A Parallelized Approach of The Moving Window & Multi-resolution Raster Correlation Indices For Geospatial Problems Within The Python Language

Owen Smith

December 4, 2020

Code is available at https://github.com/ocsmit/mwinpy

Abstract

Common pixel by pixel or cell by cell comparison methods often fail to take into account necessary spatial information with in raster data. Thus, spatially aware comparison indices are key for important research and analysis which utilizes categorical raster datasets. Costanza's moving window and multi-resolution index provide a base with which comparison indices can be computed for temporal categorical datasets. However, there is no maintained system which provides these indices and any base level implementation will be inherently slow due to the linearly iterative nature of the indices. Proposed are two algorithms which can effectively split data between parallel CPU cores and speed up computation for both similarity indices exponentially. Additional tests and figures are presented detailing outputs and capabilities within a common scripting language for GIS.

1 Introduction & background

Within geospatial research, categorical raster datasets are widely used as they are mosaics and matrices reflective of real world data. Common examples of categorical spatial data includes the National Land-cover Database (NLCD) (Homer et al., 2004) or the State Soil Geographic data base (STATSGO) (Schwarz and Alexander, 1995). Additionally the creation and analysis of categorical raster data is a key focuses of much research such as land cover & land use (LULC) mapping (Gonçalves et al., 2011; Hansen et al., 2000), forest disturbance mapping (Senf and Seidl, 2020; Oeser et al., 2017), and for landscape ecology (Dale, 2000; Riitters, 2019). Within the field of landscape ecology in particular, understanding spatial patterns is key to proving spatial non-randomness and analyzing the relationships between geography and the agents of ecological change (Perry et al., 2002). Common methods for analyzing categorical raster data typically utilize pixel by pixel approaches and are effective in many cases when comparing coarse data sets such as land use generated from Moderate Resolution Imaging Spectroradiometer (MODIS) imagery (Friedl et al., 2010). However when comparing change in higher resolution categorical data sets, pixel by pixel approaches are not capable of picking up spatial patterns that become more prevalent at finer scales (Kuhnert et al., 2005; Netzel and Stepinski, 2014). To account for the lack of spatial awareness in pixel by pixel approaches, several different approaches have been proposed such as the fuzzy set approach (Hagen, 2003) and various moving window aggregation methods (Kuhnert et al., 2005; Li and Wu, 2004). Within landscape ecology data transformation methods for analyzing

spatial patterns can be separated into three types - majority rule aggregation, resampling, and magnification. Majority rule aggregation area and resampling are similar in the sense that they are both forms of aggregation which result in coarser raster maps. The differences however lie with the methods by which each operates, where majority rule based aggregation works through a moving window where the each window votes on the value with ties broken by a random value, and resampling works through randomized but systematic sampling of the raster map. Multiple software's have implemented different moving window methods such as FRAGSTATS (McGarigal, 1995), Map Comparison Kit (Visser and De Nijs, 2006), Pronto Raster (Hagen-Zanker, 2016), and GRASS `r.pi` (Wegmann et al., 2018), and the Spatial Modeling Environment (SME) (Maxwell and Costanza, 1997; Costanza and Voinov, 2003). However, the moving window and multi-resolution comparison index as implemented by Costanza (1989) in particular has a large number of interesting and viable uses as it builds upon the moving window concept within landscape ecology. Designed as a method to get the goodness of fit between two model outputs, it can effectively take into account spatial patterns that standard statistical measures, e.g. R^2, may fail to take into account. The multi-resolution index can subsequently be applied for uses such as raster correlation and temporal change in addition to its initial purpose. SME was the primary method for implementing the moving window and multiple resolution algorithms having been developed initially by Costanza and others, however to the knowledge of the Author it is no longer maintained and no other system offers an implementation.

Proposed is a Python module wherein an algorithm capable of computing the moving window and multi-resolution comparison indices parallel across the Central Processing Unit (CPU) is implemented. The usage of parallel computing will enable the algorithm to take advantage of modern computing which can speed up array based math operations in Python and achieve speeds comparable to compiled languages such as C and C++ with little overhead (Dalcin et al., 2011; Cai et al., 2005; Wagner et al., 2017). Additionally, as the Python language has cemented itself solidly beside `R` as the primary scripting language for geospatial data science, a Pythonic implementation is key to enable the incorporation of the multi-resolution comparison index into more workflows.

