An algorithm for generating geometric buffers for vector feature layers

Sumeet BHATIA*, Viral VIRA, Deepak CHOKSI and P. VENKATA CHALAM

207 Spatial Analysis Lab, Centre of Studies in Resources Engineering, Indian Institute of Technology, Bombay 400076, India

(Received 10 February 2012; final version received 22 April 2012)

The paper presents an algorithm for constructing geometric buffers for vector feature layers and dissolving those buffers using a sweep-line approach and vector algebra. The algorithm works by first constructing a geometric buffer for a vector feature layer, then dissolving each single geometric buffer for that feature layer, and finally dissolving the overlapping buffers of the entire layer. The algorithm has been implemented successfully in a commercial Geographical Information System software package.

Keywords: geometric buffer; spatial analysis; algorithm

1. Introduction

A buffer is a zone with a specified distance surrounding a spatial data feature. The process of creating a buffer, or buffering, refers to the creation of this zone. Spatial data usually consist of different types of spatial objects. Object types include points, poly-lines and polygons. Buffers can be generated around each of these spatial features. In each case, the buffer takes its shape from the feature.

In Geographical Information System (GIS) applications, buffer zones are vector polygons, no matter whether they are generated from a point, poly-line or polygon feature. For highly convoluted spatial features, such as meandering rivers, the generated buffer zones tend to overlap (1). The boundaries of overlapping buffer zones are dissolved to produce a single coherent buffer zone for the entire spatial feature. There are many applications for coherent geometric buffers in the spatial analysis functions of a GIS; for example, when an educational institute wants to make sure that every inch of main campus is covered by wireless network. The institute has deployed wireless access point at various points on campus. Now, the goal is to find if the wireless network covers the entire campus area. For the buffer distance, assume that the wireless range is 600 m. First, the buffer for each wireless access point is created. Then, the overlaps formed due to adjacent points are removed. Subsequently, the region of the institute that does not fall under the resultant buffer polygon is the area without wireless network coverage.

Buffers are used to establish critical areas for spatial analysis or to indicate proximity or accessibility conditions. Computational geometry operations are required to establish buffer zones for vector data.

Given the wide range of buffering applications, it is considered to be essential functionality in vector based geographic data processing (2). For this reason, GIS research community frequently develops new and different methods for constructing geometric buffers. However, the algorithmic details are proprietary and hence are not available to the scientific community. The algorithmic documentation for constructing buffers for two point poly-lines has been well documented (3). However, in real world GIS applications, the dataset may not necessarily contain only two point poly-lines. The algorithm discussed here provides an approach to deal with multiple point poly-lines – a topic not dealt with in this documentation. Buffering algorithms usually emphasise visualisation. The algorithm described in this paper facilitates spatial analysis functionalities in addition to visualising coherent buffers. This is achieved by computing and storing all the associated buffer points as a separate polygon layer in a spatial database.

The paper is divided into three sections. Section 1 describes the methodology for constructing geometric buffers on different vector feature layers. Section 2 gives an approach to dissolve an individual geometric buffer for a spatial feature and Section 3 illustrates the procedure to dissolve overlapping buffers of an entire vector feature layer.

2. Methodology to construct geometric buffer

The algorithm presented in this paper constructs the geometric buffer for a vector feature layer by determining
all the buffer points. The buffer is then oriented in a counter-clockwise direction. Considering that different approaches are required to generate buffers for point, poly-line, and polygon features; the section discusses the methodology used for each vector feature layer. However, irrespective of the input layer, the output buffer is always a vector polygon layer.

2.1. Construction of geometric buffer for a point layer
The algorithm requires a point layer around which the buffer has to be generated and a buffer width as an input parameter. A point is used to create a circular polygon of specified radius as its buffer. The buffer points for a point layer are determined at a distance equal to buffer width from the input point. The parametric equations of a circle are used to determine the required buffer points. Depending on the desired accuracy, the user can determine the interval at which the buffer points will be plotted for the circular buffer. The final buffer for a point layer is as shown in Figure 1.

![Figure 1. Representation of a buffer for point layer.](image)

As illustrated by Figure 1, buffers due to multiple points overlap. These overlapping buffers are dissolved by the removal of overlapping sections. The technique to dissolve the overlapping sections of the buffers is described in Section 4.

