Tracking Severe Storms using a Pseudo Storm Concept

Yong Zhang,
Robert E. Mercer, John L. Barron
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada, N6A 5B7
\{mercer;barron\}@csd.uwo.ca

Paul Joe
King City Radar Station
Meteorological Services of Canada
Toronto, Ontario, Canada, M3H 5T4
Paul.Joe@ec.gc.ca

Abstract—Tracking storms in radar images can be conceived of as a problem of tracking deformable objects. Our current relaxation labelling-based tracking algorithm that represents these deformable objects as “fuzzy” points can track objects that undergo shape deformations. One other type of deformation is the splitting of an object into multiple objects or the merging of multiple objects into one object from one image to the next. With our current algorithm, tracks are interrupted when such events happen in image sequences. We remove this deficiency of the current algorithm by adding the concept of a Pseudo Storm to its representational repertoire. With only minor modifications to the current algorithm, the new algorithm can track deformable objects that undergo both merging and splitting events. The new pseudo storm tracking algorithm outperforms our previous storm tracking algorithm for Great Lakes Doppler precipitation datasets.

Keywords—Tracking, Fuzzy Algebra, Relaxation Labelling Algorithm, Merging and Splitting Storms, Pseudo Storm, Doppler Radar

I. INTRODUCTION

Doppler radar [1] is an important meteorological observation tool [2]. Providing severe storm reflectivity and radial velocity data is one of its principal modern applications. Our research has focused on enhancing Doppler radar data by computing 3D full velocity via optical flow methods [3], [4], [5], [6], and tracking storms using a fuzzy algebra representation framework and a relaxation labelling algorithm [7], [8], [9], [10], [11], [12], [13], [14]. Here, we focus on the storm tracking aspect of our research.

Our current storm tracking algorithm consists of a 3D flood fill algorithm, a “fuzzy” point representation of storms, and a relaxation labelling algorithm. While this tracking algorithm deals with the changing shape of storms, it does not compute appropriate storm tracks when storms merge with other storms or when a single storm splits into smaller storms. We solve this merge/split problem by using the concept of pseudo storm, a virtual object that is proposed to handle representational issues posed by merging and splitting storms. This new object requires us to make minimal changes to our original storm tracking algorithm. The modified pseudo storm tracking algorithm produces tracks that allow merging and splitting of storms. We maintain all merged/split storm tracks. Picus et al. [15] also handle newly merged or split storms using a mean-shift kernel-based tracker that only kept the longest track, stopping other Merged) tracks or starting new split tracks. We demonstrate the effectiveness of our new algorithm using data taken from the Detroit and Cleveland Doppler radars (see Figure 1).

II. STORM DETECTION AND REPRESENTATION

A NEXRAD Doppler reflectivity dataset consists of 15 or 16 cones of data. Each cone wall has a different but constant angle (0° to about 20°) with respect to the flat ground. Each cone wall is composed of 360 equally spaced rays. Each ray contains 600 data points. The data points each represent the atmospheric reflectivity of the pyramidal frustums surrounding the real 3D points which are equally spaced along the ray. Reflectivity is a measure of the precipitation density, which we use to detect storms.

An actual storm is detected as a large cluster of connected data voxels with reflectivity values greater than 35 dB. The method to connect the appropriate data voxels, described in our previous work [9], [11], [12], [13], [14], [16], [17], uses a recursive flood fill algorithm. It examines all neighbouring...
data voxels and clusters all connected voxels with reflectivity values greater than 35 dB into storms.