2 Methods and materials

Within Python the Geospatial Data Abstraction Library (GDAL) (GDAL/OGR Contributors, 2020) in conjunction with the NumPy library (Harris et al., 2020), the algorithm can be implemented while maintaining important spatial data and providing geospatial input and output (I/O). For parallel processing the Joblib library (Joblib Contributors, 2020) is used as it touts optimized processing for the NumPy library and its only dependency is Python itself. Through parallel processing time of the algorithm can be sped up immensely, however it requires a implementation from the top down.

2.1 Indices

The moving window and the multi-resolution index are interlinked with the multi-resolution providing what is essentially a weighted aggregation for the moving window index. The moving window comparison is defined by Costanza (1989) as

$$F_w = \frac{\sum_{s=1}^{t_w} \left[1 - \frac{\sum_{i=1}^{p} |a_{1,i} - a_{2,i}|}{2w^2}\right]}{t_w} \tag{1}$$

where F_w is the correlation between 0 and 1 where 1 indicates complete similarity and 0 indicates

none , w is the window size, s is the index for moving windows, t_w is the number of windows with the window size w, $a_{1,i}$ and $a_{2,i}$ represent the numbers of cells with category i in rasters 1 and 2, respectively, and p is the number of categories. However, Eq. (1) operates off the assumption that within every window size w, there will be the same number of values. Within geospatial data this not the case however. To take into account no data cells commonly found in geospatial raster formats Eq. (1) can be rewritten as

$$F_w = \frac{\sum_{s=1}^{t_w} \left[1 - \frac{\sum_{i=1}^{p} |a_{1,i} - a_{2,i}|}{2n_s}\right]}{t_w} \tag{2}$$

Eq. (2) safely removes the assumption of the number of real data within a given window by replacing w^2 with n_s where n_s is the number of data cells in the window. The multiple resolution coefficient was created by Costanza (1989) to aggregate resolutions into a weighted coefficient and is defined by

$$F_t = \frac{\sum_{w=1}^{n} F_w e^{-k(w-1)}}{\sum_{w=1}^{n} e^{-k(w-1)}} \tag{3}$$

where n is the total number of window resolutions to iterate, and k is the constant weight where $k = 0$ gives all windows the same weight and $k = 1$ gives the first several resolutions more weight. The multiple resolution algorithm can give added context to a map scene and effectively extended the information gleaned from the moving window comparison algorithm.

2.2 Row by row approach

Two different methods for parallel processing were explored. The first version, seen in Algorithm 1, iterates over each row, splits the row into a subset matrix with the required neighbor rows for window size w, and appends the output vector of F_w values to a shared vector array. This design is intended to reduce the memory overhead for large datasets and only read a single row and its neighbors for each core used at a time.

However, problems quickly arise however with this method when the resulting shared vector is reshaped into the original i, j dimensions of the categorical rasters for the resulting raster map. As the output F_w values are appended to the shared array, it becomes difficult to append values in the correct order without using more memory than needed to create a dictionary index or artificially adding significant time to the process through a forced delay to each queuing subset computation. Furthermore, as GDAL is used to read geospatial data, serialization issues are encountered when attempting to iteratively read only certain sections from raster files without ever loading the entire raster maps to memory. Thusly Algorithm 2 is proposed to approach the issues of Algorithm 1.

2.3 Subsection approach

Algorithm 2 approaches the problem of parallel processing slightly different from Algorithm 1 and takes a more standard approach to splitting data across cores. While Algorithm 1 continuously queues a row and its neighbors to a core as they are iterated, Algorithm 2 splits the data sets evenly across the number of cores being used with the last core taking an even segment plus the remainder of rows. This method falls more in line with traditional approaches to parallel processing than the process described in Algorithm 1. Furthermore, Algorithm 2 solves the problems that arise with Algorithm 1 when appropriately generating output raster maps as each core computation has a slight delay introduced before starting. This is doable without any significant increases in time as

