International Journal of Applied Earth Observation and Geoinformation 44 (2016) 205–216

Contents lists available at ScienceDirect

International Journal of Applied Earth Observation and Geoinformation

journal homepage: www.elsevier.com/locate/jag

A computational framework for generalized moving windows and its application to landscape pattern analysis

Alex Hagen-Zanker

Department of Civil and Environmental Engineering, University of Surrey, Guildford GU2 7XH, United Kingdom

ARTICLE INFO

Article history:
Received 31 May 2015
Received in revised form
23 September 2015
Accepted 23 September 2015

Keywords:
Moving window
Gradient model
Landscape metric
Multi-scale

ABSTRACT

Land cover products based on remotely sensed data are commonly investigated in terms of landscape composition and configuration; i.e. landscape pattern. Traditional landscape pattern indicators summarize an aspect of landscape pattern over the full study area. Increasingly, the advantages of representing the scale-specific spatial variation of landscape patterns as continuous surfaces are being recognized. However, technical and computational barriers hinder the uptake of this approach. This article reduces such barriers by introducing a computational framework for moving window analysis that separates the tasks of tallying pixels, patches and edges as a window moves over the map from the internal logic of landscape indicators. The framework is applied on data covering the UK and Ireland at 250 m resolution, evaluating a variety of indicators including mean patch size, edge density and Shannon diversity at window sizes ranging from 2.5 km to 80 km. The required computation time is in the order of seconds to minutes on a regular personal computer. The framework supports rapid development of indicators requiring little coding. The computational efficiency means that methods can be integrated in iterative computational tasks such as multi-scale analysis, optimization, sensitivity analysis and simulation modelling.

1. Introduction

The field of landscape ecology broadly studies interdependencies between ecological functioning and aspects of landscape pattern. Over the years a wide range of methods and tools have been developed to support such study. An overwhelming share of these methods is based on the patch matrix model (PMM). Introduced by Forman and Godron (1981), this model is based on delineation of the landscape into relatively homogenous sub-areas distinct from their surrounding matrix. Key aspects of ecological functioning are the size of patches, the distances between patches, edge areas that exist between adjacent patches and the existence of networks of patches (Forman and Godron, 1981). Even though originally based on ecological theory and linked to concepts such as species diversity, the PMM and associated methods have been adopted as a more general means of objectively characterizing and comparing patterns of land cover and land cover change, including urban landscapes where ecological concerns are secondary (Herold et al., 2002; Luck and Wu, 2002; Seto and Fragkias, 2005; Wang et al., 2014).

A critical aspect of any landscape analysis is spatial scale, which traditionally is understood to be determined by the spatial extent of the study area and the grain or resolution of its measurement units (Turner, 1989). However, scale is increasingly seen as a characteristic of the analysis of the data, rather than the data itself. Many studies investigate landscape patterns at multiple scales (Chen et al., 2013; Cushman and Landguth, 2010; Fan and Myint, 2014; Johnson et al., 2004; Myint et al., 2015; Plexida et al., 2014; Saint-Geours et al., 2014; Wickham et al., 2007; Zurlini et al., 2007). Moving window analysis is a common approach to such multi-scale analysis. In moving window analysis, each location is associated with the landscape patterns present in the spatial window surrounding it. The size of the window determines the scale of the analysis.

The PMM is widely adopted, but there has been increased recognition of limitations associated with this model. The discrete delineation and categorization of landscape elements is criticized and a gradient perspective, or gradient method (GM), is promoted instead that represents landscapes using continuous spatial variables (Cushman et al., 2010; Lausch et al., 2015; McGarigal and Cushman, 2002). This perspective brings landscape pattern analysis in line with the much longer established gradient analysis (Whittaker, 1967). Characteristically, the outcome of a moving window analysis is not a single scalar describing the overall landscape,

E-mail address: a.hagen-zanker@surrey.ac.uk

http://dx.doi.org/10.1016/j.jag.2015.09.010
1569-8432/

but a new spatial variable that describes how a particular aspect of landscape structure varies over the studied area. Therefore, moving window analysis is a prominent means of developing a gradient perspective on landscape structure, even so if the source data is categorical in nature and based on the PMM.

In a recent discussion and comparison of GM and PMM, Lausch et al. (2015) note that the application of the GM is not yet as widespread as the theoretical benefits would suggest; In their analysis they emphasize that the uptake of GM methods is hindered by issues related to unfamiliarity and technical barriers: "*requires GIS and remote sensing expertise, less intuitive*", "*require [] powerful computer capacity*", "*lack of standardized continuous surface metrics*" (Lausch et al., 2015). The current paper aims to reduce such barriers by introducing a generic computational framework for moving window based analysis that supports and eases the application of GM. The computational framework reduces the complexity of developing new indicators by separating the logic of specific landscape indicators from that of traversing a moving window over the study area. Furthermore, the framework is designed to be computationally efficient; notably, the computational cost scales with the size of the study area but not with the size of the window as a naïve implementation would. A third advantage of the framework is that it facilitates distance weighted moving windows, which are common in geoinformation science (e.g. kernel density estimation) but not normally used in window based analysis of the patch matrix.

This is not the first effort towards the computational support of moving window based analysis of landscape indicators and some notable existing frameworks and tools are: FRAGSTATS (McGarigal et al., 2002), the r.le package in the R language (Baker and Cai, 1992) that works with GRASS (Neteler et al., 2012), and focal statistics (Tomlin, 2013) implemented in various GIS packages. The framework of Estreguil et al. (2014) uses focal statistics as a pre-processing step for further landscape analysis. Hagen-Zanker (2006) presents a generalized approach to moving window based analysis of spatial patterns that are implemented in the Map Comparison Kit software (Visser and de Nijs, 2006). The current paper differs from these earlier approaches by generalizing image processing techniques into a framework that is both flexible and efficient.

The computational framework introduced in this article makes use of well-established image processing techniques such as box-filtering techniques (McDonnell, 1981). The key contribution of this article is a conceptual and pragmatic development from a method to efficiently compute specific moving window based statistics, such as mean and variance to a generic computational framework suitable for any moving window based indicator, or at least a wide variety of indicators. The central idea of the computational framework is to keep running values of a set of variables that make up the state of an indicator as the window moves over the map and pixels, edges and patches come into and go out of view; at any time the value of the indicator can be derived from the state. The framework separates the logic of the moving window from that of the indicator. The moving window logic is about bookkeeping and tracing of the incoming and outgoing elements, whereas the logic of the indicator is limited to updating the state and deriving the indicator value from the state. Under this framework, the development of a new indicator is thus only concerned with the internal logic of the indicator itself and not the bookkeeping surrounding it.

Inversion of control means indicators can be assessed pixel-by-pixel, a step at a time, giving access not only to the calculated indicator value but also the underlying state variables. This is useful, because it allows combining of diverse indicators at a variety of spatial scales, which creates opportunities for distance-weighted moving windows that will be explored in this article.

The computational framework has limitations; most notably it will require indicators that can be computed incrementally.

