length. Other than that, we try to keep things as simple as possible.

3.3.1 Defining Move types

Suppose we are considering a move from γ to γ'. The candidate state γ' is generated from γ by a randomly chosen update. There are several classes of update. The various birth and death updates are distinguished by the number of interior and boundary vertices they create and delete. Updates conserving vertex number are divided into recolourings and vertex moves. The former leave the vertices in their positions, but change the way edges are connected, while the latter move a vertex to a new position, maintaining its edge connections.

We formalise this by grouping the moves into M types. There is a transition probability $q_m(\gamma \to d\gamma')$ associated with each move type, $m = 1, \ldots, M$. We visit the move types either randomly or in some systematic manner. This is completely analogous with the strategies available in pixel-based imaging where the move types correspond to sets of pixels which can be selected for updating. As in the pixel-based case, it is therefore sufficient to show for each $q_m(\gamma \to d\gamma')$ that the conditions of Assertion 3.1 hold.

3.3.2 Defining the Update Generation Probability Measures

We now give the moves and explain how we generate them. These are the moves we have implemented in the software described below. It is worthwhile studying at least one move, as we encounter some instructive problems in specifying them. The moves which create and delete triangular subgraphs from the interior of $\mathcal{D}$ are representative. These two moves are set out in this section. The rest of the moves are in Appendix B.

We begin a move by going down the tree shown in Figure 3. At each fork there is a probability ζ, controlling the direction of descent at that fork, which we are free to choose. We usually fix these probabilities at the start of the simulation. A root of the tree may correspond to an update type, or a group of types. If we arrive at a type, then we generate a candidate state using a move of that type. The interior death operations form a group of types. If we land in this group we select a vertex at random from the interior, and decide which update type to take by looking at the graph around that vertex. We have to choose a death action appropriate for the local topology. By this strategy we avoid heavy rejection from events where, for example, we choose triangle death, but a vertex belonging to a larger polygon. The alternative, keeping a list of the objects we are able to update, seems to be an unnecessary overhead. The probability to arrive at the root for move m is η_m, which is a simple function of the ζ's. We need the η's at the acceptance step, to calculate α.

We refer to a square observation window $\mathcal{D}$. $E_b(\gamma)$ is the set of boundary edges of γ on $\partial\mathcal{D}$. A boundary edge is a line segment on $\partial\mathcal{D}$ between two boundary vertices. Boundary edges which meet at a corner (as for example edges e and e' in Figure 15C) are counted as distinct boundary edges (though corners are not counted as boundary vertices). A *casting box* is a square box of area a, oriented with sides parallel to the sides of the observation window and centred at some specified point. Positions in $\mathcal{D}$ are given in terms of coordinate axes labeled y and z. A is the area of $\mathcal{D}$. For each move m, $\eta_m > 0$ is the probability that at a given MC transition we choose to generate the candidate state using a move of that type. We check detailed balance for each move.

Interior Triangle Birth and Death

> The problem here is to create appropriately sized triangles for the process being simulated. For example, one might cast a small box at random into $\mathcal{D}$ and then throw three points in it. However this introduces an unwanted infinitesimal quantity: the box position. Moreover the triangle generation measure is not uniform conditional on it being possible to generate the triangle in the first place. This is difficult to balance efficiently in the triangle death operation. We proceed as follows.

Move 1: Birth The move type is chosen with probability η_1. Next, referring to Figure 11A, pick a point $u \subset \mathcal{D}$. Pick two further points, v and w, at random, in a casting box centred at u. The points are joined by edges and we consider adding the triangle $\triangle uvw$ to γ. If v lies within a casting box centred at w, as in Figure 11A itself, the triangle is *short*, otherwise, as in Figure 11B, it is *long*. Our triangle generation process generates any given short triangle six ways; there are just two ways to generate a given long triangle. Let $\delta(uvw)$ equal six if $\triangle uvw$ is short and two if it is long. The generation probability q_1 for the birth transition $\gamma \to d\gamma'$, is

$$q_1(\gamma \to d\gamma') = \eta_1 \times \frac{du dv dw}{Aa^2} \times \delta(uvw)$$

Move 2: Death The move type Interior Death is chosen with probability η_2. Next, pick a vertex at random. If it is in a triangle, try to delete the triangle. We first check that the triangle could have been born by our triangle birth process. To do this we check to see that each vertex has at least one other vertex inside its casting box. If not, as in Figure 11C, the move is rejected. Otherwise the generation probability q_2 for the death transition $\gamma' \to \gamma$, is

$$q_2(\gamma' \to \gamma) = \eta_2 \times 3/|V_i(\gamma')|.$$

Considering a triangle birth move and its reverse death,

$$\begin{aligned} \frac{q(\gamma' \to d\gamma)}{q(\gamma \to \gamma')} &\propto 1/du dv dw, \\ \frac{\nu(d\gamma')}{\nu(d\gamma)} &= du dv dw, \end{aligned}$$

so that the compatibility condition in Equation 12 is satisfied. Any triangle born can be killed. Reversibility has been achieved by rejecting death moves that do not have a corresponding birth.

3.4 Harris Recurrence

Since Q is stationary with respect to the transition probability measure we have constructed, it follows that if a *unique* stationary measure exits for the Markov chain we have defined, then that measure must be Q. In this section we show that some such measure exists. We begin by defining Harris recurrence. Let $\mathcal{B}_{\mathcal{D}}$ be the Borel sigma field of $\mathcal{D}$.

Definition 3.1 *Suppose that $\{X_n\}$ is a Markov process taking values on a measurable space $[\Gamma_{\mathcal{D}}, \mathcal{B}_{\mathcal{D}}]$ then μ is a recurrence measure of $\{X_n\}$ if, for $S \in \mathcal{B}_{\mathcal{D}}$,*

$$\mu\{S\} > 0 \quad \Rightarrow \quad \Pr\{X_n \in S, \text{ for some } n > 0 | X_0 = \gamma\} = 1, \qquad \forall \gamma \in \Gamma_{\mathcal{D}}.$$

If a recurrence measure exists, then $\{X_n\}$ is Harris recurrent.

Assertion 3.2 Harris [15] *Suppose $\{X_n\}$ has a recurrence measure μ. Then $\{X_n\}$ has a unique stationary measure, which is stronger than μ.*

κ stronger than μ means $\mu\{S\} > 0 \Rightarrow \kappa\{S\} > 0$, for $S \in \mathcal{B}_{\mathcal{D}}$.

Assertion 3.3 *Let* $\mathbf{0} \in \mathbf{\Gamma}_{\mathcal{D}}$ *be the empty state, of dimension zero, with no edge or vertices. The indicator function for the empty state,* $\mathrm{I}\{\gamma = \mathbf{0}\}$, *is a recurrence measure for* $\{X_n\}$.

We prove this result in Appendix A. In fact Q itself had better be a recurrence measure of any useful MCMC process we construct, and one can show directly, using the equations of detailed balance, that this is the case. We have chosen as simple a recurrence measure as we can to highlight the fact that it is enough to show recurrence on just one special set in $\mathcal{B}$ (in fact just one state, since $\mathbf{0}$ is an atom of Q). Recurrence incorporates irreducibility because we insist the process hits the sets of positive recurrence measure from *any* starting state.

4 The Implementation

We want an implementation that is easy to understand, efficient and easy to alter. Above all, it must stand a chance of being correct. We have this in mind as we specify the model, and design the sampling algorithm.

4.1 Implementing the Algorithm

We were assisted in writing functions answering basic geometric queries, and in choosing data structures, by reading elementary texts in computational geometry. Computational geometers ask themselves, first, "how will the running time increase as the problem size increases?", and second, "does the algorithm have a satisfactory implementation?". We ignored their advice in at least two respects.

First of all, we felt that the more advanced data structures suggested, which focus on achieving a good worst case performance, were unnecessarily elaborate for a first implementation. We created a relatively simple set of data structures, based on a division of $\mathcal{D}$ into a grid of boxes. We aimed to get good expected time performance for a spatially homogeneous process with structure concentrated around a single length scale. The grid localises the heavily used queries, "does this edge intersect an edge of γ?", and "which data points lie within this region?". We tried to keep the low level, query-answering, section of the code separate from the the high level section, which contains the representation of the current state of the Markov chain, and maintains detailed balance at an update.

Our philosophy in this respect was: allow the low level part of the programme to get complicated but make sure it is reasonably fast; keep the high level part as simple as possible, sacrificing speed for transparency wherever necessary.

The second departure from good practice was giving vertices floating point positions in $\mathcal{D}$. A computational geometer would use integers, as it is possible to produce more reliable code by that device. It is however rather tedious to write such code, as one must handle degenerate states explicitly. Fortunately for us, the prior sets the scale for polygons, so problematic structures, such as *very* short edges, and collinear pairs of edges adjacent at a vertex, appear with negligible probability. However, boundary objects still need special treatment.

Further details are given in Appendix C.

4.2 Testing the Implementation

Even moderately complex programmes usually contain bugs. Code that cannot be tested against external certainties is particularly bug prone.

As we built the data structures described in Appendix C we wrote checking functions for them. These visit each part of the data structure checking for consistency. These functions were gathered together in a grand checking function, which has been left in place [20]. It usually adds a small amount to the run time. Further details of internal checks for MCMC are given in Appendix D. In this section we describe the external checks we applied.

