Towards a generic theoretical framework for pattern-based LUCC modeling: Allocation revisited: Formal foundations and bias identification

François-Rémi Mazy, Pierre-Yves Longaretti

Highlights

Allocation Revisited: Formal Foundations and Bias Identification

François-Rémi Mazy,Pierre-Yves Longaretti

- An allocation bias criterion is formulated from a necessary self-consistency requirement.
- Error- and bias-free algorithms are required to remove mathematical and algorithmic inconsistencies.
- All tested pattern-based LUCC models are formally incorrect and biased to varying levels, but Dinamica EGO is close to be error- and bias-free.
- A comprehensive Land Use [and cover] Model (CLUMPY) based on error- and bias-free allocation algorithms has been developed and is freely available.
- For algorithmic testing and validation — instead of spatial location validation — comparison and validation must be performed in explanatory variable space, not on maps.

Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling

Allocation Revisited: Formal Foundations and Bias Identification

François-Rémi Mazy[a], Pierre-Yves Longaretti[a,b,*]

[a] *Université Grenoble Alpes, CNRS, Inria, Grenoble INP, LJK, 38000 Grenoble, France*
[b] *Université Grenoble Alpes, CNRS-INSU, IPAG, CS 40700, 38052 Grenoble, France*

ARTICLE INFO

Keywords:
Land Use Change, Land Cover Change, LUCC, Model Development, Model validation, Model Accuracy, Bias, Allocation, Probability, Transition.

ABSTRACT

Pattern-based land use and land cover change (LUCC) modeling is implemented in various software, such as the CLUE family, LCM or Dinamica EGO. These tools are now relatively mature, but their conceptual foundations are not fully discussed in the literature. In particular, these modeling environments exhibit substantially different behaviors for the same problem and the same calibration data and the origin of this difference of behavior cannot be identified on the basis of the published information.

These inter-model inconsistencies are revisited *ab initio*, focusing on allocation. The concept of allocation bias is defined and allows us to pinpoint a number of conceptual errors, biases and algorithmic inaccuracies in the existing LUCC modeling environments we have tested. We describe error- and bias-free allocation methods, implemented in our own software CLUMPY, which is shown to outperform these software in terms of formal correctness and accuracy.

For the purpose of error and bias identification, it is crucial that validation be performed in explanatory variable space, not on maps.

1. Introduction

Land Use and Land Cover Change (LUCC) is one of the major drivers of global change worldwide (Lambin and Geist, 2006), and LUCC models are now widespread to study these changes and their impacts (Agarwal et al., 2000; Verburg et al., 2006; Bielecka, 2020). A growing number of studies have been devoted to model comparison (Mas et al., 2014, 2022; Prestele et al., 2016; Alexander et al., 2017; García-Álvarez et al., 2022). Such studies question the robustness and soundness of the conclusions drawn from such models (Verburg et al., 2019).

The present work is part of a systematic analysis of the formal and algorithmic foundations of a specific category of models, namely, pattern-based LUCC models. Our objective is to identify the origin of the discrepancies in results pointed out in the literature just mentioned for a given problem and set of data and for this specific category of models, in order to reduce the uncertainty in model projections. Among the modeling environments implementing this approach, we focus on the CLUE family (Verburg et al., 1999), the latest member being CLUMondo (van Asselen and Verburg, 2013; van Vliet et al., 2015), Dinamica EGO (Soares-Filho et al., 2002, 2013), and Idrisi LCM (Eastman et al., 1995).

Pattern-based LUCC models compute the evolution of a rasterized study area in a discrete time approach through a Markovian assumption. They are organized in a small number of modules, each of which is particularly critical for our purposes. Our own algorithms are organized in three modules: calibration, estimation and allocation. The calibration and estimation modules characterize the probability that a pixel changes state per time step. The allocation module effectively allocates state changes per time step.

The present paper focuses on allocation, while our previous paper dealt with model calibration and estimation (Mazy and Longaretti, 2022b).Each paper is self-contained and can be read independently of the other, but important methodological points are given in section 2 of Mazy and Longaretti (2022b) and not repeated here; these points do explain the rationale and relevance of the research strategy adopted in our work.

Our present objective is to identify and illustrate errors and biases, either conceptual or algorithmic, in some of the existing pattern-based LUCC modeling environments by contrasting them with an error- and bias-free implementation.

*Corresponding author
francois-remi.mazy@inria.fr (F. Mazy); pierre-yves.longaretti@inria.fr (P. Longaretti)
ORCID(s): 0000-0001-8405-0141 (F. Mazy); 0000-0002-4940-0756 (P. Longaretti)

Such a comparison turned out to be mandatory in order to achieve our objective. The present paper therefore also introduces our (demonstrably) bias and error-free allocation method, but its main purpose remains to identify in a systematic manner the various errors and inefficiencies of existing pattern-based LUCC modeling environments. In order not to distract the reader from this primary objective, these formal and algorithmic developments are provided in dedicated Appendices..

Previous analyses (e.g., Mas et al. 2014) dealt with this question by documenting differences in outcome between different modeling environments, on a same set of problems. This is adequate to point out the existence of the issues we address here, but leaves users and modelers alike without a clear understanding of the origin of the problem, and, more critically, without means to elaborate a correct modeling strategy avoiding these problems. Here we identify the origin of the differences of behavior of different software for the same case study by focusing on what pattern-based LUCC models should do at the probabilistic and statistical level, and not at the level of comparison of output maps. This is a more powerful strategy for the present purpose, which furthermore allows us to focus on a single allocation time step (as all time steps are algorithmically equivalent).

The errors and biases we identify and quantify here are not all equally important, and some software do perform significantly better than others in this respect, as will become apparent in the course of the analysis presented in this paper. Furthermore, a limited subset of these errors and biases is most likely sufficient to pinpoint the origin of the difference of behavior and results observed between various modeling environments for quite a large spectrum of case studies, while some are most likely of little practical import in a wide array of actual problems. Still, we try to identify them in a systematic way as it is impossible (at least for us) to elaborate an *a priori* typology of actual contexts in which one or the other of these errors and biases is expected to be truly negligible. By being aware of their existence, model developers and possibly users as well can check *a posteriori* if these sources of error are indeed negligible. In fact, the lack of knowledge of potential sources of conceptual, methodological and algorithmic errors in the existing modeling environment prevents serious and robust error and sensitivity analyses to be performed. This also prevents the reproducibility of obtained results to a large extent (as exemplified by the spread of results on the same problem pointed out above), whereas such reproducibility is a basic requirement of quantitative scientific research.

This problem is partly rooted in the opacity or imprecision in the description of the modeling choices made in existing modeling environments. This is a wide-spread problem: none of the modeling environments we have looked at is described in the literature in a way allowing the reader to reproduce readily the formal choices and algorithmic implementation adopted there, although some software, most notably Dinamica EGO, do provide a substantial level of documentation for modelers and users.

On the contrary, all our own software (CLUMPY, see Appendix A) algorithmic choices are described in detail in this work, and, whenever possible, we have clarified or reverse-engineered (with the help of information provided by developers) the choices made in the existing modeling environments we have tested where this was needed for our present purpose.

In this work, we illustrate our strategy of error and bias identification on Dinamica EGO and Idrisi LCM, besides our own new algorithms. The rationale of this choice is the following. Both are popular statistical pattern-based LUCC modeling software (even though the fact that LCM belongs to this category of models may not be clear to users from the available research papers and documentation). They provide two nearly extreme opposites in terms of freedom of modeling choices and steepness of the learning curve for new users. Dinamica EGO offers a lot of freedom and options at each step of the modeling process, in particular through the use of its very modular model builder. As a consequence, it is somewhat difficult to master by unexperienced users, although its documentation and tutorials are quite extensive. Conversely, LCM streamlines as many of the modeling choices in an automatic way, leaving much less freedom to the user, but also making it much simpler to use. These two opposite philosophies seemed to provide an interesting ground to test our error and bias identification strategy.

This paper is organized as follows. Section 2 recalls our notations and the relevant concepts of probability theory we use in our formal analyses. A generic allocation framework is then set up (section 3). It defines what an allocation method is in principle, and characterizes the concept of unbiased allocation; this characterization is essential to identify the root of the differences of behaviors between — as well as modeling errors in — existing modeling environments. Section 4 proposes two unbiased allocation method. The first one, designed for mono-pixel patches is presented as a reference method and the second one, devoted to multi-pixel patches, is the current method used in our own software. Section 5 introduces a generic method to identify major allocation biases through a set of controlled problems of increasing complexity. The penultimate section (section 6) examines in a systematic way the errors and biases present in LCM and Dinamica EGO, following the method of the previous section. Our allocation method is included in this

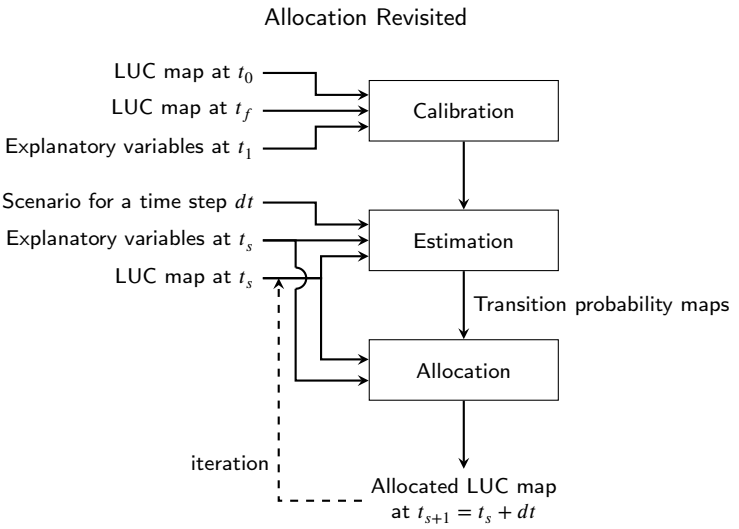


Figure 1: General architecture of our spatially explicit LUCC model.

evaluation. Our major findings and conclusions are summarized in the last section (section 8). More technical matters, in particular formal proofs and detailed descriptions of our algorithms, are collected in a number of accompanying Appendices. Reading these Appendices is not necessary to follow the arguments of the main text.

The definitions of concepts unfamiliar in the LUCC community are boxed, in order for readers to find them more easily in the text if need be.

2. Preliminary concepts and notations

Our LUCC model is organized in three modules: calibration, estimation[1] and allocation (Fig. 1). The first two determine the transition probabilities of the pixels. Calibration is performed using two raster land use and cover (LUC) maps of the same area at two distinct times t_0 and t_1 as well as a set of explanatory variables provided by the user which characterize the changes studied. Then, the estimation module determines the transition probabilities of a LUC map at time t_s with the same set of explanatory variables. The user also specifies the global transition probabilities (global amounts of change between different LUC states); this is usually done through scenarios. The details of our calibration-estimation procedure is described in a previous article (Mazy and Longaretti, 2022b).

In this paper, we focus exclusively on the allocation module. This one takes as input a LUC map at time t_s and it is then a question of generating a new LUC map which is statistically consistent with the transition probability maps obtained previously (Fig.1). The details of what this means in terms of probabilities will be presented in the rest of the article.

LUCC scenarios are often defined by overall volumes of change that indicate the rate of pixels from one state to transition to another, regardless of the associated explanatory variables. In our approach, this quantity appears in the estimation module through Bayes formula (see section 2.2.1). This point is not discussed again here; for more details, see our calibration paper (Mazy and Longaretti, 2022b).

Fig. 1 shows a simplified sketch of this three module structure (calibration, estimation and allocation). In fact, although this is omitted here for simplicity, the allocation module also calls sub-functions of the estimation one. For example, in our method (section 4.2), some probability densities are recalculated during allocation.

2.1. Definitions and Notations

We introduce a number of definitions and notations for the quantities that generically appear in pattern-based LUCC modeling. Some of these have already been used in other modeling environments, in particular Dinamica EGO. These notations are summarized in Table 1.

Land use and cover states Each LUC state is represented by a positive natural number (the null value is used for unknown state or pixels outside the study area, a common convention in GIS environments). The notation u is

[1] Estimation is often not identified as such in existing software.

used for a pixel state at time t_s (the beginning of the allocation step of interest) and v for this pixel state at the end of the allocation step.

Raster quantities We call J the set of pixels associated with initial state u at $t = t_s$. Let $\#J$ be the number of pixels in this set. Each pixel in J is identified by its index j which can be used as a superscript. One has $J = \{1, \cdots, n_J\}$. Similarly, we use the notation J_v for pixels in initial state u and final state v in the allocation step starting at time t_s.

Explanatory variables Explanatory (or ancillory, or predictor) variables – EVs – are quantities defined on pixels, and considered relevant for the statistical prediction of LUC state changes, due to their correlation with past observed changes. Also, some explanatory variable are dynamic, and their maps need to be updated once the allocation time step is completed.

Let d be the number of continuous explanatory variables. We define the explanatory variable space as a point space $\mathcal{D}$. By construction, $\mathcal{D} \subset \mathbb{R}^d$.

Explanatory variables are designated by $\boldsymbol{z}$ and their set by $\boldsymbol{Z}$. Specific explanatory variables are identified with a subscript k as z_k. These quantities are indexed by j when they refer to pixel j ($\boldsymbol{z}^j$, z_k^j).

Discrete variables are important in actual LUCC modeling case studies, but are ignored in the present work. Such variables do not pose any specific calibration problem, and the allocation methods discussed later on can be applied to an arbitrary mix of discrete and continuous variables. Including such variables however makes the various formal expressions more cumbersome. Ignoring discrete variables therefore alleviates notations and formal expressions without any real loss of generality for the objective of the paper. This is further discussed in section 7.

It is customary in the LUC modeling literature to bin continuous variables so that continuous and discrete variables can be placed on the same footing. We refrain from doing this, for consistency with our previous work on calibration (Mazy and Longaretti, 2022b).

Patch quantities LUC state transitions involve a number of contiguous pixels (patch). Pivot-cells are pixels selected during the allocation method and used as seeds of a pixel patch (see section 2.3). We use the c superscript to identify the probabilities and probability densities of these pivot-cells.

Patches are characterized by two parameters: the patch surface σ and the patch elongation e (more precisely defined in section 2.3).

Ensemble cardinals As a reminder, the prefix # is used to designate the cardinal (number of elements) of a finite set. It may also represent the expected number of a set, which may be a real number.

Probability distributions To alleviate notations, all discrete probability distributions are noted P and all continuous probability density distributions are noted ρ. The various distributions of interest are distinguished by their list of variables. When confusion is possible, a superscript is added to P or ρ to resolve the ambiguity.

Expectation values In the remainder of this work, the expectation values of any random variable Y is noted $E(Y)$ and evaluated either over an appropriate probability distribution or through a large number of repetitions of the same random event (usually a repetition of the same allocation time step).

2.2. Reminder on relevant elements of probability theory and LUCC probability distributions

Two probability distributions play a prominent role in pattern-based LUCC models:

1. $P(v|u, \boldsymbol{z})$ describes the transition probability of a pixel to state v, knowing the initial state u and the explanatory variables of this pixel.
2. $\rho(\boldsymbol{z}|u, v)$ describes the probability density[2] of explanatory variables knowing the initial (u) and final (v) states.

[2]Probabilities are dimensionless but probability densities have the dimension of an inverse volume in explanatory space $\mathcal{D}$.

element	set	definition
u, v	V	LUC state at t_s and t_{s+1}
j	J (size n_J) or J_v (size n_{J_v})	Index of pixels of state u at t_s (resp. initial state u at t_s and final state v at t_{s+1})
z_k, z_k^i, z_k^j	$\mathcal{D}_k$	kth estimation explanatory variable value for pixels in J
$\boldsymbol{z}$, $\boldsymbol{z}^i$, $\boldsymbol{z}^j$	$\mathcal{D}$	d-tuple (or column vector or point) of estimation explanatory variables for pixels in J
$\boldsymbol{Z}$	not defined	n_J-tuples of values of $\boldsymbol{z}$ taken by pixels in J
σ, e	$\mathbb{R}$	Patch size and elongation
$\delta\sigma$	$\mathbb{N}$	Pixel size (= 1 by convention)
$\#X$	$\mathbb{N}$	Cardinal (number of elements) of the finite set X. By extension, approximate number of elements in the mean
$E(Y)$	$\mathbb{R}$	Expectation value of a random variable Y

Table 1
Summary of the notations and definitions.

These probability distributions are defined on a single but arbitrary pixel. Even though continuous explanatory variables $\boldsymbol{z}$ (as well as some discrete ones) are usually deterministic functions of position (and therefore known for each pixel), by drawing pixels at random, one defines a simple but not exclusive way through which the different quantities involved in these probability distributions can be made into random variables.

This being said, it is essential to understand the difference of meaning of these probability distributions between the calibration and simulation stages. The quantities estimated in the calibration-estimation procedure are $\rho(\boldsymbol{z}|u,v)$ and $\rho(\boldsymbol{z}|u)$. These are probability distributions in the frequency sense; for example, $\rho(\boldsymbol{z}|u)$ counts the number of pixels at $\boldsymbol{z}$ within $d\boldsymbol{z}$ with initial state u in explanatory variable space $\mathcal{D}$, divided by the total number of pixels in u. Even though we relied on a kernel density estimation method to evaluate these probabilities, this method can be seen as a weighted pixel counting method, with a weight defined by the kernel function. On the contrary, during allocation, the probability distribution $\rho(\boldsymbol{z}|u,v)$ is an *a priori* distribution: one assumes that the distribution of explanatory variables at t_s knowing state u at t_s and for a transition to state v at t_{s+1} is the same as the calibrated distribution[3]. This is not true of $\rho(\boldsymbol{z}|u)$, which is reestimated at each time step through the frequency definition of probabilities.

2.2.1. Transition probability and Bayes rule

The two probability distributions that have just been introduced above are related through Bayes rule:

$$P(v|u,\boldsymbol{z}) = P(v|u) \times \frac{\rho(\boldsymbol{z}|u,v)}{\rho(\boldsymbol{z}|u)}. \tag{1}$$

This formula is particularly useful as it isolates an element with no spatial dependence, $P(v|u)$ — which is usually considered as specified by a scenario during the simulation/allocation phase — from the (usually spatially dependent) probability densities $\rho(\boldsymbol{z}|u,v)$ and $\rho(\boldsymbol{z}|u)$ which are determined by the calibration-estimation process. The notation P is reserved for categorical (discrete) random variables, while ρ refers to continuous ones. We consider only continuous explanatory variables, but mixed continuous and discrete ones are possible; in this case, the probability distribution is a probability density as well. Extension to mixed continuous and discrete variables is readily dealt with (see section 7).

Note that all probabilities involving v are defined per time step, i.e., over the calibration time span during the calibration phase, and the time-step itself during the allocation phase. Usually $\rho(\boldsymbol{z}|u,v)$ is defined directly from the calibration data, when the time-step is equal to the calibration time span (modulo possible scenario adjustments, an

[3] A scenario may specify changes in this probability distribution, if needed by the modeler, but this does not change the fact that this is an *a priori* distribution.

option which is usually ignored); otherwise, this probability must be accordingly modified, as such probabilities usually depend on the duration of the time step. This question will be further discussed in section 6.1.3.

2.2.2. Closure relations

The probabilities and probability densities defined above should verify the following closure relations:

$$\int_{z\in\mathbb{R}^d} \rho(\boldsymbol{z}|u)d\boldsymbol{z} = 1, \tag{2}$$

$$\int_{z\in\mathbb{R}^d} \rho(\boldsymbol{z}|u,v)d\boldsymbol{z} = 1. \tag{3}$$

Furthermore, although allocation algorithms generally use a method of allocation in patches and not pixel by pixel, the transition probabilities are defined so that the proportion of pixels changing state must be equal on average to the expected level defined by $P^*(v|u)$, i.e.,

$$\int_{z\in\mathbb{R}^d} P(v,\boldsymbol{z}|u)d\boldsymbol{z} = \int_{z\in\mathbb{R}^d} P(v|u,\boldsymbol{z}) \times \rho(\boldsymbol{z}|u)d\boldsymbol{z} = P^*(v|u), \tag{4}$$

These relations are exact for the exact probability distributions. They are enforced as well on the probability distribution estimates produced by our calibration-estimation method.

2.3. Pixels vs patches: assumptions in brief

In fact, pixels do not change state independently, but in patches of contiguous pixels. Dinamica EGO seems to be the only software trying to implement such an approach in a self-consistent way. Patches are characterized by a central pixel, called pivot-cell[4], and a number of characteristics, the most important being the patch size and shape.

Let us first introduce our definition of patch merging, as the occurrence of such mergings during allocation is a source of bias:

Patch merging Two patches formed during the same allocation time step are said to be merging if they have common pixels or a partially common boundary, so that the union of the two patches presents no spatial discontinuity. Patches are implicitly assumed to be simply connected spatial entities.

Patch merging may occur in real life, e.g., in between two calibration maps. However, such merging is reflected and taken into account in the patch distribution derived from the calibration data or specified by the user, which is precisely why if merging is allowed in an allocation step with patch characteristics drawn from this patch distribution, a bias occurs as this will produce an excess of large patches with respect to this distribution, which already takes into account patch merging between calibration dates. In this respect, patch allocation is not identical patch production in the real world although it reproduces the expected probability distribution. Consequently, preventing patch merging is mandatory to obtain a bias-free allocation method (see section 6.3). Patch merging is prevented algorithmically in a self-consistent way (section 4.2).

Finally, we make the following assumptions in the course of the present work:

1. The pivot-cell probability distribution in explanatory variable space is independent of other patches characteristics (size and shape). As any other pixel, a pivot-cell is identified by its index i or j.
2. Multi-pixel patches are characterized by their size (σ) and a shape parameter (e) only, with probability distribution $\rho(\sigma, e|u, v)$. Size and shape are assumed to be independent variables.
3. The various probability distributions of pivot-cells with respect to explanatory variables are identical to the distributions characterizing the entire pixel population: $\rho^c(\boldsymbol{z}|u,v) = \rho(\boldsymbol{z}|u,v)$.

[4]The developers of Dinamica EGO have introduced this definition.

Patch sizes can be measured in surface or pixel number. For simplicity, we assume in this work that they are measured in pixel number, so that, conventionally, the size of pixel, $\delta\sigma$ is set to unity: $\delta\sigma = 1$. Note that the probability distributions of size and elongation are discrete by nature. However, for simplicity, when these quantities are not obtained from calibration data (the rule so far in LUCC modeling), they are given by a probability distribution provided by the modeler. Such probability distributions are easier to specify with continuous variables. Consequently, either probability density distributions — $\rho(\sigma|u, v)$, $\rho(e|u, v)$ — or discrete distributions — $P(\sigma|u, v)$, $P(e|u, v)$ — are used depending on the context.

These assumptions imply that the overall probability of change of pivot-cells, $P^c(v|u)$, must be related to $P(v|u)$. This relation is specified in section 4.2. Furthermore, small enough time steps are required in order for pixels to undergo at most one state change per time-step. This requirement is in general not very constraining.

Note finally that our assumptions imply that Bayes rule for pivot-cells takes the form

$$P^c(v|u, \boldsymbol{z}) = P^c(v|u) \times \frac{\rho(\boldsymbol{z}|u, v)}{\rho(\boldsymbol{z}|u)}. \tag{5}$$

These assumptions are discussed in Appendix B.

3. Allocation method and bias: definitions and constraints

Now that the various relevant probability distributions have been defined and their properties and relations clarified, let us return to the discussion of section 2.2, which is not sharp enough yet to specify how probabilities should unambiguously be used in an LUCC simulation. For this purpose, it is necessary to introduce a few more definitions.

Occasionally in the remainder of this paper and related appendices, we make reference to some results of sampling theory. The interested reader can find detailed information about this, e.g., in the recently reprinted classical text of Cochran or the most recent and possibly more accessible one by Lohr (Cochran, 1977; Lohr, 2019).

3.1. Sampling design, allocation procedure and allocation method

We introduce several definitions in the following[5]:

Sampling design A sample is a collection of elements of interest (here, pixels) taken in a set (here, the set J of pixels in state u at t_s). A sampling design is a random process through which these elements are selected to constitute a sample. From a theoretical point of view, a sampling design is defined once the probability of occurrence of all possible samples is specified. From a practical point of view, the problem is to design an algorithm producing samples according to a given law of probability. Quite often, producing an algorithm is a way to define (possibly implicitly) the probability law of all possible samples and its associated characteristics. If instead the desirable properties of the sample probability law are specified beforehand, the difficulty resides in enforcing these properties exactly in the sampling algorithm(s).

Allocation procedure An allocation procedure is an algorithmic process through which a final state is attributed to pixels in given sample. An allocation procedure specifies how and in which order pixels in a sample are processed.

Allocation method In the context of this work, an allocation method is defined by the combination of a sampling design and an allocation procedure.

Our first allocation method applies to pivot-cells only. A simple allocation method dealing with this is presented in 4.1. It is well suited to mono-pixel patch allocation. Mono-pixel patches are clearly artificial and unrealistic, but this setting nevertheless allows us to identify a number of problems in existing software (sections 5 and 6). Our mono-pixel allocation method has a number of desirable features, in particular it is error- and bias-free (section 3.2). Our next allocation method deals with extended patches. The difficulty here lies in enforcing the correct transition probability law while allocating pixels in patches. These methods are presented in section 4.

[5]The sampling definition is standard in sampling theory. The other two are specifically taylored for the present purpose.