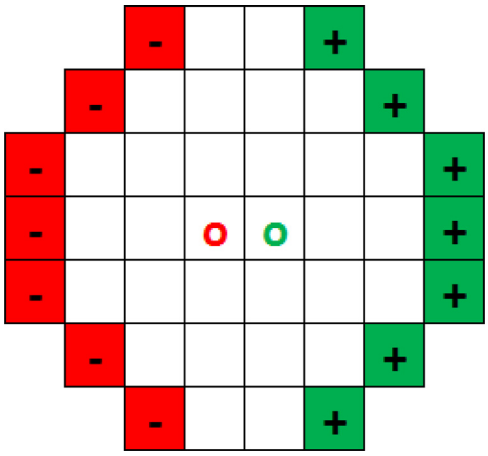

Fig. 1. As the circular window move from left to right, green pixels are added and red pixels are subtracted. The red 'o' marks the center of the window before the move and the green 'o' after the move.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Nevertheless, a wide range of indicators is feasible and to demonstrate, a variety of indicators is implemented. This article will first detail the method that constitute the computational framework, then the specific indicators implemented under the framework and subsequently apply the indicators on urbanization data for the UK and Ireland. This application is intended as a stress-test and a demonstration of the computational framework's ability to aid the interpretation of large remote sensing based land cover products. The discussion will consider limitations and future developments in greater detail.

2. Method

2.1. Moving window and box-filtering techniques

Moving average filter methods are common in image processing (Glasbey and Jones, 1997; McDonnell, 1981). A naïve approach to the moving average filter is to iterate over each pixel in the image and for each pixel in the image iterate over all pixels in the surrounding window to calculate their count and sum and subsequently compute the mean. The computational cost of this approach is $O(NM)$ where N is the number of pixels in the image and M is the number of pixels in each window. This approach is naïve because it fails to take advantage of the circumstance that the window of one pixel largely overlaps with that of the next pixel.

A more efficient algorithm computes the count and summation of pixel values in the window centred on the first pixel. But for the second, and every subsequent pixel it only updates the count and summation by adding the pixels that are in the window surrounding the next, but not the previous one and subtract the pixels that are in the window surrounding the previous, but not the next pixel. Thus, when the window is moving from left to right only the left and right facing pixels on the circumference of the window need to be processed. The cost of this algorithm is $O\left(N\sqrt{M}\right)$. Depending on window size, this can be a huge gain in efficiency compared to the naive approach. The proposed computational framework uses this algorithm for circular windows (Fig. 1).

In the case of rectangular windows (including square windows), a further efficiency gain is made. In this case, the mean over a window is computed as the mean over a number of column elements. As the window moves from one pixel to the next it is not necessary to account for each individual pixel that comes into view, but simply the column element to the right is added and the column element to the left is subtracted. The column elements need to be

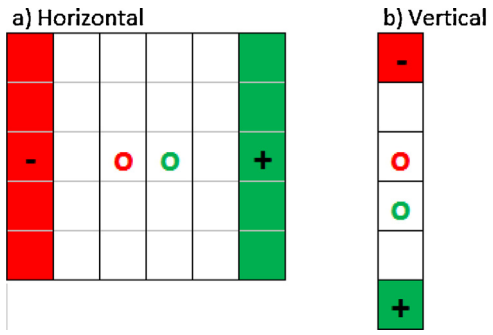

Fig. 2. Square windows are updated in two stages. (a) Horizontal movement is based on the addition and subtraction of column elements. (b) Vertical movement adds and subtracts pixels to and from column elements.

pre-computed for each column in the first row, but subsequently are updated for each next row by adding a pixel to the bottom of the column element and subtracting one from the top. This algorithm for rectangular windows is highly efficient, because for each pixel processed it only needs four operations (Fig. 2). Hence, the computational complexity is $O(N)$ and indeed independent of the size of the window. This method is called box-filtering and detailed by McDonnell (1981).

Variations of these methods exist for octogonal and other polygonal windows (Glasbey and Jones, 1997), but this article only applies the aforementioned algorithms for circular and square windows.

2.2. Generic moving windows

A core principle in software development is to avoid code duplication, this helps avoiding mistakes, eases maintenance, and makes the effort of optimization more economical. From that perspective it follows to implement the algorithms introduced in the previous section in a generic and reusable manner. The algorithms described in the preceding section are for the calculation of the window mean value. However, the basic structure of the algorithm would not be different if a moving window were applied for any other indicator, for instance the mode, variance, maximum value or something more complex. A generic implementation of the algorithm must therefore not be premeditated on one particular indicator, but be applicable for any indicator. Such flexibility can be achieved computationally through polymorphism. Under polymorphism, different types share a common interface which allows them to be used interchangeably. Here, different indicators can be used with the moving window algorithm, provided that they conform to the interface that the moving window algorithm expects. In particular, the framework specifies the following requirements upon an indicator type:

1. A function to initialize the indicator (init);
2. A function to add a pixel value to the indicator (add_element);
3. A function to subtract a pixel value from the indicator (subtract_element);
4. A function to add an indicator of the same type to the indicator (add_subtotal);
5. A function to subtract an indicator of the same type to the indicator (subtract_subtotal);
6. A function to extract the value from an indicator (extract).

It is instructive to consider the 'mean' indicator in terms of these functions. A first observation is that only keeping track of the value of the mean is not sufficient: it is undetermined how the mean value of a window changes as the value of a single pixel is added or subtracted. It is necessary to keep track separately of the number of pixels in the window (count) and their summation (sum). We call these variables that need to be traced the state of the indicator. Given the count and sum, adding or subtracting the value of a single pixel is elementary: count is incremented or decremented by the value 1 and sum is incremented or decremented by the pixel value. The initialization function simply sets count and sum to zero, and the indicator extraction function computes the mean as the quotient of sum and count. Thus, these are all the ingredients for a moving window analysis based on adding and subtracting pixel values. For the case of rectangular windows the further functions to add and subtract subtotals are required, because of the addition and subtraction of column elements. Thus by implementing the mean indicator using the six required functions, it can be used both for square and circular windows. And, by specifying the algorithm for square and circular windows using just the six required functions, they can be used with many other indicators beyond just the mean indicator.

It should be noted that the computational complexity of the algorithms will depend on the complexity of the indicator functions. Therefore the cost will be $O(N \times (\text{Extract} + \text{Add} + \text{Subtract}))$ for the square window variant and $O\left(N \times \left(\text{Extract} + \sqrt{M} \times (\text{Add} + \text{Subtract})\right)\right)$ for the circular window variant. Where Extract, Add and Subtract are the complexity associated with respectively the extract, add and subtract functions. This is relevant as it will mean that the cost of some indicators will increase with the number of categories (i.e. land cover classess).

Programming languages support polymorphism through various paradigms and it is a key feature of object-oriented programming. The implementation for this article is based on static polymorphism using template meta-programming in C++ (Abrahams and Gurtovoy, 2005) where the set of generic requirements upon a type is called a concept.

2.3. Moving windows for patches

A next step is for the framework to be able to account for patches and not just pixels. The approach taken is to use a pre-processing step, similar to that in the r.le plugin (Baker and Cai, 1992). This pre-processing consists of the recognition and enumeration of patches in the input map as well as the computation of patch properties such as their size and perimeter. Finally, an intermediary raster layer is created that associates each pixel with the index of the patch that it belongs to.

It is obvious that as the window moves over the study area, patches come in and out of view. However, the window will not align neatly with patches, and practically at all times some patches will be partially inside the window. The appropriate strategy to deal with such partial patches will depend on the particular application. It is therefore important that the computational framework supports alternative strategies. Three possible strategies are suggested that are all possible within the framework:

1. Only full patches are included in the tally;
2. Partial patches are fully accounted for in the tally;
3. Partial patches are accounted for proportional to their area within the window.

These strategies can be readily accommodated by letting the state of patch based indicators include the area of each patch within the window. As pixels or subtotals are added or subtracted the area of the patch within the window is updated along with any other state variables.

