Best Track: Real Time

David Harrison
OU CIMMS / NSSL

Updated June 27, 2017

Introduction:

Best Track: Real Time (BTRT) is a Python package designed to read in the output of a third-party storm identification and tracking algorithm (i.e. WDSS-II segmotion; ProbSevere) and improve upon that algorithm's tracking by correcting unjustifiable breaks in an object's track. BTRT is loosely modeled after the WDSS-II besttrack algorithm, but modified to support real-time processing of an operational model's output. The intended usage, syntax, and theory behind this package are described in detail within this document.

Terminology:

The following terminology will be used throughout this document:

- Cell - A single geometric object produced by a third-party algorithm at a single time step. This will usually represent a single storm cell in most contexts, but could be an area of low-level rotation, an area of high MESH, etc. depending on the third-party algorithm settings.

- Track - A series of cells at different time steps which represent the entire lifespan of the single identified object

- Real Time - The BTRT code is run on the newest data files as they are produced by the third-party algorithm. The algorithm only has knowledge of the current objects and their history

- Post-processing / archiving - The BTRT code is provided with a complete set of data files to process sequentially.

- ProbSevere - A statistical model produced by NOAA / CIMSS that predicts the probability that a storm will produce severe weather in the near-term. BTRT is able to read in both the original ascii files and the new json files produced by the ProbSevere algorithm.

- Segmotion - An algorithm designed by Valliappa Lakshmanan (formerly OU CIMMS / NSSL) to identify and track objects on spatial grids. Segmotion produces a number of

output files, including a netCDF file, with information on the identified objects. However, BTRT currently only processes the xml PolygonTable files.

Usage:

BTRT can be integrated into an existing Python project for real time applications, or the provided scripts can be used as a standalone program for post-processing. We will break down the usage instructions for each type of application.

Real-Time Applications

For real-time applications, it will likely be desirable to integrate the BTRT code within another Python project so that the algorithm can be routinely run on new data. In order to do this, first be sure that the contents of this git repository are downloaded and extracted into a directory that your Python project has access to.

Next, import besttrack_RT into your project as follows:

```
import besttrack_RT as btrt
```

BTRT can then be run by including the following line in your code:

```
btrt.besttrack_RT(currentTime, inDir, inSuffix, historyPath,
bufferTime, bufferDist, historyTime, outDir, ftype, outtype)
```

Let's take a look at each of these parameters:

Parameter	Type	Description
currentTime	datetime.datetime	The date / time of the most recent file to be processed
inDir	string	The directory containing the files to be processed
inSuffix	string	The lowest subdirectory of the input directory. This is useful if files are spread out in a hierarchy. e.g. Use `'scale_0'` to get all files in the hierarchy `cref/01/scale_0/`, `cref/02/scale_0/`, etc. Use `''` if there is no subdirectory.
historyPath	string	The full file path (including extension) of the history json file. If no file exists, specify what you want it to be saved as (i.e. `'./history.json'`)

bufferTime	integer	The time threshold (minutes) to use when associating cells with a track. Usually try to keep this value the same as or slightly larger than the size of timestep produced by the third-party algorithm.
bufferDist	integer	The distance threshold (km) to use when associating cells with a track. Recommend around 10 km.
historyTime	integer	The amount of timesteps (minutes) to keep old cells in the history file. Larger values will make the history file large and may slow down the code. Recommend 30 minutes for real-time processing of 2 to 5 minute timesteps.
outDir	string	Filepath where the new files will be saved. **Warning**: If outDir is the same as inDir, the original files will be overwritten!
ftype	string	Type of input files to process (ascii, json, or xml)
outtype	string	Type of file to produce (ascii, json, xml, or seg_json). Note: Not all ftypes are compatible with all outtypes. See **Output**.

Example:

```
currentTime = datetime.datetime.strptime('20150506_203639',
      '%Y%m%d_%H%M%S')
inDir = '/localdata/ProbSevere/json/20160901'
outDir = '/localdata/ProbSevere/json/test/'
historyPath = '/localdata/ProbSevere/json/test/history.json'
bufferTime = 3 # minutes
bufferDist = 10 # km
historyTime = 30 # minutes

btrt.besttrack_RT(currentTime, inDir, '', historyPath, bufferTime,
bufferDist, historyTime, outDir, 'json', 'json')
```

Once the code has finished running, new files for each timestep will be available in the specified output directory. These files will contain the same information as the original input files, except that each object's track ID will be modified according to the track corrections produced by

BTRT. Additionally, there will be a new field added to the file containing the object's original track ID. Segmotion files will be modified further to show updated attributes such as object age and speed. A full breakdown of the various outputs can be found in the **Output** section.