It is implicitly assumed that pixels undergo at most one state change per time step. Generally speaking (and unless some pixel states cannot be changed once reached), this implies in turn that the maps used for calibration must be obtained at close enough dates, in which case all amounts of LUC state change will be small enough. This must also apply to simulation time steps for the small changes quantified in the calibration data to be meaningful in a simulation, especially that a number of explanatory variables are updated along the simulation. Also, this requirement makes the avoidance of patch merging much easier to enforce.

Once some pixels (in state u at time t_s) have undergone a change in a given time-step, they must be taken out of the pool J to ensure that they do not undergo another state change. This is referred to as sampling without replacement in sampling design theory[6]. LUCC models make use of an allocation procedure in an explicit way, but the related sampling design is often either implicit, simplistic, or not clearly described. This is unfortunate, because sampling designs without replacement (the ones of interest here) are well-known to lead to algorithmic difficulties in sampling theory. Because of this, the (explicit or implicit) assumptions made about sampling design often lead to conceptual errors and biases in existing software. These in turn are important to identify, as they figure among the primary reasons of the differences of behavior between different software for a given problem.

To sum up, an unambiguous use of probability theory in LUCC models relies on unambiguous definitions of sampling designs and allocation procedures (i.e., allocation methods). To proceed further, one first needs to characterize possible biases in the allocation methods, as this serves as an ultimate test of correctness and relevance of a sampling method. This is the object of the next section.

3.2. Bias: definition

For an allocation method to be accurate, it is necessary that the expected mean number of pixels in each final state from any given initial state, as well as their mean distribution in $\boldsymbol{z}$, are respected. In other words, for pixels, one requires that[7]

$$E(\#J_v) \equiv \hat{P}(v|u)\#J = P(v|u)\#J, \tag{6}$$

$$E(\#J_{\boldsymbol{z},v}) \equiv \hat{\rho}(\boldsymbol{z}|u,v)E(\#J_v)\delta\boldsymbol{z} = \rho(\boldsymbol{z}|u,v)E(\#J_v)\delta\boldsymbol{z}, \tag{7}$$

where by definition $\hat{P}(v|u)$ (resp. $\hat{\rho}(\boldsymbol{z}|u,v)$) is the post-allocation value of $P(v|u)$ (resp. $\rho(\boldsymbol{z}|u,v)$) and $\delta\boldsymbol{z}$ is a small volume around $\boldsymbol{z}$. $\#J_{\boldsymbol{z},v}$ refers to the number of pixels in initial state u and final state v. Note that $\hat{\rho}(\boldsymbol{z}|u,v)$ is independent of $\delta\boldsymbol{z}$. The second relation therefore requires that the pre- and post-allocation probability distribution of explanatory variables for given initial and final state are identical, while the first requires the same property of the global probability of change.

Similar relations apply to pivot-cells (*mutatis mutandis*)

$$E(\#J^c_v) = P^c(v|u)\#J, \tag{8}$$

$$E(\#J^c_{\boldsymbol{z},v}) = \hat{\rho}^c(\boldsymbol{z}|u,v)E(\#J_v)\delta\boldsymbol{z} = \rho^c(\boldsymbol{z}|u,v)E(\#J^c_v)\delta\boldsymbol{z}. \tag{9}$$

$\#J^c_v$ is the number of pivot-cells in final state v, and $\#J^c_{\boldsymbol{z},v}$ the number of pivot-cells in final state v and with explanatory variables $\boldsymbol{z}$ within some small volume $\delta\boldsymbol{z}$.

Similarly, it is desirable that the patch characteristics probability distribution are also satisfied, e.g., for patch surfaces

$$E(\#J_{\sigma,v}) = P(\sigma|u,v)E(\#J^c_v), \tag{10}$$

where $\#J_{\sigma,v}$ is the number of patches in final state v and size σ.

It is sufficient to ascertain that these relations are preserved in a single time step, due to the repetitive algorithmic nature of a simulation, once an allocation method is defined.

[6] Conversely, a sampling design with replacement puts pixels back in the pool, allowing them to be selected again.

[7] Note that because the initial state is known, $E(\#J) = \#J$. These relations does not necessarily imply that explanatory variables have been binned, see Longaretti and Mazy (2022) — hence their designation as pseudo-numbers, as these are not required to be integers. Such numbers are used only in the formal proofs. In the tests of section 6, these relations are algorithmically tested by using directly our bin-free calibration estimation procedure; see section 6 for details. Finally, these relations do not assume that the total number of pixels undergoing a given LUC state transition is exactly enforced, contrarily to common practice in LUCC modeling. In the end, we do enforce such amounts of change. We momentarily avoid doing this, mostly for mono-pixel patches. If exact amounts of change are enforced, $E(\#J_v) = \#J_v$ in these relations.

In Eq. (8), $P^c(v|u)$ is related to $P(v|u)$ as follows[8]

$$E(\#\boldsymbol{J}_v^c) = \frac{E(\#\boldsymbol{J}_v)}{E(\sigma)}. \tag{11}$$

Consequently:

$$P^c(v|u) = \frac{P(v|u)}{E(\sigma)}, \tag{12}$$

so that Eq. (8) is automatically satisfied if Eq. (6) is satisfied and if the distributions of explanatory variables and patch size are independent, as assumed in this work.

The same argument holds for Eq. (9). Indeed, the argument of footnote 8 applies as well to the relation between $\#\boldsymbol{J}^c_{\boldsymbol{z},v}$ and $\#\boldsymbol{J}^c_{\boldsymbol{z},v}$: $E(\#\boldsymbol{J}_{\boldsymbol{z},v}) = E(\#\boldsymbol{J}^c_{\boldsymbol{z},v})E(\sigma)$. Combining this with Eqs. (11) and (7), and taking into account the assumption $\rho^c(\boldsymbol{z}|u,v) = \rho(\boldsymbol{z}|u,v)$ (discussed and justified to some extent in Appendix B) on top of the independence between explanatory variables and patch sizes leads to Eq. (9).

We are now in position to state our definition of an unbiased allocation procedure[9]

Unbiased allocation definition If the requirements of identity of these pre- and post-allocation probability distributions expressed above in Eqs. (6) through (10) are satisfied, the allocation method producing the post-allocation distributions is by definition unbiased.

It is useful to elaborate somewhat on the meaning and rationale of these requirements. Let us focus on Eq. (7) for definiteness. This relation *must* be preserved in a statistical sense as $P(v|u)$ and $\rho(\boldsymbol{z}|u,v)$ are the only probability distributions that do define the state change process. More precisely, the requirement here is that the post-allocation probability distributions $P(v|u)$ and $\rho(\boldsymbol{z}|u,v)$ — i.e., measured once pixels have changed state in a time step — are identical to the pre-allocation ones — i.e., as they appear in Bayes rule Eq. (1) which defines the transition probability — except for statistical noise. There is nothing mysterious or far-reaching here, this just a self-consistency requirement[10], i.e., we wish to check that allocation methods do enforce fair sampling of all relevant probability distributions.

However, this basic self-consistency requirement is not satisfied in the modeling environments we have tested (we suspect this is the norm and not the exception), with sometimes rather substantial consequences for the reliability and robustness of LUCC simulations.

Let us conclude this section by commenting upon the relation between biases and formal errors. Formal errors are errors of probability theory and statistics. Any such error will lead to a bias once the formalism is implemented in an algorithm. For example, a lack of attention to the difference between sampling designs with and without replacement is a formal error. This type of error inevitably produces algorithmic biases. Some errors may also occur directly at the level of the algorithm design, and will also produce biases. Detecting biases therefore points towards an error, either of theory or of algorithmic design. Pinpointing the source of error may be challenging.

3.3. Comment on real space performances

It must be pointed out that comparing performances on, e.g., $\rho(\boldsymbol{z}|u,v)$, i.e., in explanatory variable space $\mathcal{D}$, is a much more powerful way to identify errors and biases than comparing maps in real space. Indeed, on a single allocation time step, and because changes per time step are small, there is little chance that allocations would be performed at the same spatial locations, especially over a single time step.

[8] This relation follows directly from our assumption of independence of pivot-cells and patches statistical properties (at given initial and final LUC state). On the one hand, the average total quantity of change for the state transition considered is $E(\#\boldsymbol{J}_v)$. On the other hand, this same quantity can also be computed as follows. The expectation value (average) patch size is $E(\sigma)$. This applies for every pivot-cell, so that the total quantity of change (averaged over patch sizes) is $\#\boldsymbol{J}_v^c E(\sigma)$. Finally, averaging the number of pivot-cells itself leads to $E(\#\boldsymbol{J}_v^c)E(\sigma)$ for the average total quantity of LUC state change.

[9] In sampling theory, it is pointed out that, especially in contexts where variances can be large (e.g., leading to a factor of error of several in the estimates of quantities of interest), a slightly biased sampling design is acceptable if the variance is substantially reduced. In our experience, variances are not large enough in LUCC modeling to motivate us to look into this question.

[10] The comparison must of course be performed before updating dynamic explanatory variables, to be meaningful.

On the other hand, it is mandatory that they occur at statistically equivalent location, otherwise a bias occurs. Such equivalent locations can only be properly identified in explanatory variable space $\mathcal{D}$, not physical space. We are aware that this way of proceeding goes against the most common methods of analysis of LUCC model performances in the literature, but we believe that the power of this approach for the present purpose (identifying errors and biases), that will be apparent in section 6, will be sufficiently convincing in itself to justify this departure from current habits. This is true irrespective of the fact that over the course of a long simulation where a significant fraction of pixels has changed LUC state, some (possibly significant) overlap between projected and actually observed changes should be present.

This does not imply that comparisons in real space are not useful for other purposes. For example, such real space comparisons may point towards the fact that important explanatory variables may have been overlooked if spatial pattern differences are unrealistically large[11]. Also, improving spatial allocation accuracy in actual case studies is definitely an important overall goal of LUCC modeling. In fact, the work presented in this paper is also a contribution to this goal, as formal and algorithmic errors and biases produce systematic deviations from spatial accuracy. Also, spatial accuracy will clearly be better over a number of time steps than on a single one.

4. Unbiased Allocation Methods

Having at hand unbiased allocation methods is necessary to pinpoint the origin of conceptual and algorithmic errors and biases in existing LUCC models and modeling platforms. We describe two such methods here, one for mono-pixel patches (uSAM) and one for multi-pixel patches (uPAM). The mono-pixel patch method is formally shown to be unbiased in Appendix C.2, and a proof for the multi-pixel patch method is given in in Appendix E. The multi-pixel patch method is also seen to be unbiased on a case by case basis in section 5.

The uSAM algorithm enforces quantities of change only statistically per time step, while uPAM enforces them exactly. However, an alternative uSAM-like algorithm enforcing quantities of change exactly is provided in Appendix D, as this constitutes a convenient first step for the proof exposed in Appendix E. Enforcing quantities of change is not strictly necessary in principle. However, as performing multiple allocation runs of the same simulation is usually not performed, enforcing quantities of change is definitely a useful constraint for actual case studies.

Our strategy of error and biases identification involves an averaging over a large number of realizations of the same time step, in order to remove statistical noise and isolate systematic effects. Because of this, designing error- and bias-free algorithms enforcing exactly quantities of change is not necessary. However, it is definitely useful, first because this is the common choice in the LUCC modeling community, and second because this enforcement may induce biases by itself if not dealt with carefully. These two reasons motivated us to devise a multi-pixel patch allocation that is not only error- and bias-free but also enforces quantities of change.

4.1. Unbiased Simple Allocation Method (uSAM)

A mono-pixel bias-free allocation method can be defined following the procedure introduced in section 3.1:

Sampling design Every pixel in J (state u at t_s) is tested for possible selection as a pivot-cell. This is hardly a sampling design, but it turns out that testing all pixels leads to a computationally more efficient allocation method than selecting a smaller subset through pruning[12].

Allocation procedure All pixels in the sample (in this case, all pixels) are allocated a final state (which can be identical to the initial one) with the help of the multinomial sampling test algorithm MUST of Appendix C. This algorithm tests all possible final states at once. The resulting mono-pixel patches may be adjacent, as we ignore patch properties in this sampling method and do not try to prevent patch merging for mono-pixel patches.

The fact that this method is bias-free is shown in Appendix C.2.

[11] This was pointed out by one of the paper's referees

[12] Dinamica EGO introduced the concept and process of pruning, which in effect constitutes its sampling procedure. Pruning is a pre-selection process designed to focus on the pixels most likely to undergo a transition. This idea has been implemented by Dinamica EGO and IDRISI LCM in two different ways (LCM does not use the term pruning, but its allocation procedure corresponds to maximal pruning of the probability distribution). The resulting procedure leads to a biased allocation as shown in section 6.1.

4.2. Unbiased Patch Allocation Method (uPAM)

A multi-pixel patch allocation algorithm cannot simply consist in adding a patch formation procedure to the mono-pixel patch algorithm described in the previous section. Such a method presents a bias when quantities of change are enforced, and the resulting post-allocation pixel probability distributions will differ from the pre-allocation ones. Also, patch merging must be avoided in order for the post-allocation patch characteristics probability distribution to be also identical to the pre-allocation one (see section 2.3).

To avoid these problems, we devised a rather involved sampling design without replacement in order to avoid patch merging. The root of the difficulty is due to the fact that probability distributions are defined in $\boldsymbol{z}$ space, whereas the question of patch merging is a spatial one. This difficulty requires a specific adaptation of standard sampling theory procedures.

Our allocation method involves two different types of probability distribution updating:

- The first type is related to the step by step patch formation just mentioned at a given time t_s. All pixels belonging to the newly formed patch must be removed from the pixel population before the next patch can be formed. In fact, the induced limitations of possibilities for new pivot-cells selection and patch formation while constituting the patch sample is precisely why the overall post-allocation probabilities $\rho^c(z|u,v)$ and $P^c(v|u)$ can be identical to the pre-allocation ones. In practice, this implies that the pivot-cell probability distributions $\rho^c(\boldsymbol{z}|u)$ must be updated after every patch construction.

- The second type consists in updating the $\boldsymbol{z}$ values of dynamic explanatory variables (i.e., variables that change in the course of the simulation, e.g., distance to urban areas due to urban sprawl). This type of updating must be performed *after* the post- and pre-allocation probability distributions are compared to check for possible biases. In particular, $\rho^c(z|u,v)$ must not be updated after each new patch is formed at time step t_s, as it embodies an *a priori* preference among possible $\boldsymbol{z}$ values for a given transition $u \to v$, defined pixel by pixel, and independently of the number of pixels that may or may not have already changed state. This *a priori* distribution holds by definition for the whole time step allocation process. Because of this, in our model architecture, all dynamic updating are performed at the beginning of the next time step (see Fig 1).

The structure of our patch allocation method is described below and is represented in a schematic way in Fig. 2. It makes use our sampling design and allocation procedure in an iterative way. This iterative strategy differs slightly from our definitions above; it is adopted to produce a formally correct sampling method without replacement:

Sampling design A subset of potential pivot-cells is created at random by applying the MUST algorithm to the total pixel pool with $P(v|u,\boldsymbol{z})$ as the probability distribution of final states, and keeping the pixels that actually did undergo a change of LUC state (i.e., pixels with $v \neq u$). We then draw a single such potential pivot-cell at random along with its chosen final state. All other potential pivot-cells are then dismissed. If state v_0 is selected, MUST is then applied with $P(\sigma, e|u, v_0)$ to affect a patch size σ and elongation e to the newly drawn pixel.

Allocation procedure ("patcher") Our patching algorithm attempts to create a patch around this pivot-cell with a size and elongation drawn as just described (this step is detailed in appendix J). This patch creation can either be a success or a failure. A failure occurs in two cases:

1. the produced patch is adjacent to a previously created patch at the same time step which leads to patch merging;
2. the number of pixels available to create the patch is not large enough to produce the expected patch size (e.g. if the selected pivot-cell falls in too small an enclosed area bounded by different initial states than u). In this case, no pixels are allocated.

Whether it succeeds or fails, this patch creation algorithm returns the subset of pixels that have been processed. If the procedure succeeds, the new patch is kept. In both cases (success or failure), the allocation procedure has been completed and stops. The exact implementation of our patch construction algorithm is presented in Appendix J.

Iteration and updating After completion of the allocation procedure, three updating steps are necessary before proceeding any further:

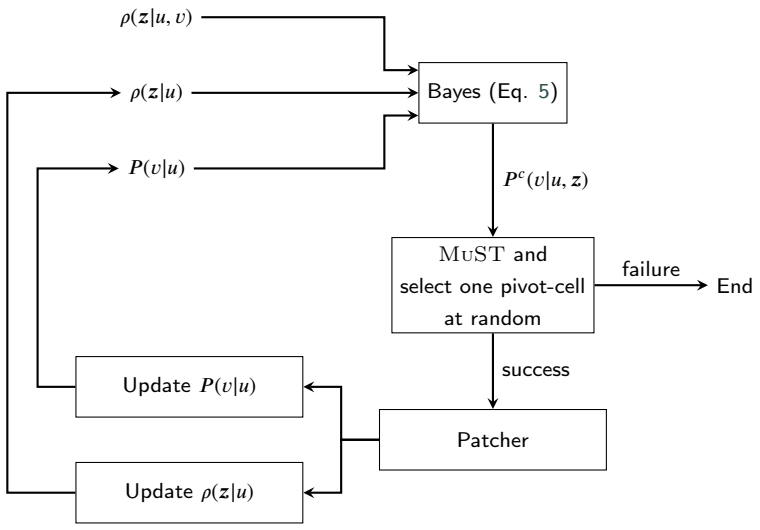


Figure 2: Unbiased Patches Allocation Method (uPAM) workflow.

1. If the patching procedure was successful, we modify $P(v|u)$ to take into account the allocated area: $P(v|u)$ is updated to $P(v|u) - \sigma/\#J$ where σ is the number of pixels in the newly created patch and $\#J$ is the current number of pixels in u state (i.e., corrected for pixel removals in previous iterations).
2. The subset of pixels returned by the patching algorithm are removed from J (whether or not the procedure was successful) and $\#J$ is accordingly updated.
3. Their removal implies to update $\rho(z|u)$, which is done with our calibration method (Mazy and Longaretti, 2022b).

The sampling design and allocation procedure are then called iteratively until $P(v|u) < 0$ for all $v \neq u$, or the updated set J is empty or the procedure cannot form a new patch. If the total quantities of change are small, the first outcome (the desired one) is by far the most probable.

It should be noted that if we parameterize this method to generate only one-pixel patches ($\sigma = 1$), the allocation method obtained is equivalent in the mean[13] to the one defined for single-pixel patches (section 4.1), and exactly equivalent to the algorithm introduced in Appendix D. Also, this allocation procedure covers both the patcher and expander procedure of Dinamica EGO and other software (see Appendix J.3).

In Appendix E, the fact that this allocation method is bias-free is proved. In addition, a pseudo-code of this method is also given in the same appendix (algorithm 2).

5. A progressive strategy for the identification of allocation biases

In this section, we describe the method we have devised to identify the origin of allocation discrepancies observed between several existing software (Mas et al., 2014). We first describe how these software are parameterized and how the results are processed (section 5.1). Then, our systematic strategy is described (section 5.2). Finally, we define a set of case studies specifically designed for this purpose (section 5.3).

5.1. Error identification strategy

First, we specify the transition probability maps $P(v|u,z)$. These maps are exactly known by construction and will therefore be used as reference maps in the analysis of biases in the allocation procedure of various software (including ours). To this effect, the calibration step is ignored in the model by inserting these transition probability maps just prior

[13] The distribution of pivot-cell numbers around the mean will be different but the mean number of pivot-cells will be identical. As we care only about preserving the mean number of pixels in line with the definition of a bias-free algorithm, this difference is unessential at the level of formal correctness adopted in this work and for the present purpose.

to the allocation step in the software under consideration[14]. This ensures that all modeling environments we have tested do have a common well-defined and controlled starting point for their respective allocation methods, independently of what their calibration methods prescribe from calibration data.

Alternatively, we could instead produce pivot-cells and patches from our assumed probability maps and let the tested software proceed through calibration and allocation. However, this would add noise to the process as well as possible calibration errors and biases. The proposed strategy allows us to focus on allocation issues only.

In section 3.2 we defined bias on the basis of expectation values. On the other hand, performing a comparison of pre- and post-allocation probability distributions is unavoidably affected by statistical noise. We can estimate the variance of this noise, but this may not give a sufficiently precise approximation of the required expectation values of the post-allocation probability distributions, Eqs. (7) and (9). In order to reduce this statistical noise, a Monte Carlo approach is adopted: the same single allocation step is performed a number of times, and the post-allocation probability distributions are obtained by averaging over this series of realizations, thereby substantially reducing the statistical noise and providing more precise estimates of the expectation values we wish to compare across the various allocation methods we have tested. This is presented in more detail in Appendix I, where the setting of these Monte Carlo simulations is specified to a given tolerance level.

Next, the post-allocation transition probabilities are estimated to compare them to the pre-allocation ones. This amounts to applying the calibration and estimation modules to the pre- and post-allocation maps. We propose to use the calibration-estimation method based on kernel density estimation introduced in Mazy and Longaretti (2022a,b) which was shown to have the best accuracy among all the calibration methods we have tested and compared. Even if a different calibration-estimation method were chosen, sticking to a single one to compare different allocation methods is also essential to ensure that the post-allocation transition probabilities have been computed in the same way. The accuracy of our method ensures that most observed differences in performance of the allocation methods we have tested is due to the allocation methods themselves.

5.2. Observed sources of bias

The post-allocation transition probability map is compared to the transition probability map used as input of all the allocation method. In the same way as proposed in Mazy and Longaretti (2022b), we could compute the average of the mean absolute error of each Monte Carlo simulations. This calculation gives an indication of the bias in an integrated way over all pixels. However, in the context of a bias identification process, it is interesting to characterize the deviation from the expected probability in a more precise way. We therefore show the pixel distribution probability $\rho(\boldsymbol{z}|u,v)$ itself, averaged over all Monte Carlo simulations.

5.3. Setting test problems of increasing complexity

Detecting an allocation bias in a real case study is difficult. The same point applies to artificial case studies whose complexity in terms of number of transitions and explanatory variables is close to a real case. Indeed, several biases of different origin may overlap, influence the result or even interact with each other. Moreover, it would be difficult to identify precisely the origin of a bias when the complexity of the case study prevents us from understanding *a posteriori* the behavior of the allocation algorithm.

Because of this, we focus on a series simple case studies of increasing complexity to perform this analysis. This allows us to detect errors and biases one by one and especially to identify their origin. To this effect, we have chosen to create test problems in graded contexts as follows:

1. Mono-pixel patches and single transition setting.
2. Mono-pixel patches and multiple transitions setting.
3. Multi-pixel patches and single transition setting.
4. Multi-pixel patches and multiple transitions setting.

This progressive method does not ensure that all errors and allocation biases will be captured, but it did help us to identify quite a few, because of its systematic nature. Also, it illustrates why by comparison our uSAM and uPAM method are error- and bias-free. A series of controlled settings implementing the above progression is described just below. In the present article, the last setting (item 4 in the above list) did not bring out any new result, but we keep it

[14]Dinamica EGO and Idrisi LCM provide such a functionality although this requires a non-standard use of Idrisi LCM, whereas CLUMondo does not, so that the allocation procedure of CLUMondo has not been analyzed independently of the calibration one (Mazy and Longaretti, 2022b).

in this list as this may reflect more limitations encountered with the tested algorithms than an actual absence of new bias in such a context.

These settings are quite far from what a user is likely to encounter in actual studies. They have been chosen in order to make more apparent the various sources of biases.

5.3.1. Mono-pixel patches and single transition setting

We set up a disk of 300 pixels of radius and spatially uniform initial LUC state u (i.e., $\#\boldsymbol{J} \gtrsim 280000$). This represents the initial LUC map of our test problem. The only explanatory variable z is the distance in pixel unit to the center of the disk, *i.e.* the radius. By construction:

$$\rho(z|u) = 2\pi z \#\boldsymbol{J}^{-1} \tag{13}$$

We define a single transition $u \to v$. The probability distribution $\rho(z|u,v)$ is a uni-variate normal law of z, with mean value $z_m = 150$ and standard deviation $\sigma_z = 40$:

$$\rho(z|u,v) = \frac{1}{(2\pi\sigma_z^2)^{1/2}} \exp\left(\frac{(z-z_m)^2}{2\sigma_z^2}\right). \tag{14}$$

We set $P(v|u) = 0.02$. We generate mono-pixel patches with the uSAM allocation method. From the analysis of Appendix I, with a confidence interval of 0.01 for 95% of the distribution, the required number of Monte Carlo simulations is 273.

5.3.2. Mono-pixel patches and multiple transitions setting

We set up a rectangle of 1000×250 pixels and spatially uniform initial LUC state u as initial LUC map. The only explanatory variable z is equal to the distance in pixel unit to the left side of the rectangle. By construction, $\rho(z|u) = N_y^{-1}$ where N_y is the number of pixels in a row, *i.e.* 250.