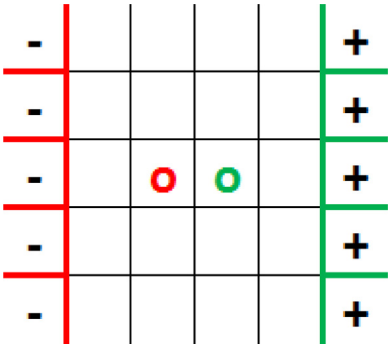

Fig. 3. The moving window updating scheme for a circular window of edges. The window consists of both horizontally and vertically oriented edges.

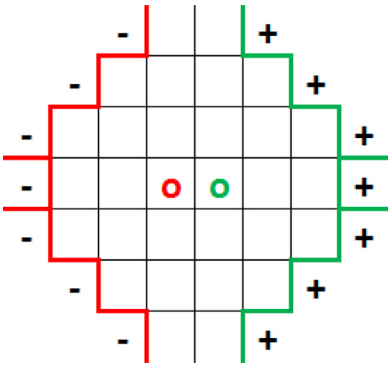

Fig. 4. The moving window updating scheme for a square window of edges makes use of column elements, similar to the square window of pixels.

2.4. Moving windows for edges

A third unit of measurement, after pixels and patches, supported by the framework is edges. Edges are here considered as the face between adjacent pixels. The efficiency considerations are identical to those of the pixel based moving window: a naïve implementation would iterate over each edge in the window surrounding each pixel, a more efficient implementation accounts for the edges that come into and out of the window as its centre moves from one pixel to the next (Fig. 3); and for rectangular windows this is further optimized by pre-aggregating column-elements of edges (Fig. 4). From this perspective, edge windows are not much different from pixel windows, however there are a few complicating factors:

- When is an edge inside the window? When one of its adjacent pixels is within the window, or both? In the application that follows it is assumed that an edge is inside the window when both pixels are inside,
- The tallying and aggregating of edges is more complex as there are both horizontal and vertical edges to take account of.

The generic concept for edge-based indicators is near-identical to that of the pixel-based indicators. The only difference is that the add_element and subtract_element functions do not take pixel values as argument, but edge values. Whereby an edge value consists of the pair of pixel values of either adjacent pixel. The state of an indicator can for instance include and edge matrix detailing how frequent pairs of land cover types are adjacent to each other.

2.5. Implementation as iterators

The information that is computed and maintained in an indicator as the window moves over the study area may be more extensive than the final extracted value. In instances, it can be useful to have access to these intermediate variables. To facilitate this, the framework develops the moving window algorithms as iterators: they move one pixel at a time until instructed to take a further step. This inversion of control makes it possible to interweave a number of iterators and use the state of one indicator as input to another. One application of such interweaving would be to define indicators at a more atomic level, for instance instead of implementing one indicator for the mean, one could implement separate sum and count indicators and combine those to calculate the mean. This article explores another avenue however: by interweaving the computation of indicators at a variety of window sizes, it becomes possible to combine these into distance-weighted indicators.

2.6. Distance and location weighting

The ability to step over the study area pixel-by-pixel and query in each step the state of the indicator is relevant to the ability of providing distance weighted indicators. The idea of these is to calculate indicators for a range of window sizes and to combine the indicators for each pixel using a weight factor for each window size. Thus, the requirements upon the generic indicator concepts are extended to include a weight parameter in the add_subtotal and subtract_subtotal functions.

A further development of the framework is to allow weighting by location; the relative weight of each pixel can be specified in a separate raster layer. Such weighting can have a number of uses, including the option to give pixels outside of the study area a weight of zero. With this additional weight parameter, the generic indicator concept is complete (Table 1).

2.7. Specific indicators

A selection of widely used landscape pattern indicators is implemented according to the concepts described in the preceding sections. The selection of indicators is primarily informed by the intention to present a stress-test for the computational framework; not just by the size of the dataset, but also by evaluating a diverse variety of indicators.

2.7.1. Area-weighted mean patch size

Here, area-weighted mean patch size is calculated for urban patches only. Patches that are only partly within the window are weighted proportional to the area within the window. Notwithstanding, the measurement of the patch area is only constrained by the study area, and not the window size. The following indicator is computed:

$$I_{\text{size},x} = \frac{\sum_i A_{x,i} S_i}{\sum_i A_{x,i}}, \quad (1)$$

where $I_{\text{size},x}$ is the area-weighted mean patch size indicator for the window centred on pixel x The index i iterates over all patches, and A_{xi} is the area of patch i within the window centred on x. S_i is the size of patch i measured in pixel units. The delineation of patches is based on Queen Contiguity, i.e. diagonal neighbours are considered adjacent.

The computational implementation of this indicator preprocesses the input data to delineate patches and tabulate their sizes. The state of the indicator consists of variables for both $\sum_i A_{x,i} S_i$ and $\sum_i A_{x,i}$ and performs the division when the indicator value is extracted from the object.

Table 1
Functions of the indicator concept.

Function name	Parameters	Behavior
init	(–)	Set the state to its neutral value
add_element	element	Update the state by including the effect of the value associated with one pixel or edge
	weight (optional)	The weight parameter is required for location weighting only
subtract_element	element	Update the state by undoing the effect of the value associated with one pixel or edge
	weight (optional)	The weight parameter is required for location weighting only
add_subtotal	element_subtotal	Update the internal state by including the effect of a pre-aggregated group of pixels or edges (i.e. merge)
	weight (optional)	The weight parameter is required for distance weighting only
subtract_subtotal	element_subtotal	Update the internal state by undoing the effect of a pre-aggregated group of pixels or edges (i.e. unmerge)
	weight (optional)	The weight parameter is required for distance weighting only
extract	(–)	Compute the iterator value from the internal state

2.7.2. Patch-weighted mean shape index

Here, patch-weighted mean shape index is calculated for urban patches. The shape index for individual patches takes the following form (Bogaert et al., 2000):

$$C_i = \begin{cases} n^2 = S_i & \rightarrow \frac{P_i}{4n} \\ n^2 < S_i \leq n \times (n+1) & \rightarrow \frac{P_i}{4n+2} \\ n \times (n+1) < S_i & \rightarrow \frac{P_i}{4n+4} \end{cases}, \quad (2)$$

where $n = \lfloor \sqrt{S_i} \rfloor$ is the integer part of the square root of the patch size, P_i is the patch perimeter (in pixel units) and C_i is the patch shape index.

Each patch has the same weight, regardless of size. Nevertheless some will only be partially within the window centred on a given pixel. Therefore, patches are weighted by the proportion to which they are within the window:

$$I_{\text{shape},x} = \frac{\sum_i \frac{A_{x,i}}{S_i} C_i}{\sum_i \frac{A_{x,i}}{S_i}}, \quad (3)$$

where $I_{\text{shape},x}$ is the patch weighted shape index iterator for the window centred on pixel x.The computational implementation of this indicator pre-processes the input data to delineate patches and tabulate their sizes and shape indices. The state of the indicator consists of variables for both $\sum_i \frac{A_{x,i}}{S_i} C_i$ and $\sum_i \frac{A_{x,i}}{S_i}$ and performs the division when the indicator value is extracted from the object.

2.7.3. Shannon diversity

Shannon diversity, is a measure of the variability of the composition of an area. It is based on Shannon's information theoretic concept of entropy:

$$I_{\text{diversity},x} = -\sum_c p_{x,c} \log p_{x,c}, \quad (4)$$

where $I_{\text{diversity},x}$ is the Shannon diversity indicator for the window centred on pixel x, $c$iterates over all classes (e.g. land cover categories) and $p_{x,c}$ is the fraction of the area of the window centred on pixel x that is of class c.