Require: w ▷ Window size, single integer
Require: A ▷ Array one
Require: B ▷ Array two
$d \leftarrow \left\lfloor \frac{w}{2} \right\rfloor$ ▷ Number to get neighbors of cell
$i, j \leftarrow$ matrix shape
$O \leftarrow [\,]$ ▷ Intialize empty vector
for $ii \leftarrow A_0 \rightarrow A_i$ **do**
    $\text{AM} \leftarrow A_{(ii-d \rightarrow ii+d)}$ ▷ Get subset of matrices A, B with window size w
    $\text{BM} \leftarrow B_{(ii-d \rightarrow ii+d)}$
    for $jj \leftarrow \text{AM}_{,0} \rightarrow \text{AM}_n$ **do** ▷ Assigned to open core
        $h,\ w \leftarrow$ shape of matrix subset
        $m \leftarrow$ central h index of matrix subset ▷ Middle of row of subset
        $v1 \leftarrow$ neighbors of window size w for $\text{AMatrix}_{m,\,jj}$
        $v2 \leftarrow$ neighbors of window size w for $\text{BMatrix}_{m,\,jj}$
        $N1, C1 \leftarrow \left\{\text{unique values in } v1\right\}, \left\{\text{counts of unique values}\right\}$
        $N2, C2 \leftarrow \left\{\text{unique values in } v2\right\}, \left\{\text{counts of unique values}\right\}$
        $\mathbb{U} \leftarrow N1 \cup N2$ ▷ Universal set of unique values
        for $i \leftarrow |\mathbb{U}|$ **do**
            $d \leftarrow |C1_i - C2_i|$
            $f \xleftarrow{+} \begin{cases} 1 - \frac{d}{2w}, & \text{if } d \neq 0 \\ 1, & \text{otherwise} \end{cases}$
        end for
    end for
    $O \xleftarrow{+} f$ from each core
end for
$F_w \leftarrow \sum \left(\frac{O}{i \times j}\right)$

Algorithm 1: Method 1 for a parallel moving window algorithm

the delay is minuscule (less than a second) and instead of multiplying the delay by tens of thousands of rows, it is only multiplied by upwards of tens of cores.

Require: w ▷ Window size, single integer
Require: A ▷ Array one
Require: B ▷ Array two
$c \leftarrow$ number of cores to use
$i, j \leftarrow$ matrix shape if shape A = shape B
$s \leftarrow \lfloor \frac{i}{c} \rfloor$ ▷ The range for each matrix subsection for each core
$r \leftarrow i \bmod c$ ▷ The remainder of cells that will be added to the last core
$d \leftarrow \lfloor \frac{w}{2} \rfloor$
$O \leftarrow [\,]$
$f \leftarrow 0$
for $ii \leftarrow 0 \rightarrow c$ **do** ▷ Get upper boundaries for each matrix subsection
    $g \xleftarrow{+} \begin{cases} f+s+r, & \text{if } ii_c \\ f+s, & \text{otherwise} \end{cases}$ ▷ Add remainder to last subsection
    $f \xleftarrow{+} s$
end for
for $ii \leftarrow g_0 \rightarrow g_n$ **do** ▷ Create list of full i ranges to split the matricies
    $sl \leftarrow \begin{cases} [0 : ii+d], & \text{if } 0 \\ [ii-d : ii], & \text{if } n \\ [ii-d : ii+d], & \text{otherwise} \end{cases}$ ▷ Take into account overlapping cells for each section
end for
for $ii \leftarrow sl$ **do** ▷ With each iteration the Worker function is passed to a new core
    $O \xleftarrow{+}$ Worker$(A[sl_{ii}, j], B[sl_{ii}, j])$ ▷ Results for each core are appended in order
end for
$tw \leftarrow |O| \neq -1$ ▷ Get count of cells that have data
$Sim \leftarrow \sum(\frac{O}{tw})$

Algorithm 2: Method 2 for a parallel moving window algorithm

When the subsections are first computed the cells at the edges lose the neighboring data cells which would be needed for the window size w. Without being addressed, the loss of data within the data set will lead to edge artifacts within the output raster map that increase with window size and ultimately reduce accuracy of the resulting F_w similarity value. To account for lost data along the edges of each subsection, each boundary is given d additional rows at the beginning and the end of the subsection where $d = \lfloor \frac{w}{2} \rfloor$. An exception to how context is added to the edges are the first and last splits which are only given d rows at the beginning *or* end of their subsections respectively. These additional rows are not iterated over outside the subsection they are initially allocated for since they only serve to re add edge context which is removed by through splitting the data.