After the storms are detected, the center of each storm is represented as a 3D fuzzy point [12]. At present, we represent a 3D fuzzy point as an ellipsoid [18]. To create the fuzzy point that represents the center of a storm, we first calculate the center of mass of each storm as \( \bar{c} \), \( \bar{y} \) and \( \bar{z} \). This provides us with a Euclidean point representing the center of the storm. Because our present representational framework uses fuzzy points, we need to find an appropriate ellipsoid. To do this for the storm with Euclidean center \( (x, y, z) \), we generate the \( 3 \times 3 \) covariance matrix of the point density as:

\[
\Sigma = \begin{bmatrix}
\sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\
\sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\
\sigma_{xz} & \sigma_{yz} & \sigma_z^2
\end{bmatrix}.
\]

We compute the eigenvectors, \( \hat{\mathbf{e}}_i \), and their corresponding *eigenvalues*, \( \lambda_i \), from this matrix for each storm. We use the three eigenvectors as the three axes of the ellipsoid and the square root of each eigenvalue, \( \sqrt{\lambda_i} \), as the length of the corresponding radius of the ellipsoid.

It should be noted that our storm detection procedure depends solely on the connectivity of storm points in the real dataset. Storm data points will not be classified as belonging to a single storm if they are not directly connected, even if they are in neighbouring locations in the sequence’s images. From our experimental results, we observed that the results from the original tracking algorithm (see Section IV below) show that the storms move relatively slow and tend to remain in the same location over time. However, the storms’ shapes change significantly over time, due to changes in the distribution of storm data points. These observations show the storms are highly deformable over time. Often, large storms split into a number of smaller distinct storms or a set of adjacent smaller storms merge into a large storm. Once one of these merging or splitting events occur, the original tracking algorithm fails to find the appropriate match between two storms in consecutive images. To handle merging and splitting of recognized storms we reintroduce the idea of a pseudo storm. The idea in this work extends the 2D pseudo storm concept first proposed in Krezeski et al. [16] to a 3D setting. Although the concept of pseudo storm was introduced in that work, there was no implementation or experimental results.

### III. Pseudo Storm

The concept of a pseudo storm is used to remove or attenuate the abrupt changes in storm trajectories, to accommodate the merging and splitting of storms and to generate smooth trajectories that reflect a real storm’s motion. The main deficiency of the original storm tracking algorithm is that it can associate only one storm in an image with one track. If storms split or merge, the tracking algorithm will fail to find matches for some of the storms, and will have to begin new tracks for these unmatched storms. Our solution to this problem is to add an enhanced notion of pseudo storm [16] to our tracking algorithm.

**Definition** A pseudo storm is a grouping of two or more real storms in an image that have roughly the same coverage as a real storm in the previous or following image.

A pseudo storm can be thought of as a collection of several real storms that can be considered as single storm for tracking and labelling purposes. Adding pseudo storms to the set of trackable storms provides alternative choices for the tracking algorithm, resulting in better trajectories.

Not all smaller storms with the same coverage as another larger storm in adjacent images will be considered a pseudo storm. The smaller storms’ total volume, \( V_p \), will have to at least 70% of the volume of the larger storm, \( V_n \), in the previous or next image. We also require that all individual candidate storms of a pseudo storm be at least 10% of \( V_n \). To simplify the process, we assume that pseudo storms can only be generated from real storms (so pseudo storms cannot be parts of other pseudo storms). However, because the overlapping storms can be in either the previous or next images, we do allow one real storm to belong to two pseudo storms at the same time, but these pseudo storms must be in different images: ones based on the previous image are denoted as pre-pseudo storms and ones based on the following image are denoted as next-pseudo storms. This ensures that two pseudo storms that are generated from two different images cannot overlap. The detailed algorithm for detecting pseudo storms can be found in Appendix 3.

