MULTI-SCALE AGGREGATION OVER SLIDING WINDOWS

Anne M. Denton
Department of Computer Science
North Dakota State University
Fargo, ND 58108-6050
anne.denton@ndsu.edu

ABSTRACT

Aggregates over sliding windows are an important part of many analyzes over raster images, including computation of basic statistic quantities, regression analysis, fractal dimensions, and topographic analysis. For those analyses, the most appropriate window size is not always obvious a priori and the window sizes may be very large. An algorithm is presented for aggregating windows iteratively with a performance that is logarithmic in the window size. The main limitations of the algorithm are that aggregation functions must be additive in the sense that performing aggregations based on prior aggregates must be possible, and that window sizes are powers of two. When those assumptions can be met, the algorithm can be used in a variety of contexts that would otherwise either require a substantial reduction of resolution or take excessive computation time.

1 Introduction

In geographic information systems (GIS) sliding or moving windows are used extensively for smoothing as a way of denoising and towards gathering focal statistics estimates [1, 2, 3]. Arguably, the need for sliding-window-based smoothing has increased substantially with the prevalence of high-resolution remotely sensed data that are collected by small satellites, drones or airplanes. While such high-resolution sources offer resolutions that cannot be achieved with conventional satellites like Landsat, the individual pixel values are commonly noisy. Current GIS offer great flexibility in the selection of the shape of sliding windows with the goal of providing an appropriate and adaptable aggregate at the center of the window [4]. However, this level of generality means that the user has to select one size, and the appropriate size for aggregation may not be known beforehand. Increasing the window size can reduce noise, but eventually it may wash out features, and the most expressive size may only become clear after doing the analysis.

With windows that are powers of two, aggregates from one iteration can be aggregated to create windows of twice the size. Note that while the presented approach is limited to windows that are a power of two, high-resolution images typically require processing over such large windows that even the limitation to powers of two creates ample processing choices. Conventional choices of window sizes for smoothing are odd numbers such as 3, 5 or 7 to ensure that there is one raster point at the center of the window that can be aligned with the derived features. The presented approach does not have that property but, for high-resolution data, an offset by half a pixel is often not serious, and can also be resolved by the GIS reraster operation if needed.

The proposed technique is useful in a large number of applications besides denoising. It is applicable for any application for which the aggregation measure is additive, i.e. the result of the aggregation can be computed by aggregating prior aggregates [5]. Clearly, there are many types of processing, such as computing a median, for which this is not the case. However, even seemingly complicated quantities, such as fractal dimensions, often require nothing but sum and max functions, both of which are additive and allow use of the presented techniques [6]. The same is true for general linear regression, including polynomial regression, which exclusively relies on additive aggregates regardless of the basis function and can be computed as discussed here [7]. Some other processing objectives, such as slope and curvature computations, can be developed based on the same aggregation strategy [8, 9]. Other applications, such as sliding-window generalizations of wavelet transforms, are possible.

Figure 1 shows the concept. In the first aggregation step a sliding window aggregation is done over windows of size 2×2. When aggregates over 4×4 windows are computed in the second step, all the relevant 2×2 aggregates are

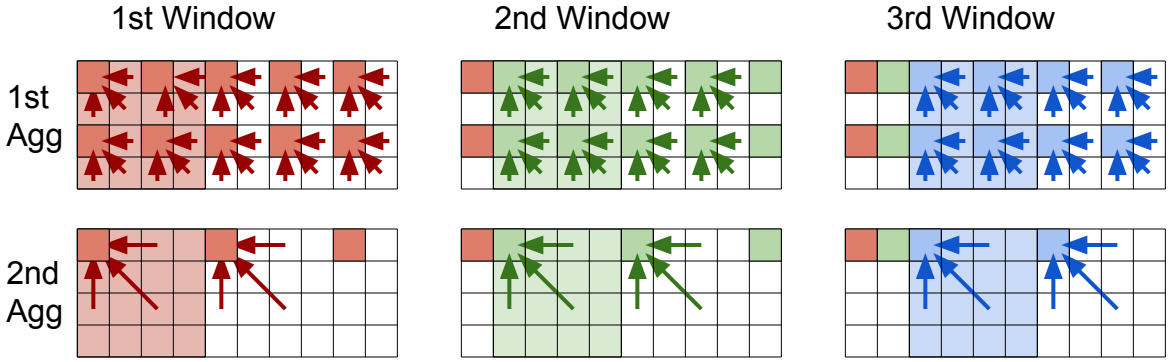

Figure 1: Schematic depiction of the aggregation process. The top row shows the first aggregation to 2×2 windows; the bottom row the second aggregation from 2×2 to 4×4 windows. From left to write the first three windows are highlighted.