The state of this indicator consists of variable for the total window area (because of boundary effects) and the area per class. The division to find $p_{x,c}$ and the log-sum only take place when the indicator value is extracted from the object.

2.7.4. Most-common class

The most-common class is the class which takes the largest area of the window and is defined as follows:

$$I_{\text{common},x} = arg\max_c p_{x,c}, \quad (5)$$

where $I_{\text{common},x}$ is the most-common class indicator for the window centred on pixel x.

The state of the indicator consists of variables to keep track of the area per class and keeps those in a continuously updated priority queue (Knuth, 2005 (vol. 3)). The indicator value is then determined by the class in front of the queue.

2.7.5. Edge density

Edge density measures the proportion of non-like pixel adjacencies over all pixel adjacencies:

$$I_{\text{edge},x} = \frac{\sum_c \sum_d E_{x,c,d} - \sum_c E_{x,c,c}}{\sum_c \sum_d E_{x,c,d}}, \quad (6)$$

where $I_{\text{edge},x}$ is the edge density indicator for the window centred on pixel x; $c$and $d$iterator over the classes and $E_{x,c,d}$ is the number of edges within the window centred on pixel $x$that face both a $c$and $d$class pixel. This means edges are double counted (as $E_{x,c,d} = E_{x,d,c}$) and for consistency $E_{x,c,c}$ is *twice* the number of edges within the window that face a $c$class pixel on both sides.

The state of this indicator consists of separate tallies of $\sum_c \sum_d E_{x,c,d} - \sum_c E_{x,c,c}$ and $\sum_c \sum_d E_{x,c,d}$ and updates these for every edge that is added to or subtracted from the window.

2.7.6. Location and distance weighting

The indicators described before do not make use of location and distance weighting. However it is a straightforward extension when the (additional) weights are applied on the linear components in the state of the different indicators, or area weights are replaced by the sum of weight over the area (Table 2). Note that all location weighted variants are extensions: using uniform weights the variants are identical to the original. This article does not make intensive use of location weighting, however binary weights are used as a means to compute $I_{\text{size},x}$ and $I_{\text{shape},x}$ only for patches of class 'urban'.

2.8. C++ library

The computational framework is implemented as a C++ library. The library provides moving window algorithms for circular and square windows. The library is light-weight in the sense that it

Table 2
Location and distance weighted extensions.

Indicator	Unweighted	Location weighted	Location and distance weighted
$I_{\text{size},x}$	$\frac{\sum_i A_{x,i} S_i}{\sum_i A_{x,i}}$	$\frac{\sum_i w_{x,i} A_{x,i} S_i}{\sum_i w_{x,i} A_{x,i}}$	$\frac{\sum_k w_k \times \left(\sum_i w_{x,i} A_{x,i} S_i \right)_k}{\sum_k w_k \times \left(\sum_i w_{x,i} A_{x,i} \right)_k}$
$I_{\text{shape},x}$	$\frac{\sum_i \frac{A_{x,i}}{S_i} C_i}{\sum_i \frac{A_{x,i}}{S_i}}$	$\frac{\sum_i \frac{w_{x,i}}{W_i} C_i}{\sum_i \frac{w_{x,i}}{W_i}}$	$\frac{\sum_k w_k \times \left(\sum_i \frac{w_{x,i}}{W_i} C_i \right)_k}{\sum_k w_k \times \left(\sum_i \frac{w_{x,i}}{W_i} \right)_k}$
$I_{\text{diversity},x}$	$-\sum_c p_{x,c} \log p_{x,c}$	$-\sum_c \frac{w_{x,c}}{W_x} \log \frac{w_{x,c}}{W_x}$	$-\sum_c \frac{\sum_k w_k \times (w_{x,c})_k}{\sum_k w_k \times (W_x)_k} \log \frac{\sum_k w_k \times (w_{x,c})_k}{\sum_k w_k \times (W_x)_k}$
$I_{\text{common},x}$	$\arg\max_c p_{x,c}$	$\arg\max_c \frac{w_{x,c}}{W_x}$	$\arg\max_c \frac{\sum_k w_k \times (w_{x,c})_k}{\sum_k w_k \times (W_x)_k}$
$I_{\text{edge},x}$	$\frac{\sum_{x,d:c \neq d} E_{x,c,d}}{\sum_{c,d} E_{x,c,d}}$	$\frac{\sum_{x,d:c \neq d} E^w_{x,c,d}}{\sum_{c,d} E^w_{x,c,d}}$	$\frac{\sum_k w_k \times \left(\sum_{x,d:c \neq d} E^w_{x,c,d} \right)_k}{\sum_k w_k \times \left(\sum_{c,d} E^w_{x,c,d} \right)_k}$

Table note: the notation for $I_{\text{edge},x}$ is shortened to fit into the table. $w_{x,i}$ is the total weight of the area of patch i within the window. W_i is the total weight of patch i,$w_{x,c}$ is the total weight of the area of class c within the window. W_x is the total weight of the area within the window, $E^w_{x,c,d}$ is total weight of edges facing c and d. $E^w_{x,c,c}$ is twice the total weight of edges facing c on both sides. w_k is the weight for window k,and $(\cdots)_k$ means that the subscript k is omitted for all variables within the brackets.

Table 3
Indicative computation time on a basic laptop.

(a). Circular windows, computation time (s)

Radius	Size (2)	Shape (2)	Diversity (2)	Diversity (4)	Common (2)	Common (4)	Edge (2)
10	95	90	23	30	43	39	498
20	150	157	43	47	69	72	891
40	259	258	78	83	133	136	1732

(b). Square windows, computation time (s)

Radius	Size (2)	Shape (2)	Diversity (2)	Diversity (4)	Common (2)	Common (4)	Edge (2)
10	37	41	7.5	12	8.3	11	29
20	39	42	8.3	14	8.2	11	30
40	42	45	8.8	15	8.3	11	29
80	40	44	10	15	8.2	11	31
160	42	47	10	15	8.3	11	31

(2) Indicates that the analysis took place on the 2-category map, (4) means the 4-category map. The time required is for two moving window analyses (for the 2000 and 2006 datasets) the computation of the difference, reading inputs and writing outputs. The computer used is an Intel Core i5-4310CPU @ 2.70 GHz.

has limited dependencies on other libraries: it relies on the Geodata Abstraction Library (www.gdal.org) and a number of BOOST libraries (www.boost.org). The library is header-only which means that users only need to build and install GDAL and BOOST that are both widely used and well-documented. The library makes use of modern standard C++ language features (C++11) and therefore needs to be compiled by an up-to-date compiler, for instance Microsoft Visual Studio 2012. The library is available as open source under a highly permissive license (https://github.com/ahhz/moving_window).

3. Application and results

The computational framework is tested on the CORINE land cover dataset of the UK and Ireland for 2000 and 2006 (Büttner, 2014). This data consists of 3388×4628 pixels (15.7 megapixel) and has been reclassified to a dataset of four classes: urban, nature, agriculture, and water; which in turn has been reclassified urban/non-urban for this application. All methods are applied on the full dataset covering the UK and Ireland, for the discussion of results however this section will focus on a smaller area centered on London (Fig. 5). The CORINE dataset like any land cover dataset contains errors; moreover it is based on observations and analysis at a specific spatial scale. The consequences of these limitations are beyond the scope of this article and the data is considered as-is. Data errors are most likely to affect the analysis of change over time, as relatively little change occurs in the period 2000–2006 and hence a large share of the observed change may be due to errors (Fig. 6).