We define two transitions: $u \to v_1$ and $u \to v_2$. The probability distributions $\rho(z|u,v_1)$ and $\rho(z|u,v_2)$ are defined as piecewise functions:

$$\rho(z|u,v_1) = \begin{cases} \left(\frac{2}{5}N_y\right)^{-1} & \text{if } \frac{1}{5}N_y \le z < \frac{3}{5}N_y \\ 0 & \text{else} \end{cases}, \tag{15}$$

$$\rho(v_2|u,z) = \begin{cases} \left(\frac{2}{5}N_y\right)^{-1} & \text{if } \frac{2}{5}N_y \le z < \frac{4}{5}N_y \\ 0 & \text{else} \end{cases}, \tag{16}$$

With this setting, some pixels may undergo both transitions with the same probability, some may only change state to one final state, whereas others have no probability of state change. We set $P(v|u) = 0.05$ for both v_1 and v_2; this is relatively large but required to observe the multiple transition bias (section 6.2) over a single time step. Mono-pixel patches are generated with the uSAM allocation method. From appendix I, with a confidence interval of 0.01 for 95% of the distribution, the required number of Monte Carlo simulations is equal to 420.

5.3.3. Multi-pixel patches and single transition setting

The synthetic data is exactly the same as the one used in section 5.3.1, transition probability included. The only change concerns the patch size expectation. Dinamica EGO's patcher provides only one way to characterize patch sizes which are drawn from a log-normal distribution. *De facto*, we used the same distribution in this case in order to analyze Dinamica EGO's results. We set the log-normal mean to 50 and its variance to 5. Concerning the patch design process, we chose to construct circular patches in order to focus on possible differences between the pre- and post-allocation patch size and $\rho(z|u,v)$ distributions.

Details about the relation between Dinamica EGO's parameters and the log-normal probability law are given in appendix F.3. Our own patch construction algorithm is described in Appendix J.

6. Analysis of conceptual errors and biases in LCM and Dinamica EGO

We identify errors and biases by comparing several allocation methods:

- Dinamica EGO 4.0.10 *Landscape Patcher* block (DE),
- IDRISI Selva Land Change Modeler (LCM),
- Our unbiased Simple Allocation Method (uSAM) developed for this study and described in section 4.1 (used for mono-pixel patches).
- Our unbiased Patches Allocation Method (uPAM) developed for this study and described in section 4.2 (used for multi-pixel patches).

The uSAM and uPAM methods provide controlled points of comparison, as these have been designed to be error- and bias-free. The fact that they are bias-free is proved in Appendices C.2 for uSAM and E for uPAM.

We have reprogrammed the part of Dinamica EGO and the complete LCM allocation methods for comparison with the original software, in order to have confidence of our understanding of their workings. This has also allowed us to identify the origin of the sometimes subtle problems we point out. Dinamica EGO and LCM allow us to bypass the calibration step and to directly fill in a transition probability map as the input of the allocation module as discussed in section 5.1. Although CLUMondo was evaluated in our analysis of calibration issues (Mazy and Longaretti, 2022b), this model is not discussed in this section for reasons explained in the conclusion of this paper (section 8).

6.1. Mono-pixel patch and single transition: pruning bias

The pruning bias is revealed by the test problem described in section 5.3.1. This bias is certainly the most important identified in this work. It is present both in Dinamica EGO and LCM, although they operate in different ways, and although LCM developers do not use the term of pruning (but apply the concept in a maximal way their allocation algorithm). This bias is not exhibited by design by the uSAM and uPAM algorithms because no pruning is performed. Pruning can be avoided in Dinamica EGO but not in LCM.

6.1.1. Analysis

Let us first briefly describe the notion of pruning. In principle, and without making any specific assumption, the allocation of pivot-cells to final states is a two-step process at any time step t_s in a simulation, as described in section 3.1. However, this procedure tests all pixels, and although theoretically correct, this could be inefficient. Indeed, most random draws of pixels are performed uselessly as most pixels will not change state, so that most of the computational time is in effect spent on checking that no transition occurs. There are a priori two ways to deal with this problem: design computationally efficient algorithms to counterbalance this drawback or design a way to reduce the pool of pixels tested for transition. Ihe simplest way to implement this second option is to rely on the fact that the transition from u to v is more or less probable depending on $\boldsymbol{z}$ and keeping only the most probable pixels for further allocation testing, i.e., by truncating the probability distribution. The resulting pre-selection is called pruning in Dinamica EGO, and adopted in both Dinamica EGO and LCM.

In this logic, the two-step allocation method described above becomes a three-step one, the first two forming sub-steps of the sampling design:

1. Pruning (pre-selection) of an ensemble of appropriate pixels for transitions from u to other states v.
2. Random selection of a pixel in this pruned ensemble — if the pruned ensemble is larger than the number of pixels expected to undergo a transition $\#J\,P(v|u)$.
3. Allocation of a final state to this pixel.

Both Dinamica EGO and LCM implement pruning by selecting the pixels most likely to undergo a transition during the pre-selection step. However, they do not agree on what "most likely" means in this statement. They also implement widely different allocation procedures. Dinamica EGO's pruning approach truncates the probability density function of the transition $u \to v$ by keeping only the pixels of highest probability $P(v|u,\boldsymbol{z})$. By construction, the bias is indeed significant for high levels of pruning as will become apparent in the illustrative example discussed next. Truncating at a fixed number of times of the quantity of change (the strategy adopted by Dinamica EGO) does not ensure that the number of pixels pre-selected in this way does encompass most of the probability distribution. A better strategy would

have been to truncate at some given fraction of the cumulated distribution (e.g., 90%). This would have given some control on the magnitude of the resulting bias.

LCM is even more radical. It only selects the exact number of pixel with the highest probability of $\rho(z|u,v)$ to achieve the allocation. The post-allocation probability density is strongly biased. The pruning procedures of Dinamica EGO and LCM are described in Appendices F and G and analyzed in more detail in Appendix H.

As previously stated, one may also consider more computationally efficient algorithms to solve the problem of processing large numbers of pixels (e.g., tens of millions). The Multinomial Sampling Test described in appendix C makes use of a unique multinomial test to allocate a final states to pixels, whatever the number of different final states under consideration. It is sufficiently efficient and runs sufficiently fast on present fdat laptops or desktops to avoid pruning altogether, even in large case studies (number of pixels $\sim 10^{10}$). As a result, pruning can be altogether avoided to run large LUCC problems, and is not used in our uSAM and uPAM methods.

It is also possible in principle to elaborate pruning algorithms not relying on truncation of the relevant probability distribution. By design such pruning algorithms are unbiased. We have not pursued this line of research though, as our preliminary algorithms using this form of pruning did not display any significant computational speed-up with respect to the ones we use and which avoid pruning.

6.1.2. Illustration

The post-allocation comparison method (sections 5.1 and 5.2) is applied to Dinamica EGO (DE), LCM[15] and uSAM. For each models, the post-allocation probabilities have been averaged over 273 allocations, as indicated earlier. The expected number of transited pixels is given by Eq. (6), i.e., $P(v|u)\#\boldsymbol{J} = 5671$. The Monte-Carlo approach is applied to each model except LCM which is not a random allocation and always returns the same allocated map. Dinamica EGO enforces the number of allocated pixels and its algorithm ensures this exact amount of 5671 transited pixels. uSAM has a different approach : the final number of pixel is a statistical variable[16] whose mean value is $P(v|u)\#\boldsymbol{J}$. The 273 simulations have given in averaged 5669.34 transited pixels with a standard deviation of 70 pixels (1.2% of dispersion).

Fig. 3 represents the returned allocated maps and Fig. 4 is a graph of the post-allocation distribution $P(v|u,\boldsymbol{z})$.

The post-allocation results of each model are compared to the expected transition probability distribution. We make here some comments on the outcome of this comparison:

uSAM The uSAM allocation algorithm output is equal to the expected distribution (Fig. 4), and this is reflected by the spatial distribution of pixels having changed state displayed on Fig. 3a.

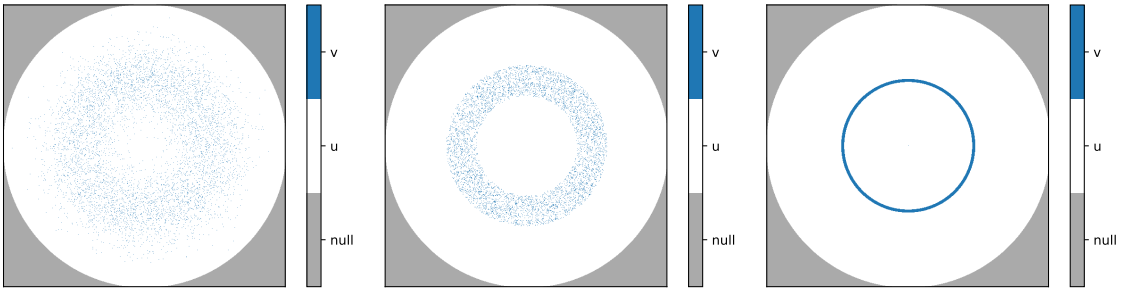
DE Three different pruning factors have been tested for Dinamica EGO : $F = 10$, $F = 20$ and $F \geq 50$. Dinamica EGO orders pixels in decreasing order of $P(v|u,\boldsymbol{z})$ and selects a number of pixels equal to $F\,P(v|u)\#\boldsymbol{J}$ in this decreasing order. Consequently Dinamica EGO omits a significant part of the actual pixel probability distribution in z (Fig. 4). Dinamica EGO's pruning method confines transited pixels to a ring of limited radial extent (Fig. 3b). The higher the pruning factor, the lower the bias of the output. In this test, $P(v|u)^{-1} = 50$ was chosen. Thus, $F \geq 50$ corresponds to no-pruning as illustrated on Fig. 4. The asymmetry observed for lower F values with respect to the maximum of $\rho(z|u,v)$ is a consequence of the fact that Dinamica EGO prunes by truncation the $P(v|u,\boldsymbol{z})$ probability distribution instead of $\rho(\boldsymbol{z}|u,v)$ itself as pointed out in appendix H.

LCM The distribution returned by LCM is highly biased (Figs. 4 and 3c). The asymmetry on Fig. 4 is similar to the one produced by Dinamica EGO, but exaggerated as in effect our setting implies that LCM assumes $F = 1$ in the language of Dinamica EGO. Indeed, as for the other two methods, we have used the probability of transition $P(v|u,\boldsymbol{z})$ as input of the allocation method. The result would probably have been worse, had we used $\rho(\boldsymbol{z}|u,v)$, i.e., the actual distribution truncated by LCM, as discussed in Appendix H.

The uSAM and uPAM (not illustrated here) algoritms return a better post-allocation probability density function but test a larger volume of pixels because no pruning is performed. This is possible due to the high efficiency of

[15] Even if Dinamica EGO and LCM have been successfully recoded and implemented in our LUCC modeling software CLUMPY, results presented in this paper are real outputs of the software.

[16] It is possible to design a monopixel patch allocation algorithm that enforces exactly the expected number of pixel state changes at each run, see Appendix D. But this is not necessary, as both the uSAM algorithm and this alternative algorithm are unbiased and produce the same expected number of pixel state change on average over all Monte Carlo runs, and within the chosen tolerance for these runs.



(a) uSAM (b) DE $F = 10$ (c) LCM

Figure 3: Pruning bias illustrated through resulting LUC maps of one allocation time step for 3 allocation methods : (a) uSAM (unbiased distribution); (b) Dinamica EGO (DE) with $F = 10$; (c) Idrisi LCM.

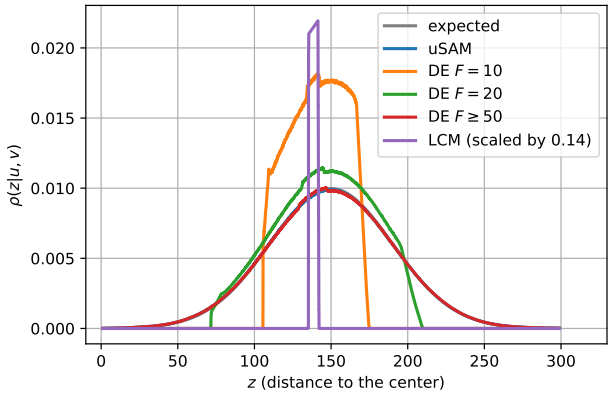


Figure 4: Pruning bias illustrated through the $\rho(z|u,v)$ post-allocation distribution averaged over 273 Monte Carlo draws compared to the pre-allocation distribution for several allocation methods: uSAM, Dinamica EGO (DE) with pruning parameters $F \in \{10, 20, 50\}$ and Idrisi LCM scaled by 0.14 (in order to fit in the graph). Note that grey (expected), blue (uSAM) and red (DE $F \geq 50$) lines are nearly exactly superposed.

modern computers and random number generators, and by optimized programming, even in an interpreted language like Python[17].

6.1.3. Multi-step vs single step induced bias

The pruning bias has yet another unexpected consequence. It relates to the fact that the post-allocation distribution $P(v|u,\boldsymbol{z})$ for a given time span depends on the number of steps chosen to perform such a simulation. This is not acceptable, as this number of steps is largely arbitrary: simulation results cannot depend on the (arbitrary) time step of the simulation.

To illustrate this point we performed the same simulation as in the previous section but we have divided the time step by a factor 3 and therefore ran the simulation for 3 time steps. The chosen number of Monte Carlo simulation is still 273. The transition probability map is adjusted in order to have the same expected distribution after 3 steps than a single step in the previous setting (this mostly consists in adjusting the total quantity of change per time step).

The results of this test are the following. uSAM provides a satisfying post-allocation $P(v|u,\boldsymbol{z})$ which is exactly the expected one, and the same as previously. On the contrary, a significant additional bias is obtained with Dinamica EGO, for the following reason. By construction, if the same pruning factor F is kept (as most users do when using Dinamica EGO), the multi-step process selects a lower number of pixels at each sub-time step compared to the original run. This mechanically increases the bias due to pruning. In order to reach the same level of bias, the pruning factor would have to be multiplied by the number of sub-time steps. Indeed, in this example, the bias level obtained for a

[17] Quite a few Python libraries run in C, and are used in our algorithms, which explains this efficiency.

single step allocation with $F = 10$ (Fig. 4) is the same as for a 3 sub-steps allocation with $F = 10 \times 3 = 30$.

This consistency check (independence of the obtained allocation statistical results on the chosen time-step) does lead to the same, extremely biased output for LCM.

6.2. Mono-pixel patch and multiple transitions: multiple transition bias

The test problem of 5.3.2 reveals a multi-transition bias. This bias can appear when there is competition on a pixel for several final states that all have a non-zero probability of occurring. If the allocation algorithm is not specifically designed to be bias-free, the pixels that can change state to several final states are *in fine* under-allocated with respect to their transition probability. This bias is ultimately rooted in the fact that pixels must be removed from the pool in the course of an allocation method without replacement.

6.2.1. Analysis

The description of Dinamica EGO's allocation method in appendix F allows us to understand the multi-transition bias displayed by this software. If we assume that unselected pixels are kept in the sample instead of being discarded, the pixels that can undergo two state transitions are put back in the pool after the first rejection test twice less often than those that can only undergo one transition. So their relative probability of being drawn later also decreases while the allocation probability remains unchanged (but this relative probability decreases by much less thab a factor of 2, since the pruning factor makes the transited pixels only a fraction of those removed from the sample). Thus, as the allocation progresses, the pixels corresponding to a single final state are drawn relatively more often with an unchanged allocation probability. In fine, pixels whose possible final state is unique are thus over-represented compared to the pixels wit multiple possible final states. By construction, this problem does not arise when only one final state exists.

LCM, on the other hand, uses an overly simplistic allocation method (see Appendix G) resulting in very large biases inhibiting the observation of the multiple transition bias.

A method which considers independently all pixels and all final states at once such as uSAM does not present this kind of bias. The uPAM method recalculates between each patch the transition probability by updating the quantity $P(\boldsymbol{z}|u)$ to avoid this source of bias.

6.2.2. Illustration

The post-allocation comparison method (sections 5.1 and 5.2) is applied to Dinamica EGO, LCM and uSAM. For each models, the post-allocation probabilities have been averaged over 420 runs of the same allocation step, as discussed earlier.

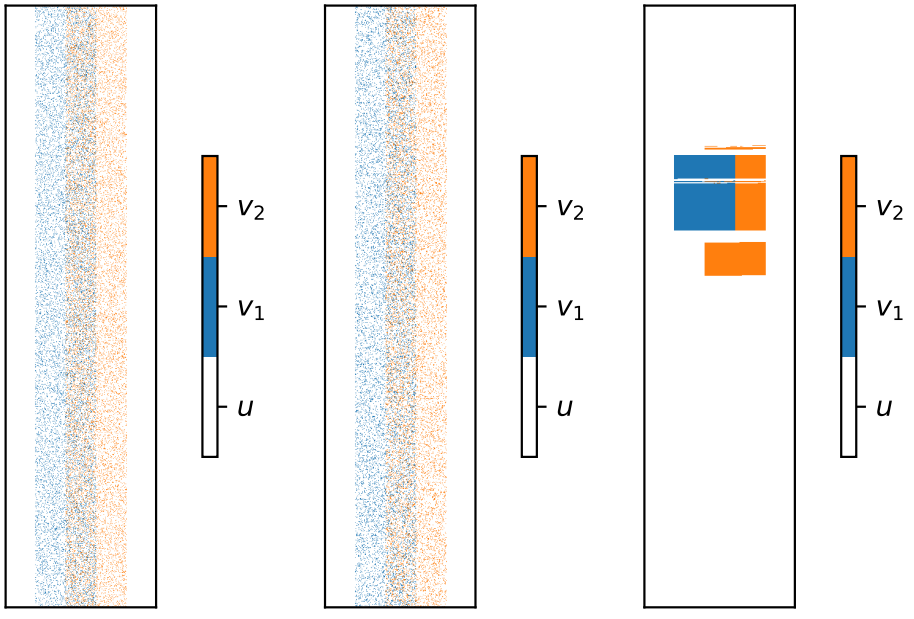
The results obtained can be summarized as follows:

uSAM The uSAM algorithm is bias-free for both transitions by design, as checked on Fig. 6a. The resulting LUC map is corresponds to the expected spatial distribution (Fig. 5a).

Dinamica EGO Dinamica EGO's pruning factor is set to $F = P(v|u)^{-1} = 20$ which implies that all pixels $P(v|u, \boldsymbol{z}) > 0$ are selected. This set up excludes pruning as a potential source of bias. As described above, the probability to draw a pixel with two possible final states decreases as tje allocation proceeds. Fig. 6b illustrates this bias, whereas it is not visible to the naked eye on the associated map (Fig. 5b). This bias is cumulative: it increases with the number of time steps in a simulation. It may therefore affect real case studies global simulations to a more significant level than visible here, a point that can only be ascertained through dedicated analysis in more realistic settings.

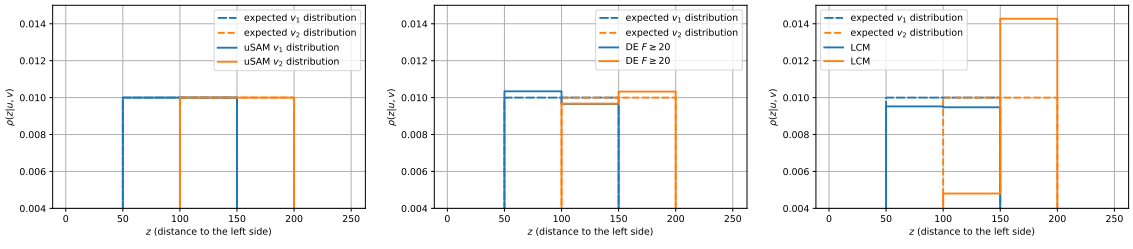
Idrisi LCM LCM pruning method described in Appendix G favors pixels with high values of $\rho(\boldsymbol{z}|u, v)$. However, since the transition probabilities are equal, the algorithm struggles to allocate pixels in a satisfying way. Indeed, as illustrated in Fig. 5c, an arbitrary set of pixel is allocated to v_1 and the transition to v_2 is accomplished in a second step. Also, the allocation algorithm breaks the spatial symmetry of the distribution. This follows because probabilities being uniform, the algorithm makes arbitrary selection choices, possibly seeded by the machine precision; this feature is maximized by our artificial setting, but is nevertheless a weakness of the allocation algorithm. As a result, Fig.6c shows a very biased distribution of $\rho(\boldsymbol{z}|u, v_2)$.

Again, only the uSAM method achieves an unbiased allocation on this multi-transition case study. The uPAM method, not shown here, has the same behavior.



(a) uSAM (b) DE ($F = 21$) (c) LCM

Figure 5: Multi-transition bias illustrated through resulting LUC maps of one allocation time step for 3 allocation methods: (a) uSAM; (b) Dinamica EGO (DE) with $F = 21$; (c) LCM. As the source of bias is small for Dinamica EGO, it is not apparent on this map, but is visible on the probability distributions shown on the next figure. The source of bias is large for LCM.



(a) uSAM (b) DE ($F = 21$) (c) LCM

Figure 6: Multiple transition bias illustrated through the $P(v|u,z)$ post-allocation distribution averaged over 420 Monte Carlo draws compared to the pre-allocation distribution for 3 allocations methods: (a) simple unbiased allocation method; (b) Dinamica EGO (DE) with $F = 21$; (c) LCM.

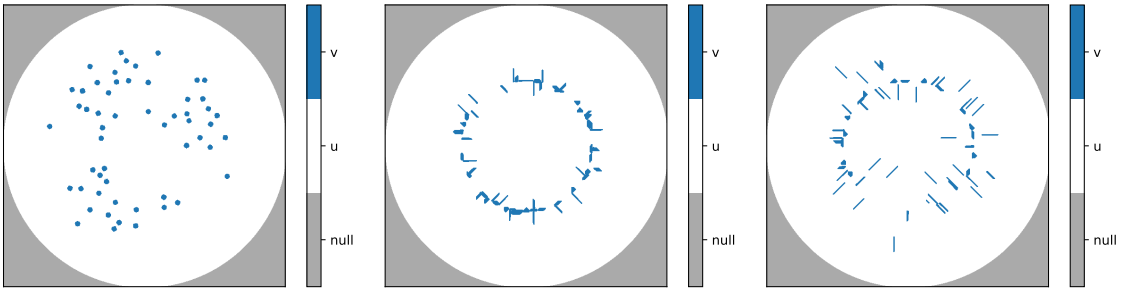
6.3. Multi-pixel patch and single transition: patch merging bias

The test problem of 5.3.3 reveals a bias that is inherent to the generation of multi-pixel patches, by producing spurious patch mergings (patch merging is defined in section 2.3).

6.3.1. Analysis

During the patch construction process, two patches of the same transition can merge and constitute together a single patch whose total size is approximately the sum of the two sizes. More than two patches can also merge. This produces an over-representation of the associated patch sizes compared to the pre-allocation size probability distribution. This constitutes a bias (see section 3.2.)

This bias can be avoided by checking during patch construction if mergings occur, as described in Appendix J.



(a) uPAM (b) Dinamica EGO ($F = 10$) (c) Dinamica EGO ($F = 50$)

Figure 7: Patch size bias illustrated through resulting LUC maps for 2 allocation methods : (a) uPAM (no bias); (b) Dinamica EGO with $F = 10$; (c) Dinamica EGO with $F = 50$ (no pruning). The patch merging biase results in spurious elongations and exxagerated sizes of the produced patches.

6.3.2. Illustration

The post-allocation comparison method (sections 5.1 and 5.2) is applied to Dinamica EGO and uPAM. LCM is excluded from this study because it does not provide any patch construction method due to its extreme form of pruning. But we have seen in previous subsections that LCM's way of producing contiguous areas of change is intrinsically biased; the problem examined here is of a more complex nature, and is inaccessible to LCM. For each model, the post-allocation probabilities are averaged over 273 repetitions of the same allocation time step.

Post-allocation results are compared to the expected probability distribution, leading to the following results:

uPAM The uPAM algorithm uses a patch construction process which avoids patch merging (see Appendix J). It returns both the expected $\rho(z|u,v)$ distribution (Fig. 8a) and the expected $\rho(\sigma|u,v)$ distribution (Fig. 8b)[18].

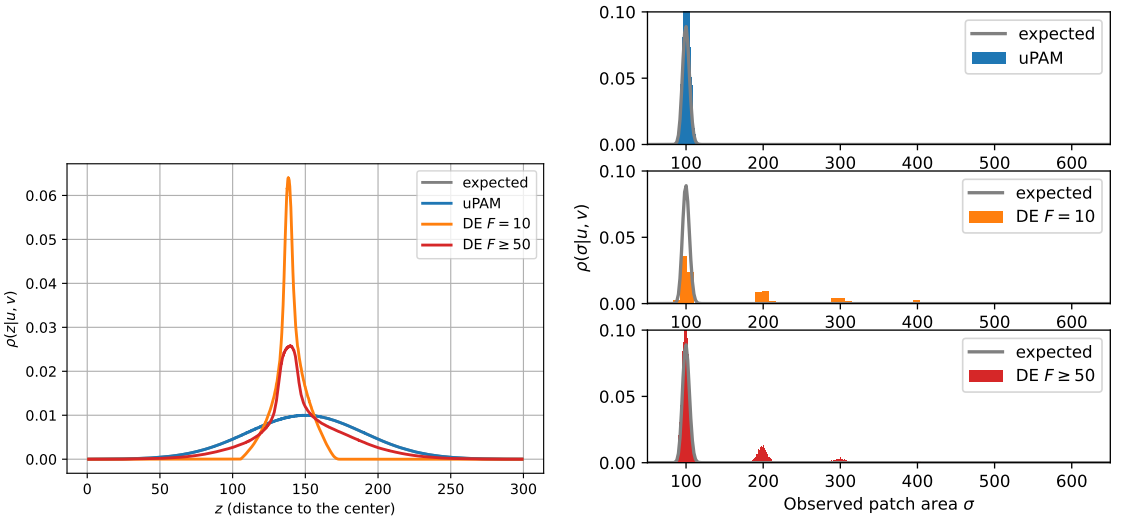
Dinamica EGO Dinamica EGO's behavior is displayed for two different pruning factors: $F = 10$ which inevitably produces a pruning bias (see section 6.1) and $F = 50$ which corresponds to an absence of pruning (all pixels are selected for the allocation process). The patch size distribution presents some small spurious replications of the original distribution at patch sizes that are multiples of the original mean size (Fig. 8b. These are produced by patch merging, as described above. More precisely, double ($s \approx 200$), triple ($s \approx 300$) and even quadruple patches are produced for $F = 10$ and are also noticeable on the resulting LUC maps (Fig. 7b and 7c). A small pruning factor reduces the number of pruned pixels and therefore increases merging occurrences. Moreover, the post-allocation distribution of $\rho(z|u,v)$ is quite different from the one expected one, and this even without pruning for $F = 50$ (Fig. 8a). This last point is rather surprising, but we have not been able to pinpoint the origin of this behavior.