### IV. The Original Relaxation Labeling Algorithm

#### A. Fuzzy 3D Point Algebra

A set of Euclidean points in 3D space within a 3D ellipsoid can be described as a fuzzy point. This ellipsoid represents an uncertain distribution of a 3D Euclidean point and called a fuzzy 3D point. According to Tang et al. [12], [13], [14], the following definitions related to fuzzy points are required:

1. **A 3D fuzzy point** \( E(c, r) \) is defined as an ellipsoid with center \( c = (x, y, z) \), three radii \( r = (r_x, r_y, r_z) \) and three mutually orthogonal direction vectors \( e = (\hat{e}_x, \hat{e}_y, \hat{e}_z) \) as the three axes.
2. **A fuzzy vector** \( \bar{E} \), from a fuzzy point \( E_1 \) to another fuzzy point \( E_2 \) is defined as the infinite set of all displacement vectors from point \( E_1 \) to \( E_2 \).
3. **A fuzzy magnitude** of a fuzzy vector \( \bar{E} \) is defined as the set of all magnitudes of all vectors in \( \bar{E} \) and is denoted as \( ||\bar{E}|| \).
4. **A fuzzy distance** is defined as the set of all fuzzy magnitudes from one fuzzy ellipsoid to the second one. Usually we present the maximum value and the minimum one as \( d_{\text{max}} \) and \( d_{\text{min}} \).
5) A **fuzzy angle** subtended by a non-zero fuzzy vector $\vec{Q}$ relative to another non-zero vector $\vec{P}$ is the set of angles subtended by and displacement $\vec{p}$ in $\vec{P}$ to another displacement $\vec{q}$ of fuzzy vector $\vec{Q}$. A fuzzy angle is denoted as $\langle \vec{P}, \vec{Q} \rangle_\theta$.

We use these definitions to describe our tracking algorithm.

**B. Disparity Between Two Storms**

To track storms over a series of images, first we build the links between two potentially matched storms in the consecutive images as the **disparity** of two storms. We denote the $j^{th}$ image in the image dataset as $I_j$. We denote the storm, $S_j$, in Image $I_j$ as the disparity’s **tail**, and the storm, $S_{j+1}$, in Image $I_{j+1}$ as its **head**. In order to evaluate how strong the link between them is, we investigate the similarities between the two storms in several aspects, and denote these similarities as **certainty values**. Each certainty value is in the range $[0, 1]$, and represents the similarity of the two storms in one aspect, such as their size certainty, $f_s(S_j, S_{j+1})$, their position certainty, $f_p(S_j, S_{j+1})$, the storms’ velocity certainty, $f_v(S_j, S_{j+1})$, and their orientation certainty (the similarity of the axis orientations of the ellipsoids representing the fuzzy points), $f_o(S_j, S_{j+1})$.

A disparity can be built only if all these certainties are higher than preset thresholds. All the certainties are weighted and summed together, giving the **overall certainty** value, $f(S_j, S_{j+1})$ for each disparity, $\vec{d} = S_jS_{j+1}$, which is also in the range $[0, 1]$. Weights have been assigned to the different certainties according to their importance. Thresholds and weights have been decided by trial and error. These overall certainty values become the initial (iteration 0) certainty values, $p_0(\vec{d})$, for each disparity, $\vec{d}$, in the relaxation labelling procedure, described in Section IV-D. These certainty values are also used in the computation described next.

**C. Adjacency Between Two Disparities**

The set of **adjacencies** is based on pairs of adjacent disparities having a storm in common. For each preceding disparity $S_jS_{j+1}$ and succeeding disparity $S_{j+1}S_{j+2}$ having the storm $S_{j+1}$ in common in Image $I_{j+1}$ an adjacency $A_{j+1}$ is built.

Extending the work of Tang et al. [12], [13], [14] we have defined several compatibility values to measure the strength of an adjacency:

1) the **minimum length compatibility**, $C_{d_{\text{min}}}$,
2) the **maximum length compatibility**, $C_{d_{\text{max}}}$,
3) the **fuzzy angle compatibility**, $C_{\theta}$,
4) the **size compatibility**, $C_s$,
5) the **position compatibility**, $C_p$,
6) the **orientation compatibility**, $C_o$,
7) the **velocity compatibility**, $C_v$.