2.2. Construction of geometric buffer for a poly-line layer
A poly-line layer can be connected, disconnected or a combination of both. Each poly-line includes a start node, an end node, and intermediate nodes in between the start and end node. Initially, the algorithm considers a single disconnected poly-line and forms a buffer for it. The buffer for a poly-line is created with the help of its nodes. Two adjacent nodes are taken at a time and a buffer segment is generated for these nodes. The method to create a buffer for a two point poly-line segment is described by Zalik et al. (3). Real world GIS datasets will however contain multiple point poly-lines. The input parameters for this algorithm are the poly-line layer around which the buffer has to be generated and the buffer width.

![Figure 2. Representation of orientation of line segments.](image)

Buffering poly-line data is a little more complicated than buffering point data due to the fact that a poly-line is made up of multiple line segments. A band of specified distance is created by taking the perpendicular distance from each node of the poly-line on either side. The orientation of each line segment within a poly-line decides the angle between them. Figure 2 highlights this fact.

In Figure 2, if the poly-line is traversed in the direction ABCDEF, then the orientation of the line segment CD will be $-45^\circ$. However, if the same poly-line is traversed as FEDCBA, then the orientation of the line segment DC will be calculated as $135^\circ$. Thus, the same line segment has different orientation values, yet the same slope depending on the direction in which it is traversed. This discussion makes it clear that distinction needs to be made between these cases depending on the orientation of the line segments. These cases are depicted in Figure 3.

The distinction between case 1 and case 2 is decided based on the following two conditions

If $((\text{Numerator} < 0 \ \text{AND} \ \text{Denominator} < 0) \ \text{OR} \ (\text{Numerator} > 0 \ \text{AND} \ \text{Denominator} < 0))$, then it concludes that case 1 has been encountered.

If $((\text{Numerator} > 0 \ \text{AND} \ \text{Denominator} > 0) \ \text{OR} \ (\text{Numerator} < 0 \ \text{AND} \ \text{Denominator} > 0))$, then it concludes that case 2 has been encountered.

Where

\[
\text{Numerator} = \text{StartVertex} \cdot y - \text{NextVertex} \cdot y
\]

\[
\text{Denominator} = \text{StartVertex} \cdot x - \text{NextVertex} \cdot x
\]
The buffer points are calculated depending on the case encountered. The parametric equations of a line are used to determine the buffer points. The determined buffer points from two parallel buffer segments ((0–1) and (2–3)) as shown in Figure 4. After determining the two parallel line segments, all the remaining line segments of the poly-line are processed in a similar way. Next, a line intersection test is performed to eliminate overlapping region between adjacent buffer segments. The standard parametric equations of a line and Cramer’s rule are used in the intersection test.

If the adjacent buffer segments intersect the end point of those buffer segments are truncated. However, if no intersection occurs a curve is inserted between those end points. The number of points in the curve depends on the central angle of the curve. On one hand, if there are too few points in the curve the visual effect becomes unsatisfactory, especially if the output is produced on large plotters, or when the user zooms in the results. On the other hand increasing the number of points in the curve results in a significant increase in memory requirements and processing time. Thus there has to be a trade-off in deciding the number of points in the curve. Depending on the desired accuracy the user determines the required number of points in the curve. This procedure complicates the implementation, but produces accurate and compact results. This interpolation of curves is required to ensure that all buffer points are obtained. The obtained buffer points are used to store the entire buffer as new vector polygon layer. The resulting polygon layer facilitates spatial analysis functionalities. Figure 5 depicts the case where truncation and insertion of curve between the end points are required.

The final step adds the bounds to the parallel buffers by capping the start and end node of poly-line with semi-circles of radius equal to the buffer width. For a non-symmetric geometric buffer appropriate curves are used to cap the buffer end points. The final representation of the segment buffer for a disconnected poly-line is as shown in Figure 6. As seen in Figure 6(a) the individual buffer needs to be dissolved after construction of the geometric buffer whereas cases like in Figure 6(b) need not be dissolved. The method to dissolve individual buffers of a vector feature layer is described in Section 3.

If only a two point poly-line is considered, dissolving individual buffers will not be required. However, a two point poly-line is not a generic representation of poly-line data in real world GIS applications. The datasets in practice are bound to contain multiple nodes, thus creating overlapping individual buffers.