We measure certain state functions at intervals through the run, and record their values. After the run we analyse the output, estimating the mean and variance of each time series, as well as getting standard errors for each estimate. We compare the values we obtain with the analytic results given in Section 2.2. We recommend vigorous testing at very small values of p, because it is at very low densities one stands a chance of detecting small departures from the measure.

We used the output analysis described in [21]. This gives an automatic windowing procedure (the *initial monotone sequence estimator*) for computing standard errors from the autocorrelations of a Markov process. We computed the integrated autocorrelation time (IACT, τ) of each state function, and estimated its standard error, in an approximation due to [22] and suitable if $L >> \tau$, L the window size on the autocorrelation function. To check the consistency of our standard errors we compute the asymptotic standard deviation of each autocorrelation function, and plot the 2σ confidence interval on top of the autocorrelation function. We have also done many runs at lower statistics and noted that the full 3σ range of estimated errors is reached and not exceeded.

We drop 1% of each run to allow for *warm up*, that is, for the chain to reach equilibrium. This is in all cases much longer than the integrated autocorrelation time, as can be seen in Table 1 below. Moreover, we hold off sampling for a time of length roughly ten times the IACT, using a rough estimate of the IACT obtained from short experimental runs. Note that this is not always enough, as the time to enter equilibrium is not the same as the IACT [22]. We were obliged to subsample our MCMC output, using the same rough IACT values to set the sample interval.

Results are given in Tables 2 and 3. Standard errors are given in brackets. 0.023(10) has estimated standard deviation 0.01. We do not detect any significant departure from the Arak measure.

5 Performance

5.1 Criteria

In this section we give some performance statistics for the software. First we set out the criteria. We want the variance, $\mathrm{var}(\hat{f})$, of the estimated expectation of any state function we measure to be as small as possible. $\mathrm{var}(\hat{f})$ falls off with the run length n of the MCMC like $\mathrm{var}(f) \times \tau_f/n$, where $\mathrm{var}(f)$ is the variance of f itself. So, as [22] explains, n/τ_f can be thought of as the number of effectively independent samples in a run of length n. For a given computing machine, $1/\tau$ measures power: at what rate *in seconds* do estimated variances go down ?

However, some functions of the state decorrelate much more slowly than others. These are the physicists' *slow modes* [22]. If one function, $f(\gamma)$ say, is particularly important to us, we look at τ_f. However we like to check on a handful of other fundamental IACT's. We can't tell if the system is stuck permanently in some subspace of $\Gamma_{\mathcal{D}}$, but we might catch it hopping very slowly from one to another. We have measured the IACT of the following functions:

1. the number of edges in the interior;
2. the number of vertices on each boundary edge;
3. the number of intersections between the pattern, γ, and vertical and horizontal interior test segments of length one;
4. the colour at one fixed point in $\mathcal{D}$;
5. the number of polygons.

When one is producing a single final interpretation of the posterior, for example the MAP classification, other performance criteria are relevant. However, the user may prefer to base their next level of interpretation on a set of samples, rather than on a single state. It would be unsafe in general to produce these from a single run, as the set of samples may be biased by some MCMC state trapping. So again the IACT is no use to us as a performance measure on the posterior. We consider a synthetic problem generated by sampling the prior, and then observing it, under artificial noise, at random data points in $\mathcal{D}$. We do simulated annealing to get an estimate of the mode in the posterior. We explain how we anneal, and how we justify annealing, in Appendix E. From the mode we get a classification of the data points black or white. We report the best and worst misclassification rates over 10 runs alongside the average runtime. We present some examples of typical restorations.

5.2 Performance on the Prior

In Table 4 we give the IACT in seconds, against p, the intensity of the Arak process. The slowest mode proved to be the number of edges, so the IACT is $\tau_{|E_i|}$. As a benchmark, the IACT for our MCMC, on an Arak process with an average of around 120 edges, is 0.8(1) CPU seconds on a Sun Sparc 10.

When p is less than about 0.01, $\tau_{\chi(x)}$ gets very big (see Figure 4, $\tau_{\chi(x)}$ is given in units of updates); $\tau_{\chi(x)}$ is the IACT for the colouring at some fixed point x in $\mathcal{D}$. This phenomena occurs because, at small p, the system is almost trapped in states of one colour, either black, or white. This is not a problem, because we could easily introduce global colour inversion moves, and in any case, $p = 0.01$ is not useful outside checking. A similar near-trapping happens in the one dimensional Ising model, when the update scheme is single-site. The linear term in p in Equation (6) (corresponding to Goldstone modes [23] made up of highly corrugated objects) is a sign that there is a way out of the trap.

It is interesting to experiment with different update sets. When we are not experimenting, we use an automatically chosen set of updates and update type selection probabilities. We use Equations (5) and (6) to set these and the area of the casting box. Thus the casting box should be of such a size that the expected number of vertices in it is some small number. However, we have more than enough move types to get Harris recurrence, and so we try and decide which are the key moves. In [12] the authors find, in a study of the Strauss process,

p	n updates$\times 10^6$	IACT(E_i) updates	successful % updates	uncertainty % estimate	CPU time hrs mins secs
0.001	6	76(2)	0.5	5.4	k1m 22s
0.01	8	110(4)	4	1.9	k1m 47s
0.1	12	257(9)	4	0.8	k2m 16s
1.0	40	2800(200)	18	0.3	k18m
2.0	40	15000(2000)	17	0.5	s70m
3.0	40	30000(5000)	18	0.4	k18m
4.0	600	75000(6000)	19	0.1	k4h 47m

Table 1: Run parameters for the MCMC measurements in tables 2 and 3 below. The IACT's given are for the number of edges, since this was always the largest IACT, up to standard errors. Percentage uncertainties are calculated from the standard errors of the same statistic (Table 3). n is the total number of updates in the run. One update is one transition in our discrete time Markov chain . Time measurements prefixed with a k are reported for a Sun Sparc 10, and s indicates a Sun Sparc 2. The former seems to be a factor of about 3.8 faster than the latter.

p	$<|\gamma \cap \hat{e}|>$ theory	Interior $\hat{e}$ measured	Interior $\hat{e}$ variance	Boundary $\hat{e}$ measured	Boundary $\hat{e}$ variance
0.001	0.002	0.00209 (15)	0.00209 (15)	0.00215 (14)	0.00214 (14)
0.01	0.02	0.01949 (42)	0.01960 (47)	0.01933 (37)	0.01925 (37)
0.1	0.2	0.2001 (18)	0.2010 (19)	0.2004 (18)	0.2012 (20)
1.0	2	2.0053 (91)	1.993 (15)	1.9962 (88)	1.985 (16)
2.0	4	4.020 (32)	4.131 (66)	4.008 (27)	3.995 (50)
3.0	6	6.098 (60)	5.83 (12)	5.948 (55)	5.93 (11)
4.0	8	7.975 (29)	8.200 (80)	7.990 (26)	7.896 (69)

Table 2: Comparison of theoretical and estimated expectations on the Arak prior. The expected number of intersections $<|\gamma \cap \hat{e}|>$ between the pattern, γ, and a test edge $\hat{e}$, is Poisson, with mean and variance $2p|\hat{e}|$, [1]. Results for boundary (lower) and interior (horizontal centred) test segments of length one are shown.

p	$< \lvert E_i \rvert >$ theory	measured
0.001	0.00401	0.00428 (23)
0.01	0.04126	0.04111 (78)
0.1	0.5257	0.5255 (40)
1.0	16.569	16.534 (53)
2.0	58.28	58.03 (25)
3.0	125.12	125.00 (50)
4.0	217.10	216.96 (27)

Table 3: Comparison of theoretical and estimated expectations. Theory against estimated values for the expected number of edges in the interior of a square window on the Arak prior, Equation (6).

p	IACT(E_i) seconds
0.1	k0.0029(1)
1.0	k0.075(5)
2.0	s1.58(21)
3.0	k0.8(1)
4.0	k2.2(2)

Table 4: The integrated autocorrelation time for our MCMC on the prior, computed on standard machines, and measured in seconds. Time measurements as for Table 1. IACT's are given for different values of the intensity p. When $p = 4$ there are well over 200 edges in a typical realisation. This table is one of the main results in this paper.

p	# Data Points	$N \times 10^6$	% Misclassified, 10 runs		CPU time per run
			best	worst	mins secs
1	500	0.6	3.8	6	s30s
1	1000	0.6	1.2	2.4	s50s
3	10000	0.6	1.6	2.4	k91s
5	20000	1.2	2.7	3.6	k151s

Table 5: Point classification results from MAP state estimated by annealing. N is the length of the annealing schedule in Markov chain transitions (see Appendix E). Time measurements as for Table 1. Percent misclassification rates are the best and worst of ten runs, each on data observed from the same *true* polygonal pattern.

that birth and death operations are most effective, and simple repositioning of points not as useful as one might expect. In Figure 5 we see if anything similar happens here. We remove all recolour operations, $\zeta_2 = 1$ in Figure 3, and then vary the proportion of vertex shift operations, moves 4, 5 and 6, to Birth-Death operations. We vary ζ_1 over the range 0.1 to 1.0, giving results for $\tau_{|E_i|}$ in units of CPU seconds. It is possible to find other statistics behaving quite differently, but this is typical. In our case vertex shift operations seem to do some good.

5.3 Performance on the Posterior

We analyse the posterior defined by Equations (8) and (4). Noise is additive Gaussian. Variance is given with respect to `black` $= -1$ and `white` $= 1$. We estimate the mode by annealing, as described in Appendix E. The initial state is always $\mathbf{0}$, the empty state.