Again, the uPAM algorithm is free of bias compared to Dinamica EGO with respect to this particular potential source of bias. Note also that the patches formed by Dinamica EGO are not circular. The transition probability plays a predominant role in the construction of the Dinamica EGO patch, to the detriment of the elongation distribution which is highly distorted here with respect to the imposed choice (Fig. 7c). A related but different observation is also made in Appendix F.3.2.

6.4. Multi-pixel patch and multiple transitions

We do not present any simulation results for this setting. We have checked that the previous biases all occur in such a more realistic context. However, we have not yet been able to identify a possible new source of bias in the test problems we have simulated so far.

[18] A very slight discrepancy exists for higher values of z, which is not visible on the figure. This is due to the fact that, although the seed pixel has been chosen correctly, the circular patch built around it does not take into account the explanatory variable distribution (distance to the center). Selected pixels are slightly more numerous for higher z than for lower ones. This phenomenon is negligible in actual case studies because the transition probability is taken into account by our patch construction method (see Appendix J) and because of the relatively small extension of the patches.



(a) $\rho(\boldsymbol{z}|u, v)$ (b) Histograms of observed sizes

Figure 8: Patch size bias illustrated through (a) the transition probability $\rho(\boldsymbol{z}|u, v)$, and (b) the size frequency histogram post-allocation distributions averaged over 273 Monte Carlo draws compared to the pre-allocation distributions for 2 allocations methods: uPAM and Dinamica EGO with $F = 10$ and $F = 50$ (no pruning). On both subfigures, the expected result and the outcome of uPAM are nearly exactly superposed. Dinamica EGO produces spurious patches at twice (two patches merging), three times (three patches merging) and even four times the mean size of the actual size distribution (chosen narrow enough in order to make the phenomenon more obvious).

7. Discussion

Except for the pruning bias, the bias levels associated to the various sources of bias discussed in this section seem moderate at first glance. However, one must keep in mind that all biases identified here are cumulative, so that their effect will become much more noticeable over the course of a whole simulation than it is for a single time step. Because of this, even apparently small biases may contribute to the difference of results between different modeling environment for the same case study and the same data.

In this work, we have ignored discrete explanatory variables and have also used a single continuous ones. There is no doubt that the sources of bias found are present as well for any mix of discrete and continuous variables. Using more than one variable would definitely be more realistic, but would make the setting more complex, and the display of the bias in explanatory variable space would be more complex as well. Furthermore, the existence of the biases we have shown is not dependent on the type or number of variables used. We therefore aimed at the simplest configuration: a single continuous explanatory variable.

Formally, and introducing $\boldsymbol{\gamma}$ to designate a t-uple of discrete explanatory variables, the generalization to an arbitrary mix of discrete and continuous variables is operated in the expressions given in this article by substituting $\boldsymbol{z}, \boldsymbol{\gamma}$ to $\boldsymbol{z}$ everywhere, and including sums over γ on top of integrals of $\boldsymbol{z}$ wherever appropriate.

A side issue mentioned a few times in the course of this paper and which has a major role in section 2.3 is that pattern-based LUCC modeling requires small enough time steps because small fractional total amounts of transitions are needed for the overall procedure to be self-consistent. Validation based on spatial accuracy of allocation is difficult for small amounts of change as many statistically equivalent sites are available for producing required LUC changes at each simulation time step; this point is quite clear, e.g., on Fig. 4 where qualitative statistical similarities or differences are immediately spotted. In fact, expecting spatial accuracy in such a context amounts to requiring a high level of predictability, a somewhat unrealistic expectation in a single time-step for a modeling approach that is statistical by design. Instead, statistical accuracy should be expected.

Although, e.g., Pontius (2002) has proposed a number of methods to cover the position/quantity accuracy spectrum, these are not adapted in light of the preceding remarks for our specific purposes. Indeed, we have explained in section 3.3 that although improving the spatial accuracy of projections is an important goal in itself (and one requiring

an improvement of the understanding of the causes driving LUC changes), one must find ways to evaluate the correctness of calibration and allocation methods independently of spatial accuracy, data quality or possible errors in model set-up (see section 2 of Mazy and Longaretti 2022b). The criteria of section 3.2 constitute a relevant and complete set of validation requirements in this respect. For example, if a validation map is available, comparing the projected and existing probability distributions (once a correct allocation method is available and calibration has been adequately performed) may provide the modeler with a useful tool to track either errors in explanatory variables identification or in scenario formulation, if calibration data are precise enough.

To summarize this very important point, a major feature of our process of error and bias identification is that the comparison is performed on the pre- and post-allocation probability distributions in explanatory variable space and not physical space, contrarily to the comparison and validation strategies widely used in the field, but we stress again that this conclusion is purpose-dependent. This point is also hopefully made clear through the test problems of section 6.

8. Conclusion

The present paper provides another step towards a systematic reinvestigation of pattern-based LUCC modeling from a mathematical and statistical point of view, following our analysis of calibration methods (Mazy and Longaretti, 2022b). This endeavor is motivated by the fact that the discrepancies pointed out in the literature (notably in Mas et al. 2014) for the projections obtained from various models for the same data and scenarios can be ascribed to differing modeling and algorithmic choices across the available modeling environments. However, a significant number of these choices cannot be freely made by the modeler as they are constrained by probability theory, and the algorithms used must be designed not to violate probabilistic and statistical constraints. Our objective is precisely to pinpoint where such constraints have been ignored or not correctly implemented in existing modeling strategies, in order to provide alternative conceptually correct, quantitatively precise and computationally efficient algorithms for pattern-based LUCC modeling. Such algorithms are in turn needed to achieve our primary purpose (identifying and eventually removing the sources of difference of behavior on the same problem and same data).

Such an undertaking should eventually encompass all the aspects of the problem, namely, calibration, allocation, and formal and algorithmic validation (which, as pointed out earlier unavoidably differs from usual validation methods in LUCC modeling). Furthermore, another overarching element of pattern-based LUCC modeling strategies should also be examined more closely, namely, the Markovian, discrete time formulation of LUCC simulations. The present work focuses on allocation.

Our analysis is designed to bring out the requirements that any modeling framework based on non-parametric probability distributions must satisfy. In particular, allocation methods are characterized by two steps: sampling design and allocation procedure (section 3.1). This two step characterization is motivated by concepts in sampling theory, where sampling designs without replacement (the ones of interest for LUCC modeling) are well-known to be difficult to formalize and implement. This approach constitutes the theoretical backbone of the present paper.

Unbiased allocation methods are defined in section 4 for both mono-pixel (unbiased Simple Allocation Method – uSAM) and multi-pixel patches (unbiased Patches Allocation Method – uPAM). In addition, Appendix J presents our patch construction method.

The study of existing allocation methods was motivated in particular by the need to identify sources of error and differences of outcome observed *a posteriori* on different software. This is possible only if one has a criterion allowing us to pinpoint errors and biases, and *our bias criterion, based on a necessary consistency requirement (section 3.2), is instrumental* in this respect. Furthermore, we propose a rather systematic method of identification of allocation biases. The idea is to compare software outputs in test problems of increasing complexity in order to detect as many biases as possible and at the same time identify their origin. This method imposes *a priori* the relevant transition probabilities in controlled settings, which allows us to focus only on the allocation module.

We have tested our method of allocation bias identification on two non-parametric pattern-based LUCC modeling environments, Dinamica EGO and Idrisi LCM, and compared it to our own uSAM and uPAM algorithms. This process allowed us to identify quite a few sources of errors and biases, to pinpoint their origin in a precise way, and to check that our proposed algorithm are indeed error- and bias-free for any practical purpose. Note that most of the biases found in this way are generic, i.e., not specific to the controlled setting in which they have been identified and studied. Our major results — besides the conception of the uSAM and uPAM algorithms of section 4 — are that Dinamica EGO implements an incorrect pruning procedure and a somewhat biased allocation algorithm, a point that is made more

obvious when multiple transitions are present. Pruning can be avoided in Dinamica EGO, therefore avoiding its major source of bias; the other sources can be corrected by correcting its allocation algorithm. On the contrary Idrisi LCM is highly biased due to pruning (although the developers of the software do not use the term "pruning"), and its simple allocation procedure is theoretically incorrect; most LCM failures relate directly to its allocation procedure. Pruning cannot be avoided in LCM, as it is inherent to its allocation strategy. Another source of problem is that its pruning strategy is not applied to the formally relevant probability distribution for this purpose.

We feel that allocation in both modeling environments could be made error- and bias-free with reasonable additional work, this objective being easier to reach for Dinamica EGO than for LCM. To achieve this purpose, these modeling environments (and others, for that matter) should have access to error- and bias-free allocation methods that are also algorithmically efficient. We point out for now that although no pruning strategy is used in the algorithms we proposed in section 4, they are nevertheless computationally efficient.

CLUMondo (van Vliet et al., 2015) is another popular modeling software and we now discuss it briefly. CLUMondo (and the other members of the CLUE family of models) implements a very different modeling strategy, based on a single LUC map to determine the LUC probability distribution. Moreover, these probabilities are determined by logistic regression (parametric method), which greatly constrains the shape of the distribution, and this produces an important lack of precision in the calibration module (Mazy and Longaretti, 2022b). Allocation follows from the single map probability distribution. Pixels are selected for transition on two criteria: highest probability and a weight depending on the distance to pixels already in the final state of interest (no indication about the setting of this parameters are given however). In case of multi-transitions, the user can indicate a priority order for each. This very basic approach is by nature highly biased. This being said, we could not include CLUMondo in our benchmark comparison tests because this software does not allow the user to bypass the calibration step, whereas this is required by our protocol. Also, CLUMondo is generally very complicated to configure for testing outside conventional cases which makes it difficult to use in such a benchmark.

It is important to note that the methods developed in this paper are not intended for actual case studies. They are geared only towards assessing whether allocation algorithms are formally error- and bias-free. This assessment needs to be performed only once. The resulting errors and biases, if found should be corrected or the algorithm abandoned if the error or bias is deemed to important in actual case studies — and this depends to some extent of the specific objectives of the intended case study. Still, in order not to raise concerns about this for each case study, it is advisable to use only error- and bias- free algorithms.

Other sources of errors than the ones discussed in the present work are important in actual case studies, namely, possible errors in the identification of explanatory variables, and errors in the calibration data themselves, in particular typological misclassifications, which are not uncommon in remote sensing data. This last source of error is definitely important or even dominant, especially if pruning is avoided, or if a computationally efficient unbiased pruning strategy is available. However, quantifying precisely the effect of errors in the data requires to first be able to dispense with sources of calibration and allocation errors. Indeed, such a quantification can again be performed through the use of (preferably realistic) controlled settings. This would in particular allow us to quantify the relative importance of different sources of error in actual case studies, or to the very least elaborate and characterize a strategy to this effect. In this respect, the present work constitutes a first and unavoidable step towards such a goal, which is of more direct interest and importance to LUCC modelers. Such an endeavor will also give modelers a better grasp of the usefulness of the algorithms we have designed, and such a study would also be much more in line with their modeling practices.

Another source of concern comes from the intrinsic variability in results for a specific problem due to the wide spectrum of modeling methods (and not only in pattern-based LUCC models), only briefly mentioned at the beginning of the introduction. We believe — or maybe just hope — that this spread in obtained results could be significantly reduced if arbitrary choices were eliminated in a systematic way in these various alternative approaches, as was endeavored here for pattern-based LUCC modeling.

References

Agarwal, C., Green, G.L., Grove, M., Evans, T., Schweik, C., 2000. A review and assessment of land-use change models: dynamics of space, time and human choice. Gen. Tech. Rep. NE-297. Newton Square, PA: U.S. Department of Agriculture, Forest Service, Northeastern Research Station. 61 p. doi:10.2737/NE-GTR-297.

Alexander, P., Prestele, R., Verburg, P.H., Arneth, A., Baranzelli, C., Batista e Silva, F., Brown, C., Butler, A., Calvin, K., Dendoncker, N., Doelman, J.C., Dunford, R., Engström, K., Eitelberg, D., Fujimori, S., Harrison, P.A., Hasegawa, T., Havlik, P., Holzhauer, S., Humpenöder, F., Jacobs-Crisioni, C., Jain, A.K., Krisztin, T., Kyle, P., Lavalle, C., Lenton, T., Liu, J., Meiyappan, P., Popp, A., Powell, T., Sands, R.D., Schaldach, R.,

Stehfest, E., Steinbuks, J., Tabeau, A., van Meijl, H., Wise, M.A., Rounsevell, M.D.A., 2017. Assessing uncertainties in land cover projections. Global Change Biology 23, 767–781. doi:10.1111/gcb.13447.

van Asselen, S., Verburg, P.H., 2013. Land cover change or land-use intensification: simulating land system change with a global-scale land change model. Global Change Biology 19, 3648–3667. doi:10.1111/gcb.12331.

Bielecka, E., 2020. Gis spatial analysis modeling for land use change. a bibliometric analysis of the intellectual base and trends. Geosciences 10, 421. doi:10.3390/geosciences10110421.

Cochran, W.G., 1977. Sampling Techniques, 3rd Edition. Wiley.

Eastman, J.R., Jin, W., Kyem, P., Toledano, J., 1995. Raster Procedure for Multi-Criteria/Multi-Objective Decisions. Photogrammetric Engineering & Remote Sensing 61, 539–547.

García-Álvarez, D., Camacho Olmedo, M.T., Van Delden, H., Mas, J.F., Paegelow, M., 2022. Comparing the structural uncertainty and uncertainty management in four common land use cover change (lucc) model software packages. Environmental Modelling & Software 153, 105411. doi:https://doi.org/10.1016/j.envsoft.2022.105411.

Lambin, E.F., Geist, H. (Eds.), 2006. Land-Use and Land-Cover Change. Springer Berlin Heidelberg. doi:10.1007/3-540-32202-7.

Lohr, S.L., 2019. Sampling: Design and Analysis: Design and Analysis, 2nd edition. Chapman and Hall/CRC. doi:10.1201/9780429298899.

Longaretti, P.Y., Mazy, F.R., 2022. Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling. A maximum relevance / minimum redundancy selection procedure of explanatory variables. Submitted to Environmental Software and Modeling .

Mas, J., García Álvarez, D., Paegelow, M., Domínguez-Vera, R., Castillo-Santiago, M., 2022. Metrics Based on a Cross-Tabulation Matrix to Validate Land Use Cover Maps, in: Land Use Cover Datasets and Validation Tools. Validation Practices with QGIS. Springer, Cham, pp. 127–151. doi:10.1007/978-3-030-90998-7_8.

Mas, J.F., Kolb, M., Paegelow, M., Camacho Olmedo, M.T., Houet, T., 2014. Inductive pattern-based land use/cover change models: A comparison of four software packages. Environmental Modelling & Software 51, 94–111. doi:10.1016/j.envsoft.2013.09.010.

Mazy, F.R., Longaretti, P.Y., 2022a. A Formally Correct and Algorithmically Efficient LULC Change Model-building Environment, in: Proceedings of the 8th International Conference on Geographical Information Systems Theory, Applications and Management - GISTAM, SciTePress. pp. 25–36.

Mazy, F.R., Longaretti, P.Y., 2022b. Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling. An accurate and powerful calibration-estimation method based on Kernel density estimation. Environmental Software and Modeling .

Pontius, R.G., 2002. Statistical methods to partition effects of quantity and location during comparison of categorical maps at multiple resolutions. Photogrammetric Engineering & Remote Sensing 68, 1041–1049.

Prestele, R., Alexander, P., Rounsevell, M.D.A., Arneth, A., Calvin, K., Doelman, J., Eitelberg, D.A., Engström, K., Fujimori, S., Hasegawa, T., Havlik, P., Humpenöder, F., Jain, A.K., Krisztin, T., Kyle, P., Meiyappan, P., Popp, A., Sands, R.D., Schaldach, R., Schüngel, J., Stehfest, E., Tabeau, A., Van Meijl, H., van Vliet, J., Verburg, P.H., 2016. Hotspots of uncertainty in land-use and land-cover change projections: a global-scale model comparison. Global Change Biology 22, 3967–3983. doi:10.1111/gcb.13337.

Soares-Filho, B., Rodrigues, H., Follador, M., 2013. A hybrid analytical-heuristic method for calibrating land-use change models. Environmental Modelling & Software 43, 80–87. URL: https://www.sciencedirect.com/science/article/pii/S1364815213000236, doi:10.1016/j.envsoft.2013.01.010.

Soares-Filho, B.S., Coutinho Cerqueira, G., Lopes Pennachin, C., 2002. Dinamica — A stochastic cellular automata model designed to simulate the landscape dynamics in an Amazonian colonization frontier. Ecological Modelling 154, 217–235. doi:10.1016/S0304-3800(02)00059-5.

Verburg, P.H., Alexander, P., Evans, T., Magliocca, N.R., Malek, Z., Rounsevell, M.D.A., van Vliet, J., 2019. Beyond land cover change: towards a new generation of land use models. Current Opinion in Environmental Sustainability 38, 77–85. doi:10.1016/j.cosust.2019.05.002.

Verburg, P.H., Kok, K., Pontius, R.G., Veldkamp, A., 2006. Modeling Land-Use and Land-Cover Change, in: Lambin, E.F., Geist, H. (Eds.), Land-Use and Land-Cover Change. Global Change - The IGBP Series. Springer. Global Change - The IGBP Series, pp. 117–135. doi:10.1007/3-540-32202-7_5.

Verburg, P.H., de Koning, G.H.J., Kok, K., Veldkamp, A., Bouma, J., 1999. A spatial explicit allocation procedure for modelling the pattern of land use change based upon actual land use. Ecological Modelling 116, 45–61. doi:10.1016/S0304-3800(98)00156-2.

van Vliet, J., Malek, Z., Verbug, P., 2015. The CLUMondo land use change model, manual and exercises.

Appendices

A. Software

In this article we introduce our own environment for LUCC modeling, CLUMPY (Comprehensive Land Use [and cover] Model in PYthon). As indicated by the name, this is a Python package which provides a complete framework that includes our own calibration, estimation and allocation algorithms for statistical pattern-based LUCC modeling, as well as an optimized semi-automatic method of selection of explanatory variables. This open-source software is provided under a GNU public license and is freely available on the GitLab of our institution: https://gitlab.inria.fr/fmazy/clumpy/ (no username or password required). It is currently developed by François-Rémi Mazy and a beta version is being tested as part of the work presented here. A first stable version along with its documentation is planned for the fall of 2022 and a Graphical User Interface (GUI) will be also provided in the spring or summer of 2023. No specific hardware is required and CLUMPY aims to be user-friendly and usable on any computer, although it has only been tested on Linux OS at the time of writing. The software is mainly written in Python, but some critical

element	set	definition
$\#\boldsymbol{J}_{\boldsymbol{z},v}$, $\#\boldsymbol{J}^c_{\boldsymbol{z},v}$	$\mathbb{N}$	Number of pixels (resp. pivot-cells) in a small volume $\delta\boldsymbol{z}$ around $\boldsymbol{z}$ in explanatory variable space
s_j, δ^z_j, $\delta_{i,j}$	$\mathbb{N}$	Bivalued (0 or 1) variables defined for counting purposes
n, n_w, n_b	$\mathbb{N}$	Numbers of balls, white balls, black balls in an urn (Appendix D)
$n^{(p)}$, $n^{(p)}_w$, $n^{(p)}_b$	$\mathbb{N}$	Same as above at the pth draw (Appendix D)
N, N_v	$\mathbb{N}$	Number of balls in a generalized urn sampling problem (Appendix E)
$N^{(p)}$, $N^{(p)}_v$	$\mathbb{N}$	Same as above at the pth draw (Appendix E)
Δ, $\boldsymbol{z}_\Delta$, $\delta\boldsymbol{z}_\Delta$	not defined, $\mathbb{R}^d$	Bin in explanatory variable space of (small) volume $\delta\boldsymbol{z}_\Delta$ and with nearly constant explanatory variable value $\boldsymbol{z}_\Delta$
$N_{u,\Delta}$; N_u, N_v	$\mathbb{N}$	Number of pixels in bin Δ, in initial state u; number of these pixels in final state u, v
$p_{v,\Delta}$	$\mathbb{R}$	Shorthand notation for $P(v\|u,\boldsymbol{z}_\Delta)$
$E_X(Y)$	$\mathbb{R}$	Expectation value of random variable $Y(X)$ with respect to the probability distribution of variable X (sometimes, the distinction between the random variable Y and its possible values y is not made)

Table B1
Summary of the notations and definitions used in Appendices B, C, D and E.

elements such as the eKDE method have been written in Cython to improve computation efficiency. The whole program with its documentation weighs 477kb (zip file) at the time of writing.

B. Patch-related probability distributions

This Appendix introduces more formally patch probability distributions. It also discusses and gives some justification to the assumptions made in section 2.3.

This and the following three Appendices introduce a number of quantities that have not been defined earlier. These new quantities are rather numerous, and for convenience and reference, are collected in Table B1.

B.1. Patch probability distribution

In this work, patch distributions are characterized by their pivot-cell distribution in explanatory space, their size and their elongation (i.e., a minimal way to characterize patch shapes); the last two are defined in real, 2D space. As a matter of fact, patch characteristics do not seem to be a point of much focus in LUCC studies. Our assumptions about patch probability distributions combined to our patch construction method (Appendix J) produce patches with some degree of realism in their randomness.

On the formal level, let us introduce, for each initial and final LUC state pair (u, v), the patch pivot-cell explanatory variables set $\boldsymbol{z}^c$, and shape parameters T: $T = (\sigma, e)$, where σ is the patch size and e the patch elongation. The patch probability distribution can therefore be written $\rho^p(\boldsymbol{z}^c, \sigma, e|u, v)$. It is conditional to a given LUC state transition, as indicated by the notation. This distribution generalizes to patches the pivot-cell distribution $\rho^c(\boldsymbol{z}^c|u, v)$ introduced in section 2.3. Note that the distribution $\rho(\boldsymbol{z}|u)$ has no patch equivalent, as patches exist only for a given transition.

Elongation is formally defined and characterized in Appendix J.2. Our definition allows the user to calibrate the elongation distribution, if needed, because elongations are well-defined on any patch, and because our patch construction algorithm does enforce in a statistical way the chosen elongation. The size distribution can also be calibrated from calibration data. However, this calibration approach has apparently not yet been used in the LUCC community for

these two patch characteristics, and for the purpose of the present work, specific size and elongation distributions are assumed instead, for simplicity.

The present work also assumes that the three patch parameters $\boldsymbol{z}^c$, σ and e are statistically independent:

$$\rho^p(\boldsymbol{z}^c, \sigma, e|u, v) = \rho^c(\boldsymbol{z}^c|u, v) \times \rho(\sigma|u, v) \times \rho(e|u, v), \tag{B1}$$

where the p superscript is dropped from the distributions of size and elongation, for simplicity. There is no *a priori* reason for this assumption to hold in real case studies, but in our experience, this does seem to be at least approximately true. This independence assumption is adopted here for the sake of simplicity. It may be relaxed in the future if needed, and with appropriate calibration data. If this is the case, the allocation algorithms and associated proofs of absence of bias will need to be extended.

B.2. Identity of pivot-cells and pixels complete distributions

This is the last assumption made in section 2.3:

$$\rho^c(\boldsymbol{z}^c|u, v) = \rho(\boldsymbol{z}^c|u, v). \tag{B2}$$

This assumption is natural enough, but its conditions of validity are not obvious. There is however a sufficient condition under which it is satisfied. To see this, let us define a characteristic patch scale $L_p = E(\sigma)^{1/2}$, and introduce the smallest of the characteristic explanatory variables scale (in physical space), L_z. If $L_p \ll L_z$, or even $L_p \lesssim L_z$, the distributions $\rho(\boldsymbol{z}|u, v)$ and $\rho(\boldsymbol{z}|u)$ will be nearly uniform over a patch. Thus the probability density distribution of all pixels is constant over any patch, and this includes the pivot-cell, which is sufficient to justify the above assumption.