These raster maps have been processed using the framework described in this article to create the scale-specific maps of the various indicators. Each moving window indicator surface is computed both for the 2000 and 2006 dataset and subsequently the cell-by-cell difference between the two indicator surfaces is computed. The computation took place on a regular laptop computer and the required time for processing each indicator was recorded (Table 3). The required computation time is in accordance with the theoretical expectation that computation cost of circular window scale in proportion to the radius of the window, whereas square window computations are independent of window size. The results confirm that the computation time required for the Shannon diversity and most-common class indicators is sensitive to the number of classes and this has significant impact on the computation time for square windows. The edge density indicator is relatively expensive: the computation using a circular window at a radius of 40 pixels (10 km) takes close to 30 min, for a square window it is 30 s.

Table 4
Benchmark comparison to Fragstats.

	Fragstats, square window 41×41 (min)	Proposed square 41×41 (min)	Proposed circular $r = 20$ (min)
Edge density	122	0.20	7.8
Mean patch size	119	0.24	0.58

A key motivation for the presented computational framework is increased computational efficiency. We therefore compared the computation time with that of FRAGSTATS version 4.2 (McGarigal et al., 2012). As expected efficiency gains are much greater for square windows than circular windows (Table 4). Furthermore the efficiency gain for edge-density methods is less than the patch size method. In the worst case the efficiency gain is a factor 9 and in the best case a factor 600.

The edge density indicator surfaces for circular windows at multiple scale show that over time the edge density has changed, but not uniformly so. The surfaces allow identifying areas of increase and decrease of edge density, moreover these areas differ depending on the scale at which the analysis takes place (Fig. 7). At all scales it is obvious that edge density decreases within the urban boundaries of London. At the outskirts the pattern of change is diverse and noisy at the scale of 2.5 km, at 5 km scale the pattern is also diverse but shows a regular intermittent pattern of edge density decrease and increase. At 10 km scale only little change in edge density at the outskirts of London is recorded.

Fig. 8 presents the same analysis as Fig. 7 but based on square instead of circular windows. The results are qualitatively very similar, and roughly the same areas and spatial scales of edge density change are identified. It is also apparent however, that the edge-density surfaces include discontinuities that directly reflect the shape and size of the windows. These border effects are most pronounced in the 2000–2006 difference maps based on the 10 km radius of both the square and circular windows. Such discontinuities may be considered artefacts of the discrete delineation of the windows.

An alternative that prevents or reduce the effects of the discrete delineation of the window is the use of a distance-weighted window, whereby further distance bands contribute less to the particular landscape indicator. For the edge density indicator, such distance weighted indicators are presented in Fig. 9. Indeed, just using three distance bands (Table 5) almost complete removes the window-boundary artefacts.

Further results with a variety of indicators help identifying spatial and scale specific variability in spatial structure. The results for area weighted patch size (Fig. 10a) are a reminder of the degree to

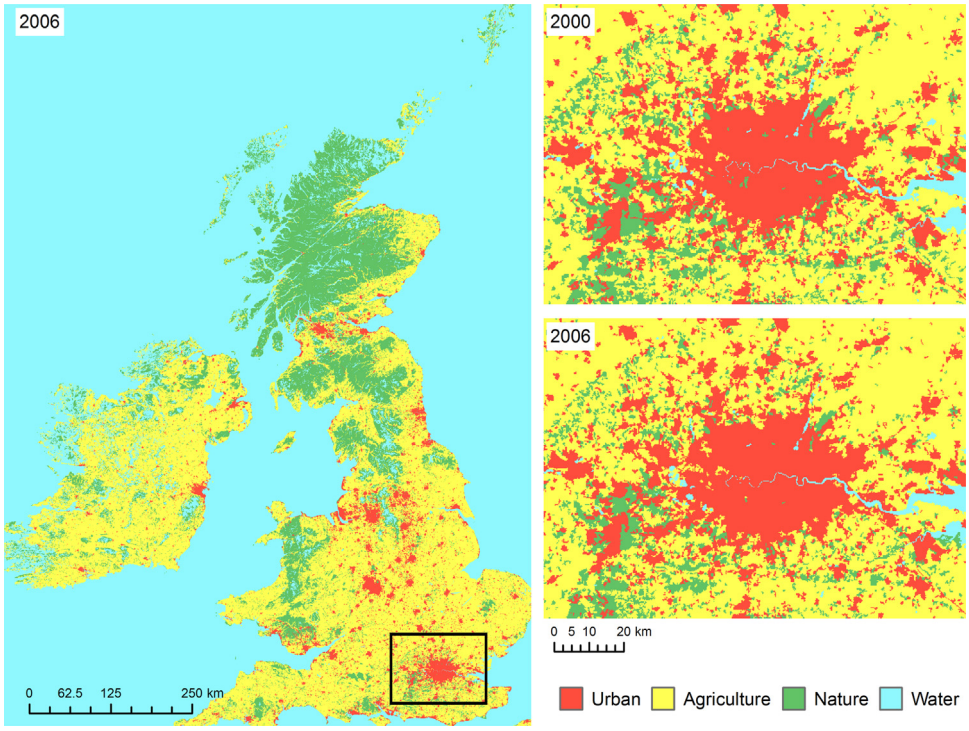

Fig. 5. Test dataset: land cover clipped from EU CORINE and reclassified to four classes.

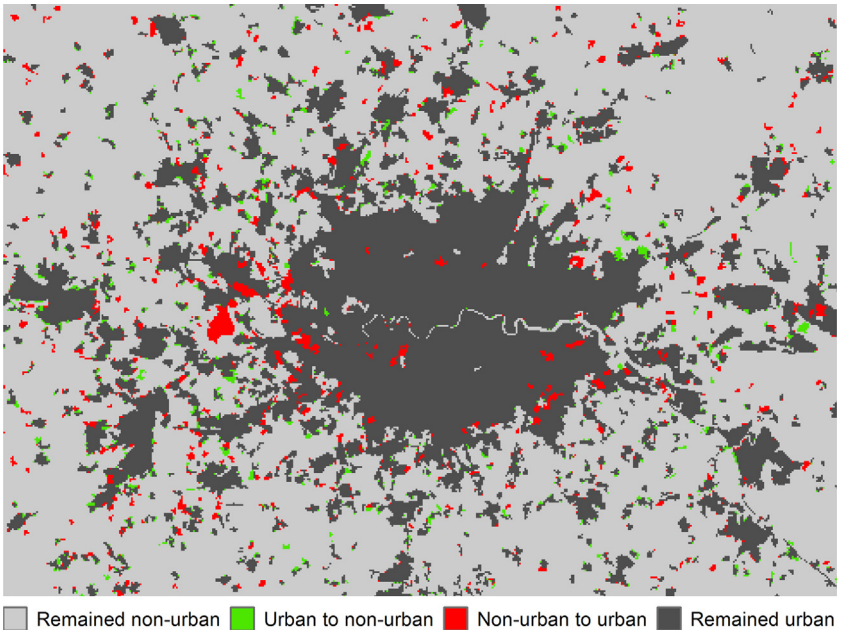

Fig. 6. Change in urban extent over the period 2000–2006.

Table 5
Distance bands and weights for the distance weighted edge density indicator.

Distance range (km)	Weight
0–2.5	0.4
2.5–5	0.2
5–10	0.1

which large patches dominate this indicator. The results for most common class shows that the method are not limited to gradient analysis of landscape structure, but can be a means of map generalization as well (Fig. 10b). The results for Shannon diversity indicate a stark contrast between the Northeast and Southwest segments of the map (Fig. 10c).