To enable the approach of Algorithm 2 the *Worker* function as shown in Algorithm 3 is used. This function facilities the F_w computation across all cores by allowing the initialized object to compute the range of splits needed before assigning each subset to a separate core for the bulk of the heavy computation. With the *Worker* function assigned to each core with respective submatrices, it firstly checks if the iterated cell is a NoData value, returning a value of -1 to be masked if it is and continuing the algorithm if it is not. The function subsequently carries out the rest of moving

window algorithm across the submatrices relying on set operations in conjunction with vectorized NumPy functions where possible.

function WORKER(A, B)
  for $i, j \leftarrow A[i, j] \rightarrow A[n, n]$ **do** ▷ Iterate over i, j of matrix
    if $A[i, j]$ **and** $B[i, j] = \text{NaN}$ **then**
      $f \overset{+}{\leftarrow} -1$ ▷ Assign the value of -1 if NoData value
    else
      $v1 \leftarrow$ neighbors of window size w for A$[i,\ j]$
      $v2 \leftarrow$ neighbors of window size w for B$[i,\ j]$
      $N1, C1 \leftarrow \{\text{unique values in } v1\}, \{\text{counts of unique values}\}$
      $N2, C2 \leftarrow \{\text{unique values in } v2\}, \{\text{counts of unique values}\}$
      $\mathbb{U} \leftarrow N1 \cup N2$ ▷ Create universal set of unique values
      for $ii \leftarrow |\mathbb{U}|$ **do** ▷ Iterate over each unique value
        $d \leftarrow |C1_{ii} - C2_{ii}|$ ▷ Subtract count of each unique value
        $f \overset{+}{\leftarrow} \begin{cases} 1 - \frac{d}{2w}, & \text{if } d \neq 0 \\ 1, & \text{otherwise} \end{cases}$ ▷ If $d = 0$, then similarity is 1 for $A[i, j], B[i, j]$
      end for
    end if
  end for
  return f ▷ Return vector of similarity values computed on core
end function

Algorithm 3: Worker function for Algorithm 2

3 Multi-resolution implementation

The most important usage of the moving window index is its incorporation into the weighted multi-resolution index which will return an improved and more spatially aware similarity index than that of just a single moving window value. The aggregation occurring with each increasing window will provide a similarity for an increasing area which can be described by the simple mathematical expression of

$$A = (n_s \times b) \times (n_s \times h) \tag{4}$$

where n_s is the number of data cells in a window, b is the width of each cell in map units, and h is equal to the height of each cell in map units.

The multi-resolution implementation is subsequently achieved by building upon Algorithm 2 where the parallel processing at each window size (w) iteration is contained within the moving window object. With it, the relationships between increased window sizes and similarity can be examined as in Figure 3 in addition to how the method as a whole reflects the similarities between spatial data.

4 Results & Discussion

When parallelized the time taken for the moving window correlation saw an exponential decrease in computational time taken, as can be seen in Figure 2, where sensitivity analysis performed for two datasets with the size 5528×5569 shows a negative exponential relationship between time and