Let us now show this is a slightly more formal way. To this effect, let us introduce some small volume $\delta\boldsymbol{z}$ around some specific value $\boldsymbol{z}$ in explanatory variable space. Because continuous explanatory variables are function of spatial location, this small volume will implicity define some scale L_δ in physical space[19]. We choose $\delta\boldsymbol{z}$ such that $L_p \ll L_\delta \ll L_z$ (but the argument holds approximately even if these inequalities hold only approximatively, i.e., if $L_p \lesssim L_\delta \lesssim L_z$ instead). Let us also define $E(\#\boldsymbol{J}_{\boldsymbol{z},v})$ as the average number of pixels at $\boldsymbol{z}$ within $\delta\boldsymbol{z}$, for givne initial and final state u, v. By definition

$$E(\#\boldsymbol{J}_{\boldsymbol{z},v}) = E(\#\boldsymbol{J}_v) \int_{\delta\boldsymbol{z}} d\boldsymbol{z}\, \rho(\boldsymbol{z}|u, v) \simeq E(\#\boldsymbol{J}_v)\rho(\boldsymbol{z}|u, v)\delta\boldsymbol{z}. \tag{B3}$$

Because all transited pixels belong to a patch, the expected cumulative patch size[20] σ_T at $\boldsymbol{z}$ within $\delta\boldsymbol{z}$ is:

$$\sigma_T = E(\#\boldsymbol{J}_{\boldsymbol{z},v}) = E(\#\boldsymbol{J}_v)\rho(\boldsymbol{z}|u, v)\delta\boldsymbol{z}. \tag{B4}$$

On the other hand, by definition of $\rho^p(\boldsymbol{z}^c, \sigma, e|u, v)$ the same expected cumulative size is given by:

$$\sigma_T = E(\#\boldsymbol{J}^c_v) \int_{\delta\boldsymbol{z}} d\boldsymbol{z} d\sigma de\, \sigma\rho^p(\boldsymbol{z}^c, \sigma, e|u, v) \simeq E(\#\boldsymbol{J}^c_v)E(\sigma)\rho^c(\boldsymbol{z}|u, v)\delta\boldsymbol{z}. \tag{B5}$$

From Eq. (11), $E(\#\boldsymbol{J}_v) = E(\#\boldsymbol{J}^c_v)E(\sigma)$ and the comparison of Eqs. (B4) and (B5) leads to $\rho^c(\boldsymbol{z}|u, v) = \rho(\boldsymbol{z}|u, v)$, i.e., the required result.

C. Multinomial Sampling Test algorithm

This Appendix deals with monopixel patches. These are rarely if ever encountered in actual maps, but are a useful tool in the present work, either as a first step in multipixel patch design algorithms, or as a tool to identify sources of bias. Because only monopixel patches are dealt with here, the notations ρ^c and ρ, and $\boldsymbol{z}^c$ and $\boldsymbol{z}$ can be used interchangeably in the present Appendix.

[19] It may define several such scales, as $\boldsymbol{z}$ may be found at several spatial locations, in which case L_δ is the largest of the various scales associated to the various locations.

[20] i.e., the total patch area in the small explanatory variable space volume.

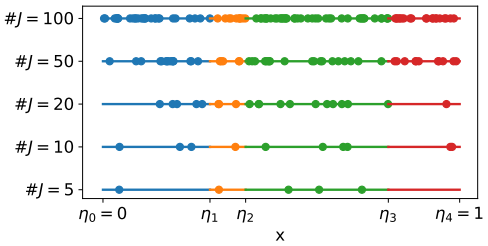


Figure C1: Outcome of the multinomial sampling test in a simple idealized setting with four possible final output states. These output states are sections of the unit interval. The length of these sections have been chosen arbitrarily. The distribution of random draws along the unit section are represented from bottom to top for an increasing number of draws (5, 10, 20, 50, 100 respectively).

C.1. Test description

For any pixel j, u^j and $\boldsymbol{z}^j$ are known as well as the transition probability $P(v|u^j, \boldsymbol{z}^j)$ for all v (Eq. 1). Let us define η_w as the cumulative sum[21] of $P(v|u^j, \boldsymbol{z}^j)$:

$$\forall\, w \in V, \quad \eta_w = \sum_{v \in V \,|\, v \leq w} P(v|u^j, \boldsymbol{z}^j). \tag{C1}$$

By convention, $\eta_{-1} = 0$ (there is no $w = -1$ state but this starting value is needed in the logic of the test). Then, a unique random float is sufficient to test simultaneously all possible transitions for this pixel. For any real random number ξ in $[0, 1[$, one can find an index w such that $\eta_{w-1} \leq \xi < \eta_w$. The final state $v = w$ in then affected to this pixel. This is a well-known test in probability theory, based on the fact that the random variable Y defined on the cumulative distribution function of a probability distribution $P(x)$ [i.e., $Y = P(x \leq X)$] has a uniform probability. By definition of a random draw, this test enforces Eq. (1) for each pixel, and the resulting distribution of N pixels will satisfy Eqs. (6) and (7) (see next subsection).

In plain words, we consider output states as sections of the unit interval with length equal to the various transitions probability; the total length is then equal to one (as the transition $u^j \to v = u^j$ is included in the process). Intuitively, it is clear that distributing pixels at random on this unit interval will produce relative number of pixels in each possible final state proportional to the length of each interval, on average (Fig. C1), and therefore comply with $P(v|u^j, \boldsymbol{z}^j)$, on average.

This test presents several advantages: all possible final states are tested at the same time, and the order of the sections in the unit interval is irrelevant. Algorithm 1 right below (Multinomial Sampling Test, hereafter MuST) provides a pseudo-code implementation of this allocation method.

Algorithm 1: Multinomial Sampling Test MuST

Data:
The set of all pixels indices $\boldsymbol{J}$,
The set of LUC states V,
$\{P(v|u^j, z^j)\}_{\forall j \in J,\, \forall v \in V}$, the transitions probabilities
Result: A new state is drawn for each pixel according to transitions probabilities

$\forall j \in J,\ \forall w \in V$, compute η_w according to (C1)
$\forall j \in J,\ \xi^j \leftarrow \text{RAND}(0, 1)$
foreach $v \in V$ sorted in the reverse order **do**
  $\forall j \in J \mid \xi^j < \eta_w,\ v^j = w$
return $\{v^j\}_{\forall j \in J}$

[21] For simplicity, we consider that the set of final states V is ordered and the values of its elements are consecutive. This set is identified to a series of integers $0, 1, \ldots$. By convention $w = 0$ is an unaffected state (pixels outside the map, or pixels with unknown state).

In this algorithm, line 2 attributes a random number to all pixels simultaneously[22]. Line 4 is a test of the embedded inequality, and the loop at lines 3-5 stops for the first value v (in decreasing order) that satisfies this inequality.

Note that the same algorithm can be used *mutatis mutandis* for any discrete probability distribution. Note also that this multinomial sampling test does also in effect realize a binomial sampling test for any given final state v: one just has to regroup all segments η_w with $w \neq v$ after the algorithm has run to achieve this purpose.

C.2. Statistical properties of the MUST algorithm

The fact that this algorithm is unbiased is ultimately rooted in the fairness of the random number generator used. Modern random number generators are highly robust in this respect[23]. In the proof of absence of bias of the MUST algorithm presented below, this property is represented by the assumption that each particular pixel trial faithfully represents a Bernoulli trial in the mathematical sense.

We can proceed with this proof by introducing another type of samples on the pixel population. They refer to the transition from u to v. Let us define first a random variable S_j with possible values s_j for each pixel j, such that $s_j = 1$ if the pixel has been selected in the sample to change state from u to v, and $s_j = 0$ otherwise; S_j is a Bernoulli variable associated to pixel j. The samples of interest here, $S_{u,v}$, do collect the values of s_j produced by the allocation method: $S_{u,v} = (s_1, s_2, ...)$.

By construction, the probability distribution of s_j is a Bernoulli probability distribution:

$$p_j = p(s_j) = P(v|u, z^j)^{s_j} \times \left[1 - P(v|u, z^j)\right]^{1-s_j}, \tag{C2}$$

and the probability of realizing a given sample $S_{u,v}$ is given by $\prod_j p_j$ (this fully specifies the sampling design for these new samples, according to our earlier definition). Indeed, if the random number generator is truly random, the result of the trial of pixel j produces the state v with probability p_j, by construction.

This allows us to calculate various statistical quantities on the samples $S_{u,v}$. This makes use in an intermediate step of the number $\#J\, g(\boldsymbol{z}, u) d\boldsymbol{z}$ of pixels in initial state u and explanatory variables $\boldsymbol{z}$ within some small (infinitesimal) volume $d\boldsymbol{z}$. By definition, $g(\boldsymbol{z}, u) = \rho(\boldsymbol{z}|u)$ (this quantity is known as the degeneracy factor in physics).

For example, the number of pixels in final state v in a given sample can be obtained from $\#J_v \equiv \sum_j s_j$, and the mean expected number of these pixels follows from

$$E(\#J_v) = E\Big(\sum_j S_j\Big) = \sum_{s_j, j} s_j p_j = \sum_j P(v|u, z^j) \tag{C3}$$

$$= \#J \int_z P(v|u) \times \frac{\rho(\boldsymbol{z}|u, v)}{\rho(\boldsymbol{z}|u)} \times g(\boldsymbol{z}, u) d\boldsymbol{z} \tag{C4}$$

$$= P(v|u)\#J \int_z \rho(z|u, v) d\boldsymbol{z} = P(v|u)\#J. \tag{C5}$$

The last equality recovers the expected result. This follows essentially from the transformation of a sum over j in an integral over $\boldsymbol{z}$.

Even though no such compact expression is available for the variance, it is useful to recall the result. Because all S_i are independent random variables, the variance of their sum, $V(\sum S_j) = \sum V(S_j)$, i.e.:

$$V\Big(\sum_j S_j\Big) = \sum_j P(v|u, z^j)\left[1 - P(v|u, z^j)\right] = \sum_j p_j(1 - p_j). \tag{C6}$$

In fact, this formulation allows us to recover much more, as the probability distribution of the $S_{u,v}$ samples is exactly known from Eq. (C2). For example, a slightly different form of the previous calculation gives us the expected probability distribution with respect to z for a given initial and final LUC state of the sample, which, in the mean, should be $\rho(z|u, v)$.

[22] RAND(0,1) stands for a random number generator in the interval [0,1]. It is advisable to use optimized libraries to generate such numbers for all pixels at once.

[23] The number of draws needs to be ridiculously large to achieve a noticeable level of non-randomness in the series of numbers generated in this way, and this non-randomness bears on correlations anyway, i.e., a property of no relevance to the present problem. In any case such large numbers are never reached — by far — in our tests.

To show this, let us define an indicator variable δ_j^z such that $\delta_j^z = 1$ when $z^j = z$ within some small range $\delta\boldsymbol{z}$ and $\delta_j^z = 0$ otherwise. Let us define yet another sample, $S_{z,u,v} = (s_1\delta_1^z, s_2\delta_2^z, ...)$. By definition, $S_{z,u,v}$ allows us to collect all pixels that have been drawn with the value z within $\delta\boldsymbol{z}$ of the set of explanatory variables, and allocated to v from u. One has (with the notation used in section 3.2):

$$\begin{aligned} E(\#J_{\boldsymbol{z},v}) &= E\Big(\sum_j S_j\delta_j^z\Big) = \sum_{s_j,j} s_j p_j \delta_j^z = \sum_{j,z^j=z \text{ within } \delta z} P(v_f|z^j, v_i) = \\ &= P(v|u) \times \frac{\rho(\boldsymbol{z}|u,v)}{\rho(\boldsymbol{z}|u)} \times \#J\, g(\boldsymbol{z},u)\delta\boldsymbol{z} \\ &= P(v|u) \times \rho(\boldsymbol{z}|u,v)\#J\,\delta\boldsymbol{z}, \end{aligned} \tag{C7}$$

so that

$$\frac{E(\#J_{\boldsymbol{z},v})}{E(\#J_v)} = \rho(\boldsymbol{z}|u,v)\delta\boldsymbol{z}. \tag{C8}$$

This last quantity is the fraction of pixels in $\boldsymbol{z}$ within $\delta\boldsymbol{z}$ for given states u and v. Therefore, the associated probability density $\rho(\boldsymbol{z}|u,v)$ is recovered from relevant expectation values, as advertized.

In other words, the pre-allocation probability distribution $\rho(\boldsymbol{z}|u,v)$ (i.e., before the sample is drawn) is also the post-allocation distribution in the mean, for the transition u to v (i.e., the mean distribution of $\boldsymbol{z}$ values over a large number of samples). Note that consequently our simple allocation method is unbiased, as we have just proven that it satisfies Eqs. (6) and (7) from Eqs. (C5) and (C7). On the other hand, the assumption that the quantities of change must be small does not seem to have been used anywhere. Remember however that small amounts of change are required for pixels to be expected to undergo a single transition per time step, and this assumption underlies our sampling method.

D. A mono-pixel algorithm enforcing quantities of change

Because pivot-cells are assumed to be statistically independent from one another, the MUST algorithm does attribute final states independently of how many pixels have (or not) already been selected in the sample for any final state v. As a result, the test does not ensure that the number of pivot-cells will always be equal to the expected mean [Eq. (11)] for each final state v, although, as demonstrated in the previous Appendix, this is true on average over a larger number of iterations of the allocation procedure (i.e., iterations of single-step simulations). In fact, as the probability distribution of repeated Bernoulli test is a binomial distribution (a well-known result of probability theory) the number of selected pivot-cells may turn out to be zero, even if the probability of transition to the intended final state is not vanishing (but the probability of this to occur is usually extremely small).

Besides the potential interest of this Appendix in itself, the arguments and proofs presented here constitute a stepping stone for the proof of absence of bias of the uPAM algorithm presented in the next Appendix.

D.1. Sampling without replacement from exactly known populations

It is in fact possible to enforce quantities of change exactly, but with a different algorithm. Enforcing such quantities of change is routinely done in LUCC modeling, as usually a single simulation of future projections (allocations) is performed, and as respecting the quantities of change provided through scenarios therefore becomes desirable in such a context. To achieve this purpose, however, one must change the sampling probability at every draw.

To show this explicitly, let us first examine a related but different problem. Namely, let us consider an urn containing initially (before balls are drawn from the urn) $n_w^{(1)}$ white balls and $n_b^{(1)}$ black balls, for a total $n^{(1)} = n_w^{(1)} + n_b^{(1)}$ balls. Balls are drawn without replacement at random from the urn. At the first draw, the probability of drawing a white ball is $p_w^{(1)} = n_w^{(1)}/n^{(1)}$, and similarly $p_b^{(1)} = n_b^{(1)}/n^{(1)}$ for a black ball, with $p_w^{(1)} + p_b^{(1)} = 1$.

The subsequent probabilities of draws are changed, because the number of balls in the urn has changed. Let us assume that, e.g., a black ball has been drawn at this first draw. The urn now contains $n_b^{(2)} = n_b^{(1)} - 1$ black balls, $n_w^{(2)} = n_w^{(1)}$ white balls for a total of $n^{(2)} = n^{(1)} - 1$ balls. The draw probabilities are therefore now changed to $p_w^{(2)} = n_w^{(2)}/n^{(2)} = n_w^{(1)}/(n^{(1)} - 1)$ and $p_b^{(2)} = n_b^{(2)}/n^{(2)} = (n_b^{(1)} - 1)/(n^{(1)} - 1)$. This new probability of drawing a white ball has increased as $n_w^{(1)}/(n^{(1)} - 1) > n_w^{(1)}/n^{(1)}$ and correlatively the new probability of drawing a black ball has

decreased as $(n_b^{(1)} - 1)/(n^{(1)} - 1) < n_b^{(1)}/n^{(1)}$. A similar reasoning applies if a white ball has been drawn instead of a black one at the first draw.

Such changes of the probabilities apply at every draw, until the end of the process ($n^{(1)}$ draws performed) and are necessary so that in the end the total number of drawn white and black balls is actually $n_w^{(1)}$ and $n_b^{(1)}$, respectively.

D.2. Mono-pixel allocation algorithm without replacement and with exact quantities of change enforced

The problem just discussed provides us with a way to obtain the desired algorithm for a single LUC state change, and by extension (section D.3.4), for an arbitrary number of simultaneous state changes. The problem is first discussed in a somewhat informal and intuitive way. More formal approaches are then progressively introduced. This discussion also serves as an introduction to the analysis of our multi-pixel algorithm (Appendix E).

The reasoning is simpler if continuous explanatory variables are first binned. Such a context is relevant in itself (as binning of explanatory variables is often performed in LUCC modeling software), but also as a first step for the non-binned context of section D.4.

To this effect, we focus on a specific but arbitrary bin Δ of explanatory variables $\boldsymbol{z}$. For simplicity, the bin is assumed to be small enough so that the value of $\boldsymbol{z}$ is nearly constant in the bin and approximated as $\boldsymbol{z}_\Delta$ (say, to the percent level for definiteness; alternatively we can define $\boldsymbol{z}_\Delta$ such that the second equality is true in the following equation). The total number of pixels in initial state u in this bin is thus given by[24]

$$N_{u,\Delta} = \#J \int_\Delta \rho(z|u) d\boldsymbol{z} = \#J\, \rho(z_\Delta|u) \delta\boldsymbol{z}_\Delta \equiv \#J\, P(z_\Delta|u), \tag{D1}$$

where the integral is performed over all $\boldsymbol{z}$ in the bin and where $\delta\boldsymbol{z}_\Delta$ is the small volume of the bin in explanatory variable space. Each pixel in the bin has therefore a probability $p_{v,\Delta} = P(v|u, z_\Delta)$ of undergoing a transition from state u to v, and a probability $1 - p_{v,\Delta}$ of not undergoing a transition, at the initial draw. Finally, we wish to impose that the total number of transited pixels in the bin is equal to its expectation value, i.e., $p_{v,\Delta} N_{u,\Delta}$.

In order to enforce this total number of transited pixels exactly, it is sufficient to associate to each pixel draw a ball draw similar to the one described above. This can be done by assuming, e.g., that white balls stand for no state transition, and black balls stand for the considered $u \to v$ transition. Thus, the initial number of balls in the urn is $n^{(1)} = N_{u,\Delta}$, the initial number of black balls is $n_b^{(1)} = p_{v,\Delta} N_{u,\Delta}$ and the initial number of white balls is $n_w^{(1)} = (1 - p_{v,\Delta}) N_{u,\Delta}$ ($\#J$ is assumed to be large enough so that these numbers can be assimilated to the nearest integer with negligible fractional error).

A possible algorithm is then the following:

Sampling design Draw at random a pixel in the pool of $N_{u,\Delta}$ pixels.

Allocation procedure Draw at random a ball from the urn, and associate to the pixel the corresponding outcome of the ball draw (transition or no transition depending on the ball color). Remove the drawn pixel and ball from their respective pools, update the ball numbers and ball draw probabilities as explained in section D.1.

Iteration Repeat the process until all pixels (and all balls) have been drawn.

Clearly at the end of the process, the required number of pixels will have transited from the discussion of section D.1. The algorithm is then repeated for all possible bins to complete the allocation process. This algorithm is not the only possible one (or even the simplest one) but has relevant features for the algorithm we propose for multi-pixel patch transitions.

D.3. A more direct algorithm enforcing quantities of change

Using an urn and balls is helpful to understand intuitively the necessity of updating pixel numbers and probabilities in sampling without replacement. But this detour is unnecessary, and the algorithm can be formulated directly and equivalently on pixels as there is a one to one correspondence between pixels and balls and their respective probabilities of draw.

[24] Discrete explanatory variables if any are not included in this expression, to simplify notations. Adding them does not change the argument. This remark holds throughout this Appendix.

D.3.1. Algorithm structure

From this argument, the algorithm can be reformulated as follows.

Sampling design Draw at random a pixel in the pool of $N_{u,\Delta}$ pixels.

Allocation procedure Assign a final state (either u or v) with the MUST test for a single $v \neq u$ final state. After a draw is completed, the pixel is removed from the pixel pool and the probabilities of the MUST algorithm are updated as described in section D.1.

Iteration Repeat the process until the required number of transition pixels is obtained.

D.3.2. Simplified updating

The only requirement is to update the number of pixels in the bin, $N_{u,\Delta}$ and the probability $p_{v,\Delta}$ in the same way as in section D.1. From these two quantities, the probability of no transition, $1 - p_{v,\Delta}$ and the number of non-transited pixels, $(1 - p_{v,\Delta})N_{u,\Delta}$ is also obtained, and the problem is completely specified at the next draw. This allows us to define an equivalent alternative and more convenient set of updated quantities:

- First, defining $N^{(p)}_{u,\Delta} = n^{(p)}$ the number of pixels in bin Δ and $\#\boldsymbol{J}^{(p)}$ the number of pixels in state u at the p-th iteration, updating $N^{(p)}_{u,\Delta}$ is equivalent to updating $\rho^{(p)}(\boldsymbol{z}_\Delta|u) = N^{(p)}_{u,\Delta}/(\#\boldsymbol{J}^{(p)}\delta\boldsymbol{z}_\Delta)$. More formally, this follows because the first equality itself holds independently of the bin size.

- Second, from Bayes rule, Eq. (1), updating $p^{(p)}_{v,\Delta}$ is equivalent to updating $\boldsymbol{P}^{(p)}(v|u)$ once $\rho^{(p)}(\boldsymbol{z}_\Delta|u)$ is itself updated, because $\rho(\boldsymbol{z}_\Delta|u, v)$ is a probability density defining the suitability of pixels for the $u \to v$ transition, which, as such, should not be modified in the process (the fact that pixels are removed from the sample does not *a priori* affect the relevance — or lack of — of explanatory variable values for the considered transition).

Conversely one sees that updating these quantities — i.e., $\rho^{(p)}(\boldsymbol{z}_\Delta|u)$, $\#\boldsymbol{J}^{(p)}$ and $\boldsymbol{P}^{(p)}(v|u)$ — at each draw immediately specifies the new transition probabilities, under the same *a priori* given information. One expects that the updating of the quantities of change is operated through $\boldsymbol{P}^{(p+1)}(v|u) = \boldsymbol{P}^{(p)}(v|u) - \sigma^{(p)}_v/\#\boldsymbol{J}^{(p)}$ where $\sigma^{(p)}_v = 1$ if the removed pixel in bin Δ is in final state v and $\sigma^{(p)}_v = 0$ otherwise. This follows because the total number of pixels undergoing a change at iteration (p) is $\sum_\Delta p^{(p)}_{v,\Delta} N^{(p)}_{u,\Delta} = \#\boldsymbol{J}^{(p)}\boldsymbol{P}^{(p)}(v|u, \boldsymbol{z}_\Delta)\rho^{(p)}(\boldsymbol{z}_\Delta|u)\delta\boldsymbol{z}_\Delta = \#\boldsymbol{J}^{(p)}\boldsymbol{P}^{(p)}(v|u)\int \rho^{(p)}(\boldsymbol{z}_\Delta|u, v)d\boldsymbol{z}_\Delta = \#\boldsymbol{J}^{(p)}\boldsymbol{P}^{(p)}(v|u)$ and the quantity of change at iteration (p) is $\sigma^{(p)}_v$.

D.3.3. Comments

A yet even simpler version of the algorithm can be produced, if one takes into account that the order in which transitions occur (or not) does not matter. In this case, and because the order in which pixels are drawn is random, one could as well assign the first $p_{v,\Delta}N_{u,\Delta}$ draws to the $u \to v$ transition and stop the algorithm there, leaving the remaining pixels in their initial state u. From a statistical point of view, this would be strictly equivalent, and this would definitely be much more economical from an algorithmic point of view.

We ignore this route in this Appendix because it is impractical when explanatory variables are not binned (section D.4). Also, the previous algorithm is probably more intuitively related to a real world process of drawing elements at random in a pool.

D.3.4. Multiple transitions generalization

One can generalize the algorithm to an arbitrary number of simultaneous transitions in a simple manner, with the help of the MUST algorithm.

Sampling design Draw at random a pixel in the pool of $N_{u,\Delta}$ pixels.

Allocation procedure Assign a final state with the MUST test for multiple transitions. After a draw is completed, the pixel is removed from the pixel pool and the probabilities of the algorithm are updated as described above, from the updating of all $\boldsymbol{P}(v|u)$ for each v and of $\rho(\boldsymbol{z}_\Delta|u)$, i.e., $\#\boldsymbol{J}^{(p)}$ is first updated $\boldsymbol{P}^{(p)}(v|u)$ is updated to $\boldsymbol{P}^{(p)}(v|u) - 1/\#\boldsymbol{J}^{(p)}$ if the pixel has undergone a state change from u to v and unchanged otherwise [with a similar updating of $\boldsymbol{P}^{(p)}(u|u)$], and $\rho^{(p)}(\boldsymbol{z}_\Delta|u)$ is updated to $\rho^{(p)}(\boldsymbol{z}_\Delta|u) - 1/\#\boldsymbol{J}^{(p)}\delta\boldsymbol{z}_\Delta$.

Iteration Repeat the process until the required number of transition pixels is obtained for each transition.

Once a transition has attained its predefined volume of change, the associated transition probability is down to zero and can be removed from the test. The procedure ensures that this occurs before or when all pixels are drawn (this should be transparent from the analogy with sampling without replacement in an urn with the appropriate number of balls of different colors, each color standing for a given transition).

Finally repeat the algorithm for all bins to complete the allocation process.

D.3.5. Absence of bias

Let us now show that this last algorithm is unbiased. In the present context, this amounts to verifying that Eqs. (6) and (7) are satisfied.