In Table 5 we give the best and worst point misclassification rates over 10 runs alongside the average runtime in seconds, for a range of values of p. All runs have an observation process variance σ^2 of 0.5. As the variance decreases, and the amount of data goes up, optimization gets harder. However the annealing seems to deal with this well, as misclassification rates decreased with variance. These low variance results are not reported. As a benchmark, for 10000 data points at a noise variance of 0.5, and polygon patterns made up of more than one hundred edges, one expects to get a misclassification rate under 2%, in less than 5 minutes on a Sun Sparc 2.

In Figures (6) and (7) we give typical examples of restored patterns taken from the runs in Table 5. In each case the estimate (solid) is superimposed on the truth (dashed). Region colourings and data have been left out, but misclassified data points are shown.

Some of the problem structures which occur are presented in Figure 10. Spikes are very stable, because the tip is outside deletion range of the base, and is only weakly attracted to it. A move deleting spikes would in general have to create them, to keep reversibility. In the final stage of annealing, we are free to delete them irreversibly, because any energetically favored move is legal at zero temperature, though we have not implemented this. Also, the recolour move, Move 3, is rather inefficient, and takes a long time to join or split polygons in the way data suggests. This means the topology may come out wrong in short annealing schedules. Narrow necks may be joined the wrong way. These are fairly well treated in practice, simply by using longer annealing schedules. However it is these structures which force us to take minutes, rather than seconds, over annealing large patterns. We think they can be treated with new update types, as they arise.

6 Conclusions

We expect to be able to construct MCMC algorithms for a wide range polygon shape models, by following the prescription we outlined in Section 3. We have described how we implemented and checked one such algorithm. Although efficiency is important, simple strategies were sufficient to get the results we have presented. With this software it is possible, in a few minutes, and on readily available hardware, to reconstruct polygonal patterns made up of over one hundred edges, observed at ten thousand points scattered randomly in $\mathcal{D}$.

The problem now is to get as complete as possible an understanding of the relationship between the set of update types and the IACT for any particular statistic of interest.

One has to make sure the system can move easily between states with different values of the statistic. It is usually quite obvious what kind of update is needed to improve communication between any given pair of states. One has then to find some procedure for generating an update of the kind needed, with a reasonably efficient implementation. This is the guts of the practical part of the problem. At the design stage of the implementation one must choose data structures that will support sufficiently powerful updates.

7 Acknowledgements

This research was funded by a grant from the British Science and Engineering Research Council under the Complex Systems Initiative.

A $\{X_n\}$ is Harris Recurrent, Assertion 3.3

We show that it is always possible to delete a vertex. Since the probability for a transition to the empty state is greater than the probability for a path made up of a sequence of deletions, *there exists a finite t such that* $\Pr\{X_n = \mathbf{0}|X_0 = \gamma\} > 0$. This is sufficient to prove that the process visits $\mathbf{0}$ with certainty for some t. It follows from Definition 3.1 that $\mathrm{I}\{\gamma = \mathbf{0}\}$ is a recurrence measure.

Assertion A.1 *From any state there is a set of paths which lead, with positive probability, and in a finite number of steps, to a state with one fewer vertex.*

Proof of Assertion A.1: First we divide the pattern up into polygons. Polygons are defined as the borders of the part of the region coloured black. This removes an ambiguity at the boundary.

(a) There is always a simple polygon. We choose one.

(b) If this polygon has three vertices (or is a corner cut), and is sufficiently small (see Move 2 or Move 10), we can delete the whole triangle, with positive probability. If the triangle is too large, we shrink it, using Move 4. This can be done, and in a finite number of moves, since the triangle is empty, and the set of paths leading to the set of small triangles has positive probability measure.

(c) If the polygon has more than three vertices, it has a triangular decomposition [24]. One of the sub-triangles has two sides which are edges of the polygon [5]. If the vertex v, at which these two edges meet, is deleted, the generated state is in $\Gamma_{\mathcal{D}}$. There is a positive probability to delete it if it lies close enough to the edge e that will be created by its death (see Move 8 and Figure 14). If this is not the case then again, the sub-triangle is empty, so we can move v into deletion range of e using Move 4 repeatedly. The set of paths leading to the set of deletion positions has positive probability.

B The Update Classes Implemented

The Interior Triangle Birth and Death updates are described in Section 3.3.2. The rest are given below. We check detailed balance for each move by checking that the conditions of Assertion 3.1 are satisfied.

Move 3: Recolour The move type is chosen with probability η_3. Next, pick two distinct edges e and e' from the set of interior edges. Join the vertices to form a simple quadrilateral, as in figure Figure 12A. If two simple quadrilaterals can be formed (which will be concave), as in figures Figure 12B and C, choose one with probability one half. Of course, the choice in Figure 12C will lead to an illegal graph γ' and will be rejected at a later stage. If the move is accepted, the old edges of the update quadrilateral are replaced by the new. Then the recolour move generation probability q_3 for the transition $\gamma \to \gamma'$, is

$$q_3(\gamma \to d\gamma') = \eta_3 \times \frac{2\delta(e,e')}{|E_i(\gamma)||E_i(\gamma)-1|}$$

where $\delta(e, e')$ is equal one half if the update quadrilateral is concave and one if it is convex. The move satisfies the conditions of Assertion 3.1, since the reverse move is generated with the same probability, and $\nu(d\gamma')/\nu(d\gamma) = 1$.

Move 4: Move an Interior Vertex The move type is chosen with probability η_4. Next, pick an interior vertex v at random. Pick a point v' at random within a casting box centred at v. Consider moving v to v'; edges move with the vertex as in Figure 13A. The update region is a quadrilateral, which need not be convex, or simple. The generation probability q_4 for the transition $\gamma \to d\gamma'$, is then

$$q_4(\gamma \to d\gamma') = \eta_4 \times 1/|V_i(\gamma)| \times \frac{dv'}{a}.$$

v is in the casting box of v', so the reverse move can be generated, and

$$\begin{aligned} \frac{q(\gamma' \to d\gamma)}{q(\gamma \to d\gamma')} &\propto \frac{dv}{dv'}, \\ \frac{\nu(d\gamma')}{\nu(d\gamma)} &= \frac{dv'}{dv}, \end{aligned}$$

so we have reciprocity and compatibility; the conditions of Assertion 3.1 are satisfied.

Move 5: Move Boundary Vertex along Boundary Edge The move type is chosen with probability η_5. Next, pick a boundary vertex v at random. Referring to Figure 13B, let e and e' be the boundary edges connected to v. Pick a point v' on the boundary at random from $e \cup e'$. Consider moving v to v'; the interior edge, connected to v, moves with the vertex as in Figure 13B. The generation probability q_5 for the transition $\gamma \to d\gamma'$, is

$$q_5(\gamma \to d\gamma') = \eta_5 \times 1/|V_b(\gamma)| \times \frac{dv'}{|e| + |e'|}.$$

v' can be moved by the same process to v, and the conditions of Assertion 3.1 can be shown to be satisfied in the same way as for q_4.

Move 6: Move Boundary Vertex past Corner The move type is chosen with probability η_6. Next, pick a boundary vertex by choosing a corner of the observation window, and then one of the edges e entering the corner, at random. v is the non-corner vertex of e, if it has one. Let e' be the other edge entering the corner, as in Figure 13C. Pick a point v' on e' and consider moving v to v', moving the interior edge connected to v with the vertex. The generation probability q_6 for the transition $\gamma \to d\gamma'$, is

$$q_6(\gamma \to d\gamma') = \eta_6 \times 1/4 \times 1/2 \times \frac{dv'}{|e'|}.$$

The reverse transition is generated by the the same process, and the conditions of Assertion 3.1 are satisfied in the same way as for q_4.

Interior Vertex Birth and Death

Move 7: Birth The move type is chosen with probability η_7. Next, pick an interior edge e at random. It joins vertices u and u' say. Second, pick a point (y, z) from the neighborhood of e. The point is chosen uniformly from a box of side e, with sides oriented parallel and orthogonal to e, and centred on the point half way along e. See Figure 14A. We first choose a point at (y', z') on e and then the point (y, z) at a distance $r_1|e|$ from (y', z') in a direction orthogonal to e, with $r_1 \sim U(-1/2, 1/2)$, and the sign of r_1 determining the side of e. Label the point at (y, z) v, and consider replacing edge e by two edges connecting u and u' to v. The generation probability q_7 for the transition $\gamma \to d\gamma'$, is

$$q_7(\gamma \to d\gamma') = \eta_7 \times 1/|E_i(\gamma)| \times \frac{dv}{|e|^2}.$$

Note that the point selection process has a density with respect to the product of two one-dimensional Lebesgue measures. However dv still corresponds to a product measure since the Jacobian of the rotation into the frame of the coordinate system of v is one.