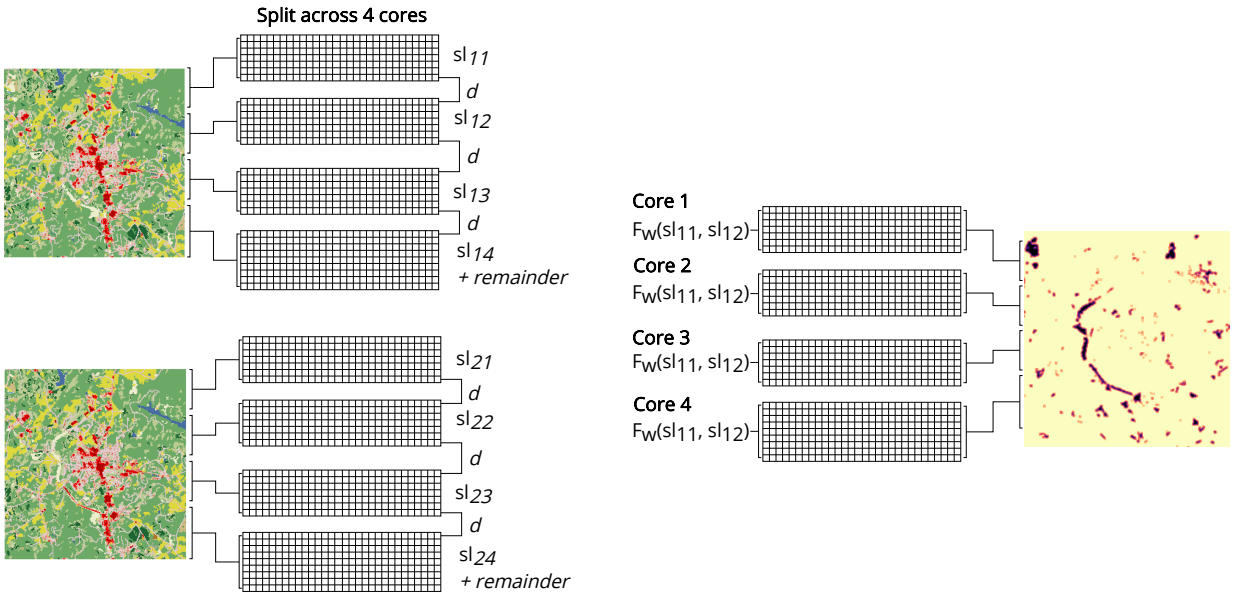

Figure 1: Roughly showing the moving window algorithm split between four CPU cores with Algorithm 2. The output raster map shows a 3×3 comparison where red is 0 and green is 1.

cores used across seven different window sizes. It is important to note that with window sizes less than 43 a sharp increase in time taken from the first to second core is seen. This increase in time is due to the time taken to split the data between cores that a single core approach does not contain. However when a window size greater than 13 is used, the implementation sees greater speeds after using thee cores. These time improvements are key to the implementation of the multi-resolution coefficient as without it, the time which the multi-resolution would take to complete for any sized area would grow linearly with increased window sizes. The effects aggregation with increasing window sizes can be seen in Figure 4.

With the moving-window index sped up, the relationships between window size and similarity can be examined. Figure 3 shows that as the window size increases by ten, so too does the similarity increase with a positive linear curve. Furthermore, Figure 3 shows the weight, k, increases the overall similarity index by approximately 2%. The weight $k = 0$, which takes into account all resulting F_w values, returns a F_t similarity index shown to be 0.8304 (83.04 % similar), where as the weight of $k = 1$ returns a similarity index of 0.8119 or 81.19 % similar.

Further information about the relationship between window size and similarity can be inferred by comparing the common pixel by pixel subtraction index which is expressed mathematically as

$$F = \frac{N_{id}}{N} \tag{5}$$

where N_{id} is the total number of cells which match and N is the total number of cells in a raster map (Kuhnert et al., 2005).

When applied to the raster data shown in Figure 3, the pixel by pixel similarity of 0.8098 is returned which is only marginally smaller than the F_t value with weight of one. The closeness between the pixel by pixel similarity index and the multi-resolution index shows the weights working as intended, while allowing the similarity index to aggregate spatial data. However, there is an approximate three percent difference between the pixel by pixel subtraction similarity and the F_t index value with the weight of zero. The increase in seen in the F_t value weighted zero when