From the discussion after Eq. (D1), the number of pixels transiting to state v in bin Δ is $p_{v,\Delta} N_{u,\Delta} = P(v|u, \boldsymbol{z}_\Delta) \times \rho(\boldsymbol{z}_\Delta|u) \#\boldsymbol{J}\, \delta\boldsymbol{z}_\Delta$. This is true independently of the number of simultaneous transitions considered and follows from the formal correspondence of the algorithm with the urn sampling without replacement and with fixed number of colored balls for each bin; in this expression, $\rho(\boldsymbol{z}_\Delta|u)$ is the initial probability density (the formal correspondence is ensured precisely because the probability and total pixel number are identical to the urn one at each step). Consequently, the total number of pixels transiting for any $u \to v$ transition (all bins) is $\#\boldsymbol{J}_v = \sum_\Delta P(v|u, \boldsymbol{z}_\Delta) \rho(\boldsymbol{z}_\Delta|u) \#\boldsymbol{J}\, \delta\boldsymbol{z}_\Delta = \int \rho(v, \boldsymbol{z}|u) d\boldsymbol{z} \#\boldsymbol{J} = P(v|u) \#\boldsymbol{J}$, i.e., the result of Eq. (6) without expectation values: the algorithm does indeed enforce the exact quantities of change, as planned.

Finally, concerning Eq. (7), and from the number of pixels $P(v|u, \boldsymbol{z}_\Delta)\, \rho(\boldsymbol{z}_\Delta|u) \#\boldsymbol{J}\, \delta\boldsymbol{z}_\Delta = \rho(\boldsymbol{z}_\Delta, v|u) \#\boldsymbol{J}\, \delta\boldsymbol{z}_\Delta$ transiting to state v in bin Δ just mentioned, one has $\#\boldsymbol{J}_{\boldsymbol{z},v} = \rho(\boldsymbol{z}_\Delta, v|u) \#\boldsymbol{J}\, \delta\boldsymbol{z}_\Delta = \#\boldsymbol{J}_v \rho(\boldsymbol{z}_\Delta|u, v) \delta\boldsymbol{z}_\Delta$, where the expression of $\#\boldsymbol{J}_v$ just obtained has been used. In other words, the algorithm also exactly enforces the quantity involved in Eq. (7), and not only in expectation value.

This completes the proof that the algorithm is bias-free.

D.4. An alternative algorithm avoiding binning of continuous explanatory variables

So far, the argument has sidestepped a potential but minor problem. The numbers $N_{u,\Delta}$ of pixels in each bin, as well as the numbers of pixels undergoing each transition in each bin is assumed to be a round number (an integer). There is no guarantee that this is the case because probabilities are real numbers, not integers.

This can be circumvented in various ways. A first approximate way is to assume that the bin sizes are small enough so that $\boldsymbol{z}$ is close to being constant in the bin, but still large enough so that the numbers just mentioned are large enough and can be approximated by an integer with sufficient relative precision. This argument is more and more exact in the limit of an infinite number of pixels, which can be taken at constant bin size, or at constant (integer) numbers per bin in the limit of infinite numbers of pixels and vanishing bin size. These limits are of course never achieved. However, as maps are more and more resolved, very large number of pixels may now be considered (e.g., up to 10^{10}) so that rounding numbers such as $N_{u,\Delta}$ to the nearest integer constitutes a negligible relative error, except in the (themselves negligible) tails of the distributions of interest.

Another way to tackle the problem is to avoid binning altogether, a step also required by the fact that one obtains better estimates of of $\rho(\boldsymbol{z}|u)$ (one of the two quantities that need to be updated at each draw) by using our estimator (Mazy and Longaretti, 2022b), which requires unbinned continuous explanatory variables.

The use of binning allowed us to produce an algorithm where the number of pixels changing state was enforced not only globally (for the whole unbinned pixel population) but also locally, i.e., bin by bin. This second requirement made somehow the question of elaborating an unbiased algorithm simpler, but at the cost of too strong a requirement; indeed, Eq. (7) only requires that the distribution is obtained statistically, not exactly for each allocation; Eq. (6) is the only one that should be met exactly, and not in expectation values.

D.4.1. Algorithm structure

The understanding gained in the previous sections of this Appendix suggests a way to elaborate such an algorithm. To this effect, let us modify the algorithm defined in section D.3.4 in the following way:

Sampling design Draw at random a pixel in the pool $\boldsymbol{J}$ of pixels.

Allocation procedure Assign a final state with the MUST test for multiple transitions, and for the $\boldsymbol{z}$ value of the pixel. After a draw is completed, the pixel is removed from the pixel pool and the probabilities of the MUST algorithm are updated as described right below.

Iteration Repeat the process until the required number of transition pixels is obtained.

The discussion of section D.3 shows that it is sufficient to update $\#\boldsymbol{J}^{(p)}$, $\boldsymbol{P}^{(p+1)}(v|u) = \boldsymbol{P}^{(p)}(v|u) - 1/\#\boldsymbol{J}^{(p)}$ if the pixel has undergone a state change from u to v and unchanged otherwise and similarly for $\boldsymbol{P}^{(p+1)}(u|u)$, and to update $\rho^{(p)}(\boldsymbol{z}|u)$. This second updating is performed with our eKDE algorithm (Mazy and Longaretti, 2022b). Note that the eKDE method enforces that $\int \rho(\boldsymbol{z}|u)d\boldsymbol{z} = 1$ so that reducing by one the total pixel number ($\#\boldsymbol{J}^{(p+1)} = \#\boldsymbol{J}^{(p)} - 1$) also reduces by one the integrated pixel density $\#\boldsymbol{J}^{(p)} \int \rho^{(p)}(\boldsymbol{z}|u)d\boldsymbol{z}$; this ensures the self-consistency of pixel numbers between the two updatings. Again, these updatings define in turn the updated transition probabilities from Bayes rule, as $\rho(\boldsymbol{z}|u, v)$ is unchanged in the process.

D.4.2. Absence of bias

Eqs. (6) and (7) are satisfied by design with this algorithm. Let us show this in an explicit way.

For Eq. (6), the algorithm has been designed to enforce the quantities of change exactly and not only in expectations value. An proof of this statement can be obtained by making again use of the correspondence between this algorithm and another form of coupled pixel/colored balls problem. Start with $\#\boldsymbol{J}$ balls labelled by the various states v. The number of balls for any given state is the enforced one, i.e., $\boldsymbol{P}(v|u)\#\boldsymbol{J}$. If the balls were used as in section D.2, the final state of a pixel would be assigned from a random ball draw. Here we use this property in the opposite way: the final state of the pixel is assigned through the algorithm, and a ball of the correct label is assigned at random to the pixel. By design the updating of the probabilities $\boldsymbol{P}^{(p)}(v|u)$ after removal of the ball/pixel is identical to the probabilities obtained from ball counting as in section D.1. This ensures that updating the probabilities and removing balls are equivalent operations, and that the number of balls/pixel of each final state v is $\boldsymbol{P}(v|u)\#\boldsymbol{J}$ once the allocation is completed.

The main feature of this proof is that it allows us to ignore the value $\boldsymbol{z}$ associated to each pixel, as it is irrelevant to the overall updated number of balls/pixels in each state v at each iteration. This is to be contrasted with the proof given in section D.3.5 for the algorithm of section D.3.4, which applied for each individual bin, with a specified $\boldsymbol{z}_\Delta$.

Eq. (7) follows because $\rho(\boldsymbol{z}|u, v)$ is enforced exactly at each iteration of the algorithm. Consider a small hypercube volume Δ in explanatory space around a specific value $\boldsymbol{z}_\Delta$. Once all pixels have been assigned a final state v, for any given final state, one can define an index δ^j for each pixel j such that $\delta^j = 1$ if $z^j \in \Delta$ and $\delta^j = 0$ otherwise. The probability $p(\delta^j = 1) \equiv p^j_\Delta = \int_\Delta \rho(\boldsymbol{z}|u, v)d\boldsymbol{z} = \rho(\boldsymbol{z}_\Delta|u, v)\delta\boldsymbol{z}_\Delta$ and the probability $p(\delta^j = 0) = 1 - p^j_\Delta$. This defines again a Bernoulli variable on each pixel, but with another probability distribution p_j:

$$p_j = p(\delta^j) = {p^j_\Delta}^{\delta^j} \times \left[1 - p^j_\Delta\right]^{1-\delta^j}, \tag{D2}$$

and leads to the looked for expectation value:

$$E(\#\boldsymbol{J}_{\boldsymbol{z},v}) = E\left(\sum_{j\in\boldsymbol{J}_v} \delta^j_\Delta\right) = \#\boldsymbol{J}_v \sum_{\delta_\Delta=0}^{1} \delta_\Delta p(\delta_\Delta) = \#\boldsymbol{J}_v \rho(\boldsymbol{z}|u, v)\delta\boldsymbol{z}_\Delta. \tag{D3}$$

Note that the logic of the argument differs from the one used in Eq. (C3): the sum has been identified to $\#\boldsymbol{J}_v$ times the expectation value of the same variable. This was possible because the number of transited pixels $u \to v$ is now enforced to its expectation value. In the process, a detour through an explanatory variable integral has been avoided, thereby avoiding the problem raised by the fact that $\boldsymbol{P}(v|u, \boldsymbol{z})$ is no longer a constant quantity throughout the allocation process. This is also to be contrasted with the proof given in section D.3.5.

E. Multi-pixel patch allocation without pixel replacement

One of the motivations of Appendix D is to provide an introduction to the discussion of the uPAM algorithm. In particular the logic of Appendix D will be followed, by designing algorithms of increasing scope and complexity until the uPAM algorithm as presented in section 4.2 can be shown to be bias-free.

E.1. A multi-pixel patch MuST-like algorithm

It is convenient to start with an algorithm that does not enforce the total quantities of change, nor the absence of patch merging. This algorithm is of little practical use, but will allow us to reach our purpose in a progressive way, by pointing out the problems raised. A single final state v is used in this algorithm.

As explanatory variables, patch size and patch elongation are statistically independent, applying the MuST algorithm to each of these will provide the looked-for allocation procedure (for simplicity we keep here the same notation for the binned and non-binned quantities):

E.1.1. Algorithm structure

Binning The distributions $\rho(\sigma|u,v)$ and $\rho(e|u,v)$ are binned[25] to $P(\sigma|u,v)$ and $P(e|u,v)$.

Sampling design The MuST algorithm of Appendix C is applied to all pixels in initial state u with the relevant probability $P^c(v|u,\boldsymbol{z})$ of Eq. (5) for each pixel. This provides a set $\boldsymbol{J}_v^c$ of pivot-cells for state v. For each such pivot-cell j^c, the MuST algorithm is then applied with the relevant $P(\sigma|u,v)$ to associate a patch size σ^{j^c} to the pivot-cell, and again with $P(e|u,v)$ to associate a patch elongation e^{j^c}.

Allocation The patch is effectively allocated with the help of the patch design algorithm of Appendix J.3 where the patch merging test is omitted. No pixel is removed from the sample in the process.

Iteration At each iteration (i.e., for each pivot-cell), the pixels are left in the pool $\boldsymbol{J}$. The rationale of this choice is to avoid dealing with overlapping patches. Only at the end (once all patch have been assigned) are patches actually constructed and possibly merged on the basis of the selected pixels.

E.1.2. Properties and limitations

At the pivot-cell selection process step, the algorithm is unbiased: the argument of Appendix C.2 applies without change. The same is true with appropriate modifications of the proof for patch sizes and elongation. Let us show this for patch sizes for definiteness (the proof is clearly formally identical for elongation). To this effect, we introduce a new random variable S_j associated to each pivot-cell in the set $\boldsymbol{J}_v^c$, with two possible values, $s_j = 1$ if the associated patch as size $\sigma = \sigma_0$ and $s_j = 0$ if $\sigma \neq \sigma_0$. The probability distribution of this variable is

$$p_j = p(s_j) = P(\sigma_0|u,v)^{s_j} \times \left[1 - P(\sigma_0|u,v)\right]^{1-s_j}, \tag{E1}$$

Consider samples S_σ collecting the values of s_j produced by the allocation method: $S_\sigma = (s_1, s_2, ...)$. The probability of such a sample is $p(S_\sigma) = \prod_j p_j$. The number of times σ_0 appears in a sample $\#\boldsymbol{J}_\sigma$ has an expectation value over the population of samples of length $\#\boldsymbol{J}_v^c$ given by

$$E(\#\boldsymbol{J}_\sigma) = E\Big(\sum_j S_j\Big) = \sum_{s_j,j} s_j p_j = \#\boldsymbol{J}_v^c P(\sigma_0|u,v), \tag{E2}$$

which is the required result.

This suggests that the post-allocation distributions of patch sizes and elongations is unbiased. However, this is the case *only if* patch merging is avoided. Indeed, patch merging will necessarily and systematically produce a reduction in the number of pixels that have undergone a transition. It will also alter the patch size distribution, by reducing the number of smaller patches to the benefit of larger ones. Patch merging must therefore be avoided to obtain an unbiased algorithm.

Patch merging can hardly be avoided in the above algorithm. Indeed, some amount of patch overlap will almost necessarily occur in the process (random pivot-cell draws nearly always produce spatially close pairs of pivot-cells), although the total number of overlaps will be fractionally small for small enough quantities of change.

Also, the patch allocation may fail at step (iii) of Fig. J1 if no suitable neighbor is found. This occurs when the location of the pivot-cell is drawn in a very constrained surrounding (few available pixels of the same initial state in the neighboring structures of Appendix J.1 used in the patch design algorithm).

[25] The distribution of patch sizes and elongations, although specified in the continuous limit, are intrinsically discrete as pixels are discrete, but they are more simply specified in the continuous limit, and in any case, one may wish, for various reasons, to bin them in a coarser way than their natural discretization.

E.2. A constrained multi-pixel patch algorithm enforcing quantities of change and avoiding patch merging

E.2.1. Context

As in Appendix D, we first wish to produce a bias-free algorithm when explanatory variable space distributions are binned. To this effect, the definitions of Appendix D.2 are adopted and extended. All possible final states v are included in this section.

More precisely, consider a given bin Δ of volume $\delta\boldsymbol{z}_\Delta$. It is assumed again that $\rho(\boldsymbol{z}|u)$ is nearly constant in Δ. As before, the total number of pixels in the bin is $N_{u,\Delta} = \#J\int_\Delta \rho(z|u)d\boldsymbol{z} \simeq \#J\,\rho(z_\Delta|u)\delta\boldsymbol{z}_\Delta \equiv \#J\,P(z_\Delta|u)$, and the number N_v of pixels undergoing a transition from u to v is enforced to $N_v = N_{u,\Delta}P(v|u,\boldsymbol{z}_\Delta) \equiv \#J_{\boldsymbol{z}_\Delta,v}$; for $v = u$ this stands for the number of pixels not undergoing a transition. By construction $N_{u,\Delta} = \sum_v N_v$.

Furthermore, binning is chosen such that the scales L_p and L_z defined in Appendix B.2 obey the ordering assumed there ($L_p \ll L_z$); this assumption is unessential inasmuch as binning is avoided at the end, but helps to simplify the argument. Elongation is left aside: as there is no constraint associated to this, and as it is statistically independent by assumption from the other quantities, including it does not change the argument. Finally, we assume that patch formation cannot fail in step (iii) and (viii) of Fig. J1 due to lack of suitable neighbors. Again, this assumption will be lifted when discussing the uPAM algorithm (section E.3) and is made to avoid side issues.

The binning just performed allows us to focus on allocation in a single bin at a time. It is convenient in a first step to consider pixel final state allocation by bunch σ while ignoring their spatial location (i.e., ignoring for the time being that they change state by patches of spatially contiguous). This approach will be justified in section E.2.5.

E.2.2. Generalized urn and balls algorithm

The problem can first be approached by generalizing the argument of Appendix D.1. Let us consider an urn containing initially (before balls are drawn from the urn) $N_v^{(1)} = N_{u,\Delta}P(v|u,\boldsymbol{z}_\Delta) \equiv \#J_{\boldsymbol{z}_\Delta,v}$ balls labeled v (for all possible final states of the problem, i.e., with $v = u$ included) for a total $N^{(1)} = \sum_v N_v^{(1)} = N_{u,\Delta}$ of balls.

Balls are drawn without replacement at random from the urn. At the first draw, the probability of drawing a v ball is $p_v^{(1)} = n_v^{(1)}/n^{(1)} = P(v|u,\boldsymbol{z}_\Delta)$, with $\sum_v p_v^{(1)} = 1$. At iteration (p), the number of v balls is $N_v^{(p)}$ for a total number of balls $N^{(p)} = \sum_v N_v^{(p)}$, and with probabilities of draw $p_v^{(p)} = N_v^{(p)}/N^{(p)}$.

Balls are drawn iteratively in a modified way with respect to Appendix D.1. A ball is drawn at random. Let us say that its v label is $v = v_0$. The difference with the previous urn and ball model is that now, we wish to remove at each iteration a number $\sigma_{v_0}^{(p)}$ balls from the urn ($\sigma_{v_0} \ll N_{v_0}^{(1)}$) and with $\sigma_{v_0}^{(p)} = s^{(p)}$ if $v_0 = u$. Usually $s^{(p)} = 1$ but occasionally, $s^{(p)} > 1$ can be chosen to mimick patch construction failure; otherwise $\sigma_{v_0}^{(p)}$ mimicks patch construction. How this is done is not relevant, but, for example, one may look into the urn and pick the remaining $\sigma_{v_0} - 1$ balls at random.

Quite clearly, the updated number of balls are $N^{(p+1)} = N^{(p)} - \sigma_{v_0}$, $N_{v_0}^{(p+1)} = N_{v_0}^{(p)} - \sigma_0$ and $N_v^{(p+1)} = N_v^{(p)}$ for $v \neq v_0$. The new draw probabilities are $p_v^{(p+1)} = N_v^{(p+1)}/N^{(p+1)}$ for all v, with $\sum_v p_v^{(p+1)} = 1$ as expected.

Iterations stop when all $v \neq u$ balls are drawn from the urn (except for possible small mismatches between $\sigma_{v_0}^{(p)}$ and the remaining number of v_0 balls at the penultimate iteration for v_0; σ_0 can, for example, be completed by switching the required number of u balls with v_0 balls coming from outside the urn). Obviously, this takes at most $N^{(1)}$ iterations (the number of iterations if balls were drawn one by one from the urn), and at the end, all v balls will have been drawn from the urn (with a possible slight overshoot in v ball numbers as just described).

E.2.3. Constrained multi-pixel patch algorithm structure

Following again the lead of Appendix D, we may implement the previous iterative sampling directly on pixels with the same numbers and probability updating, without reference to urns and balls, for each bin Δ. The formal correspondence with the urn model just described will ensure that both the required numbers of pixel state change will be obtained for each final state v, respecting the expected initial probabilities of state change, and that this is achieved precisely by updating these probabilities and the total number of balls remaining in the urn.

Therefore, the algorithm proceeds as follows:

Binning The distribution $\rho(\sigma|u,v)$ is binned to $P(\sigma|u,v)$.

Sampling design A pixel is drawn at random in bin Δ. The MUST algorithm of Appendix C is applied with $P(v|u, \boldsymbol{z})$. If state $v_0 \neq u$ is selected, MUST is then applied with $P(\sigma|u, v_0)$ to associate a patch size σ to the newly drawn pixel.

Allocation If $v_0 \neq u$, a patch of size σ is constructed with the help of the patch design algorithm of Appendix J.3 around the drawn pixel, which becomes the pivot-cell of the newly created patch. We will check at the end that the distribution of pivot-cells is the correct one, although the probability of draws used is the one for the total pixel population of bin Δ, $P(v|u, \boldsymbol{z}_\Delta)$, and not the pivot-cell one $P^c(v|u, \boldsymbol{z}_\Delta)$.

Updating If no state transition occurred, the pixel is removed from the pool. If the patch is effectively allocated, all pixels are removed from Δ. If the patch construction fails (patch merging is detected or too few suitable neighbors), the pivot-cell and all pixels used in the attempted patch construction are removed from Δ. The updating is performed as follows after iteration (p) (the definitions of pixel numbers N, N_v are the same as in section E.2.2):

- If a state transition occurs, $N_{v_0}^{(p+1)} = N_{v_0}^{(p)} - \sigma$. Similarly $N^{(p+1)} = N^{(p)} - \sigma$ starting with $N^{(1)} = N_{u,\Delta}$. For all $v \neq v_0$, $N_v^{(p+1)} = N_v^{(p)}$. Pixels number are initialized to $N_v^{(1)} = N_v = N_{u,\Delta} P(v|u, \boldsymbol{z}_\Delta) = \#\boldsymbol{J}_{\boldsymbol{z}_\Delta, v}$.
- If no state transition occurs, $N_u^{(p+1)} = N_u^{(p)} - 1$ starting with $N_u^{(1)} = N_{u,\Delta} P(u|u, \boldsymbol{z}_\Delta) = \#\boldsymbol{J}_{\boldsymbol{z}_\Delta, u}$ (i.e., $v = u$ for this quantity) and $N^{(p+1)} = N^{(p)} - 1$. For all $v \neq u$, $N_v^{(p+1)} = N_v^{(p)}$.
- If the patch construction fails, the updating removes s pixels from state u where s is now the number of pixels involved in the patch construction until the process fails (it may fail immediately after the draw, in which case only one pixel is removed): $N_u^{(p+1)} = N_u^{(p)} - s$ and $N^{(p+1)} = N^{(p)} - s$. For all $v \neq u$, $N_v^{(p+1)} = N_v^{(p)}$.
- From this updating, one can now update the probability distribution $P^{(p)}(v|u, \boldsymbol{z}_\Delta) = N_v^{(p)}/N^{(p)}$ for all v; $\sum_v P^{(p)}(v|u, \boldsymbol{z}_\Delta) = 1$ as required.
- As $\rho(\boldsymbol{z}_\Delta|u, v)$ is unchanged (the fact that pixels are removed from the sample does not *a priori* affect the relevance — or lack of — of explanatory variable values for the considered transition), Bayes rule allows us to recover $P^{(p)}(v|u)$ from $P^{(p)}(v|u, \boldsymbol{z}_\Delta)$, $\rho^{(p)}(\boldsymbol{z}_\Delta|u) = N^{(p)}/(\#J\,\delta\boldsymbol{z}_\Delta)$ and $\rho(\boldsymbol{z}_\Delta|u, v)$.
- $P(\sigma|u, v)$ is unchanged.

Iteration The sampling design and allocation steps are iterated until the required quantity of change is obtained for each transition in bin Δ (i.e., $N_v^{(p)} \leq 0$ for each v).

The procedure is then repeated for all bins to complete the allocation process. The simplified updating of section D.3.2 applies here as well, for the same reason.

E.2.4. Absence of bias

Let us show that this algorithm is bias-free. This is achieved in several steps as one needs to check Eqs. (6), (7) and (10). Note that we have shown in section 3.2 that Eqs. (8) and (9) are satisfied if Eqs. (6) and (7) are, under the independence assumption of the explanatory variables and patch size distributions adopted in this work.

Eqs. (6) and (7) are satisfied by design. This is ensured by the formal correspondence with the urn problem of section E.2.2, due to the updating of probabilities and total pixel number $N^{(p)}$ used in the algorithm, which are identical to the ones in the urn sampling problem. In particular, $\#\boldsymbol{J}_{\boldsymbol{z}_\Delta, v} = N_v$ for bin Δ and for all v is exactly enforced in the algorithm (except for the slight overshoot due to the condition $N_v^{(p)} \leq 0$ at the last iteration). The total volume of change follows immediately from summing bin volumes of change: $\#\boldsymbol{J}_v = \sum_\Delta \#\boldsymbol{J}_{\boldsymbol{z}_\Delta, v} = \#J \int \rho(\boldsymbol{z}|u) P(v|u, \boldsymbol{z}) d\boldsymbol{z} = \#J \int \rho(v, \boldsymbol{z}|u) d\boldsymbol{z} = \#J\, P(v|u)$ so that Eq. (6) is satisfied exactly and not only in expectation value, as anticipated. Finally, using this expression and Eq. (1) (Bayes rule) in the bin quantity of change leads to $\#\boldsymbol{J}_{\boldsymbol{z}_\Delta, v} = \rho(\boldsymbol{z}_\Delta|u, v)\#\boldsymbol{J}_v$, i.e., Eq. (7) is satisfied exactly in bin Δ and not only in expectation value.

At the end of the algorithm, an unknown number of pivot-cells $\#\boldsymbol{J}^c_{\boldsymbol{z}_\Delta, v}$ will have been generated in each bin Δ and for each final state v; the total number of pivot-cells in final state v, $\#\boldsymbol{J}^c_v = \sum_\Delta \#\boldsymbol{J}^c_{\boldsymbol{z}_\Delta, v}$ is also unknown. To each pivot-cell, a patch size has been affected with the MUST algorithm. The fact that this algorithm is bias-free (see section E.1.2) ensures that $E(\#\boldsymbol{J}_{\sigma, v}) = P(\sigma|u, v)\#\boldsymbol{J}^c_v$ for all pivot-cell samples of given size $\#\boldsymbol{J}^c_v$. Averaging over all possible sample sizes therefore shows that Eq. (10) is satisfied because the algorithm avoids patch merging by design.