The same process is repeated for all poly-lines in a layer. The buffer for an entire poly-line layer consisting of connected and disconnected poly-lines is as shown in Figure 7. As illustrated in Figure 7, buffers due to multiple poly-lines overlap. The technique to dissolve the overlapping buffers is described in Section 4.
2.3. Construction of geometric buffer for a polygon layer

A polygon layer can contain single polygon or a combination of two or more adjacent polygons. A single polygon is made up of multiple poly-lines. Initially, the algorithm always considers a single polygon when forming a buffer around it. A single polygon from a polygon layer and the buffer width are required as input to the algorithm.

For a polygon data, a buffer is a belt of specified buffer distance from the edge of the polygon surrounding the polygon and conforming to its shape. Buffering for polygonal surfaces uses the same concepts used for polyline buffering. For polygon data, a buffer is formed around each poly-line of the polygon, till all the poly-lines are traversed. This process is suitable for single polygon and two or more adjacent polygons wherein a topology is not formed. However, if we have two or more adjacent polygons wherein topology is formed, care needs to be taken to consistently traverse all the poly-lines of the polygon.

A significant consideration in polygon buffering is the removal of internal buffers. The highlighted polygon in Figure 8 represents an internal buffer. To determine an internal buffer, the difference between the minimum and maximum x coordinates of the two buffers is determined. This difference values are compared and the buffer having the lower value of the difference is discarded; as it forms an internal buffer.

3. Algorithm for dissolving a single disconnected polygon layer buffer

While working with poly-line and polygon buffers situations arise wherein due to the nature of the line segments, crossings are formed within the buffer area for a single poly-line or a single polygon. These crossings are unwanted and must be removed. To achieve this, the buffer must be dissolved. Figure 9 illustrates an individual polygon and its buffer which contains these unwanted crossings.

As seen from Figure 9, the crossings arise due to intersections between the buffer segments. The first step thus in removing these crossings is the determination of all the intersection points. Identifying all the intersection points can become a really complex process, with each buffer segment required to be tested against all the others. This brute-force technique requires $O(n^2)$ time and considerably increases the processing time as the number of buffer segments increase. To simplify this problem, sweep-line algorithm with suitable modifications was used.

In this algorithm, there is an imagery line, swept across the plane. Geometric operations are restricted to buffer segments that either intersect or are in the immediate vicinity of the sweep-line whenever it stops, and the complete solution is available once the line has passed over all the buffer segments. The sweep-line technique is well known and has been successfully applied to many different tasks (4).

This section briefly explains how the sweep-line process has been suitably modified and implemented in our algorithm. From the set of determined buffer points, a set of edges is created and each edge is described as having a left vertex and right vertex. If a vertical edge is encountered, the buffer point with smaller y coordinate is said to be the left vertex. The edges are then sorted according to their left vertices. An algorithm with linear logarithmic efficiency has been implemented to sort the edges. An event queue (EQ) is then formed from these sorted edges. As the sweep-line traverses the plane from left to right the edges are removed one after another from the EQ and inserted in a sweep line status (SLS) data structure. Whenever an element (edge) is inserted in SLS, it is checked for intersection with all the segments currently in SLS for intersection. If an intersection occurs the details of the intersection are stored. Whenever an SLS element whose right vertex exceeds the left vertex of the element from the EQ is encountered, that particular SLS element is erased. The sweep-line process is described with the help of Figure 10.

The edges $S_1, S_2, S_3, S_4,$ and $S_5$ in Figure 10 are assumed to be buffer edges. They are first sorted and an EQ is a formed. Initially the EQ contains the edges in following order: $\{S_1, S_2, S_3, S_4, S_5\}$ SLS 

The SLS structure is initially blank. The Sweep Line is traversed across the plane from left to right and makes first contact at the left vertex of edge $S_1$. It is inserted in SLS and removed from EQ. SLS: $\{S_1\}$ EQ $\{S_2, S_3, S_4, S_5\}$

The next intersection of the sweep line is with left vertex of edge $S_2$. At this instance $S_2$ is checked for
intersection with the lone element present in SLS. No intersection point is reported. SLS: \{S_1, S_2\} EQ \{S_3, S_4, S_5\}

The sweep line traverses further and encounters intersection with left vertex of \(S_1\). Here \(S_1\) is removed from SLS as its right vertex exceeds left vertex of \(S_2\). \(S_2\) is checked for intersection with elements present in SLS. Again, no intersection is reported. SLS: \{S_2, S_1\} EQ \{S_4, S_5\}.