Post-Processing Applications

BTRT can be used to post-process data much in the same way it is used for real-time applications. Although you can write your own code to run BTRT on old data using the code shown in the previous section, we have included a stand-alone script that you can call from the command line for convenience. This program, called archiveBTRT, will read and process all files located within a specified directory that fall within a certain time range. The method for running this code is as follows:

From a command terminal, simply run the command:

```
python archiveBTRT.py start end inDir outDir -it -ot -bt -bd -ht -cd
```

Many of these parameters and flags are similar to those shown for real-time processes above, but let's go over them again!

Parameter	Default	Description
Start (required)		The beginning of the desired time range in yyyymmdd.
End (required)		The end of the desired time range in yyyymmdd. The end date is inclusive. So, for example, if you wanted to process all files in May 2015, you would use a start time of 20150501 and an end time of 20150531. Currently, BTRT does not support time ranges on scales smaller than a day.
inDir (required)		Location of the source files to be processed (same as inDir above)
outDir (required)		Where to save the output (same as outDir above)
-it --itype	ascii	Type of input file (ascii, json, or xml; same as ftype)
-ot --otype	ascii	Type of output file (ascii, json, xml, or seg_json; same as outtype)

-bt --buffertime	3 (min)	The time threshold (minutes) to use when associating cells with a track. (same as bufferTime)
-ht --historytime	30 (min)	Amount of timesteps (minutes) to keep old cells in the history file (same as historyTime)
bd --bufferdist	10 (km)	The distance threshold (km) to use when associating cells with a track. (same as bufferDist)
-cd --convectiveday	False	Set this flag if your directory is organized by convective day

Example:

```
python archiveBTRT.py 20140501 20140501
/localdata/ProbSevere/segmotion/20140501/
/localdata/ProbSevere/segmotion/test2/ -it xml -ot seg_json -bt 6 -bd
10 -ht 30 -cd
```

Once complete, the new files will be available in the specified output directory. Unlike for real-time applications, archiveBTRT will automatically delete the history file when it has completed processing the desired time range. This is intended to help save on disk space and avoid leaving unnecessary files which the user has to clean up later.

Input:

BTRT currently supports input from ProbSevere and WDSS-II Segmotion in the form of ascii, json, and xml files. These files need to have a specific format for BTRT to work properly, which we will now cover for each extension.

ASCII

ASCII files were what ProbSevere produced up to the summer of 2017 when they changed over to a json output. In order to allow for post-processing of older data, we have maintained support for the original ascii file format. In the ascii format, each file represents a single timestep, and every line in that file is a single object at that time step. The various attributes are separated by a colon, and values within a list (i.e. lat/lon) are separated by commas. Here is an example of a single object in a ProbSevere ascii file:

```
RAD:3:204 J/kg:22.7 kts:2048Z 0.33
in.:N/A:N/A:43.68,-106.55,43.68,-106.54,43.69,-106.53,43.69,-106.52,4
3.68,-106.51,43.67,-106.50,43.66,-106.50,43.65,-106.50,43.65,-106.51,
```

```
43.65,-106.52,43.65,-106.53,43.66,-106.54,43.67,-106.55,43.68,-106.55
:88286:5.92795:-0.575045
```

For BTRT to work properly, the following standards must be followed within the ascii file:

- The list of lats and lons must be the 8th attribute in the line
    - Each item must be separated by a comma
    - The order must be lat, lon, lat, lon, …
- The object's track must be the 9th attribute in the line
- The object's probability must be the 2nd attribute in the line
- The object's motion east must be the 10th attribute in the line
- The object's motion south must be the 11th attribute in the line
- The filename must follow the format: SSEC_AWIPS_PROBSEVERE_YYYYMMDD_hhmmss.ascii