Move 8: Death The move type Interior Death is chosen with probability η_2. Next, pick an interior vertex v at random, connected via edges e' and e'' to vertices u and u' (if u and u' are on the same side of $\partial\mathcal{D}$, or connected by an edge, one of the other moves in the group is selected). Consider deleting e', v and e'', and creating a new edge e connecting u and u' directly, the reverse of the move shown in Figure 14A. We must first check that the vertex v could have been born in a birth from edge e, *ie*, that v lies within the $|e| \times |e|$ birth box around the edge we will create. If this is not the case, as in Figure 14B, then the move is rejected. Otherwise, the generation probability q_8 for the transition $\gamma' \to \gamma$, is

$$q_8(\gamma' \to \gamma) = \eta_2 \times 1/|V_i(\gamma')|.$$

Considering the vertex birth move acting on vertex v, and its reverse death,

$$\begin{aligned} \frac{q(\gamma' \to d\gamma)}{q(\gamma \to \gamma')} &\propto 1/dv, \\ \frac{\nu(d\gamma')}{\nu(d\gamma)} &= dv, \end{aligned}$$

so we have compatibility. Any vertex born can be killed. Reversibility has been achieved by rejecting death moves that do not have a corresponding birth. The conditions of Assertion 3.1 are satisfied.

Boundary Triangle Birth and Death

Move 9: Birth Pick a boundary edge e at random. Pick two points u and v uniformly on e. Let e' be the proposed new boundary edge between u and v. See Figure 15A. Pick a point w uniformly in a square box of side $|e'|$ positioned in the interior of $\mathcal{D}$ with one side on e'. Consider adding triangle $\triangle uvw$ to γ. The generation probability q_9 for the boundary triangle birth transition $\gamma \to d\gamma'$, is

$$q_9(\gamma \to d\gamma') = \eta_9 \times 1/|E_b(\gamma)| \times \frac{2du\,dv\,dw}{|e|^2|e'|^2}.$$

The factor of 2 comes from the indistinguishability of u and v in the birth process.

Move 10: Death There are two ways into boundary triangle death. The first is as follows. The move type Boundary death is chosen with probability η_{10}. Next, referring again to Figure 15A, chose a boundary edge e'. If it belongs to a boundary triangle consider deleting the triangle from γ. The second entry is via the group Interior Death. This is chosen with probability η_2. We then choose an interior vertex. If it belongs to a boundary triangle we consider the same operation. The reason for the two entries is explained in Section 3.3.2. In both cases, before proceeding with the update we must check the interior vertex of the boundary triangle lies inside a square box of side $|e'|$ positioned in the interior of $\mathcal{D}$ with one side on e'. If not, then the boundary triangle could not have been born by the reverse operation, and the move is rejected. Otherwise, the generation probability q_{10} for the boundary triangle death transition $\gamma' \to \gamma$, is

$$q_{10}(\gamma' \to \gamma) = \eta_{10} \times 1/|E_b(\gamma')| + \eta_2 \times 1/|V_i(\gamma')|$$

where η_{10} denotes the probability a boundary triangle death is attempted from a boundary edge, and η_2 is the probability it is attempted from an interior vertex.

Considering a boundary triangle birth move and its reverse death,

$$\begin{aligned} \frac{q(\gamma' \to d\gamma)}{q(\gamma \to \gamma')} &\propto 1/dudvdw, \\ \frac{\nu(d\gamma')}{\nu(d\gamma)} &= dudvdw, \end{aligned}$$

so we have compatibility. Any boundary triangle born can be killed. Reversibility has been achieved by rejecting death moves that do not have a corresponding birth. The conditions of Assertion 3.1 are satisfied.

Corner Cut Birth and Death

Move 11: Birth The move type is chosen with probability η_{11}. Next, pick a corner. On each of the edges e and e' entering the corner, pick a point uniformly at random. Call these points u and v respectively. Join u and v with an edge, as in Figure 15B, and consider adding u, v and the edge to γ. The generation probability q_{11} for the corner cut birth transition $\gamma \to d\gamma'$, is

$$q_{11}(\gamma \to d\gamma') = \eta_{11} \times 1/4 \times \frac{dudv}{|e||e'|}.$$

Move 12: Death The move type is chosen with probability η_{12}. Next, pick a corner. Referring to Figure 15C, let u and v respectively be the vertices at the opposite end of the edges e and e' entering the corner. If u and v are connected by an edge, consider removing u, v and the edge from γ'. The generation probability q_{12} for the corner cut death transition $\gamma' \to \gamma$, is

$$q_{12}(\gamma' \to \gamma) = \eta_{12} \times 1/4.$$

Considering a corner cut birth move and its reverse death,

$$\begin{aligned} \frac{q(\gamma' \to d\gamma)}{q(\gamma \to \gamma')} &\propto 1/dudv, \\ \frac{\nu(d\gamma')}{\nu(d\gamma)} &= dudv, \end{aligned}$$

so we have compatibility. Any corner cut that can be born can be selected at the death stage, and can therefore be killed. The conditions of Assertion 3.1 are satisfied.

C Further details of the Implementation

The programme is written in C. Memory demand increases with p, but is always slight. Deleted structures are stacked, and reused in the usual way. Memory is allocated only when the stacks are exhausted. In this approach, the demand increases slowly through a run as the dimension of the state reaches new maxima. However for $p = 10$, with over 1000 edges, the code never occupies more than 400 kilobytes, which is peanuts. For $p = 3$, with 10000 data points, our benchmark problem, the maximum size was around 1 megabyte, and most of this is data.

C.1 Data Structures

Edges and vertices are structures. Polygons are doubly linked lists of alternating edges and vertices. Each edge and vertex has a `next` and `prev` pointer field, so that it can fit into a polygon list. Besides being accessible via these lists, edges and vertices can be referenced via arrays. Thus `e[i], i=0,1...num_edges-1` is a pointer to an interior edge structure, and `v[j], j=0,1...num_vertices-1` is a pointer to an interior vertex structure. This allows us to choose at random an element of γ, by sampling from an array. Arrays are allocated dynamically at the start of a run, and their size depends on the density parameter p which controls the number of edges. Their size is fixed through a run, so we have to be generous in allocating them. Equations (5) and (6) are used to set sizes online. These fixed arrays do not occupy much space because they are arrays of pointers to structures, and not arrays of structures. There are other advantages in the pointer array structure. The `v[j], j`$\geq$ `num_vertices` point to deleted vertices, so that this section of the array acts as the vertex *stack*. To delete a vertex, all we have to do to keep the array of live vertices packed is swap a couple of pointers, and decrement `num_vertices`. We do not need to copy the fields of the structures. All vertex and edge types are handled in this way.

Once a move has been generated, we have to check that the new graph state is legal. This simply means checking its edges do not cross each other. We localise this checking using a grid of boxes. For each box in the grid, one linked list records the data points in the box and another records the edges in and crossing the box. The structures in these lists are called *tags*. Tags are links, containing a pointer to the next tag in the chain, and a pointer to the edge or data point or box they actually tag. If an edge e is deleted, we must visit each box e crosses, and delete from the box's list of edge tags the edge tag corresponding to e. However, we do not want to have to search the box's list of edges for e's tag.

To avoid searching lists for specific edge tags, edges maintain a list of box tags. When an edge e is created, we have to visit each box it crosses and check the edge does not cross edges already in place. The edge keeps a tag for that box (in a linked list of such tags). Box tags are double tags. The box tag corresponding to box b points, not only to b itself, but also to the edge tag in b's list of edge tags corresponding to e. When an edge is deleted, we go though its list of box tags and remove the edge tags they point to from the lists those edge tags lie in.

C.2 Initialising and Maintaining Low level Data Structures

Maintaining the low level data structures is an overhead. The main trick is to *build as you go*. Since many candidate updates are rejected, it would be a waste of time to fully install and delete the support structures for these failed edges. We build the linked list of box tags for an edge e one box at a time, as each box is checked for edges crossing e. The edge tags for the new edge are not installed in the boxes it crosses. If the move is accepted, we go back through the list of box tags, visiting each crossed box, only now installing an edge tag marking the new edge intersecting the box.

There is another issue which must be treated if the simulator is to be reasonably efficient. The grid should respect the scale of the process simulated over it. The only important scale in the prior is $1/p$. However the number of data points per unit area is a second length scale, independent of $1/p$, coming in in the posterior. It would be natural to have two systems of grids, with a fineness appropriate for the key scales of the processes they localise. This seemed to be unnecessarily complex. After all, in sensible problems the scale of the data density and the scale of the polygonal interpretation are related. The grid fineness is chosen adaptively at run time, when the two key scales are known.

C.3 Programme Flow

We begin a move by going down the tree shown in Figure 3. Once an update type m is chosen, we use it to generate the candidate state. There is one function for each update type. The chosen update function builds a small subgraph called the *update polygon*. The edges and vertices of this polygon are in γ, γ', or both. A pointer to this update polygon is passed to a central routine which performs the Metropolis update. First of all it checks that γ', as implied by the update polygon, will be a legal graph. If γ' is legal another function makes a list of data points in the update region. Convex update regions are identified and passed into a special section of code, as these can be handled with particular efficiency. Next, the update polygon and the list of data points inside it are passed to a set of functions which evaluate α in Equation (11). The acceptance step is then executed. If accepted, the update polygon is used to install the edges and vertices of γ', and delete those of γ, and the estimated colour at each data point in the update region is switched from black to white or *vis versa*.

We note that there is no notion of *polygon colour* in our algorithm. Data points keep track of their own current estimated colour. If they find themselves in an update region their colour changes to its opposite. If the estimated colouring is consistent with γ at the first state in the MCMC, this is enough for it to remain consistent as γ evolves. As a consequence of our approach, we have no direct way of telling what the colour at any given point in $\mathcal{D}$ actually is. We would have to compute that colour by counting the number of intersections between γ and a Jordan curve connecting the point to one of the data points, where colour values are maintained. However at no stage is colouring information for points other than data points needed.