The sweep line then encounters intersection with left vertex of \(S_2\) and an intersection of \(S_3-S_4\) is observed. The details of intersection \(I_1\) are stored and the process is iterated till all the buffer edges are encountered. All the intersection points are available when the sweep line has completed its traversal across the plane. SLS: \{S_2, S_1, S_4\} EQ \{S_3\}

This approach addresses the drawback of a sweep line algorithm which fails to report the intersection point if two segments intersect such that end point of one segment lies on the other segment and the three end points are collinear. The algorithm takes care for proper storing of the details associated with such a scenario. All the intersection points are initialised as “not visited”. The points \(A, B, C, D, E, F\), and \(G\) in Figure 9 represent the intersection points that are formed due to crossings in the buffer. After determining intersection points, the next step is determination of the correct buffer points to plot the buffer outline. To make sure that we do not start with a point that lies within the buffer area, the buffer point that has either the minimum or maximum \(x\) coordinate is first determined. The point with minimum or maximum \(x\) coordinate will always lie on the periphery of the buffer. For the course of this discussion we proceed by determining the point with minimum \(x\) coordinate value.

The buffer outline is plotted from the buffer point having minimum \(x\) coordinate value. A buffer is traversed till an intersection point is encountered. On encountering an intersection point the algorithm determines the immediate next point that will lie along the periphery of the buffer. To determine this “Cross Product” the geometry and the details associated with the intersection point are used. The result of the cross product of two vectors is a vector, which is perpendicular to both of the vectors being multiplied and normal to the plane containing them. An important observation here is that, while travelling in anti-clockwise direction around the perimeter, the interior is always to the left. Therefore, if at any intersection point we wish to keep to the perimeter, we must choose to turn right rather than left. By determining Cross Product of the intersection points and its associated vertices the rightmost direction is obtained (5). This principle holds true even if multiple buffer edges have a common intersection point. The Cross Product is given by determinant of

\[
\begin{vmatrix}
i & j & k \\
x_1 & x_2 & x_3 \\
y_1 & y_2 & y_3 \\
\end{vmatrix}
\]

In this illustration \(z_1 = z_2 = 0\). Substituting these values and solving the determinant result \(x_1y_2 - x_2y_1\). Depending on the value of the result, the next point along the periphery is determined. The determination of a Cross Product yields a point that lies on the perimeter of the buffer. The only exception to this is that no other intersection point can lie between the previously encountered intersection point and the next intersection point as determined by the Cross Product. Figure 11 depicts such a case.

When such a case is encountered, determine the intersection point that is nearest to the previously plotted intersection point. Suppose Figure 11 is traversed along line 1. We keep on traversing till the first intersection point \(A\) is encountered. Now point \(A\) becomes the previously plotted intersection point and point \(B\) is the nearest intersection point to point \(A\). After realizing this, the Cross Product of point \(B\) and its associated vertices is determined to get the next point along the periphery of the buffer. This process is then continued till the first plotted buffer point is encountered. The algorithm always considers buffer segments as individual two point segments; thus eliminating the need for explicit boundary case checks for determination of intersection points. The

![Figure 10. Representation of the sweep line algorithm.](image)

![Figure 11. Representation of exceptional case.](image)
final procedure for dissolving polygon buffer is as depicted with the help of Figure 9.

The buffer is plotted in counter-clockwise direction from point \( S \) till the first intersection point is encountered at “\( A \)”. Point \( S \) is the buffer point with minimum \( x \) coordinate. At intersection point “\( A \)” the algorithm computes Cross Product to identify the correct direction and the required point along the periphery of the buffer. Traversing again continues from the point determined by Cross Product till the next intersection point \( I_2 \). At this point, the same procedure is repeated to get next buffer point on the perimeter. The above procedure continues till start point \( S \) is reached. Figure 12(a) depicts the dissolved buffer of test case encountered in Figure 9 and Figure 12(b) illustrates a dissolved buffer for real world GIS data.

4. Methodology for dissolving connected vector layer buffer

As discussed before, a vector layer can be connected or disconnected layer. A typical example of poly-line layer is as shown in Figure 13(a).