All of the compatibility values are calculated based on the certainties from the two corresponding disparities. They are summed together, giving the **partial compatibility**, $C_p$:

$$C_p = \omega_{d_{\text{min}}} C_{d_{\text{min}}} + \omega_{d_{\text{max}}} C_{d_{\text{max}}} + \omega_{\theta} C_{\theta} + \omega_s C_s + \omega_p C_p + \omega_o C_o + \omega_v C_v,$$

(2)

The values for the weights $\omega_{d_{\text{min}}}$, $\omega_{d_{\text{max}}}$, $\omega_{\theta}$, $\omega_s$, $\omega_p$, $\omega_o$, and $\omega_v$ are given in Table I. The velocity and fuzzy angle weights are 0.0 because the data is profoundly different than that for which these measures were created [14]. See [19] Section 6.4.4.2 for details. The partial compatibility remains the same throughout the tracking procedure. The **overall compatibility**, $C_{\text{overall}}$, is calculated by adding $C_p$ and the overall certainties (iteratively updated by the algorithm in Section IV-D) from the preceding and succeeding disparities.

**D. Relaxation Labelling Algorithm**

We adopt a relaxation labelling algorithm to iteratively update the overall certainty of each disparity by relaxation on the overall compatibility among its adjacent disparities. At each iteration, $k$, we first calculate the values of two constraints for each disparity, the **temporal consistency constraint** and the **spatial consistency constraint**. The first constraint is meant to measure the strength of a disparity’s links to its tail’s preceding and its head’s succeeding disparities. The latter constraint measures a disparity’s certainty value relative to all of the other disparities occurring between the same images. If the constraint value is positive, it is added to the variable called **supporting evidence**, $E_s$; if negative, its absolute value is added to the variable **contradictory evidence**, $E_c$. The overall certainty value of each disparity becomes:

$$p_k(\vec{d}) = \begin{cases} \frac{1}{2} \left( 1 + \frac{\omega_s E_s + \omega_c E_c}{\omega_s + \omega_c} \right) & \text{if } E_s \neq 0, E_c \neq 0, \\ 0, & \text{otherwise,} \end{cases}$$

(3)

where $\omega_s$ and $\omega_c$ are the weights of the supporting and contradictory evidence. This procedure ends when the refinement of overall certainty converges to an acceptable level or a maximum number of iterations has been reached. If convergence is achieved, we have a set of disparities among all the image storms with stable certainty values. These values are used by a greedy algorithm to select the adjacencies, with its two disparities, that should form the storm tracks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>(\omega_s)</td>
<td>0.4</td>
</tr>
<tr>
<td>Orientation</td>
<td>(\omega_o)</td>
<td>0.2</td>
</tr>
<tr>
<td>Position</td>
<td>(\omega_p)</td>
<td>0.2</td>
</tr>
<tr>
<td>Velocity</td>
<td>(\omega_v)</td>
<td>0.0</td>
</tr>
<tr>
<td>Angle</td>
<td>(\omega_{\theta})</td>
<td>0.0</td>
</tr>
<tr>
<td>Minimum Length</td>
<td>(\omega_{d_{\text{min}}})</td>
<td>0.1</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>(\omega_{d_{\text{max}}})</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Table I**

The certainty and compatibility weights used.
V. PSEUDO STORM TRACKING ALGORITHM

In the pseudo tracking algorithm, the calculation of disparities between two storms remains the same as in the original algorithm except that now pseudo storms are candidates for the head or tail of a disparity. To accomplish this, we expand the notion of an adjacency to include $1:n$ and $n:1$ relationships between disparities. A new data structure records the connections between a pseudo storm disparity and the disparities associated with the real storms comprising it.