already available without a need to go back to the original data. As a result the aggregation of 4×4 windows only takes twice as long as the aggregation of 2×2 windows or, more generally, the aggregation of $w \times w$ for $w = 2^d$ with $d \in \mathbb{N}$, takes $d = \log_2(w)$ as long as the aggregation over 2×2 windows. The efficiency of the algorithm comes from the massive reuse of aggregates. In Figure 1 that reuse can be seen for the third window. Two of the 2×2 aggregates that are needed for this window are the same as were used for the first window. Any aggregate that is more than the window size away from the perimeter of the image will be used towards 4 higher-level aggregates at the next level. Since the algorithm leads to a doubling of window size in each iteration the complexity per raster point is $O\left(log_2(w)\right)$. In a brute force implementation, each window would have to be scanned, corresponding to a per-raster-point complexity of $O(w^2)$.

In Figure 1 it appears as if the window aggregate is treated as representing the raster point located in its top left corner, although logically the aggregate represents the center of the sliding window. Note that any further analysis is correctly attributed to the central location by means of a shift in the Geo-TIFF meta-data by $(w-1)/2$ in each dimension. Note also that, as with any sliding window approach, the output raster of an image of width N and height M only has $(N - w + 1) \times (M - w + 1)$ aggregates that are derived from full windows of size w. Windows that are centered on points that are closer than $(w-1)/2$ to the border would extend beyond the extent of the raster. If there is a wish to let the output raster have the same dimensions as the original image, the frame can be padded with values from prior iterations.

2 Concepts

The approach is possible because additive aggregate functions can be rewritten as aggregates of the previous iteration. For summation, the aggregation of y_{ij} over a window of size w ranging from $i = i_0$ and $j = j_0$ up to and including $i = i_0 + w - 1$ an $j = j_0 + w - 1$ respectively, can be rewritten as follows:

$$
\begin{aligned}
y^{(w)}_{i_0 j_0} &= \sum_{i=i_0}^{i_0+w-1} \sum_{j=j_0}^{j_0+w-1} y_{ij} \qquad (1)\\
&= \sum_{i=i_0}^{i_0+w/2-1} \sum_{j=j_0}^{j_0+w/2-1} y_{ij} + \sum_{i=i_0+w/2}^{i_0+w-1} \sum_{j=j_0}^{j_0+w/2-1} y_{ij}\\
&+ \sum_{i=i_0}^{i_0+w/2-1} \sum_{j=j_0+w/2}^{j_0+w-1} y_{ij} + \sum_{i=i_0+w/2}^{i_0+w-1} \sum_{j=j_0+w/2}^{j_0+w-1} y_{ij}\\
&= y^{(w/2)}_{i_0 j_0} + y^{(w/2)}_{(i_0+w/2) j_0} + y^{(w/2)}_{i_0 (j_0+w/2)} + y^{(w/2)}_{(i_0+w/2)(j_0+w/2)}
\end{aligned}
$$

Note that in 1, the top left corner, (i_0, j_0), is used as designation for windows of all sizes, but for any further processing, the window is considered to represent the center $(i_0 + (w-1)/2, j_0 + (w-1)/2)$. A fully recursive specification of the aggregation can be written as follows:

Algorithm 1 Iterative Aggregation

procedure AGGREGATE(d, y_{old})
  $w \leftarrow 2^d$
  $delta \leftarrow w/2$
  $N \leftarrow$ WIDTH(y_{old})
  $y_{flat} =$ FLATTEN(y)
  $y_{agg} \leftarrow y_{flat} +$ SHIFT($y_{flat}, delta$) + SHIFT($y_{flat}, N * delta$)) + SHIFT($z_{flat}, (N+1) * delta$)
  $z_{new} \leftarrow$ SHAPE($(N - w + 1, M - w + 1), z_{agg}$)
return d, z_{new}
end procedure
procedure MAIN(*file_name*,*max_d*)
  $z_0 \leftarrow$ *import_raster*()
  $d \leftarrow 0$
  while $d <=$ *max_d* **do**
    $z_{d+1} \leftarrow$ AGGREGATE(d, z_d)
    $d = d + 1$

  end while
return $z_0 \ldots z_d$
end procedure

$$y_{ij}^{(w)} = \begin{cases} y_{ij}^{(w/2)} + y_{(i+w/2)j}^{(w/2)} \\ +y_{i(j+w/2)}^{(w/2)} + y_{(i+w/2)(j+w/2)}^{(w/2)} \text{ for } w \geq 2 \\ \\ y_{ij} \text{ for } w = 1 \end{cases} \tag{2}$$