The buffer for each individual poly-line in the poly-line layer is first created and then dissolved using the approach discussed in previous sections. Such a dissolved buffer is depicted in Figure 13(b). To completely dissolve the connected poly-line layer as shown in Figure 13(b), we need to first determine the edges along which two or more geometric buffers intersect. The edges of the same geometric buffer will never intersect at this stage because each buffer has been dissolved individually beforehand. To determine intersection points, sweep line algorithm is used again. For connected layers, the unique geometric buffer number needs to be stored in addition to other details associated with the intersection point. This facilitates a jump to the next buffer when an intersection point is encountered.

The sweep line algorithm approach will determine all the intersection points. After determining intersection points, the buffers that do not intersect at all with any other geometric buffers are first plotted. However, not all intersection points are required for plotting the perimeter of the intersecting geometric buffers. To determine the required intersection points, a “Point in Polygon” test is performed to discard the intersection points that lie within other geometric buffers (6,7). A record of intersection points that have been visited is maintained. When all are accounted for, the process is terminated. After the determination of required intersection points, these points are inserted at their appropriate positions. The final step plots the required perimeter of the geometric buffer. Figure 14 is used to illustrate this process. The order of geometric buffers is determined based on the corresponding ID associated with the vector data it represents.

The process begins by selecting an arbitrary intersection point. Computing the Cross Product at this point gives the rightmost fork. Suppose point \( A \) is selected as the first arbitrary point. The algorithm then starts traversing in the direction of \( B_0 \) till the second intersection point at \( B \) is encountered. Here, it selects \( B_1 \) and moves along the boundary to point \( C \) where a jump to \( B_2 \) is required. In this way, the walkabout continues until it reaches intersection point \( A \). This is the initial point and had its marking changed to visited. We have thus identified one perimeter of the intersecting basic geometric buffers. However, all the intersection points have not been visited, thus the algorithm selects any non-used intersection point say point \( J \). The correct direction of movement is determined with the help of cross product geometry and the procedure discussed is again repeated till we reach the initial intersection point. This process is
repeated till all the intersection points have been accounted for.

After the process terminates, the total extent of the set of geometric buffers has been identified by traversing the boundaries of the geometric buffers and crossing from one geometric buffer to another at intersection points. Figure 15 gives an example of the dissolved connected poly-line layer. The same process can be used to dissolve a polygon and a point layer in which the buffers due to adjacent polygons or points overlap. Figures 16 and 17 depict the coherent buffer for point and poly-line layers, respectively.

The flow chart for the entire process of Buffer creation and dissolving the generated buffers is as depicted in Figure 18. Figure 18 is divided into four sections namely Figure 18(a–d). Each of these figures depicts a step in the algorithm.

Figure 18(a) depicts the process of buffer generation for each of the vector feature layer. Irrespective of the input layer, generated buffer will always be a vector polygon layer and stores as an intermediate spatial layer.

Figure 18(b) portrays the flow adopted for implementing the Sweep Line algorithm. The algorithm is used twice in implementation. First occurrence is while-identifying intersection points for dissolving a single disconnected polygon. The other occurrence is to identify the intersection between two or more adjacent buffers to dissolve an entire vector feature layer.

Figure 18(c) shows the steps involved in plotting the periphery of a single disconnected buffer. Such a case is encountered only for poly line and polygon layer.

Figure 18(d) illustrates the details associated with plotting the periphery of an entire vector feature layer.

5. Time complexity

In GIS applications, the datasets can be quite large. In such cases, the accuracy and efficiency of an algorithm plays a vital role. The efficiency of an algorithm is a measure of the resources required by an algorithm as a function of size of the instance. This section considers time complexity as the barometer for determining the efficiency of the algorithm.

5.1. Theoretical time complexity

The algorithm works by creating edges between the obtained buffer points. Assume that there are “n” edges for a geometric buffer. Determining the buffer points gets completed in $O(n)$ time. The insertion of event points in an EQ gets completed in $O(\log n)$ time. Similarly, we can retrieve the elements in $O(\log n)$ time.