A. Building Connections

Figure 2a shows an artificial image sequence comprised of three consecutive images: Both Images 1 and 2 have two real storms and Image 3 has one larger storm that the two storms in Images 1 and 2 have merged into. Two disparities can be recovered from Image 1 to Image 2. However the storm sizes change too much from Image 2 to Image 3 to allow any disparities to be built. In Figure 2b, a disparity can connect the pseudo storm (the dashed red ellipse) in Image 2 and the real storm in Image 3. Since this newly added disparity shares no common storms with the previous disparities, we add connection relationships to connect the pseudo storm’s and its component real storms’ disparities.

Figure 3a shows a more complicated scenario where two pseudo storms overlap one real storm. In Image 1, the upper storm splits into 2 smaller horizontally positioned storms (a pseudo storm) in Image 2. The two vertically positioned storms are linked via a pseudo storm in Image 2 to the larger storm in Image 3. Figure 3b shows the connections created to link the appropriate disparities.

B. Building Adjacencies

In addition to storing connection relationships, the adjacency structure also records the $1:n$ and $n:1$ relationships between its disparities. After introducing a pseudo storm, one and only one side of the adjacency can have more than one disparity. The preceding and succeeding disparities of an adjacency are connected by a connection that we just built. Some typical adjacency structures are shown in Figure 4a. No connection will be created for a real storm that doesn’t have a disparity associated with it.

It is possible that a real storm can have more than one disparity connected to it, as shown in Figure 4b. In this case, an adjacency is built for each possible combination. For the storms in Figure 4b, the number of adjacencies is $2 \times 2 = 4$. As shown in Figure 4c, the disparities from different real storms in a pseudo storm can be linked to the same storm in Image 3. Since two storms cannot track to the same storm, of the four possible combinations, this combination of disparities does not get an adjacency built.

The computation of the seven compatibility values for adjacencies, given in Section IV-C has been modified for pseudo storms: the preceding and succeeding disparities are considered as sets and the mean of the certainty values for each set of disparities is used to compute the compatibility values. For example, size compatibility is computed as:

$$C_s(S_j, S_{j+1}) = \frac{f_s(S_j, S_{j+1}) + f_s(S_{j+1}, S_{j+2})}{2},$$

where $f_s(S_j, S_{j+1})$ is the mean of the size certainty values of the preceding disparity set for pseudo storm $S_{j+1}$, while...
Figure 4. (a) The organization of a typical adjacency with pseudo storms; (b) building an adjacency where the real storms have multiple disparities; (c) building an adjacency in a more complex situation.

\[ f_s(S_{j+1}, S_{j+2}) \]

is the mean of the size certainty values of the succeeding disparity set for pseudo storm \( S_{j+1} \).

The relaxation labelling algorithm works in a manner similar to that which is described in Section IV-D.

When the relaxation algorithm has finished, the selection procedure chooses the disparities with the highest overall certainty values to construct the final tracks. This procedure differs from the previous greedy algorithm to allow for the new type of adjacency. In the original tracking algorithm, one disparity can connect to only one preceding disparity and one succeeding disparity. This yields a restricted set of tracks. In the pseudo tracking algorithm, a disparity can connect to more than one disparity at either its head or tail. Our new track selection algorithm uses mutual recursion between the adjacency and its disparities to make sure the final disparities are selected appropriately. Pseudo code is given in Section VIII. The final selected tracks that represent the trajectories of storms will be a network of several tracks connected to each other. The network is a better representation of the merging and splitting of the storms that have been grouped together by a pseudo storm.

VI. RESULTS AND DISCUSSION

We present some tracking results using the original and the pseudo storm tracking algorithms. These results, shown in Figure 5, illustrate how the storm tracks differ with the two algorithms. Figures 5a, 5b, 5c, 5g, 5h and 5i are generated by the original tracking algorithm, and the corresponding pseudo storm algorithm results are in Figures 5d, 5e, 5f, 5j, 5k and 5l. The large grey areas in each figure indicate non-storm data (greater than 0 dB and less than 35 dB); the other colours represent separate storms. The black ellipses drawn over these storms show the 2D projections of the 3D fuzzy points indicating the storms’ 2D sizes, shapes and orientations. The red dashed ellipses represent the pseudo storms generated by our algorithm. The blue curves are the tracks of storms including segments that are derived later in the image sequence, and the red dots show the centers of the ellipsoids being used to determine the track segment in the current image.