In the example of denoising, the summation of the raster values is the only aggregation needed, since the window size is known. For polynomial regression among raster bands, some aggregates of squares and higher powers of raster values are needed, and so are some aggregates of products between different bands. This is possible, since the raster value y_{ij} does not have to be an original value drawn from any raster map. It could instead be the original value taken to any power, the product between different bands, or any other function of one or more raster values. While this creates a need for multiple aggregations, there is no fundamental difference in the process other than initial element-wise multiplications over the arrays that hold the data.

The computation of fractal dimensions using box counting relies on using the maximum function as well as summations to produce the multiple scales that are an inherent part of the computation of the measures. Conducting the aggregation for the maximum function is as straightforward as the summation, since the maximum and minimum functions also have the property that aggregates can be computed from prior aggregates. Even an explicit representation of the geographic coordinates within the plane can be done, although it requires some additional mathematical derivations. For topographic attributes these computation are needed and can be done with the same computational complexity as the aggregation described here.

Although the recursive representation is concise, efficient implementations are iterative. When using Python with NumPy, it is moreover advantageous to use an array-based implementation, in which four one-dimensional arrays are aggregated in a single step rather than iterating over array elements one at a time and aggregating four individual raster points. Algorithm 1 shows the pseudocode. The procedure MAIN initializes the raster from input and sets the current number of aggregations d to zero. For regression or other advanced processing goals, the imported raster values may have to be processed. The AGGREGATE procedure is called from the MAIN procedure as many times as the window size is intended to be doubled (*d*).

One iteration, as shown in the function AGGREGATE, consists of flattening the array, which is assumed to hold the raster points using a row-wise storage organization, and then aggregating that array and 3 copies that have been shifted by $delta = w/2$, by $N * delta$, and by $delta + N * delta$ respectively, corresponding to a left shift by delta, an upward shift by the same distance, and a left and upward shift. The three shifts correspond to the arrows in Figure 1. Once the aggregation is complete, the resulting array is shaped to be $N - w + 1$ wide and $M - w + 1$ high.

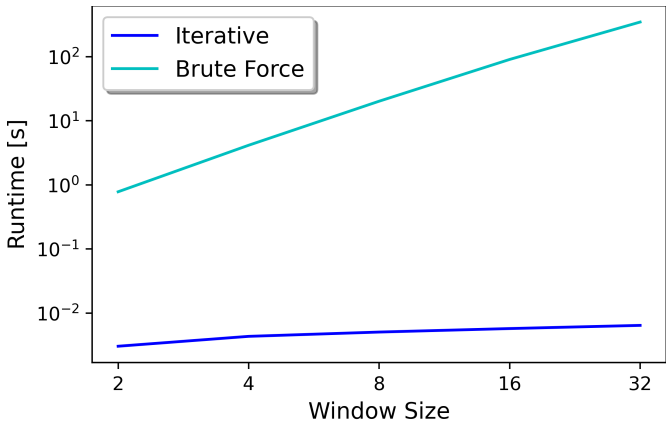

Figure 2: Runtime comparison between the described iterative algorithm (blue) and a default brute force implementation that aggregates the points in the largest window (cyan). In addition to being faster, the iterative algorithm outputs results for all power of two window sizes up to the maximum.

3 Evaluation

Figure 2 shows the comparison of a Python with NumPy implementation of the described algorithm with a default brute force algorithm that neither uses the iterative doubling of window size nor the array-based implementation. Since both implementations return the same result no quality comparison is included for this simple experiment.

In practice, most application domains of the algorithm use somewhat different algorithms: For statistics analyses, windows are used that approximate circular neighborhoods, which improve the statistical quality of the result somewhat but do not allow for the fast multi-scalar analysis. Fractal and similar computations are normally either done over non-overlapping squares or the result of using segmentation as preprocessing, which gives high-quality results but at a much lower resolution than the sliding window approach. Finally, topographic measures are typically done over windows of size 3 by 3. For high-resolution elevation models, such an approach requires smoothing as preprocessing step with a resulting loss of information.