D Internal Checks for general MCMC

To help ourselves find bugs in the building stage, we wrote several functions which check data structures for internal consistency. These should be left in place [20] as they usually add a negligible amount to the run time. Thus we check that the polygons are consistent linked lists, that each vertex and edge is in exactly one polygon and that no pair of edges intersect. Boundary and interior edges and vertices are arranged in stacks separate from the equivalent interior objects, for sampling purposes. These we check as well. We check, for example, that a boundary edge links two boundary vertices and that the boundary vertices have correct boundary-valued positions. We also check that the estimated colouring of each data point is consistent with the pattern and the other data points, by choosing one, and comparing the others to it. We join each data point to the chosen one with a line segment. If the colours of the two data points agree then γ must intersect the segment an even number of times, or zero. Finally we check that the data structures which localise geometric queries have a correct picture of γ, and that the links from γ's edges into the grid are correct.

The testing programme is looking for discrepancies larger than a floating point number `SMALL=10E-6`. In very long runs (of the order of a 100 million updates or longer) with p large and many data points, very nearly collinear triples involving vertices and vertices with data points will occur. These may generate rounding errors larger than `SMALL`, and `test_all()` will complain. However to date the dynamics of the MCMC has been consistent through these states.

When implementing MCMC with complicated states, one very satisfactory test is the following. Consider a transition $\gamma \to \gamma'$ and suppose it is accepted. In evaluating this move we had to compute an acceptance probability, $\min(1, f)$, as in Equation (10). It is sensible to check that $f(\gamma \to \gamma') = 1/f(\gamma' \to \gamma)$ up to possible rounding errors. This can often be done quite straightforwardly, if the move generating functions are set up suitably. One simply passes up the reverse move for evaluation, passing with it a warning that it is to be used for checking, and if accepted is not in fact to be used as the next state.

The most difficult problem seems to be to check the programme is not unnecessarily rejecting perfectly legal updates.

E Notes on Annealing the Arak process

In this section we explain how an estimate of the maximum a posteriori state can be obtained, and treat a small technical point.

We multiply the posterior Gibbs potential in Equation (8) by a factor $1/T$, with T a temperature. For sufficiently high T, $Q(d\gamma; T)$ flattens down to $\lambda(d\gamma)$. At small T it is concentrated on a few states of high probability density. Let Ω_0 be the set of the states maximising the density. To arrive at a state in Ω_0, we bring the system to equilibrium at large $T = T_{\max}$ and gradually lower T to some very small value $T = T_{\min}$. The function $T(t)$ is called the annealing schedule. Following [25], Van Lieshout [26] has given a schedule and $T_{\max}$, for point processes, with the property that the probability measure of the state of the Markov chain at $T_{\min}$ is equal to $Q(d\gamma; T_{\min})$. This measure tends to a uniform distribution on Ω_0, as $T_{\min} \to 0$.

Unfortunately the ideal schedule is not computationally feasible, and a more rapid descent in T must be made. This may lead to a final state which is quite obviously not modal. We use a schedule going like $T = \beta_1 + \beta_2/t^{\theta}$. β_1 and β_2 are fixed during the annealing, and depend on $T_{\max}$, $T_{\min}$ and N, the total number of updates in the run. We make sure that $T_{\max}$ is large enough to leave little preference for black or white over the whole frame, at the start of the schedule. So we have only to choose N, θ and $T_{\min}$ to set the schedule. In all runs described, we annealed to $T_{\min} = 1$ with $\theta = 0.7$, then did a short run at $T = 0.3$.

However a potential problem remains. The states in Ω_0 maximise the density. But we are free to multiply the posterior density by another density, if we simultaneously divide the base measure by the same function. This trivial operation redefines the posterior density, and we get a different set Ω_0. There is not always an obvious natural density, though there may be an inevitable one. In our case, it is common practice to write the probability measure for the Arak process with terms from the base measure in Equation (2) included as logarithms in the Gibbs potential. Then the density is maximised by states of infinite dimension, and is no use to us. The posterior Gibbs potential we minimise is the one defined by Equations (8) and (4). It is easy to see that this has a usable posterior mode, because large highly corrugated polygons (which we identify with the Goldstone modes of [23]) are penalised. In particular the empty state $\mathbf{0}$ (with two colourings of equal density) is the state maximising the prior density.

References

[1] T. Arak, P. Clifford, and D. Surgailis. Point based Polygonal models for random graphs. *Advances in Applied Probability*, 25:348–372, 1993.

[2] T. Arak. On Markovian random fields with a finite number of values. In *4th USSR-Japan Symposium on Probability Theory and Mathematical Statistics: Abstracts of Communications*. Tiblisi, 1982.

[3] D. Frenkel. Advanced Monte Carlo techniques. In MP Allen and DJ Tildesley, editors, *Computer simulation in chemical physics*, volume C397 of *Nato ASI Series*. Kluwer Academic Publications, Dordrecht, 1993.

[4] MP Allen and DJ Tildesley. *Computer simulation of liquids*. Clarendon Press, Oxford, 1987.

[5] U. Grenander. *General Pattern Theory*. Oxford University Press, 1993.

[6] D. Stoyan, W.S. Kendall, and J. Mecke. *Stochastic Geometry and its Applications*. Wiley, New York, 1987.

[7] U. Grenander and M. Miller. Representation of knowledge in complex systems. *J. R. Statist. Soc. B*, 56(3), 1993.

[8] A.J. Baddeley and M.N.M van Lieshout. Stochastic geometry models in high level vision. In K.V. Mardia and G.K. Kanji, editors, *Statistics and Images, Vol 1*, volume 20 of *J. Applied Statistics*, pages 231–256. 1993.

[9] G. Storvik. A Bayesian approach to dynamic contours through stochastic sampling and simulated annealing. *IEEE Pattern Analysis and Machine Intelligence*, 16:976–986, 1994.

[10] B.D. Ripley and F.P.. Kelly. Markov point processes. *Journal of the Lond. Math. Soc.*, 15:188–192, 1977.

[11] C. Preston. Spatial Birth and Death processes. *Bull. Int. Statistical Inst.*, 46(2):371–391, 1976.

[12] C.J. Geyer and J. Moller. Simulation procedures and likelihood inference for spatial point processes. Research Report No. 260, Department of Theoretical Statistics, Aarhus University, to appear in *Scand. J. Stat.*, 1993.

[13] P. Clifford and G.K. Nicholls. Comparison of Birth-and-Death and Metropolis-Hastings Markov chain Monte Carlo for the Strauss process. Statistics Department, Oxford, OX1 3TG, June 1994.

[14] G. Storvik. *A Bayesian Approach to Dynamic Contours.* PhD thesis, Norwegian Computing Center, Oslo, October 1992. Report No 860.

[15] T.E. Harris. The existence of stationary measure for certain Markov processes. In *Proceedings III Berkeley Symposium on Statistics and Probability*, volume 2, pages 113–115. University of California Press, Berkley, 1956.

[16] N. Metropolis, A.W. Rosenbluth, M.N. Teller, and A.H. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[17] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[18] G.E. Norman and V.S. Filinov. Investigations of phase transitions by a Monte Carlo method. *High Temperature*, 7:216–222, 1969. Translation, Journal also known as High Temperature Research USSR.

[19] P.J. Green. Contribution to discussion of paper by Grenander and Miller. RSS Meeting, 20 October, 1993.

[20] M. Blum. Program checking. In *Lecture Notes In Computer Science*, volume 560, pages 1–9. Springer, 1991.

[21] C.J. Geyer. Practical Markov chain Monte Carlo. *Statistical Science*, 7:473–511, 1992.

[22] A. Sokal. Monte Carlo methods in Statistical Mechanics. In *Cours de Troisième Cycle de la Physique en Suisse Romande, Lausanne*, 1989.

[23] K. Huang. *Statistical Mechanics*, chapter 16, pages 406–411. John Wiley & Sons, 1987.

[24] N.J. Lennes. Theorems on the simple finite polygon and polyhedron. *Am. J. Math.*, 33:37–62, 1911.

[25] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[26] M.N.M. van Lieshout. Stochastic annealing for nearest neighbour point processes with application to object recognition. *Adv. Appl. Prob.*, 26:281–300, 1994.

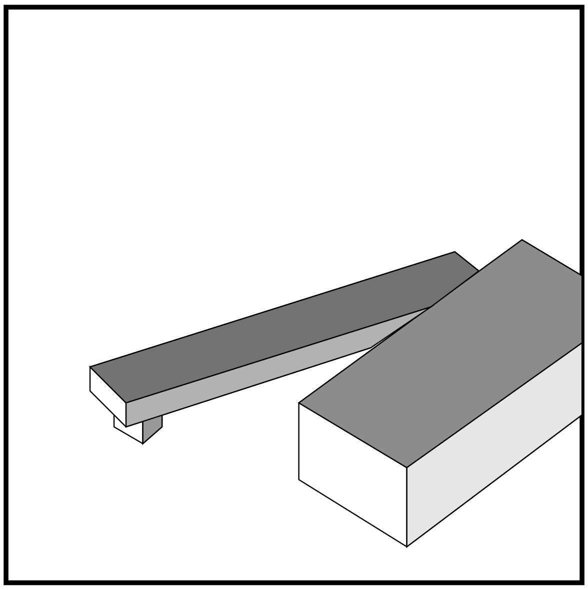

Figure 1: In this idealised geometrical picture vertices have a certain logic. Vertices of degree one (I-vertices) are absent except at the window boundary. T vertices indicate a background object partially obscured by something in the foreground. Many other rules are discernible. These rules are easily applied in the context of polygonal models, where the descriptive variables correspond to the terms of the constraints.