JSON

ProbSevere began producing JSON files during the summer of 2017 in order to include more information about each object. An example of a single object in a json file is as follows:

```
{"type": "Feature",
   "geometry": {
      "type": "Polygon",

"coordinates":[[[-81.88,37.52],[-81.87,37.52],[-81.86,37.52],[-81.86,
37.51],[-81.85,37.50],[-81.85,37.49],[-81.85,37.48],[-81.86,37.47],[-
81.87,37.46],[-81.88,37.46],[-81.89,37.46],[-81.90,37.46],[-81.90,37.
47],[-81.90,37.48],[-81.90,37.49],[-81.89,37.50],[-81.88,37.51],[-81.
88,37.52]]]
   },
   "properties": {
      "PROB":"0",
      "PROBHAIL":"0",
      "PROBWIND":"0",
      "PROBTOR":"0",
      "LINE01":"ProbHail: 0%; ProbWind: 0%; ProbTor: 0%",
      "LINE02":"-MUCAPE: 542 J/kg; MLCAPE: 427 J/kg ; MLCIN: -3
J/kg",
      "LINE03":"-EBShear: 7.8 kts ; SRH 0-1km AGL: 0 m^2/s^2",
      "LINE04":"-MeanWind 700-900mb: 4.4 kts",
      "LINE05":"-MESH: 0.04 in.",
      "LINE06":"-VIL Density: 1.00 g/m^3",
      "LINE07":"-Flash Rate: 0 fl/min",
```

```
"LINE08":"-Flash Density (max in last 30 min): 0.00
      fl/min/km^2",
"LINE09":"-98% LLAzShear: 0.002 /s",
"LINE10":"-98% MLAzShear: 0.004 /s",
"LINE11":"-Norm. vert. growth rate: 2000Z 0.4%/min (weak)",
"LINE12":"-Glaciation rate: N/A",
"LINE13":"-Wetbulb 0C hgt: 3183 m AGL",
"LINE14":"-Lapse Rate 700-500mb: 6.4 C/km",
"LINE15":"-minAvgRH 700-450mb: 21%",
"LINE16":"Avg. beam height (ARL): 3.68 kft / 1.12 km",
"LINE17":"Object ID: 60418",
"MUCAPE":"542",
"MLCAPE":"427",
"MLCIN":"-3",
"EBSHEAR":"7.8",
"SRH01KM":"0",
"MEANWIND_700-900":"4.4",
"MESH":"0.04",
"VIL_DENSITY":"1.00",
"FLASH_RATE":"0",
"FLASH_DENSITY":"0.00",
"MAXLLAZ":"0.002",
"P98LLAZ":"0.002",
"P98MLAZ":"0.004",
"MAXRC_EMISS":"2000Z 0.4%/min (weak)",
"MAXRC_ICECF":"N/A",
"WETBULB_0C_HGT":"3183",
"LR_700_500":"6.4",
"MIN_AVG_RH_700_450":"21",
"AVG_BEAM_HGT":"3.68 kft / 1.12 km",
"COLOR":"90 90 90",
"BCOLOR":"90 90 90",
"OPACITY":"0",
"BOPACITY":"100",
"WIDTH":"2",
"MOTION_EAST":"0.717",
"MOTION_SOUTH":"-4.689",
"ID":"60418",

  }}
```


The ProbSevere JSON format is still under development and many of these features may change over time. However, for BTRT to properly read the file, these standards must be followed:

- Lats and lons are located in [‘geometry’][‘coordinates’][0]
- The track is located at [‘properties’][‘ID’]
- The motion east is located at [‘properties’][‘MOTION_EAST’]
- The motion south is located at [‘properties’][MOTION_SOUTH’]
- The filename must follow the format: SSEC_AWIPS_PROBSEVERE_YYYYMMDD_hhmmss.json