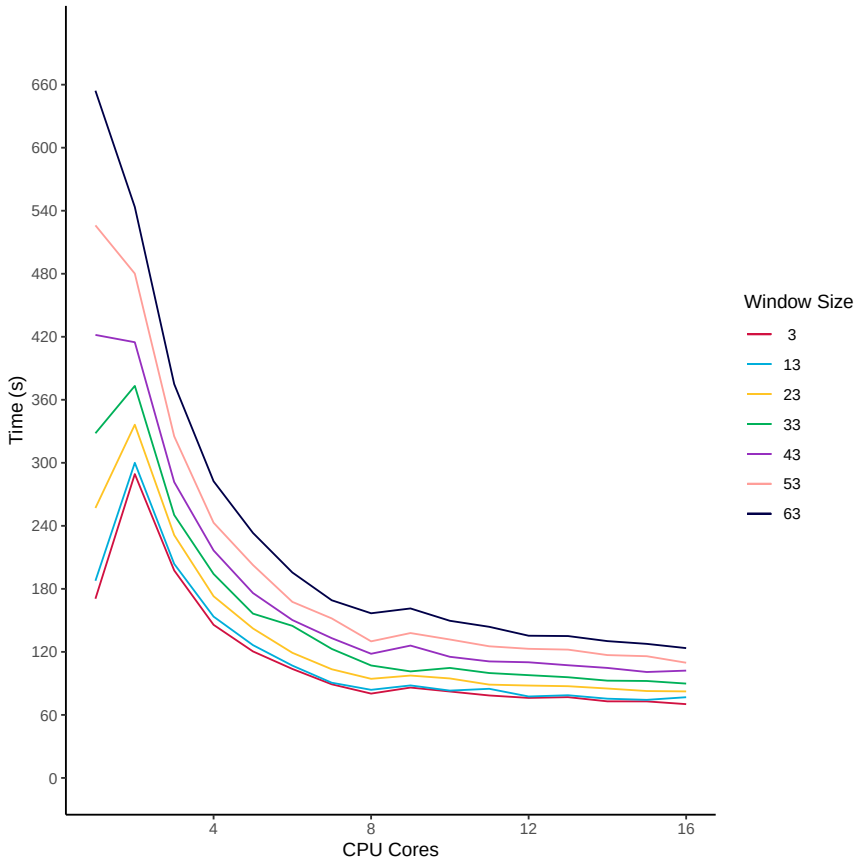

Figure 2: The processing time in seconds it takes to compute the similarity of two NLCD rasters of size 5528×5669 across an increasing number of CPU cores and window sizes.

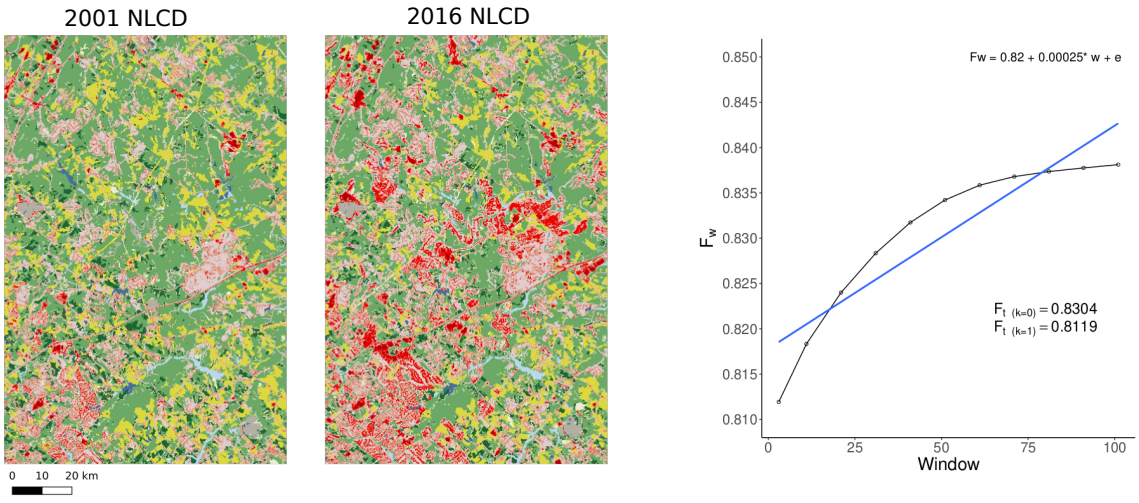

Figure 3: Both the F_w and total computed F_t values for weights 1 & 0 for the maps on the left. Additionally shown is the relationship between the F_w index and an increasing window size.

compared to the pixel by pixel value is expected. This is due to the lack of weight for the smaller window sizes leading to an F_t value which ultimately will correspond to a larger aggregation area as described by Eq. (4). The causation of the increasing similarity without weight given can be visually seen in Figure 4 as each increasing window results in an aggregation area equal to Eq. (4).