4. Discussion

The results confirm the potential and relevance of moving window analysis to quantifying the spatial variation in landscape pattern at multiple scales. It is clear in the results that landscape patterns are not uniform in space, and for instance reporting just on the landscape level change in edge density, Shannon Diversity or patch size would be misleading.

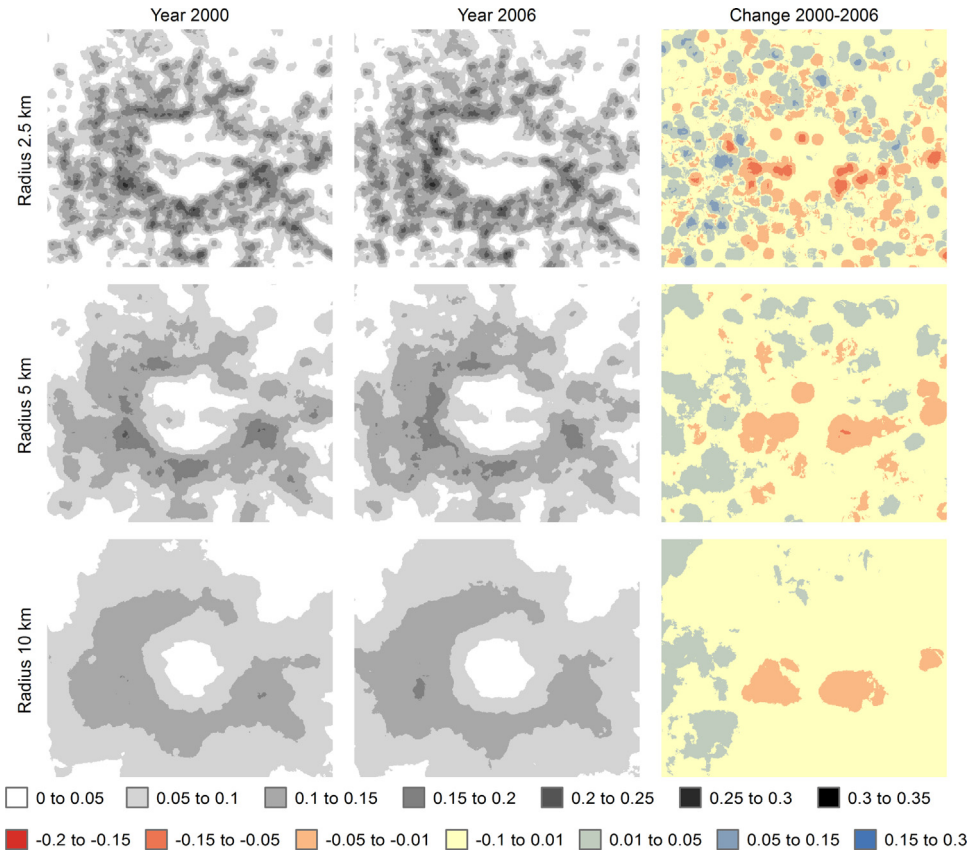

Fig. 7. Multi-scale analysis of edge density and change over time based on circular windows.

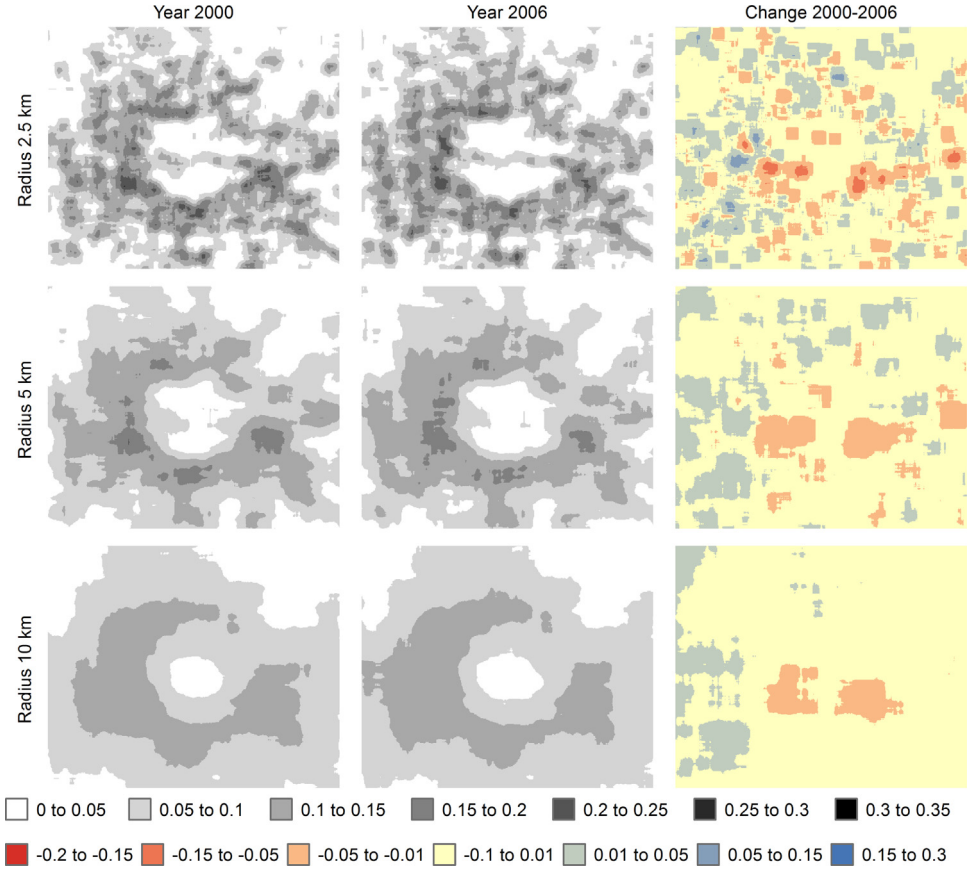

Fig. 8. Multi-scale analysis of edge density and change over time based on square windows.

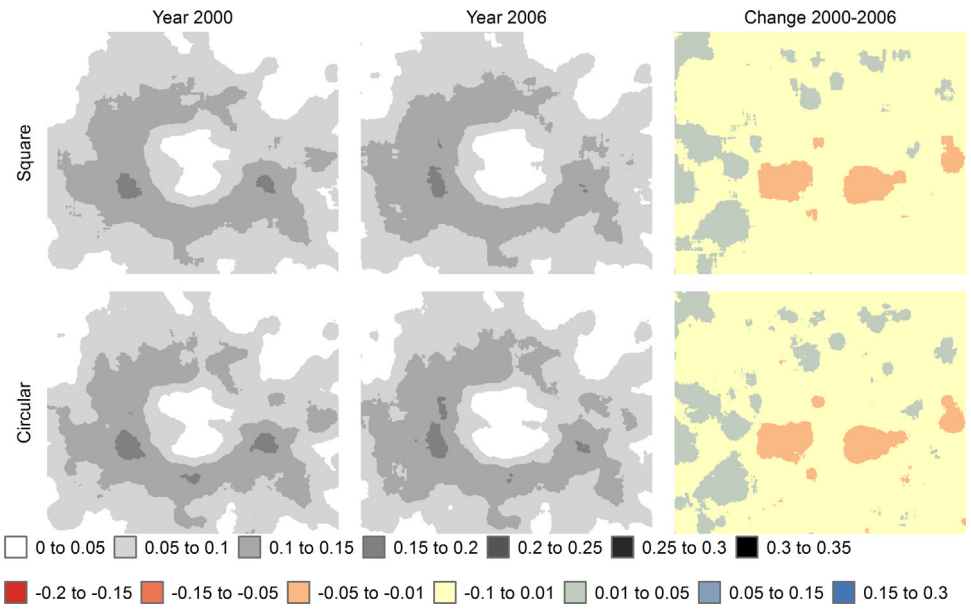

Fig. 9. Edge density analysis based on distance weighted window.