When determining, the intersections assume the worst case scenario, wherein each edge of a geometric buffer intersects with every other edge of the buffer. This gives $k=n(n-1)$ intersections so we cannot avoid the worst-case time complexity for $O(n^2)$ time. However, in general practice, far fewer intersections are expected. Since the buffer has a non-zero width, not all the edges of a geometric buffer can intersect with each other. In the most general case, the number of distinct intersections can never exceed $k=n-1$ intersections. The sweep line algorithm will then report the intersections with a time complexity of $O((n+k) \log (n+k))$. After this, each
intersection point is visited only once and thus all the intersection points will be visited in linear time $O(k)$.

In summary, the worst-case complexity will be encountered when $k = n(n-1)$ and the highest order term will generate a complexity of $O(n^2 \log n)$. However, in general practice, $k \ll n(n-1)$ and thus average time complexity is $O(n \log n)$.

5.2. Practical results

The algorithm was implemented using Microsoft Visual C++ 6.0 compiler and Microsoft foundation class library. It was tested on real GIS data representing village boundaries, state highways and on randomly distributed data, with different number of line segments.

6. Conclusion

A geo-computational algorithm for constructing geometric buffers for vector feature layers and dissolving those buffers was presented. The algorithm works by first constructing geometric buffers for a vector feature layer, followed by dissolving individual buffer for each feature of that spatial layer and finally dissolving the buffer for the entire vector layer as a whole. In the worst case, the
efficiency of the algorithm is $O(n^2 \log n)$, but for most of practical implementations, the time complexity is estimated as $O(n \log n)$. The run-time complexity of the algorithm is directly dependent on the number of intersections present between the edges of geometric buffers in a vector feature layer.

Buffering is a very essential feature for spatial analysis. This paper divulges the details of an approach used for buffering in real world GIS applications. This algorithm used the most generic form of a vector layer data to describe the methodology for creating geometric buffers and dissolving them. This algorithm also provides a means to obtain all the coordinates associated with circular arcs to facilitate the storage of all the buffer coordinates. These stored buffer coordinates can be used further for a variety of spatial analysis functionalities. The algorithm was successfully implemented and tested for a variety of complex datasets encountered in real GIS applications; and found to be stable and efficient by a variety of end users.

**Acknowledgments**

This work was an in-house project conducted at Indian Institute of Technology, Bombay.

**Notes on contributors**

Sumeet Bhatia was a research assistant at Centre of Studies in Resources Engineering, Indian Institute of Technology, Bombay and majors in Computer Science. His research interests include spatial analysis theory and algorithm development.

Viral Vira was a software development intern at Centre of Studies in Resources Engineering, Indian Institute of Technology Bombay and majors in Computer Science. His research interests include spatial analysis theory and its application in the transport and telecom domain.

Deepak Choksi is a project engineer at Centre of Studies in Resources Engineering, Indian Institute of Technology Bombay. His research interests include development of GIS software for natural resources applications and development of spatial decision support system for resources applications.

P. Venkatachalam is a professor and head at Centre of Studies in Resources Engineering, Indian Institute of Technology Bombay. Her research interests include development of GIS software for natural and human resources applications, development of spatial decision support system for resources applications and spatial statistics.

**References**

(1) Lo, C.P.; Yeung, Albert K.W. *Concepts and Techniques of Geographic Information Systems*, 1st ed.; Prentice Hall of India.

(2) Laurini, R.; Thompson, D. *Fundamentals of Spatial Information Systems*, 3rd ed.; Academic Press: San Diego, CA, 1994.

(3) Zalik, B.; Zadravec, M.; Clapworthy, G. Construction of a Non-Symmetric Geometric Buffer From a Set of Line Segments. *Comput. Geosci.* 2003, 29, 53–63.

(4) De Berg, M.; van Kreveld, M.; Overmars, M.; Schwarzkopf, O. *Computational Geometry, Algorithms and Applications*; Springer: Berlin, 1997.

(5) Gere James M.; William Weaver, Jr. *Matrix Algebra for Engineers*, 1st ed.; D. Van Nostrand: Princeton, NJ, 1965.

(6) Chong-Wei, Huang.; Tian-Yuan, Shih. On the Complexity of Point in Polygon Algorithm. *Comput. Geosci.* 1997, 23 (1), 109–118.

(7) Yang, Sheng.; Yong, Jun-Hai; Sun, Jiaguang; Gu, Hejin; Paul, Jean-Claude. A Point in Polygon Method Based on a Quasi Closest Point. *Comput. Geosci.* 2010, 36 (2010), 205–213.