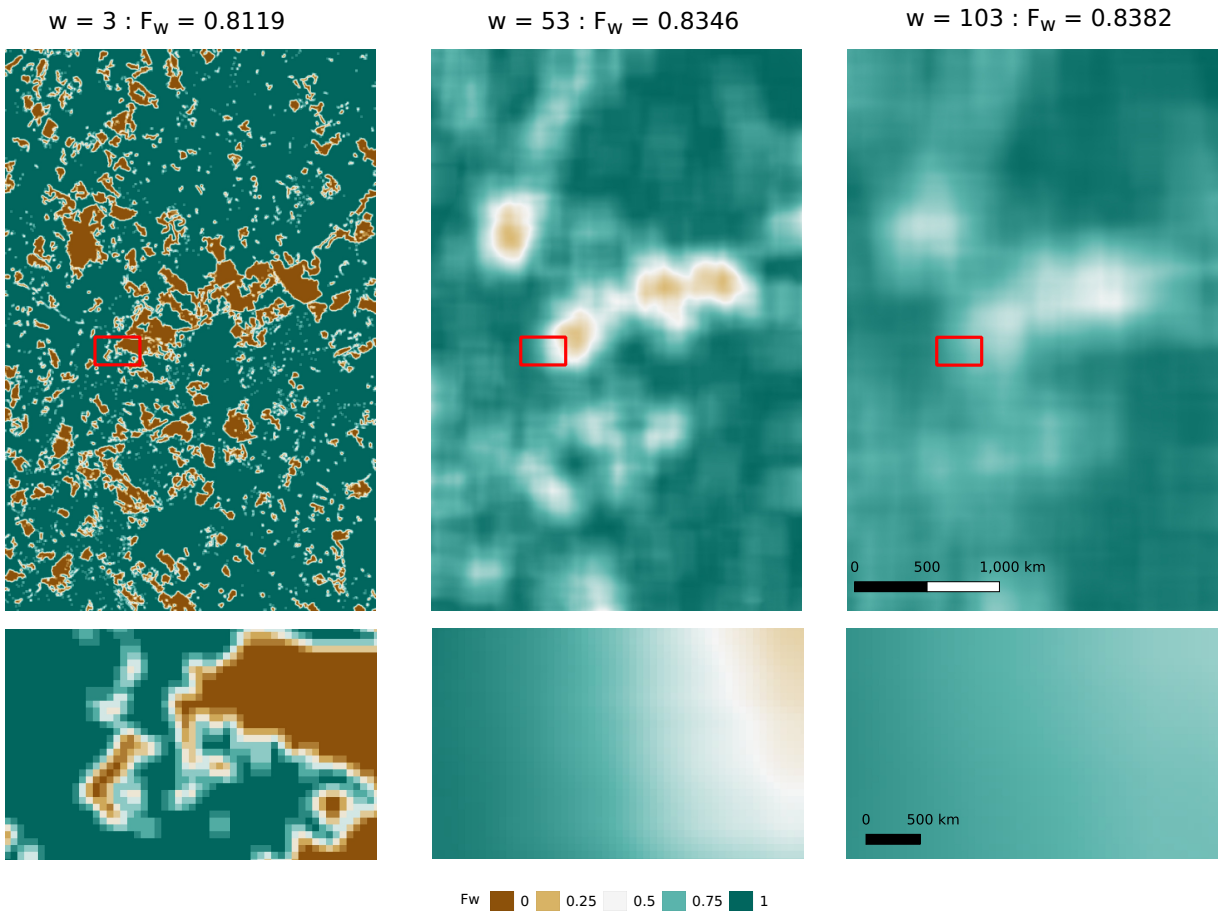

Figure 4: Resulting F_w raster map values showing the effects of increased window size on similarity with insets showing the effects of area aggregation at a small scale. Window sizes from left to right are 3, 53, and 103.

5 Conclusions

Within in the paper two algorithms are introduced to incorporate both the moving window and multi-resolution similarity index. Modern implementations of each are needed due to previous implementations going unmaintained. It is found that the parallel algorithm as implemented with Python in this paper allows for computationally efficient implementations of the moving window and its subsequent aggregation index, the multi-resolution index. Further development of a resolution independent index is underway so as to not only implement the equations proposed by (Costanza, 1989), but build upon and improve them systematically into a new index entirely for landscape ecology.

6 References

Cai, X., Langtangen, H. P., and Moe, H. (2005). On the performance of the python programming language for serial and parallel scientific computations. *Scientific Programming*, 13(1):31–56.