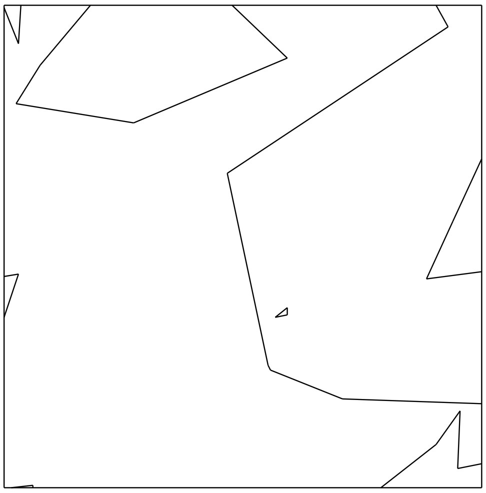

Figure 2: A sample of the Arak process with $p = 1$. Only the edges of polygonal regions are shown. Regions are coloured black or white so that the edges are colouring discontinuities. Just two colourings are possible.

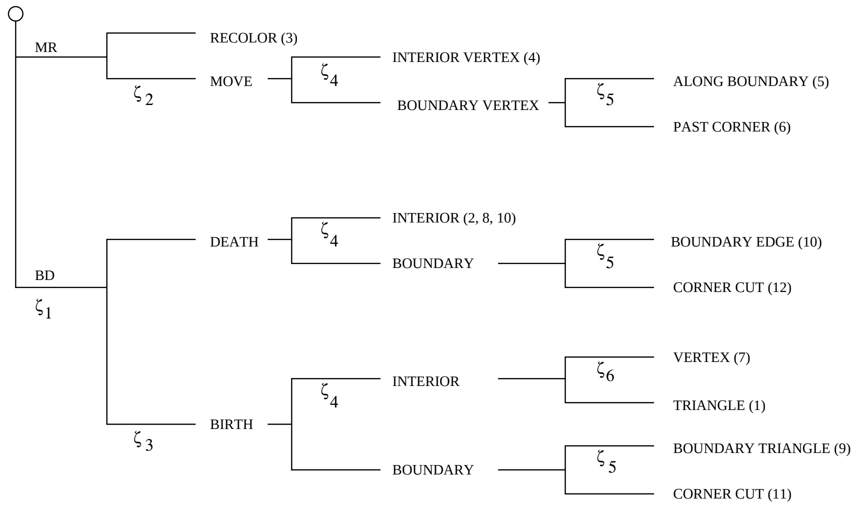

Figure 3: The tree which the programme descends to choose a move. The roots are labeled by the name and number of the corresponding update. See Section 3.3.2 and Appendix (B). For simplicity, ζ_4 weights interior choices against boundary choices, for all update types, and ζ_5 does the same job for side or corner choices, wherever they appear.

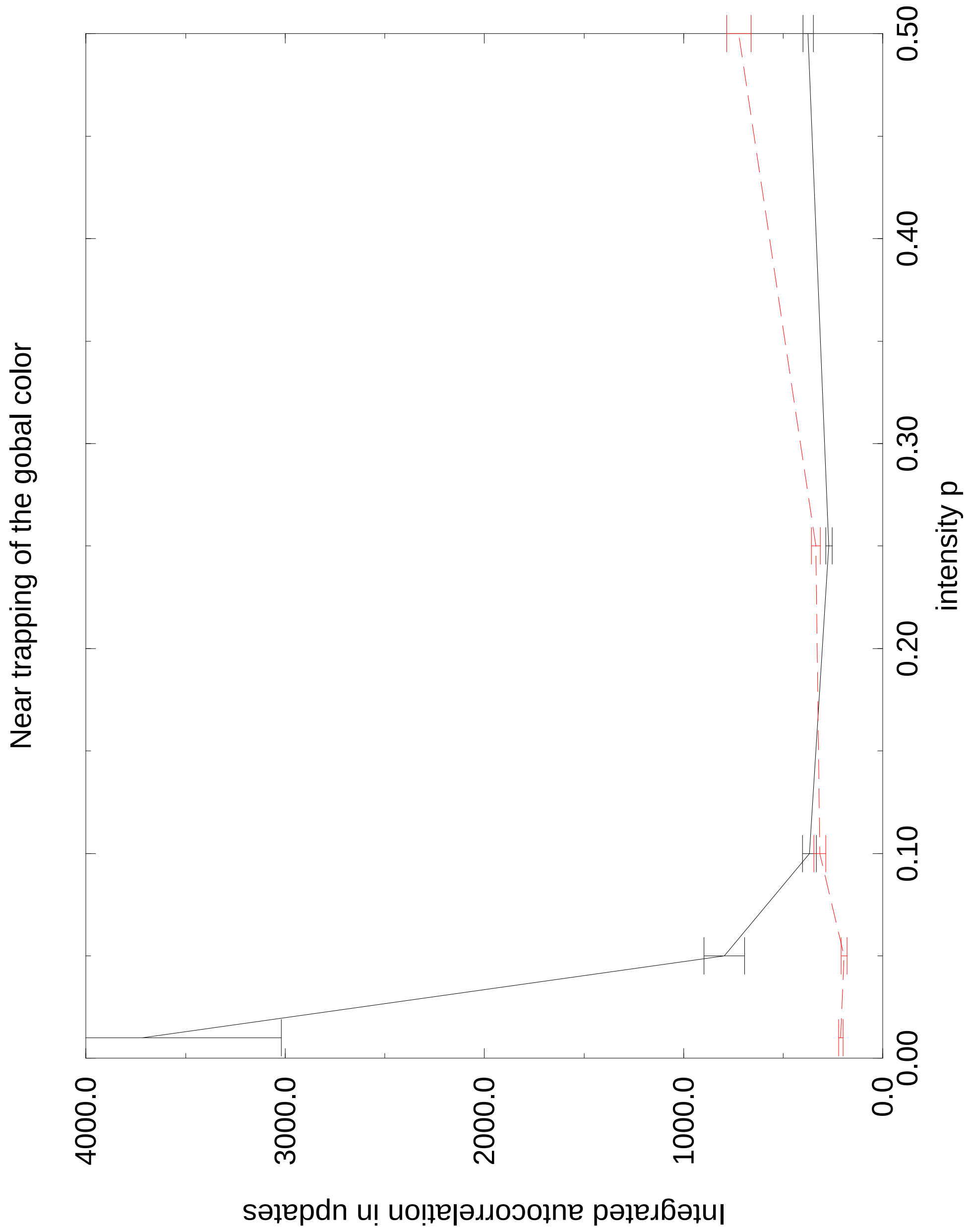

Figure 4: The solid line is the IACT for the colour at a fixed point in the interior of $\mathcal{D}$. The dashed line is the IACT for the number of edges. Error bars are one standard deviation. When p is small, or equivalently the window is small, typical realisations are of one colour. The MCMC then moves slowly from one uniform global colouring to the other, and $\tau_{|\chi(x)|}$ gets big. However it is only *nearly* trapped, see text, Section 5.2. *Note: we are obliged to change the values of the variables* ζ_4 *and* ζ_5 *with* p*. This is because the proportion of vertices on the boundary changes rapidly at small* p*. This is the cause of the uneven-ness of the graph.*