E.2.5. Spatial issues

The preceding proof leaves in the dark a point that one may find confusing: the spatial structure of patches (contiguous pixels) has been ignored. How can such an essential feature in usual 2D space be so unessential in explanatory variable space? A first element of answer has already been provided in Appendix D.3.3: as all pixels are equivalent in a bin Δ, all collections of such pixels with the right properties (correct total number for each final state) will satisfy the no-bias constraint. One may therefore as well choose contiguous pixels (forming a patch) if one wishes to.

This answer however leaves a point in the dark: not all pixels collection of the right total number are equivalent, because pivot-cells cannot be freely chosen due to the patch merging avoidance condition. Indeed, a collection of $n_{\boldsymbol{z}_\Delta,\upsilon} \simeq E(\#J^c_{\boldsymbol{z}_\Delta,\upsilon})$ pivot-cells in final state υ all grouped in the same spatial area in bin Δ is as probable as any other, in principle (at least at the level of pivot-cell selection), but cannot be realized because patches have a non vanishing spatial extent ($\sim E(\sigma)^{1/2}$): this is prevented by the removal of pixels at the iteration and updating step of the algorithm, which itself is enforced to prevent patch merging.

How is the information of patch merging prevention formally accounted for? The answer lies in the fact that the exclusion of potential pivot-cells is not incorporated in the 1-pixel probability distribution $\rho^c(\boldsymbol{z}|u,\upsilon)$, but the joint 2-pixels, 3-pixels, ..., $n_{\boldsymbol{z}_\Delta,\upsilon}$-pixels probability distributions, that is, $\rho^c(\boldsymbol{z}^1,\boldsymbol{z}^2|u,\upsilon)$, $\rho^c(\boldsymbol{z}^1,\boldsymbol{z}^2,\boldsymbol{z}^3|u,\upsilon)$, etc.

For example, $\rho^c(\boldsymbol{z}^1,\boldsymbol{z}^2|u,\upsilon) = \rho^c(\boldsymbol{z}^1|u,\upsilon)\times\rho^c(\boldsymbol{z}^2|\boldsymbol{z}^1,u,\upsilon) \neq \rho^c(\boldsymbol{z}^1|u,\upsilon)\times\rho^c(\boldsymbol{z}^2|u,\upsilon)$. The last inequality follows because $\rho^c(\boldsymbol{z}^2|\boldsymbol{z}^1,u,\upsilon) \neq \rho^c(\boldsymbol{z}^2|u,\upsilon)$ precisely because $\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{x})$ ($\boldsymbol{x}$ stands for the coordinates of in physical 2D space) and $\boldsymbol{x}^2$ cannot be chosen closer than $\sim 2E(\sigma)^{1/2}$ to $\boldsymbol{x}^1$, on average. Because the conditional and unconditional probability distributions differ, the 1-pixel distributions do not contain all the information needed to fully describe the problem (in particular, e.g., the distribution of mutual distances between patches, and not only for the same initial and final states), and multiple pixel distributions are required to quantify the missing information.

This limitation is of no direct consequence here, because joint pixel distributions are not yet in the scope of LUCC modeling (except indirectly through patch distributions and locations). If the information they carry ever becomes a concern, such joint pivot-cell distributions will also need to be calibrated.

Note finally that in the limit of small quantities of change, the mutual patch excluded "volume" in explanatory space is small or even negligible, so that the conditional probability distribution simplifies to $\rho^c(\boldsymbol{z}^2|\boldsymbol{z}^1,u,\upsilon) \simeq \rho^c(\boldsymbol{z}^2|u,\upsilon)$, and the patch merging problem is only marginally important. However, quantifying this is delicate and problem-dependent, so it is best to err on the side of safety and explicitly prevent patch merging.

E.3. uPAM algorithm discussion

The algorithm uPAM as described in section 4.2 results from a combination of the arguments of Appendices D.4 and E.2.3. In particular there are two differences between the algorithm of section E.2.3 and uPAM:

- The drawn pixel is obtained from the MUST algorithm.
- The updating of numbers and probabilities are obtained from the updating of $\#J$, $P(\upsilon|u)$ and $\rho(\boldsymbol{z}|u)$.

The first point follows the logic of Appendix D.4. Because MUST is bias-free, the sampling design of uPAM is bias-free.

The rationale of the second point is explained in Appendix D.3.2 and applies without change in the present context. In particular, the updating of the quantity of change $P^{(p+1)}(\upsilon|u) = P^{(p)}(\upsilon|u) - \sigma_\upsilon^{(p)}/\#J^{(p)}$ applies in the same way, except for $\sigma_\upsilon^{(p)}$ which accounts for the number of pixels removed from the initial pool. The same is true of the way the updating of $\rho(\boldsymbol{z}|u)$ is obtained from the explanation of Appendix D.4. The absence of bias demonstrated in section D.4.2 applies here as well, once the correspondence with the algorithm of section E.2.2 is substituted to the correspondence with the algorithm of section D.1.

These previous proofs of absence of bias were detailed enough and need not be repeated here.

E.4. Pseudo code

In addition to the workflow (figure 2) which resumes the process of this method, a pseudo-code is given in the algorithm 2.

Algorithm 2: Unbiased Patch Allocation Method UPAM

Data:
A LULC map
$P(v|u)$, the global transitions probabilities for each final state v
Result: An allocated LULC map

Get from the LULC map the set of all pixels of initial state u noted $\boldsymbol{J}$
$n(v|u) \leftarrow P(v|u)\#\boldsymbol{J}$
$\hat{\boldsymbol{J}} \leftarrow \boldsymbol{J}$
while $n(v|u) > 0$ *and* $\tilde{\boldsymbol{J}} \neq \emptyset$ **do**
  $\tilde{P}(v|u) \leftarrow n(v|u)/\hat{J}_v$
  Computes $\tilde{\rho}(\boldsymbol{z}|u)$ through the chosen density estimation method according to the pixels in $\tilde{\boldsymbol{J}}$
  Computes $P^c(v|u, \boldsymbol{z})$ through the Bayes Adjustment Process resumed in (5) and detailled in Mazy and Longaretti (2022b) (section 5.3) according to $\tilde{\rho}(v|u, \boldsymbol{z})$ and $\tilde{P}(v|u)$
  Draws pivot cells through MuST (appendix C)
  if *The MuST process returned no pivot cell* **then**
    Break the While loop
  Pick one pivot cell at random.
  Applies the patcher algorithm on this pivot cell (appendix J)
  It returns the list of pixels that has been considered by the patcher : K
  if *The patch process has been a success* **then**
    $n(v|u) \leftarrow n(v|u) - \#K$
  $\tilde{\boldsymbol{J}} \leftarrow \tilde{\boldsymbol{J}} \setminus K$

F. Dinamica EGO Allocation Method

F.1. Allocation Method Based on a Pruning Procedure

Although Dinamica EGO's code is not open source, the code documentation and discussions through Dinamica EGO's Google Group allowed us to understand and reproduce its allocation method[26]. We build on this understanding to present here a detailed algorithm structure of Dinamica EGO's allocation method for pivot-cells (the part we needed most to understand in detail for our purpose).

Sampling design Dinamica EGO ranks pixels by decreasing value of $P(v|u^j, \boldsymbol{z}^j)$ and keeps F times the number of pixels needed for the each transition of interest[27], i.e., the software prunes the number of pixels. F is specified by the user; the default is $F = 10$. If more than one transition is possible, the sum of the probabilities over all v is used instead of a single probability: $\sum_{v \neq u^j \in V} P(v|u^j, \boldsymbol{z}^j)$. The $\boldsymbol{J}$ ensemble pruned in this way is designated by $\boldsymbol{J}^F$.

Allocation procedure In Dinamica EGO, allocation follows from a series of simple rejection tests[28]. Pixels are first chosen at random in $\boldsymbol{J}$, and the global probability of transition $\sum_{v \in V} P(v|u^j, \boldsymbol{z}^j)$ is tested. If a pixel passes the test, it is kept for the second rejection test, where one of the possible state changes is chosen at random (say, v_0), and the simple rejection test is applied to v_0 with the conditional probability[29] $P(v_0|u^j, \boldsymbol{z}^j)/\sum_{v \in V} P(v|u^j, \boldsymbol{z}^j)$. The test is repeated until one of the state changes is accepted. Once a pixel has been selected in this way, it is

[26] Dinamica EGO has two different procedures to create new patches: patcher and expander. The second one is specific to creating new patches in contact with existing areas of the target LUC category. Here we ignore Dinamica EGO's expander functionality, as it does not involve new issues of principle with respect to the patcher one.

[27] The Dinamica EGO online documentation (`https://www.csr.ufmg.br/DinamicaEGO/dokuwiki/doku.php?id=guidebook_start`) is not very clear on these technical points. The most detailed information is obtained through Dinamica EGO's Google Group (`https://groups.google.com/forum/#!searchin/DinamicaEGO/allocation%7Csort:date/DinamicaEGO/kL7XE_uCBnI/9E_zWUJWPNMJ`). Some more details were provided through mail to one of us (PYL) by one of Dinamica EGO developers (Hermann Rodrigues, *private communication*)

[28] These tests proceed as binomial tests with q single probability p of acceptation and $1-p$ of rejection, similarly to the multinomial test described above restricted to a binomial outcome.

[29] The conditional probability is necessary because the pixel is now known to undergo one of the possible transitions.

used as pivot-cell of a new patch (the patch construction procedure is described below). The pixels of the patch are then removed from the pruned ensemble.

Because $J^F \subset J$, in principle, $P(v|u)$ should be changed into $P(v|u) \times (\#J/\#J^F)$ when one uses a pruned ensemble instead of the complete one, in order to reflect the fact that the expected volume of LUC state transitions is unchanged while the number of pixels to test is reduced. In practice, Dinamica EGO works with weights of evidence and enforces the volumes of changes instead of enforcing this correction. The end result is identical with using actual transition probabilities and rescaling $P(v|u)$.

The allocation procedure description leaves a point in the dark: are pixels rejected in the first test taken out of the pruned ensemble, or put back in there? In principle, they should be taken out, following the general logic of sampling without replacement that should be implemented, but it appears that Dinamica EGO's algorithm puts them back in the sample. This question is further discussed in section 6.2.1.

F.2. Recoded Pivot Cell Allocation Method

The selection and allocation of pivot cells is probably the most critical step in the analysis of potential biases. For this reason, this part of Dinamica EGO's patcher has been recoded for this study, in order to check by cross-comparison between our version and Dinamica EGO's that our understanding of this step is quantitatively and not only qualitatively correct. This applies to both mono-pixel and multi-pixel patches. We only needed to check the algorithm for mono-pixel patches, and it is therefore coded only in this context. The resulting pseudo-code is given in algorithm 3.

Dinamica EGO applies its pivot cell allocation method to each initial LUC state separately. Once the pruning process (line 1) is completed, for each possible final state v, the targeted number of pixels $N_{u\to v}$ that are required to undergo the $u \to v$ transition is determined (line 3). While all transitions are not performed (line 4), one pixel is randomly selected from the pool of pivot candidates J_v^F (line 5). Then, a rejection test with the sum of all possible transition probabilities is applied (line 6). If the pixel is rejected, it is replaced in the pool (this operation is formally incorrect, a point discussed in section 6.2.1). If on the contrary the pixel is accepted, the selection process chooses a final state randomly (line 8) and then it tests its normalized probability against a random value (line 9). If the pixel passes the test, the pixel is used as a seed (pivot cell) for a new patch (line 10). Otherwise, the process chooses a new final state randomly until a transition is accepted (line 7). Once a transition quota is reached, the corresponding transition and its probabilities are removed (line 14).

As a final comment, let us point out that the two stage process of selection of a final state and pivot cells is somewhat inefficient, as one can in fact test all final states in the same test, as in our multinomial simple test MUST described earlier.

F.3. Patch parameters

Dinamica EGO characterizes patches through two parameters: the patch size and a shape parameter called "isometry"[30].

F.3.1. Patch size

The probability distribution of patch sizes is specified by the software and cannot be changed by the user, except for its parameters. It follows a log-normal law:

$$P(\sigma) = \frac{1}{\delta(2\pi)^{1/2}} \exp\left(-\frac{[\ln(\sigma) - \mu]^2}{2\delta^2}\right), \tag{F1}$$

where μ and δ are the log-normal law parameters and ln is the natural logarithm (the probability distribution is independent of the pixels initial and final states). However, the parameters specified by the user in Dinamica EGO are the mean $E(\sigma)$ and the variance $V(\sigma)$ of the patch size. The corresponding parameters μ and δ of Eq. (F1) are obtained through the following relations:

$$\mu = \ln\left(\frac{E(\sigma)^2}{\left(E(\sigma)^2 + V(s)\right)^{1/2}}\right) \tag{F2}$$

[30] This has no relation with the mathematical concept of isometry.

Algorithm 3: Reproduced Dinamica pivot cell allocation process

Data:
The set of all pixels indices J,
V, the set of LUC states,
$\{P^c(v|u^j, z^j)\}_{\forall j\in J,\ \forall v\in V}$, the transitions probabilities,
$P^c(v|u)$, the transition scenario matrix,
the patch parameters,
F, the pruning factor

Result: Allocated LUC map.

$J^F \leftarrow \text{PRUNING}(J, \{P^c(v|u^j, z^j)\}_{j,u}, F)$
$\nu \leftarrow V \setminus \{u\} \equiv \{v \in V,\ v \neq u\}$
$\forall v \in \nu,\ N_{u\to v} \leftarrow P^c(v|u)\#J$
while $\exists v \in \nu,\ N_{u\to v} > 0$ **do**
  $j \leftarrow \text{SAMPLE}(J_v^F)$
  if RAND $< \sum_{v\in\nu} P^c(v|u, z^j)$ **then**
    while allocation is not made yet **do**
      $v \leftarrow \text{SAMPLE}(\nu)$
      if RAND $< \dfrac{P^c(v|u, z^j)}{\sum_{v\in\nu} P^c(v|v_t, z^j)}$ **then**
        Allocate pivot cell j
        Decrease $N_{u\to v}$ by 1
        Remove allocated pivot cell j from J_u^F
        if $N_{u\to v} \leq 0$ **then**
          $\nu \leftarrow \nu \setminus \{u\}$
return the allocated LUC map, *i.e.* $\{v^j\}_{j\in J}$

$$\delta^2 = \ln\left(1 + \frac{V(\sigma)}{E(\sigma)^2}\right). \tag{F3}$$

F.3.2. Patch Isometry

The shape parameter used by Dinamica EGO is called "isometry" and ranges from 0 (linear patch) to 2 (circular patch). In order to have a better idea of the impact of this parameter during the patch creation process, we have characterized the shape of the patches produced by Dinamica EGO using our own patch shape parameter, elongation, defined in appendix J.2. This analysis makes use of very simple synthetic data. We chose a uniform initial LUC map of 1000×1000 pixels with a uniform transition probability distribution toward a unique final state. The global transition probability is set to $P(v|u) = 0.005$, the pruning factor to $F = 200$ (this corresponds to no pruning) and the expected patch size to $\sigma = 50$.

We perform a number of Monte Carlo simulations, defined as follows. We run a large number of times the allocation procedure and average the observed value (here the mean elongation of the patches). The chosen number of Monte Carlo simulations is 100.

This setting is designed to focus on patch shapes. Allocated maps are generated through Dinamica EGO and we measure the elongation, Eq. (J12), of each patch (we exclude patches that have merged in order to avoid the patch merging bias, which will change the post-allocation measured isometry in uncontrolled ways).

The isometry parameter does not have any simple relation to our elongation parameter. However, one would expect some form of monotonic relation between the two if both are to constitute useful ways of representing some kind of mean and progressive deformation of a patch with respect to a purely circular patch. Our elongation parameter being directly based on moments of the patch shape is expected to display this behavior. The question we address here is whether this is also the case for Dinamica EGO's isometry parameter, which has no such direct geometric interpretation.

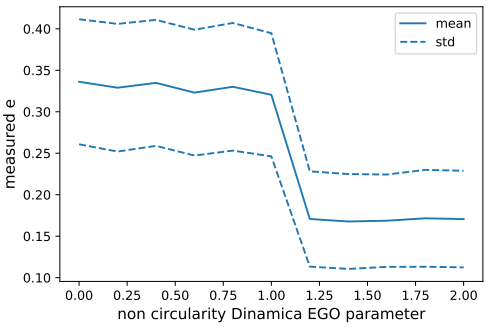


Figure F1: Measured elongation (average and standard deviation, with rook neighboring structure) of final LUC maps produced by Dinamica EGO as a function of the non-circularity parameter (isometry).

The result of this analysis is represented on Fig. F1 where we plot the measured elongation and its standard deviation obtained in our Monte Carlo simulations as a function of Dinamica EGO's non-circularity parameter (isometry). We observe a bimodal distribution of elongations with a transition around an isometry parameter of 1. The standard deviation of the measured elongation is also relatively high and independent of the non-circularity parameter.

One cannot compare the expected shape parameter with the measured one in Dinamica EGO's logic. Indeed, the isometry parameter does not produce a precisely defined type of deformation. On the other hand, our definition of elongation allows us to check this, and the measured (post-allocation) elongation correlates strictly to the imposed (pre-allocation) one (see appendix J.4). In this respect, if a shape parameter based on patch moments were calibrated on data in LUCC modeling, Dinamica EGO isometry parameter distribution would be difficult to calibrate due to its bimodality and would lead to a degraded representation of patch shapes. For this reason, and because of the absence of clear correlation between isometry and elongation, while elongation provides a clear shape control to the user, we favor elongation as a shape parameter, although the associated patch construction and characterization is more involved.

G. Idrisi LCM Allocation Method

Idrisi LCM allocation method is relatively simple. It is deterministic: no random draws are involved and the allocated pattern will always be the same if the simulation is repeated.

Sampling design The sampling is reduced to LCM chosen pruning procedure. LCM ranks pixels by decreasing value of $\rho(\boldsymbol{z}^j|u^j, v)$ (called potential maps in LCM) and keeps only the number of pixels needed for the specified transition. This is done for all transitions (Eastman et al., 1995). This constitutes an extreme form of pruning, and the sampling design becomes deterministic.

Allocation procedure If there is only one transition, there is no random pixel selection nor allocation step in LCM: all the remaining pixels after pruning are allocated. If there is more than one, allocation conflicts can arise. These are solved by a simple rule of thumb[31] allocating conflicting pixels to one or the other transition and extending the sample until the targeted number of pixels is obtained for each transition.

The choices performed by LCM ensure that transitions necessarily occur in patches (due to the underlying continuous dependence of $\boldsymbol{z}$ on spatial coordinates), but at the cost of a strongly biased transition distribution with respect to the distribution quantified in the calibration phase. Indeed, truncating any distribution to the highest value inevitably changes the post-allocation distribution with respect to the pre-allocation one (see section 6.1 and Appendix H).

H. Pruning by probability distribution truncation

Pruning is an operation of pre-selection of pixels that are most likely to undergo a given transition. This operation is not strictly necessary but is nevertheless performed by both Dinamica EGO and Idrisi LCM, most likely in order

[31] This rule does not need to be further specified here as this allocation method is not theoretically correct, as will be shown later on. For more details, see Eastman et al. (1995).

to reduce the allocation computational time. In our own algorithms, we do not need any pruning as our algorithmic implementation is intrinsically quite efficient, even for large case studies (billions of pixels).

The main tasks of this Appendix are to identify the correct probability distribution that must in principle be used in any kind of pruning strategy, and to show why, for the pruning strategy based on truncation of the probability distribution, both Dinamica EGO and Idrisi LCM pruning strategies are necessarily biased.

The problem of identifying the correct probability distribution for pruning by truncation has an inherent ambiguity that must first be dispelled. Indeed, two criteria can *a priori* be chosen for pruning, for a given $u \to v$ transition:

- On the one hand, one may choose the pixels who have the largest transition probability $P(v|u, \boldsymbol{z})$.
- On the other hand, because explanatory variables are selected on the basis of their correlation with observed transitions, one may wish to choose the pixels whose explanatory variable values maximize $\rho(\boldsymbol{z}|u, v)$ instead.

It turns out that both pruning criteria are admissible, but not in the same space. Furthermore, choosing one or the other as the distribution to be pruned may lead to differences in truncation boundaries, for the same fraction of total probability pruned.

The proof of these statements follows from a close scrutiny of Eqs. (C3) and (C5). The first equation shows that the pixels with the largest $P(v|u, \boldsymbol{z})$ do indeed contribute most to the number of transited pixels $N_{u,v}$. The next equation shows that the $\boldsymbol{z}$ values contributing most to $N_{u,v}$ once expressed as an integral in explanatory variable space $\mathcal{D}$ are those maximizing $\rho(\boldsymbol{z}|u, v)$ in $\boldsymbol{z}$ space.

To proceed further, let us assume that pruning is performed by truncating the probability distribution so that a given fraction of the total number of pixels is summed in Eq. (C3). Because $P(v|u, \boldsymbol{z}) \propto \rho(\boldsymbol{z}|u, v)/\rho(\boldsymbol{z}|u)$, if $\rho(\boldsymbol{z}|u)$ is not uniform, $P(v|u, \boldsymbol{z})$ and $\rho(\boldsymbol{z}|u, v)$ will not be maximum for the same $\boldsymbol{z}$, and the truncation boundaries in $\boldsymbol{z}$ will not be the same if one prunes by truncation on $P(v|u, \boldsymbol{z})$ or $\rho(\boldsymbol{z}|u, v)$. This point is further discussed in section 6.1.2.

This argument relies on a proof provided for mono-pixel patches. However, it applies as well to multi-pixel patches as, on the one hand, it applies to pivot cells with straightforward changes (i.e., our MUST algorithm is also an unbiased algorithm of selection of pivot-cells), and, on the other hand, to all pixels in a patch constructed around a pivot-cell as these, on average (over repeated simulations of the same time step allocation process), have the same number of pixels $E(\sigma)$, and nearly the same probability distribution as their pivot cell seed as long as their mean size $\sim E(\sigma)^{1/2}$ is small compared to the spatial characteristic scale of explanatory variables (a condition often if not always satisfied in practice). A more formal version of this argument can be provided but is not needed at the level of rigor of the present discussion.

Therefore, as long as pruning consists in keeping the most probable pixels for any state transition, either the probability distribution $\rho(\boldsymbol{z}|u, v)$ in $\boldsymbol{z}$ should be selected for probability ordering in explanatory variable space $\mathcal{D}$ (and keeping the corresponding pixels), or the probability $P(v|u, \boldsymbol{z})$ if pixels are directly ordered and not the explanatory variables themselves. However, this formal equivalence does not apply to maps. Indeed raster maps can be seen as a particular way to order pixels in a matrix form. Assigning row i_r and column i_c indices to pixels is equivalent to assigning a single index j, and ordering $P(v|u, \boldsymbol{z})$ with a single index j or two indices i_r, i_c clearly lead to the same pixel ordering and same pixel selection from this ordering. Equivalently, this ordering can be performed on location coordinates instead of row and column indices, as there is a bijection between the two. Finally the ordering and associated selection results in a specific area selection in the raster map, with exactly the same probability values and number of pixels selected. This series of equivalent statements implies that, conversely, if pixels are pruned from a map, the map must represent $P(v|u, \boldsymbol{z})$, not $\rho(\boldsymbol{z}|u, v)$.

Alternatively, a pruning of $\rho(\boldsymbol{z}|u, v)$ can be performed in explanatory variable space, not in physical (map) space with a similar line of reasoning. Here again, these arguments can be presented in a more formal way, but this is not needed at the level of rigor adopted here.

This shows that Dinamica EGO pruning is based on the correct distribution, while Idrisi LCM choice is incorrect, as it selects pixels on the basis of a map of $\rho(\boldsymbol{z}|u, v)$ (transition potential map). Indeed, choosing pixels maximizing $\rho(\boldsymbol{z}|u, v)$ on a map amounts to summing this quantity over pixels:

$$\sum_i \rho(\boldsymbol{z}^i|u, v) \propto \int_z \rho(\boldsymbol{z}^i|u, v)\rho(\boldsymbol{z}^i|u)dz, \tag{H1}$$

similarly to Eqs. (C3) and (C5). This has no simple relation with $N_{u\to v}$.

For definiteness, in the remainder of this discussion we keep the ordering of $P(v|u,\boldsymbol{z})$ as a choice of pruning strategy, as spatial maps are more natural to use than explanatory variable space $\mathcal{D}$ ordering, especially when the number of explanatory variables exceeds two.

As a limiting case, it may occur that a vast majority of explanatory variable space have zero contribution, that is, $P(v|u,\boldsymbol{z}) = 0$ over large domains of $\boldsymbol{z}$ values. Removing the corresponding pixels from $\boldsymbol{J}$ does not change anything in the final allocation, but may considerably reduce the required number of random pixel selection. By extension, removing the pixels with small probability distribution $P(v|u,\boldsymbol{z})$ should have little effect on the result, while still reducing the computational cost of the allocation, by effectively pruning the ensemble $\boldsymbol{J}$. This being said, one must keep in mind that any method of pruning by truncation of the probability distribution necessarily produces a bias with respect to the expected distribution $P(v|u,\boldsymbol{z})$: by necessity, the post-allocation distribution will differ from the pre-allocation one as Eq. (7) will be violated by virtue of Eqs. (C3) and (C5). Consequently, if this pruning strategy is adopted, the amount of pruning must be decided on the basis of a trade-off between computing efficiency and an acceptable level of bias. Alternatively one may prune more uniformly the probability distribution in order to maintain its shape, but this type of pruning strategy is considerably more involved.