Figure 5d contains a pre-pseudo storm that is based on a storm in the image prior to this one. Figure 5e contains a pre-pseudo storm that is based on the large purple storm centrally located in Figure 5d. What appears to be a potential next-pseudo storm in Figure 5e (the blue and purple storms matching the light green storm in Figure 5f) is not created because the thresholds for generating that pseudo storm are not met. Figure 5j contains two pre-pseudo storms that are based on the light green and dark green storms in Figure 5f. Figure 5k contains two pre-pseudo storms, one based on the light blue storm and the other based on the dark green (real) storm in Figure 5j. The two pre-pseudo storms in Figure 5l are based on the purple and green storms in Figure 5k.

The tracks computed using the original algorithm are shorter than those computed by the pseudo storm algorithm. These abbreviated tracks are indicative of the original algorithm’s inability to build a disparity to an appropriate storm in the next image to continue the tracks. What can be seen in Figures 5a and 5b is that the large centrally located purple storm has split into three storms. The original tracking algorithm cannot build a disparity for this large purple storm, since no storm in the next image meets all the certainty value thresholds, thus the track is not extended. On the other hand, the pseudo storm tracking algorithm is able to link the large purple storm in Figure 5d with the pseudo storm comprising the three storms in Figure 5e, thus maintaining the track. In the next image, new tracks are generated for each storm with the original tracking algorithm (Figure 5c), while with the pseudo tracking algorithm, the previous track continues (Figure 5f). This track eventually divides into two tracks as shown in the next images in Figures 5k and 5l. The pseudo storm in Figure 5h not only successfully matches the previous storm, but is also capable of dividing it into two tracks to continue tracking the storms in the following images. In the last pair of images in Figures 5k and 5l, the
Figure 5. The tracks for the six-image sequence in figures (a), (b), (c), (g), (h) and (i) have been generated using the original tracking algorithm, and the tracks in the same six-image sequence in figures (d), (e), (f), (j), (k) and (l) have been generated using the pseudo storm tracking algorithm. The sequence is extracted from a multiple radar data sequence using the Detroit and Cleveland Doppler radar data on August 19th, 2007. The pseudo storms (red ellipses) in figures (d), (e), (j), (k) and (l) are pre-pseudo storms. The pre-pseudo storm in figure (d) is based on storms in a prior image not shown here. The pre-pseudo storms in figures (e), (f), (j) and (k) are based on storms in figures (d), (f), (j) and (k), respectively. The complete tracks, that is including future disparities, are the blue curves in each figure. The red dots indicate the centers of the ellipsoids that add a disparity to the track in the current image.

Previously divided storms split into smaller storms yet again.

The pseudo storm algorithm has successfully tracked the large purple storm in the first image through to the smaller green and dark purple storms in the final image. As mentioned earlier, the potential next-pseudo storm in Figure 5e was not detected. If it had been, another track following the pseudo storm in Figure 5d to the dark blue storm in Figure 5e, and then combined with the purple storm as a pseudo storm to the light green storm in Figure 5f. This track would then have followed the purple storm in the next three images.

It can be concluded that for the original algorithm, the
continuation of a storm trajectory is always disrupted by abrupt changes to the storm’s shape, caused, for example, by the merging or splitting of the storm. The track is interrupted and a new trajectory must be started to accommodate these abrupt changes. This problem is successfully solved by introducing pseudo storm into the tracking process.