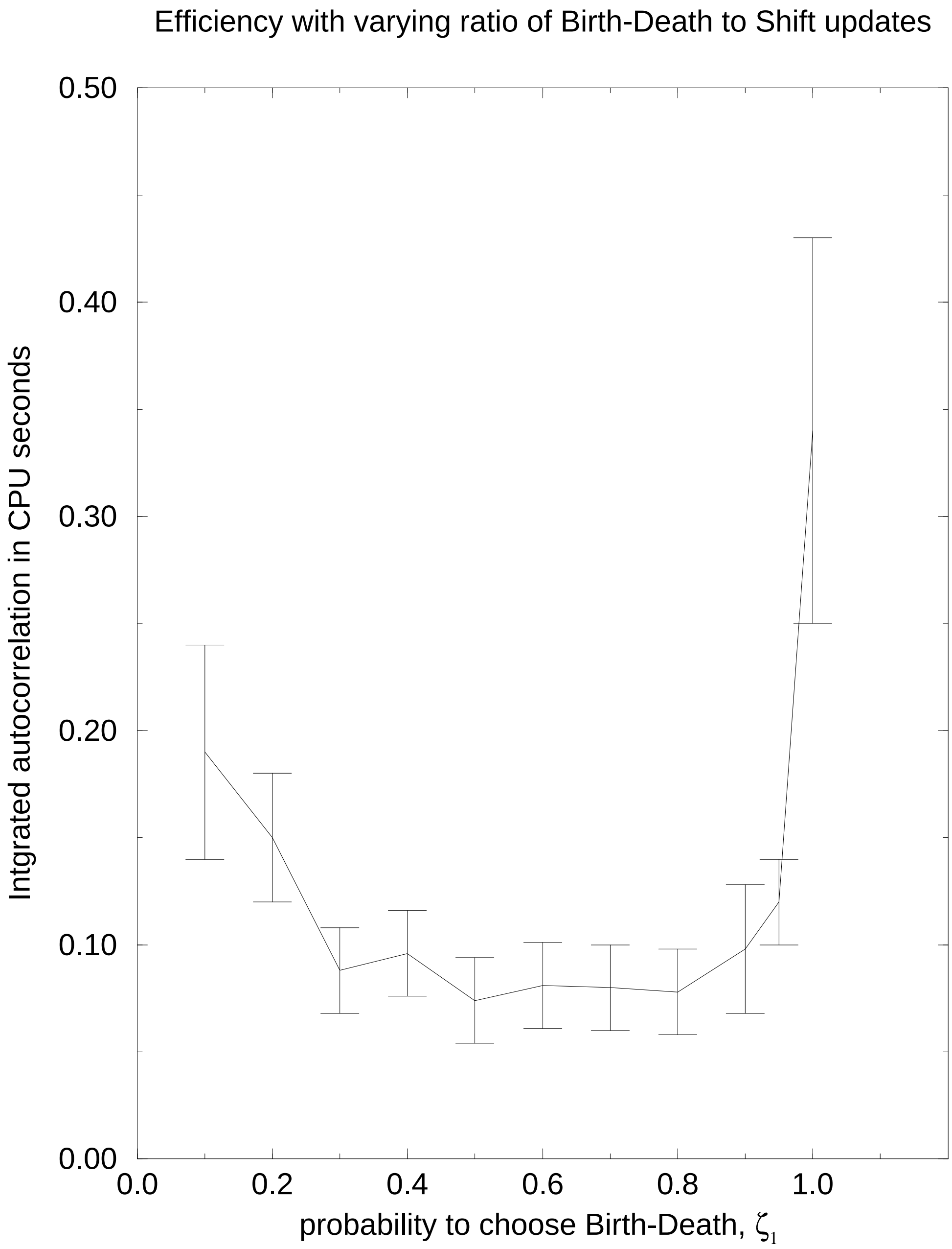

Figure 5: With ζ_2 set to one, ζ_1 is the probability to choose a Birth-Death update as opposed to a vertex shifting update. Moves do some good. If we remove all the vertex shift updates (Moves 4,5 and 6) the efficiency of our MCMC suffers; likewise if we cut down the proportion of Birth-Death too much. $\zeta_1 = 0$ is not recurrent.

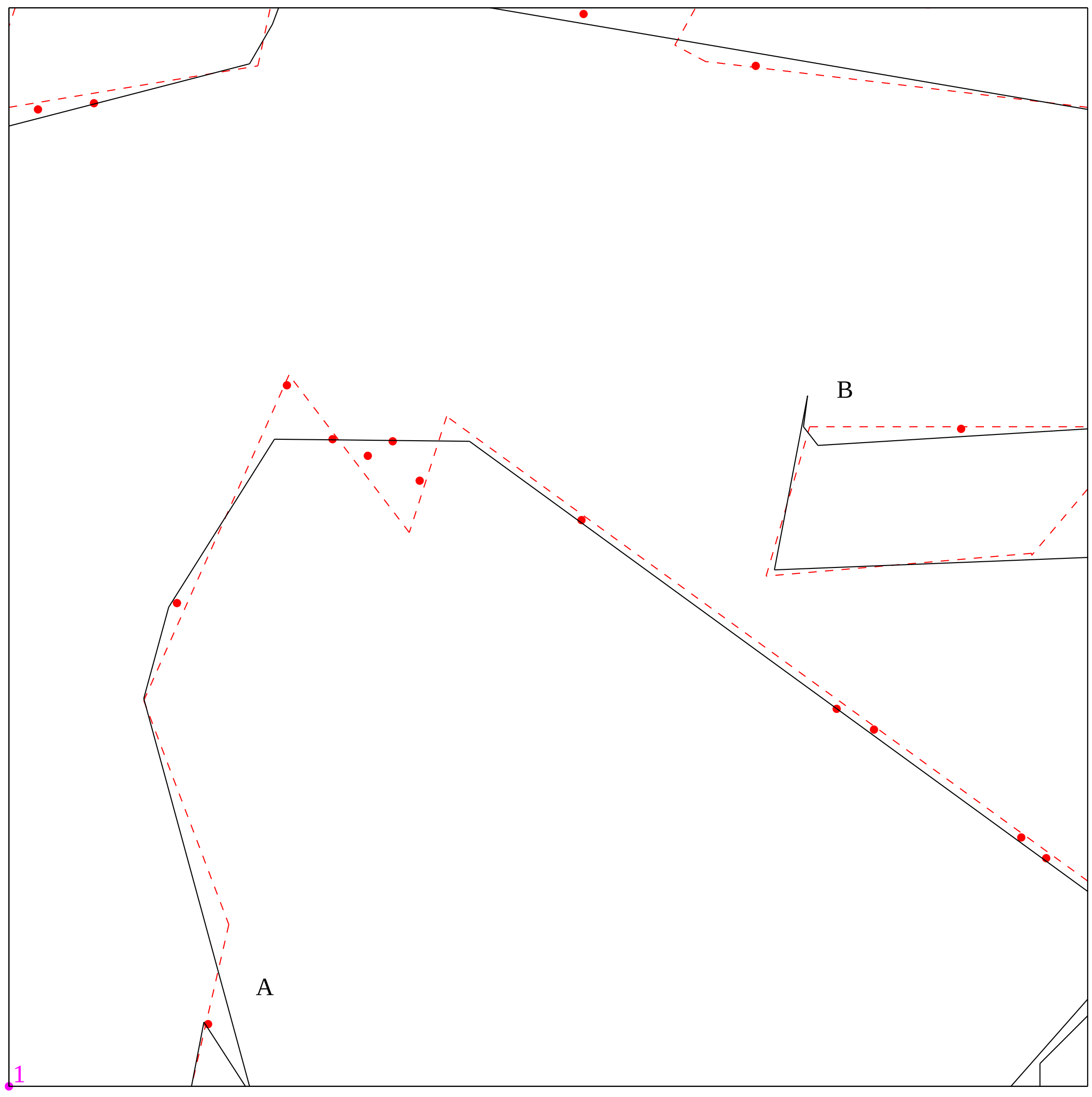

Figure 6: A sample reconstruction (solid lines) of a polygonal pattern (dashed lines) sampled from the Arak prior with $p = 1$ and observed at 1000 points. Noise variance $\sigma^2 = 0.5$, where `black=-1, white=1`. 1.7% points were misclassified. Annealing took k58 seconds. The black dots are misclassified points. Area "A" is blown up in Figure 8. "B" is a spike, see text.

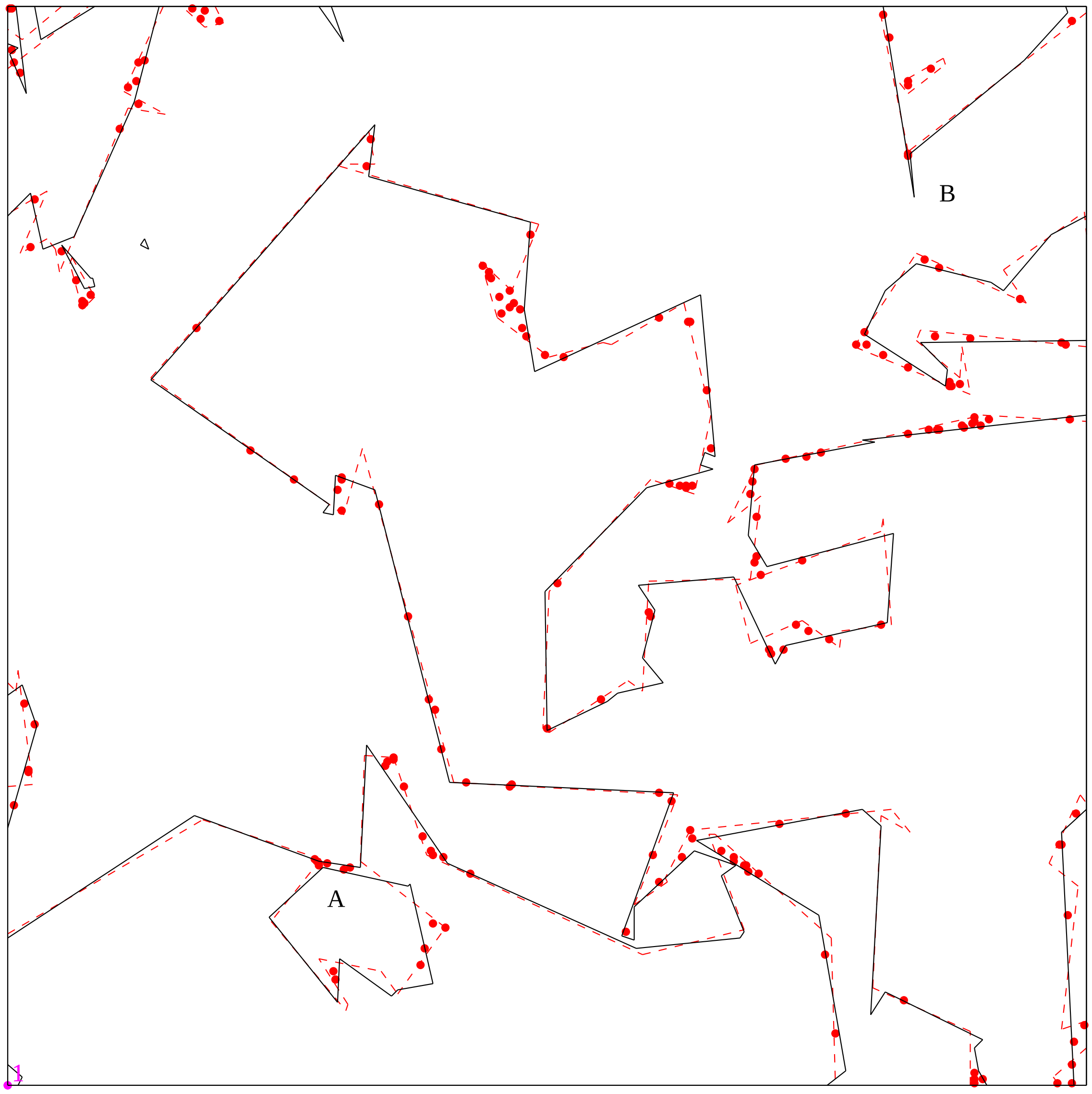

Figure 7: As for Figure 6, but $p = 3$ with 10000 data points. 1.9% points were misclassified. Annealing took around k86 seconds. Area "A" is blown up in Figure 9. "B" is a spike. Further *difficult* structures are shown in Figure 10.