This discussion can be summarized in two points:

- Dinamica LCM does choose the correct probability distribution map for pruning, $P(v|u,\boldsymbol{z})$ while Idrisi LCM does not , as it chooses $\rho(\boldsymbol{z}|u,v)$ instead[32].
- Furthermore, the level of bias of the allocation method of LCM is substantially higher than for Dinamica EGO, because the level of pruning applied by LCM is much more extreme.

I. Post-allocation probability distribution confidence interval

We estimate here the statistical noise of a bias-free allocation in order to determine the minimum number of Monte Carlo simulations needed to produce a given, predefined level of accuracy in the determination of the relevant post-allocation probability distributions. For definiteness, we focus on a single (pre-allocation) combination $(u,\boldsymbol{z})$ (within some small range $\delta\boldsymbol{z}$) and a single final (post-allocation) state v_0. By construction, this transforms our multinomial probability distribution $P(v|u,\boldsymbol{z})$ into a binomial one with $v = v_0$ or $v \neq v_0$ (without affecting the transition probability to the chosen state v_0), so that each pixel now undergoes a Bernoulli test during an allocation (this is precisely the case with our mono-pixel allocation method, section 4.1, and this remark also applies to the allocation of a final state to a pivot-cell in the multi-pixel patch context of section 4.2, as the same algorithm is used for this step).

The pivot-cell transition probability is $p_0 = P^c(v_0|u,z)$ and $1-p_0$ for no transition. The number of pixels undergoing this Bernoulli test is equal[33] to $\#\boldsymbol{J}^*\rho(\boldsymbol{z}|u)\delta\boldsymbol{z}$. Thus, if we perform n_{MC} Monte Carlo simulations, this binomial transition probability will have been tested $n_{MC}\#\boldsymbol{J}^*\rho(\boldsymbol{z}|u)\delta\boldsymbol{z}$ times in total. In these expressions, $\boldsymbol{J}^*$ is the set of pixels that is reduced as our iterative allocation method proceeds (section 4.2). However, in our iterative allocation method, $\#\boldsymbol{J}^*$ is never very different from $\#\boldsymbol{J}$ because the total number of pixels changing LUC state per time step is always a small fraction of the total $\#\boldsymbol{J}$. For simplicity, we assume that $\#\boldsymbol{J}^* = \#\boldsymbol{J}$ throughout the iterative allocation process. This amounts to approximating our iterative allocation method without replacement with an allocation method with replacement. Although this approximation is not precise enough to produce an unbiased allocation algorithm, it is sufficient for the present purpose (quantifying a number of Monte Carlo simulations to obtain the post-allocation probability distribution within a predefined confidence interval) as the looked for number of Monte Carlo runs does not need to be very precise.

In practice, we perform a series of Bernoulli trials whose characteristics have just been specified. Probability theory then implies that the expected mean for success (i.e., the mean number of transitions from u to v_0) is $n_{MC}\times p_0\rho(\boldsymbol{z}|u)\times \#\boldsymbol{J}\delta\boldsymbol{z}$. Dividing this by the number of trials $n_{MC}\#\boldsymbol{J}^*\rho(\boldsymbol{z}|u)\delta\boldsymbol{z}$ gives back the expected probability $p_0 = P^c(v_0|u,z)$.

The actual number of successes will differ somewhat from the expected mean, and dividing by the number of trials will yield an estimate of the expected probability, $\hat{P}^c(v_0|u,z)$. The question boils down to defining the number of Monte Carlo simulations n_{MC} in order for this estimate to lie close enough to the expected probability, within some tolerance level predefined by the user.

[32] For the same reason, comparing directly the maps of Dinamica transition probability with LCM maps of transition potential is meaningless.

[33] This quantity is not necessarily a natural number, depending on the choice of $\delta\boldsymbol{z}$. This point is unessential for the argument presented here.

This question is best resolved by making use of the known variance V of repeated Bernoulli tests, which, in the case at hand, is given by

$$V = \frac{P^c(v_0|u,\boldsymbol{z})(1 - P^c(v_0|u,\boldsymbol{z}))}{n_{MC}\rho(\boldsymbol{z}|u)\#\boldsymbol{J}} \tag{I1}$$

In the limit of large numbers, the Bernoulli test probability distribution is approximated by a Gaussian distribution and a very common way to approximate the needed confidence interval ΔP is based on the central limit theorem[34]

$$\Delta P = 2p_\alpha V^{1/2}, \tag{I2}$$

where p_α is the $1 - \alpha/2$ quantile of a standard normal distribution and refers to the target error rate α. This p-value quantifies the number of standard deviations required to achieve a given level of confidence α i.e., the cumulative probability distribution within $\pm p_\alpha V^{1/2}$. For the standard choice $\alpha = 0.95$ (95% confidence level), $p_\alpha \simeq 1.96$.

We can now introduce our tolerance ε as the admissible error of the observed transition probability, i.e.:

$$\varepsilon = 2p_\alpha V^{1/2}. \tag{I3}$$

For example, we may choose $\varepsilon = 0.01$ and $\alpha = 0.95$; that is, we require our estimate of $P^c(v_0|u,\boldsymbol{z})$ to be precise within 1% at an $\alpha = 95\%$ confidence level. This in turn translates into a condition on the number of Monte Carlo simulations that must be performed to achieve the required accuracy:

$$n_{MC} \geq 4\frac{p_\alpha^2}{\varepsilon^2}\frac{P^c(v_0|u,\boldsymbol{z})(1 - P^c(v_0|u,\boldsymbol{z}))}{\rho(\boldsymbol{z}|u)\#\boldsymbol{J}}. \tag{I4}$$

For simplicity, this criterion is applied throughout the explanatory variable domain and the maximum value is retained:

$$n_{MC} = \max_{\boldsymbol{z}} \frac{p_\alpha^2}{\varepsilon^2}\frac{P^c(v_0|u,\boldsymbol{z})(1 - P^c(v_0|u,\boldsymbol{z}))}{\rho(\boldsymbol{z}|u)\#\boldsymbol{J}}. \tag{I5}$$

The number of Monte-Carlo samples n_{MC} is thus fixed once α and ε are specified. In an actual case study, it is often necessary to exclude pixels whose probability density $\rho(\boldsymbol{z}|u)$ is too low from the evaluation of Eq. (I5), as these would make n_{MC} ridiculously high from regions in explanatory variable space $\mathcal{D}$ which contribute little to the transition and where $\rho(\boldsymbol{z}|u)$ is badly estimated anyway. To this effect, we used a simple exclusion method: we normalize the $\rho(\boldsymbol{z}|u)$ values of each pixel so that their sum is 1, order them in an ascending manner, compute the cumulative sum and exclude pixels whose cumulative sum falls below a certain threshold (5% for example). Again, the details of such a procedure are not critical as the number of Monte Carlo runs does not need to be specified in a very precise way.

J. Patch construction method (patcher)

In the elaboration of our patch design process (or patcher), we chose to build patches pixel by pixel from a given pivot-cell. The main reason for proceeding in this way is that it simplifies the implementation of the patcher algorithm. This choice is also motivated by a compromise between user control on the patch shape and some level of randomness in the shape construction, in order to give patches a more realistic appearance.

The following properties are expected to hold for any patch construction method:

- The patcher algorithm should not introduce any noticeable bias on the distribution $\rho(\boldsymbol{z}|u,v)$.
- The predefined patch size (as defined at the previous sampling design step, just before the patcher algorithm is called) must be reached.
- A patch elongation parameter is provided.
- The patch merging bias must be avoided (this bias is discussed in section 6.3).

[34] This approximation holds although the transition probability is low, because the typical number of pixels changing state is nevertheless large enough in practice.

- Patches must be "realistic". This is a rather subjective criterion. In practice, this means that very regular patch shapes must be avoided, as these are rarely if ever encountered in actual case studies. The construction method we provide does satisfy this criterion in our estimate, but the user has no or little control on this characteristic.

Concerning the last requirement, ideally, one should also define a measure of patch shape randomness and a related probability distribution that should be estimated on calibration data, and design a bias-free algorithm that would enforce this probability distribution during allocation. However, this level of sophistication is not yet required at the present level of precision of LUCC modeling. This is fortunate, as producing an appropriate bias-free algorithm for a shape randomness parameter would be quite challenging.

Very little attention has been devoted so far in the literature to the question of patch construction. The most elaborate existing patcher algorithm (Dinamica EGO's) enforces the distribution of patch sizes, and allows the user to define some kind of a "non-cirularity" parameter ("isometry"); however this parameter produces an essentially bimodal elongation distribution (see Fig. F1). In this respect, our elongation definition and implementation is more satisfactory, as will become apparent in the remainder of this Appendix.

In principle, both the distribution of patch sizes and elongations may be obtained from calibration data, although quantifying elongations may turn out to be quite computationally intensive. In practice though, our own software (CLUMPY) can either calibrate the distribution of patch sizes, or let the user define the mean and variance of a gaussian probability distribution, a strategy also adopted in Dinamica EGO (with a lognormal distribution instead). Concerning patch elongations, and considering the near absence of attention paid to this parameter in the literature, we do enforce a gaussian distribution, with mean and variance specified by the user. More versatile choices may be implemented in the future, if the need arises. In particular, it is always possible to define more shape parameters if needed.

J.1. Neighborhood structures

If patches are to be constructed from a pivot-cell seed (a strategy adopted here, following Dinamica EGO), it seems natural to take into account the information carried by neighboring pixels in the process. To this effect, it is useful to define first elementary neighborhood structures for every potential seed pixel.

Neighborhood structures are represented by matrices, where any given matrix element corresponds to a neighboring pixel (the pixel of interest sitting at the center of the matrix and represented by $\times$). These matrices have only two possible entries, 0 or 1; 1 means that the corresponding pixel is taken into account in the process, 0 means that it is ignored.

Two neighborhood structures are commonly used. The queen neighborhood structure includes all 8 nearest neighbors around a given pixel:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \times & 1 \\ 1 & 1 & 1 \end{pmatrix}, \tag{J1}$$

whereas the rook neighborhood structure only considers 4 of the nearest neighbors

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & \times & 1 \\ 0 & 1 & 0 \end{pmatrix}. \tag{J2}$$

The terms *queen* and *rook* are clearly inspired by the possible directions of motion of the corresponding pieces in a chess game from the central pixel. More extended neighborhood structures may in principle be considered. This option is not explored in the present work.

This notion of neighborhood structure is exploited in the patch creation algorithm of section J.3. For definiteness, we use the rook structure which is the default one in our model.

J.2. Patch elongation

A patch elongation can be defined by analogy with an equivalent ellipse. Let us recall that, from elementary geometry, the principal moments of inertia of an ellipse, I_x and I_y are related to the ellipse major axis a (along x) and minor axis b (along y) by $I_x = \pi a b^3/4$ and $I_y = \pi b a^3/4$. The ellipse eccentricity itself is given by $\epsilon = (1 - I_x/I_y)^{1/2} = (1 - b^2/a^2)^{1/2}$. This motivates us to define a patch elongation from its moments of inertia.

Generally speaking, any bidimensional shape can be characterized by its successive moments. Let J_p be a set of pixels constituting a patch. The zeroth order moment is equal to the size of the patch (i.e., the number of pixels in the patch):

$$M_{00} = \sum_{j \in J_p} \delta\sigma_j = \#J_p, \tag{J3}$$

where $\delta\sigma_j = 1$ by convention for pixels in the patch (i.e., pixel areas are normalized to unity). The first order moments are directly related to the position of the center of the patch ($x_0^c = M_{10}/M_{00}, x_1^c = M_{01}/M_{00}$):

$$M_{10} = \sum_{j \in J_p} x_0^j, \tag{J4}$$

$$M_{01} = \sum_{j \in J_p} x_1^j, \tag{J5}$$

where (x_0^j, x_1^j) are the spatial coordinates of pixel j. The normalized second order moments are given by:

$$\mu_{20} = \frac{1}{M_{00}} \sum_{j \in J_p} (x_0^j - x_0^c)^2, \tag{J6}$$

$$\mu_{02} = \frac{1}{M_{00}} \sum_{j \in J_p} (x_1^j - x_1^c)^2, \tag{J7}$$

$$\mu_{11} = \frac{1}{M_{00}} \sum_{j \in J_p} (x_0^j - x_0^c)(x_1^j - x_1^c). \tag{J8}$$

The normalized inertia tensor[35] I_p is then defined as:

$$I_p = \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix}. \tag{J9}$$

From this definition, one can introduce the reduced principle moments of inertia of the patch, i.e., the eigenvalues λ_1 and λ_2 of I_p:

$$\lambda_1 = \frac{1}{2}\left(\mu_{20} + \mu_{02} - \left[\left(\mu_{20} - \mu_{02}\right)^2 + 4\mu_{11}^2 \right]^{1/2} \right), \tag{J10}$$

$$\lambda_2 = \frac{1}{2}\left(\mu_{20} + \mu_{02} + \left[\left(\mu_{20} - \mu_{02}\right)^2 + 4\mu_{11}^2 \right]^{1/2} \right). \tag{J11}$$

Clearly, the ratio of the minor to the major axes of the ellipse having the same reduced principle moments of inertia is given by $(\lambda_1/\lambda_2)^{1/2}$ and its eccentricity by $(1 - \lambda_1/\lambda_2)^{1/2}$.

At this point, we deviate somewhat from the equivalent ellipse analogy and choose to define the patch elongation as

$$e = 1 - \left(\frac{\lambda_1}{\lambda_2} \right)^{1/2}. \tag{J12}$$

This definition is adopted as it turns out that the elongation thus defined has convenient features from a numerical point of view. The patch elongation ranges from 0 (for a circular patch) to 1 (for a linear patch).

J.3. Patch design algorithm

The patch design procedure is represented as a workflow on Fig. J1. In the following description, the Roman numerals refer to the numbered steps in this workflow. The process is iterative and begins with the patch pivot-cell. The objective is to produce some randomness in the patch shape, while preserving as best as possible the patch selected size and elongation. This procedure refers to two subprocesses (patch connectedness condition and neighbor elongation score), described right after the algorithm description.

[35] If a patch is seen as an actual 2D massive structure with each pixel mass equal to its area, this definition corresponds indeed to the tensor of inertia defined in classical mechanics.

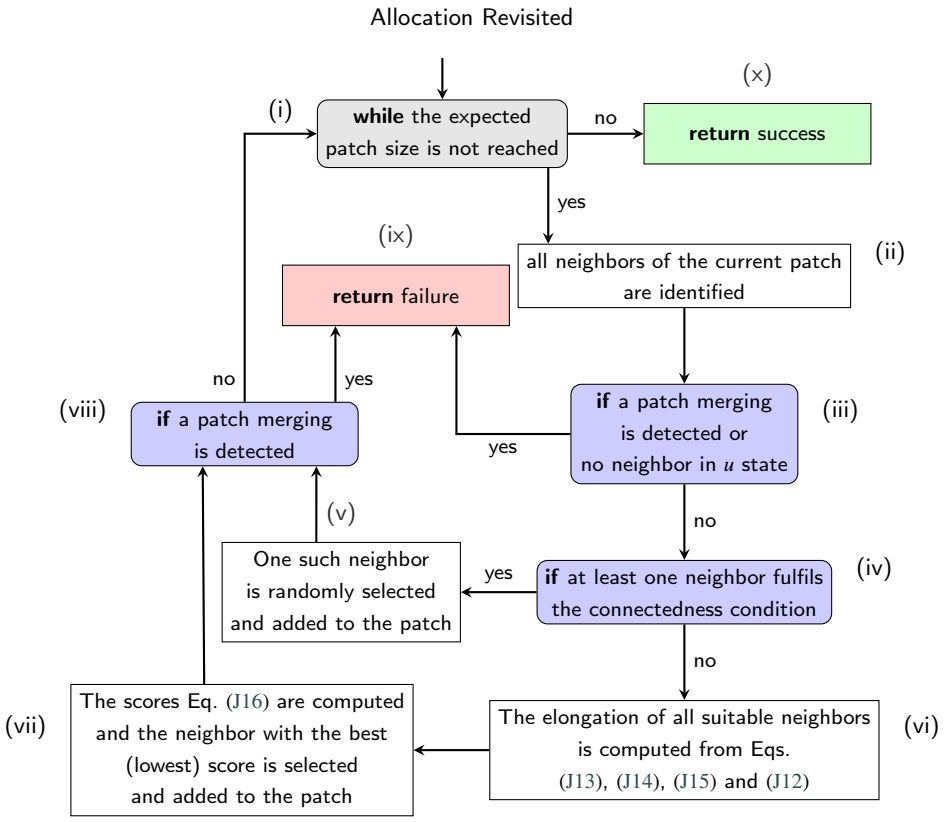


Figure J1: Patch design procedure workflow. Two parameters are required: the previously randomly selected patch size and elongation.

J.3.1. Algorithm description

The loop runs while the expected patch size is not reached (i). As the process goes on, the patch grows pixel by pixel. All neighbors of the current patch at each iteration are identified with the help of the chosen neighborhood matrix defined in section J.1, applied to each pixel of the current patch boundary (ii). Next, all identified neighbors are tested as follows. If at least one neighbor belongs to another patch of the same final LUC state v (iii), a patch merging (defined in section 2.3) has occurred; the procedure is then stopped (failure) and no new patch is created (x). The same outcome follows if no neighbor is in the pivot-cell initial state u (iii and x). Otherwise, there is at least one suitable neighbor, *i.e.* a neighbor with LUC state equal to the pivot cell initial state u (in particular, this excludes pixels who have already undergone another state transition during the same time step). Next, the patch connectedness condition (see below) is checked on all suitable neighbors (iv). If at least one pixel satisfies the connectedness condition, one such pixel (if several) is randomly selected and added to the current patch (v). Otherwise, the patch elongation associated to each pre-selected pixels in step (iv) is computed (vi). Indeed, adding one such neighboring pixel to the patch would change its elongation and one therefore needs to find the most suitable pixel to be added to the patch, if more than one such pixel is available. This is defined on the basis of an elongation score: the pixel with the best (lowest) elongation score is selected and added to the current patch (vii); the computation of the elongation score is described below. At this point, one pixel has been added to the patch, either from the patch connectedness test (v) or from the elongation score one (vii). A final patch merging test is then performed (viii). If a patch merging is detected, the patch is discarded and the procedure stops with a failure outcome (x). Otherwise, the patch growth process starts again until the excepted patch size is reached (i). Finally, the patch is allocated and the algorithm returns a successful outcome (xi).

All tested pixels (ending up in either success or failure) are removed from the pool of potential pixels for a $u \rightarrow v$ transition (set $\boldsymbol{J}$, which therefore is reduced as the iterative allocation method progresses). Whether the patcher procedure ends in success or failure, the allocation method continues as described in section 4.2 and in Fig. 2.

The patch merging test of steps (iii) and (viii) bears only on merging with other patches produced at the same time step. Merging with pre-existing areas does not lead to patch rejection. As a consequence, and if the transition probability is correctly estimated in the vicinity of existing areas (as ensured by our calibration-estimation procedure,

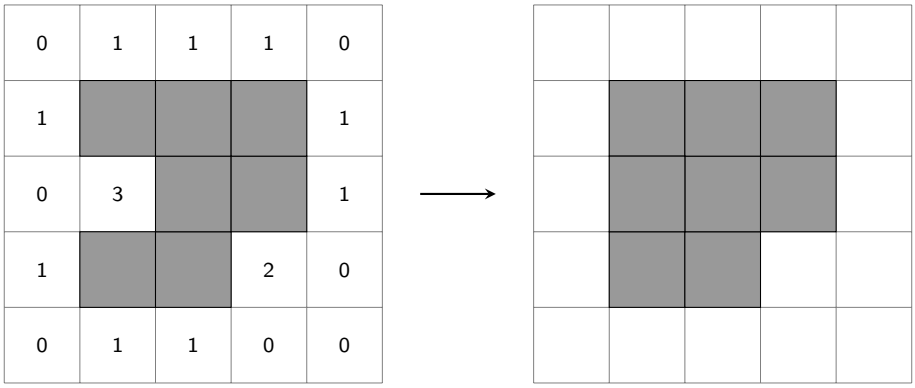


Figure J2: Allocation example on the basis of the connectedness test. The chosen neighborhood structure is rook and the connectedness condition is set to 3.

Mazy and Longaretti 2022b), our patch design procedure can also serve the purpose of the expander patch creation procedure of Dinamica EGO or other software. No specific algorithm is needed for this purpose in our software.

J.3.2. Connectedness test

A "connectedness test" has been implemented in order to prevent the formation of small areas of one or a few non-transited pixels in a patch. To this effect, we consider all pixels that have successfully passed tests (ii) and (iii) in the patch formation procedure just described. The rook neighboring structure is attached to each of these pixels. Next, we determine the number of neighbors n_{rook} in this rook structure, whose LUC state is identical to v (the final state in the transition of interest). We define a threshold number of neighbors n_c. If $n_{rook} \geq n_c$, the pixel under test is selected for allocation in (iv), and one such pixel is randomly selected in (v), if more than one is found.

Fig. J2 presents an example of this test with the threshold set to $n_c = 3$ (our default choice). Obviously, this approach is only efficient for small holes and is inefficient in case of large ones. However, such a shape is very unlikely, due to the elongation condition also included in the patch design procedure. As a matter of fact, we have never observed a failure of the test to achieve its intended purpose (i.e., preventing the formation of holes inside a patch), but we do not provide a formal proof of this statement.

J.3.3. Elongation score test

In case the connectedness test just described has failed (i.e., no new pixel has been added for the purpose of preventing the formation of a hole in the patch), all pixels selected in steps (ii) and (iii) are considered in steps (vi) and (vii). Each such pixel is tested by provisionally adding it to the current patch. This results in a change of elongation of the patch. The new potential patch elongation is computed under the assumption that the patch centroid does not change. Indeed, the addition of a pixel will only slightly change the center of mass (x_0^c, x_1^c) of the patch (unless it is composed of only a few pixels) and making this assumption greatly simplifies the calculations and leads to a more efficient process. For this purpose, for each neighbor j with coordinates (x_0^j, x_1^j), one computes new potential second order moments:

$$\mu_{20}^j = \frac{1}{M_{00}+1}\left(M_{00}\mu_{20} + (x_0^j - x_0^c)^2\right) \tag{J13}$$

$$\mu_{02}^j = \frac{1}{M_{00}+1}\left(M_{00}\mu_{02} + (x_1^j - x_1^c)^2\right) \tag{J14}$$

$$\mu_{11}^j = \frac{1}{M_{00}+1}\left(M_{00}\mu_{11} + (x_0^j - x_0^c)(x_1^j - x_1^c)\right), \tag{J15}$$

where moments M_{20}, M_{02} and M_{11} and centroid coordinates (x_0^c, x_1^c) refer to the current patch state. The neighbor elongation e^j is then computed according to the equation (J12).

Next, in order to decide which among all pixels thus characterized is eventually incorporated in the current patch, a test score is computed. The basic rationale of this score is to take two constraints into account: the closer e^j is to the

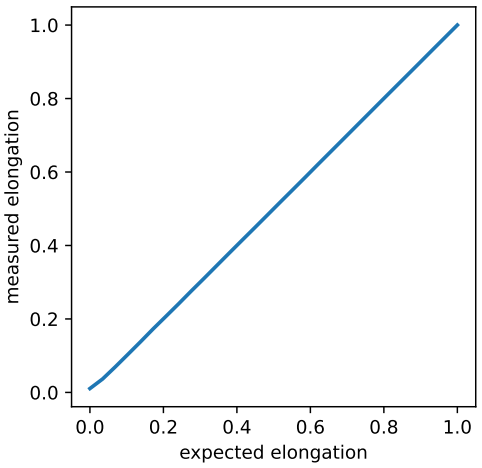


Figure J3: Measured elongation (averaged) of final LUC maps produced by the uPAM on the case study 3, as a function of the expected mean elongation.

targeted elongation patch parameter e, and the higher the transition probability $P(v|u^j, \boldsymbol{z}^j)$ of this neighbor, the most likely this neighbor is to be selected for inclusion in the current patch.

A possible score satisfying these two conditions is the following:

$$\tau^j = \frac{|e - e^j|}{P(v|u^j, \boldsymbol{z}^j)} \tag{J16}$$

where v is the final LUCC state of the LuC transition of interest. The neighbor with the lowest τ score is selected and added to the current patch (vii).

J.3.4. Patch design pseudocode

In addition to the workflow represented in figure J1 and described in the sections above, a pseudocode is given in the following.

J.4. Patch elongation validation

We are now in position to provide a justification of this rather convoluted choice of patch construction for a given elongation. This is done on an example. Namely, we take again the very simple case study of section 5.3.3, except that we now vary the patch elongation parameter from 0 to 1.

Our choice of patch construction and elongation definition allows us to obtain a controlled result, as shown on Fig. J3. The relation displayed on this figure shows the relation between the elongation selected prior to the formation of the patch (expected elongation) and the elongation measured on the actual patch (measured elongation). This relation is an average over a large number of generated patches with the same expected elongation, in order to reduce statistical sources of noise. A nearly perfect equality between expected and measured elongations is obtained. The normalized standard deviation of all measured elongations is not represented but is very low ($< 5 \times 10^{-3}$). This nearly perfect relation between imposed and measured elongation implies that if some patch shape measure based on second order moments were included in the calibration process, our patch construction parameter would be essentially bias-free.

This justification does not uniquely select the procedure we have defined. We merely checked here that this procedure is satisfactory, not necessary.