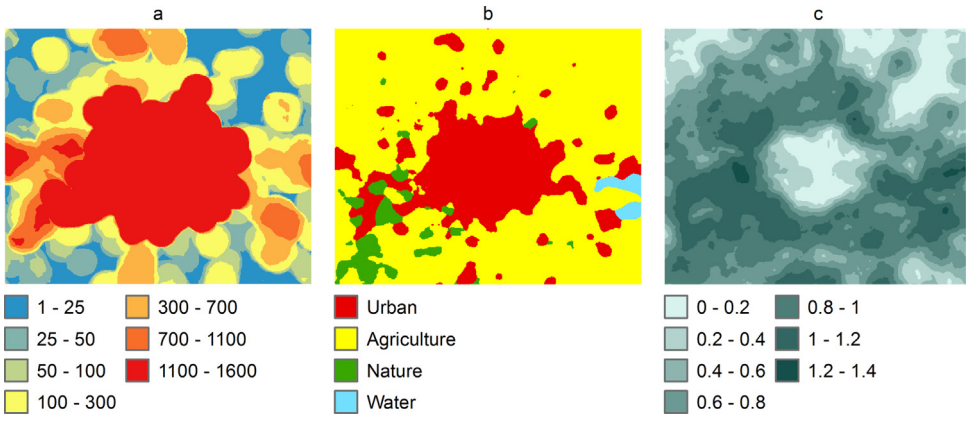

Fig. 10. Three results of scale-specific landscape indicators. (a) Area-weighted patch size, (b) Most-common class, (c) Shannon diversity.

Moving windows can adequately be used to quantify landscape pattern at a particular scale. However, it must be recognized that the discrete delineation of the window boundary is almost always an artefact. In reality, gradual or fuzzy boundaries will be more appropriate, and in recognition of the first law of geography (Tobler, 1970) a distance decay function will generally be preferred. Therefore, distance-weighted moving windows may be recommended as a more sensible alternative to the unweighted moving windows that are currently near-exclusively used.

Time is a relevant factor in doing quantitative research on high resolution data products. The strong fit between square and circular windows is therefore relevant. Square windows can be evaluated within a fraction of the time of circular windows. This has the potential to change the nature and workflow of scale specific analysis of landscape structure. Turnarounds in the order of seconds, means that landscape patterns can be analysed in a much more explorative and interactive fashion than before. The provision of square and circular windows as well as optional use of distance bands means that informed decisions can be made about the trade-off between precision and computational efficiency.

When compared to FRAGSTATS the de-facto standard in landscape pattern analysis, the proposed methods offered a speed-up of up to a factor 600 for a window size of 41×41 pixels. Importantly however, the proposed method offer an improvement in the computation complexity, meaning that for smaller window sizes the improved efficiency in relative terms will be smaller, but for larger window sizes it will be greater.

The implementation of FRAGSTATS can be described as brute force: it derives an input map separately for each window centred on each cell and applies its methods to that window. The r.le package is computationally savvier: it recognizes that some of the computational analysis is common over all windows and uses that to reduce computational cost; in particular it only identifies patches once for the whole study are and reuses this information for the analysis of each separate window. However it does not present a systematic approach to reducing computation complexity as the proposed framework does. GIS packages typically provide moving window functionality, for instance ArcGIS (Ormsby, 2010) has a tool called Focal Statistics, these tools are generally very efficient and make use of the same principles as the proposed framework, and a number of studies make use of this type of analysis for landscape analysis (Willemen et al., 2012; Zurlini et al., 2007). The disadvantage of these GIS tools is that they only support a limited range of window statistics, notably mean and variance and consequently limit the scope of the analysis.

An alternative to moving windows, that is also highly efficient is to evaluate the landscape as a lattice of non-overlapping sub-landscapes (Jasiewicz et al., 2014). The scale of the analysis is then determined by the size of the sub-landscapes. That approach is more sensitive however to discretization effects and not as amenable to multi-scale analysis, because the resolution of the result maps is determined by the scale of the analysis.

One advantage of the proposed computational framework is that it allows for the definition of indicators in a precise mathematical sense, even though the framework consists of a set of algorithms, i.e. procedural complexes of iteration, counting, addition and subtraction implemented in some programming library. It is an important benefit because it allows users to focus on the conceptual meaning of indicators rather than issues of computation. Such clarity was for instance not present in earlier approaches (Baker and Cai, 1992; Hagen-Zanker, 2006).

The main limitation to the computational framework presented here is that it relies on indicators that are computed by increments and decrements. This requires the tallying operations to be commutative, associative and invertible, which we will now call conformant. Many indicators are not immediately conformant and intermediate values called the state of the indicator need to be introduced. The computation of the indicator is then split into conformant (the various add and subtract functions) and non-conformant operation (the initialization and extract functions). Theoretically this is a highly robust framework: for the most stubbornly non-conformant indicators we can simply let all computation take part in the extract function and merely let the state consist of a matrix of copied pixel values. Pragmatically, however, this would undo the computational gains. The efficiency of the computational framework relies on the ability to efficiently express the computation of the indicators using the functions prescribed in the concept.

This separation of tasks comes at some cost. For instance, the tallying of frequencies per class means that the updating functions need to iterate over the classes making their cost proportional to the number of classes. Likewise, if a full edge matrix is part of the state, updating will be proportional to the number of classes squared. This explains the observed sensitivity of the computation time to the number of classes (Table 3).

The pre-processing of the map of classes into patch indices linked to a table of patches properties can be seen as a means of moving the non-conformant part of the computation to the initialization step. Through these mechanisms it proved possible to implement a range of indicators with diverse non-linear properties without losing the efficiency of the moving window algorithms. Future implementations of indicators will face this problem, but the experience with the indicators presented here gives confidence that solutions can be found.

One particular type of indicators that may prove challenging is formed by those based on distances between patches or pixels. Possible solutions to this problem are to model distances either as vectors or as properties of pairs of patches, which can be tallied in a similar way to patches in this article. Such an extension of the computational framework will be a topic of further research.

Other future developments, besides obvious applications and extensions with further indicators, are other moving windows algorithms. For example, there seem to be no substantial barriers to developing a 2D Gaussian filter that works efficiently with the same indicators applied in this article. Likewise, it would be of high interest to extend the framework to more formal multi-scale analysis, such as wavelet and Fourier analysis.

The framework is implemented as an open source C++ library and developed with the user in mind. A simple indicator can be developed in approximately 50 lines of code, and a program, applying an indicator with a moving window can be as short as 10 lines. The prerequisites for using the library are modest: a modern C++ compiler and the installation of BOOST and GDAL, two widely used libraries. Therefore the framework has good potential to become integrated in existing or future modelling frameworks.

Notwithstanding the fundamental nature of the framework, it facilitates types of analysis that are currently out of reach of most researchers. It becomes possible to study the variability of landscape patterns using a distance weighted window for large study areas measured at a fine resolution. The computational efficiency means that methods can conveniently be integrated in iterative computational tasks such as optimization, sensitivity analysis and simulation modelling or interactive procedures such as (map algebra) raster calculators in GIS.