VII. Conclusion

The new pseudo storm algorithm using the pseudo storm concept to model storm merging or splitting events now deals not only with storm events such as shape and size change but also with events such as storm merging and splitting. This algorithm clearly outperforms our original storm tracking algorithm by explicitly and correctly accounting for storm merging and splitting using pseudo storms as the transition device. We focussed on tracking severe storms using real Doppler data to demonstrate our pseudo storm concept. However, this concept could be extended to other similar tracking problems, such as recognizing and tracking large swarms of insects or flocks of birds. These swarms and flocks are deformable objects that can undergo merging and splitting events, yet may need to be tracked as a single object.

VIII. Appendix: Pseudo Code

Algorithm 1 Adjacency Selection SelectAdj (adj)

Require: change adj’s status and process all its disparities
Ensure: adj and all its disparities must be available
Adj.status ← CHOSEN
for each disparity, disp, in adj’s preceding disparity list do
  disp.status ← CHOSEN
end for
for each disparity, disp, in adj’s following disparity list do
  disp.status ← CHOSEN
end for
for each disparity, disp, in adj’s preceding disparity list do
  Call function SelectDisp (disp, adj)
end for
for each disparity, disp, in adj’s following disparity list do
  Call function SelectDisp (disp, adj)
end for

Algorithm 2 Disparity Selection SelectDisp (disp, adj)

Require: disp contains a list of adjacencies that use it as the preceding or following disparity
Ensure: disp is adopted by adj
if disp is adopted by adj as preceding disparity then
  for each adjacency, a, that is not adj and uses disp as preceding do
    if a.status = CHOSEN then
      Report error
    end if
  end for
if no adjacency that uses disp as following disparity has been chosen then
  Find the adjacency with highest compatibility value that uses disp as following disparity, a
  Call function SelectAdj (a)
end if
end if
if disp is adopted by adj as following disparity then
  for each adjacency, a, that is not adj and uses disp as following do
    if a.status = CHOSEN then
      Report error
    end if
  end for
if no adjacency that uses disp as preceding disparity has been chosen then
  Find the adjacency with highest compatibility value that uses disp as preceding disparity, a
  Call function SelectAdj (a)
end if
end if

Algorithm 3 Pseudo Storm Tracking Selection SelectTracks (AdjacencySet)

Require: All available adjacencies saved in AdjacencySet
Ensure: Select adjacencies to generate suitable tracks
for each AdjacencyList in AdjacencySet do
  adjacencyLeft ← AdjacencyList.num
  while adjacencyLeft > 0 do
    Find the available adjacency that has highest compatibility value, adj
call function SelectAdjacency (adj)
Collect the number of adjacencies left in AdjacencyList, m
adjacencyLeft ← m
end while
end for
Algorithm 4 Pseudo Storm Detection

Require: Three continuous images with storms detected.
Ensure: Obtain the two pseudo storm numbers for each storm

for each storm $i$ in the current image do
  for each storm $j$ in the previous image do
    if $i$'s center is in $j$ then
      PreviousStack[$j$].push($i$)
    end if
  end for
  for each storm $k$ in the next image do
    if $i$'s center is in $k$ then
      NextStack[$k$].push($i$)
    end if
  end for
end for

for each storm $j$ in the previous image do
  if PreviousStack[$j$].length $\geq 2$ then
    PreviousPseudoNumber $\leftarrow$ PreviousPseudoNumber + 1
    while PreviousStack[$j$].length $\geq 0$ do
      $i$ $\leftarrow$ PreviousStack[$j$].pop()
      $i$.PreviousStorm $\leftarrow$ $j$
    end while
  end if
end for

for each storm $k$ in the next image do
  if NextStack[$k$].length $\geq 2$ then
    NextPseudoNumber $\leftarrow$ NextPseudoNumber + 1
    while NextStack[$k$].length $\geq 0$ do
      $i$ $\leftarrow$ NextStack[$k$].pop()
      $i$.NextStorm $\leftarrow$ $k$
    end while
  end if
end for

ACKNOWLEDGMENTS

This work was partially funded through Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants to Robert E. Mercer and John L. Barron.

REFERENCES