Costanza, R. (1989). Model goodness of fit: a multiple resolution procedure. *Ecological modelling*, 47(3-4):199–215.

Costanza, R. and Voinov, A. (2003). *Landscape simulation modeling: a spatially explicit, dynamic approach*. Springer Science & Business Media.

Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A. (2011). Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139.

Dale, M. R. (2000). *Spatial pattern analysis in plant ecology*. Cambridge university press.

Friedl, M. A., Sulla-Menashe, D., Tan, B., Schneider, A., Ramankutty, N., Sibley, A., and Huang, X. (2010). Modis collection 5 global land cover: Algorithm refinements and characterization of new datasets. *Remote sensing of Environment*, 114(1):168–182.

GDAL/OGR Contributors (2020). *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation.

Gonçalves, M., Costa, J., Netto, M., and Mwasiagi, J. (2011). Land-cover classification using self-organizing maps clustered with spectral and spatial information. *Self-Organizing Maps Applications and Novel Algorithm Design*, 1:299–322.

Hagen, A. (2003). Fuzzy set approach to assessing similarity of categorical maps. *International Journal of Geographical Information Science*, 17(3):235–249.

Hagen-Zanker, A. (2016). A computational framework for generalized moving windows and its application to landscape pattern analysis. *International journal of applied earth observation and geoinformation*, 44:205–216.

Hansen, M. C., DeFries, R. S., Townshend, J. R., and Sohlberg, R. (2000). Global land cover classification at 1 km spatial resolution using a classification tree approach. *International journal of remote sensing*, 21(6-7):1331–1364.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

Homer, C., Huang, C., Yang, L., Wylie, B., and Coan, M. (2004). Development of a 2001 national land-cover database for the united states. *Photogrammetric Engineering & Remote Sensing*, 70(7):829–840.

Joblib Contributors (2020). Joblib: running python functions as pipeline jobs.

Kuhnert, M., Voinov, A., and Seppelt, R. (2005). Comparing raster map comparison algorithms for spatial modeling and analysis. *Photogrammetric Engineering & Remote Sensing*, 71(8):975–984.

Li, H. and Wu, J. (2004). Use and misuse of landscape indices. *Landscape ecology*, 19(4):389–399.

Maxwell, T. and Costanza, R. (1997). An open geographic modeling environment. *Simulation*, 68(3):175–185.

McGarigal, K. (1995). *FRAGSTATS: spatial pattern analysis program for quantifying landscape structure*, volume 351. US Department of Agriculture, Forest Service, Pacific Northwest Research Station.

Netzel, P. and Stepinski, T. F. (2014). Pattern-based assessment of land cover change on continental scale with application to nlcd 2001–2006. *IEEE Transactions on Geoscience and Remote Sensing*, 53(4):1773–1781.

Oeser, J., Pflugmacher, D., Senf, C., Heurich, M., and Hostert, P. (2017). Using intra-annual landsat time series for attributing forest disturbance agents in central europe. *Forests*, 8(7):251.

Perry, J., Liebhold, A., Rosenberg, M., Dungan, J., Miriti, M., Jakomulska, A., and Citron-Pousty, S. (2002). Illustrations and guidelines for selecting statistical methods for quantifying spatial pattern in ecological data. *Ecography*, 25(5):578–600.

Riitters, K. (2019). Pattern metrics for a transdisciplinary landscape ecology.

Schwarz, G. E. and Alexander, R. (1995). State soil geographic (statsgo) data base for the conterminous united states. Technical report, USGS.

Senf, C. and Seidl, R. (2020). Mapping the forest disturbance regimes of europe. *Nature Sustainability*.

Visser, H. and De Nijs, T. (2006). The map comparison kit. *Environmental Modelling & Software*, 21(3):346–358.

Wagner, M., Llort, G., Mercadal, E., Giménez, J., and Labarta, J. (2017). Performance analysis of parallel python applications. In *ICCS*, pages 2171–2179.

Wegmann, M., Leutner, B. F., Metz, M., Neteler, M., Dech, S., and Rocchini, D. (2018). r. pi: A grass gis package for semi-automatic spatial pattern analysis of remotely sensed land cover data. *Methods in Ecology and Evolution*, 9(1):191–199.