5. Conclusion

This article proposes a computational framework for moving window analysis of landscape pattern. It facilitates a type of analysis that is widely seen as a promising but underexplored avenue for landscape ecology research (McGarigal and Cushman, 2002). It does so by reducing computational and complexity barriers that are recognized and still persist today (Lausch et al., 2015). The framework should be uncontroversial because it builds on fundamental concepts in landscape ecology and geography information science and therefore has a wide range of potential applications in particular in the interpretation of remote sensing based land cover products.

Acknowledgements

Feedback from two anonymous reviewers has been tremendously helpful. The measuring icon used in the graphical abstract is made by Freepik and licensed under Creative Commons BY 3.0.

References

Abrahams, D., Gurtovoy, A., 2005. C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond. Addison-Wesley, Boston.
Baker, W.L., Cai, Y., 1992. The r.le programs for multiscale analysis of landscape structure using the GRASS geographical information system. Landsc. Ecol. 7 (4), 291–302.
Bogaert, J., Rousseau, R., Van Hecke, P., Impens, I., 2000. Alternative area-perimeter ratios for measurement of 2D shape compactness of habitats. Appl. Math. Comput. 111 (1), 71–85.
Büttner, G., 2014. CORINE Land Cover and Land Cover Change Products. In: Manakos, I., Braun, M. (Eds.), Land Use and Land Cover Mapping in Europe. Springer, Netherlands, pp. 55–74.
Chen, Y., Li, X., Liu, X., Ai, B., 2013. Modeling urban land-use dynamics in a fast developing city using the modified logistic cellular automaton with a patch-based simulation strategy. Int. J. Geogr. Inf. Sci. 28 (2), 234–255.
Cushman, S., Landguth, E., 2010. Scale dependent inference in landscape genetics. Landsc. Ecol. 25 (6), 967–979.
Cushman, S.A., Gutzweiler, K., Evans, J.S., McGarigal, K., 2010. The gradient paradigm: a conceptual and analytical framework for landscape ecology. In: Cushman, S.A., Huettmann, F. (Eds.), Spatial Complexity, Informatics, and Wildlife Conservation. Springer, Japan, pp. 83–108.
Estreguil, C., De Rigo, D., Caudullo, G., 2014. A proposal for an integrated modelling framework to characterise habitat pattern. Environ. Model. Softw. 52, 176–191.
Fan, C., Myint, S., 2014. A comparison of spatial autocorrelation indices and landscape metrics in measuring urban landscape fragmentation. Landsc. Urban Plan. 121, 117–128.
Forman, R.T., Godron, M., 1981. Patches and structural components for a landscape ecology. Bioscience 31 (10), 733–740.
Glasbey, C.A., Jones, R., 1997. Fast computation of moving average and related filters in octagonal windows. Pattern Recog. Lett. 18 (6), 555–565.
Hagen-Zanker, A., 2006. Map comparison methods that simultaneously address overlap and structure. J. Geogr. Syst. 8 (2), 165–185.
Herold, M., Scepan, J., Clarke, K.C., 2002. The use of remote sensing and landscape metrics to describe structures and changes in urban land uses. Environ. Plan. A 34 (8), 1443–1458.
Jasiewicz, J., Netzel, P., Stepinski, T.F., 2014. Landscape similarity, retrieval, and machine mapping of physiographic units. Geomorphology 221, 104–112.
Johnson, C., Boyce, M., Mulders, R., Gunn, A., Gau, R., Cluff, H.D., Case, R., 2004. Quantifying patch distribution at multiple spatial scales: applications to wildlife-habitat models. Landsc. Ecol. 19 (8), 869–882.

Knuth, D.E., 2005. The Art of Computer Programming. Addison-Wesley, Upper Saddle River, NJ.
Lausch, A., Blaschke, T., Haase, D., Herzog, F., Syrbe, R.-U., Tischendorf, L., Walz, U., 2015. Understanding and quantifying landscape structure—a review on relevant process characteristics, data models and landscape metrics. Ecol. Model. 295, 31–41.
Luck, M., Wu, J., 2002. A gradient analysis of urban landscape pattern: a case study from the Phoenix metropolitan region, Arizona, USA. Landsc. Ecol. 17 (4), 327–339.
McDonnell, M., 1981. Box-filtering techniques. Comput. Graphics Image Process. 17 (1), 65–70.
McGarigal, K., Cushman, S.A., 2002. The gradient concept of landscape structure: or, why are there so many patches. In: Wiens, J., Moss, M. (Eds.), Issues and perspectives in landscape ecology. Cambridge University Press, pp, pp. 112–119.
McGarigal, K., Cushman, S.A., Ene, E., 2012. FRAGSTATS v4: Spatial Pattern Analysis Program for Categorical and Continuous Maps. University of Massachusetts, Amherst.
McGarigal, K., Cushman, S.A., Neel, M.C., Ene, E., 2002. FRAGSTATS: spatial pattern analysis program for categorical maps.
Myint, S.W., Zheng, B., Talen, E., Fan, C., Kaplan, S., Middel, A., Smith, M., Huang, H.-P., Brazel, A., 2015. Does the spatial arrangement of urban landscape matter? Examples of urban warming and cooling in Phoenix and Las Vegas. Ecosyst. Health Sustain. 1 (4), art15.
Neteler, M., Bowman, M.H., Landa, M., Metz, M., 2012. GRASS GIS: a multi-purpose open source GIS. Environ. Model. Softw. 31, 124–130.
Ormsby, T., 2010. Getting to Know ArcGIS Desktop, 2nd ed. ESRI Press, Redlands, Calif.
Plexida, S.G., Sfougaris, A.I., Ispikoudis, I.P., Papanastasis, V.P., 2014. Selecting landscape metrics as indicators of spatial heterogeneity—a comparison among Greek landscapes. Int. J. Appl. Earth Observ. Geoinf. 26, 26–35.
Saint-Geours, N., Bailly, J.-S., Grelot, F., Lavergne, C., 2014. Multi-scale spatial sensitivity analysis of a model for economic appraisal of flood risk management policies. Environ. Model. Softw. 60, 153–166.
Seto, K.C., Fragkias, M., 2005. Quantifying spatiotemporal patterns of urban land-use change in four cities of China with time series landscape metrics. Landsc. Ecol. 20 (7), 871–888.
Tobler, W.R., 1970. A computer movie simulating urban growth in the Detroit region. Econ. Geogr. 46 (Suppl), 234–240.
Tomlin, C.D., 2013. GIS and Cartographic Modeling. Esri Press, Redlands, Calif.
Turner, M.G., 1989. Landscape ecology: the effect of pattern on process. Annu. Rev. Ecol. Syst. 20, 171–197.
Visser, H., de Nijs, T., 2006. The map comparison kit. Environ. Model. Softw. 21 (3), 346–358.
Wang, W.J., He, H.S., Spetich, M.A., Shifley, S.R., Thompson, F.R., Dijak, W.D., Wang, Q., 2014. A framework for evaluating forest landscape model predictions using empirical data and knowledge. Environ. Model. Softw. 62, 230–239.
Whittaker, R.H., 1967. Gradient analysis of vegetation. Biol. Rev. 42 (2), 207–264.
Wickham, J.D., Riitters, K.H., Wade, T.G., Coulston, J.W., 2007. Temporal change in forest fragmentation at multiple scales. Landsc. Ecol. 22 (4), 481–489.
Willemen, L., Veldkamp, A., Verburg, P.H., Hein, L., Leemans, R., 2012. A multi-scale modelling approach for analysing landscape service dynamics. J. Environ. Manag. 100, 86–95.
Zurlini, G., Riitters, K.H., Zaccarelli, N., Petrosillo, I., 2007. Patterns of disturbance at multiple scales in real and simulated landscapes. Landsc. Ecol. 22 (5), 705–721.