Figure 2 shows that the scaling of the iterative algorithm does indeed taper off in a double-logarithmic representation as is expected for an algorithm with logarithmic performance, while a brute force implementation is quadratic, i.e. slope 2 in a double-logarithmic plot. The array-based implementation also results in a substantial improvement in the prefactor when implemented in Python. Faster implementations of the comparison algorithm using conventional techniques are conceivable, for example through incremental updating [10]. Implementing it in a language without Python's large overhead would reduce the prefactor. However, such an implementation would not return the full multi-scalar results offered by the iterative algorithm discussed above.

Returning results for a spectrum of window sizes makes the iterative algorithm much more versatile. Instead of deciding on the desired window scale from the start, the user only has to provide an upper bound for the scales that are to be returned and can choose the best window size based on the output quality. While acceleration strategies for window-based approaches that use incremental updating can improve on the brute-force approach, they do not result in the multi-scalar output. For high-resolution data, for which analyzes can be done over a much larger spectrum of window sizes, the multi-scalar output is particularly important. The logarithmic performance moreover allows exploring a large range of different window sizes.

4 Summary

Multi-scale aggregation over sliding windows in raster data was presented that is useful in feature generation tasks for high-resolution raster imagery. The approach has logarithmic scaling as a function of the window size and can be applied in many contexts, provided that all aggregation measures are additive, i.e., can be computed as aggregates of prior aggregates. It furthermore produces all window sizes that are powers of two as byproduct of the computation. With the increases in resolution in raster data, this algorithm is expected to become increasingly more important.

5 Acknowledgements

Thank you to former and current students Rahul Gomes, David Schwartz, Jordan Goetze, Nick Dusek, Riley Conlin, Dawit Beshah, and Guy Hokanson. Thanks to Dave Franzen (NDSU Soil Science) for the agriculture perspective. This material is based upon work supported by the National Science Foundation through grant IIA-1355466.

References

[1] Konstantin Krivoruchko. *Spatial statistical data analysis for GIS users*. Esri Press Redlands, 2011.

[2] A Stewart Fotheringham, Martin Charlton, and Christopher Brunsdon. Two techniques for exploring non-stationarity in geographical data. *Geographical Systems*, 4(1):59–82, 1997.

[3] Qiong Wu, Dan Hu, Rusong Wang, Hongqing Li, Yong He, Min Wang, and Bihui Wang. A gis-based moving window analysis of landscape pattern in the beijing metropolitan area, china. *The International Journal of Sustainable Development and World Ecology*, 13(5):419–434, 2006.

[4] ArcMap. How focal statistics works. `https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/how-focal-statistics-works.htm`, Visited 28 Sept 2021.

[5] Ned Glick. Additive estimators for probabilities of correct classification. *Pattern recognition*, 10(3):211–222, 1978.

[6] Anne M. Denton and Jordan Goetze. Window-based fractal dimension as geospatial data type. In *Workshop on Data Science for Intelligent Food, Energy, and Water (DSIFER)*, ACM-KDD 2017, 2017.

[7] Anne M. Denton, Mostofa Ahsan, David Franzen, and John Nowatzki. Multi-scalar analysis of geospatial agricultural data for sustainability. In *2nd International Workshop on Big Data for Sustainable Development*, IEEE Big Data 2016, 2016.

[8] Anne M. Denton, Rahul Gomes, and David Franzen. Scaling up window-based slope computations for geographic information systems. In *18th Annual IEEE International Conference on Electro Information Technology (eit2018)*, 2018.

[9] Rahul Gomes, Anne Denton, and David Franzen. Quantifying efficiency of sliding-window based aggregation technique by using predictive modeling on landform attributes derived from dem and ndvi. *ISPRS International Journal of Geo-Information*, 8(4):196, 2019.

[10] Pramod Bhatotia, Marcel Dischinger, Rodrigo Rodrigues, and Umut A Acar. Slider: Incremental sliding-window computations for large-scale data analysis. *MPI-SWS, CITI/Universidade Nova de Lisboa, CMUTechnical Report: MPI-SWS-2012-004 September*, 2012.