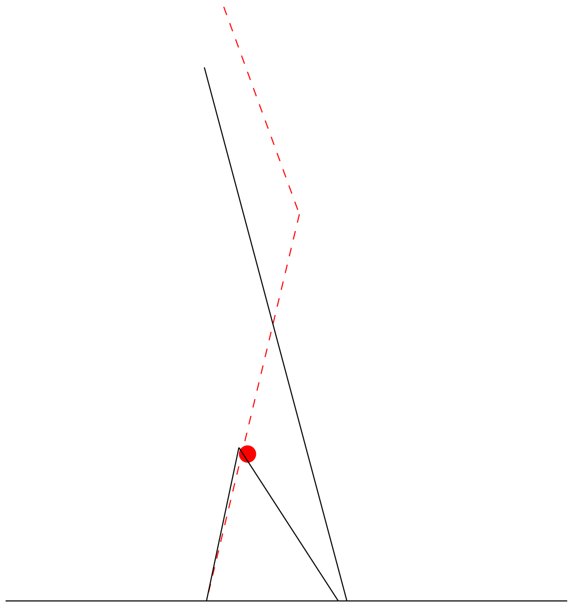

Figure 8: Area “A” from Figure 6, a form of structure “1” in Figure 10.

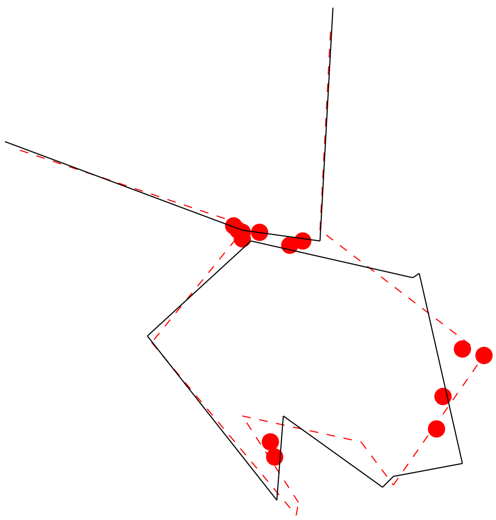

Figure 9: Area “A” from Figure 7, a form of structure “6” in Figure 10.

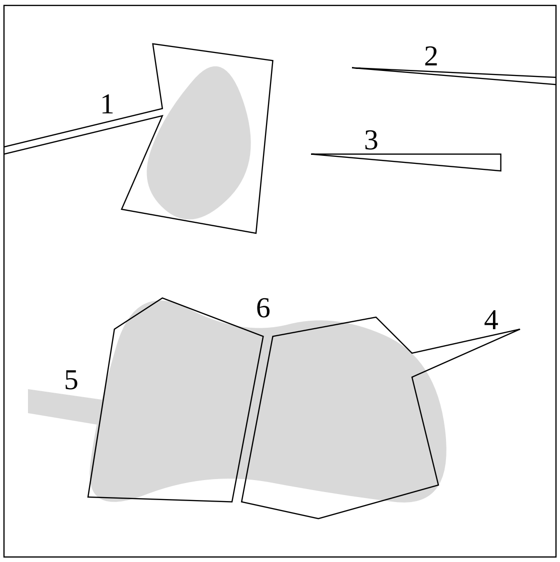

Figure 10: Problem structures. The move set we created acts only slowly on these. Number 5 is where the data (grey) indicates a collinear pair of edges, broken by a lateral extrusion.

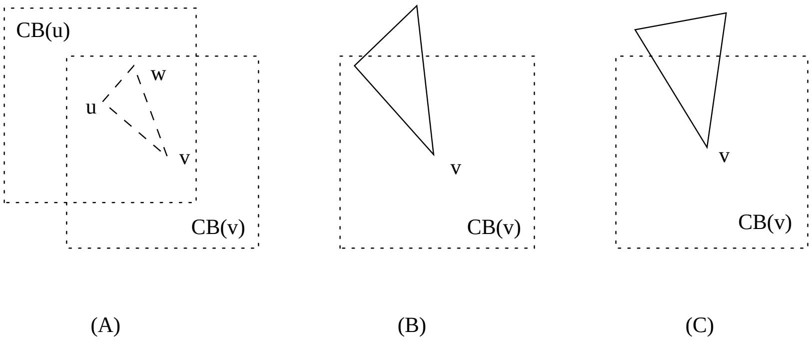

Figure 11: Updates which create and delete interior triangles. A box marked $CB(v)$ is a casting box centred at v.

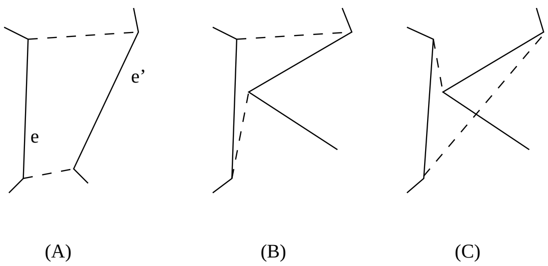

Figure 12: The recolouring move, which changes the edge configuration but has no effect on the number and position of vertices.

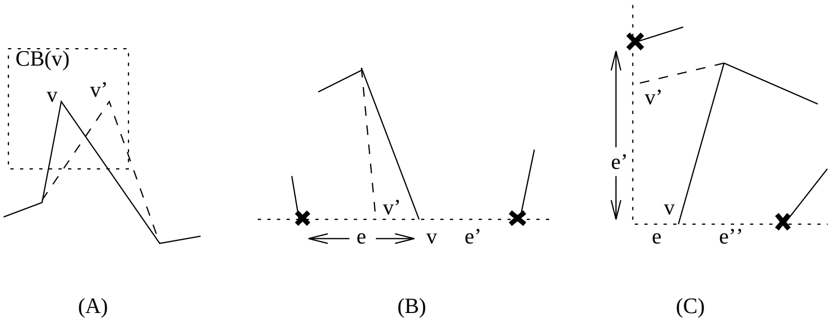

Figure 13: The updates which simply move an existing vertex to a new position, carrying its edges with it.

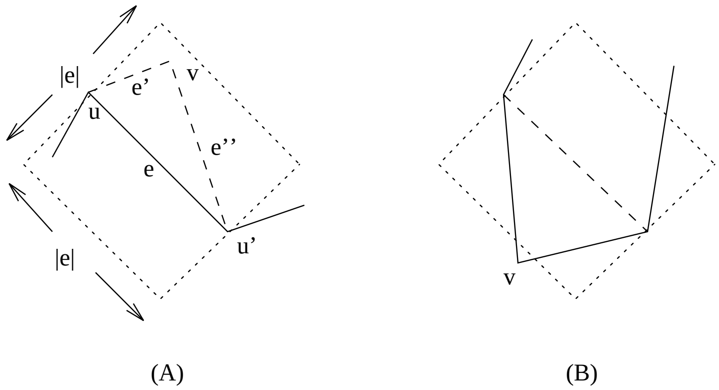

Figure 14: Updates which create and delete interior vertices. Death is not permitted in (B), as the reverse birth could not occur.

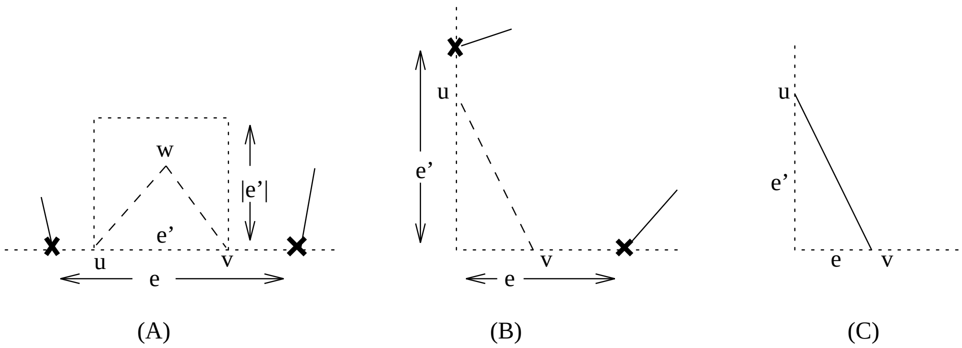

Figure 15: Updates which create and delete simple objects attached to the boundary.