XML

The WDSS-II Segmotion algorithm produces a number of output files. However, BTRT only reads in the xml files produced for the polygon table. These files will vary somewhat depending on the settings used when running segmotion, but here is a partial example of an xml file with only a single object:

```
<data>
 <datacolumn name="Age" units="s">
  <item value="255.877000"/>
 </datacolumn>
 <datacolumn name="LatRadius" units="km">
  <item value="2.725755"/>
 </datacolumn>
 <datacolumn name="Latitude" units="Degrees">
  <item value="35.223099"/>
 </datacolumn>
 <datacolumn name="LonRadius" units="km">
  <item value="0.979491"/>
 </datacolumn>
 <datacolumn name="Longitude" units="Degrees">
  <item value="-97.752802"/>
 </datacolumn>
 <datacolumn name="MotionEast" units="MetersPerSecond">
  <item value="15.061001"/>
 </datacolumn>
 <datacolumn name="MotionSouth" units="MetersPerSecond">
  <item value="-6.519001"/>
 </datacolumn>
 <datacolumn name="Orientation" units="degrees">
  <item value="76.136566"/>
 </datacolumn>
 <datacolumn name="RowName" units="dimensionless">
  <item value="1"/>
 </datacolumn>
 <datacolumn name="Speed" units="MetersPerSecond">
```

```
   <item value="16.411310"/>
  </datacolumn>
  <datacolumn name="StartTime" units="dimensionless">
   <item value="20130520-192056"/>
  </datacolumn>
 </data>
```

BTRT is able to process xml files produced using a variety of segmotion settings. However, there are a few standards which need to be followed in order for the code to work properly:

- The algorithm settings must produce a polygon table (xml file)
- The file must contain the following fields:
    - LatRadius
    - Latitude
    - LonRadius
    - Longitude
    - MotionEast
    - MotionSouth
    - Orientation
    - RowName
    - Age
    - StartTime
- The filename must follow the format: YYYYMMDD-hhmmss.json

Output:

Compatibility

Due to data limitations, not all input file types can be converted to all output file types. In fact, BTRT is currently only able to convert the xml file type to a different file type than the original source. For clarity, a table showing the compatible input and output file types is shown below:

	Input	**Output**
ProbSevere	ascii	ascii
	json	json
Segmotion	xml	xml, seg_json

Each output file type will now be described in detail.

ASCII

The ascii file produced by BTRT is exactly identical to the original source file except for two modifications. First, the object's track number (attribute 9) may be modified according to the track corrections provided by the BTRT algorithm. Second, there will be a new attribute added to the end of the line which contains the original track number (the original attribute 9) for each object. All other attributes will remain unmodified and in the same order as in the original file. However, it is not guaranteed that the objects within the file will remain in the same order.

JSON

Similar to the ascii file output, BTRT will produce a ProbSevere json file that is nearly identical to the original. The ['properties']['ID'] field of each object (feature) will be modified according to the algorithm's track corrections, and there will be a new field added at ['properties']['besttrack'], which contains the object's original track (ID) number. Note that although the internal structure of the json will remain the same, the human-readable file may appear slightly different to the original. The order of objects (features) within the new output is not guaranteed.

XML

The BTRT xml (segmotion) output files will contain a few more changes than those listed previously, but will retain the same structure as the original xml file. First, the RowName column will be renamed to Track for consistency, and the values will be updated with the new tracking information provided by the BTRT algorithm. Next, a new column labeled OldTrack will be added to contain the original track number for each object. Finally, the Age, StartTime, MotionEast, MotionSouth, and Speed attributes of each object will be recalculated from the new tracking information and updated accordingly. The order of objects is not guaranteed to be the same as in the original file, but the order of the rows will match for each attribute. All header information (lines above the <data> tag) will be identical to the original file.

SEG_JSON

The seg_json file type is currently the only custom output provided by the BTRT algorithm. The intention of this output is to convert the segmotion xml file type to a more convenient json file type. When producing a seg_json file, each object's track, old track, age, start time, motion east, motion south, and speed are calculated as for the xml output. These and the other attributes within the original xml file are then saved as a json file. The fields available within the json file will vary depending on what settings were used to run segmotion, but the new json is guaranteed to contain the attributes mentioned previously. The data column names from the original xml file will be used as the keys for the associated values, and every line in the json file will represent an individual object. ***Note that each line is a separate json object.*** So, for example, a json file with two objects would like like this:

{"LonRadius": 3.761443, "MaxRef": 40.5, "Orientation": 127.195007, "MotionSouth": -10.952001, "SubType": "scale_0", "Track": 1, "Speed": 14.886002487840784, "Age": 0.0, "Longitude": -122.7, "StartTime": "20140501_000215", "OldTrack": 1, "MotionEast": 10.082001, "scale": "0.000000", "MaxRef-10C": -99903.0, "Latitude": 49.54, "LatRadius": 6.18519, "Size": 69.094315}
{"LonRadius": 3.504732, "MaxRef": 47.5, "Orientation": 121.516052, "MotionSouth": 4.942, "SubType": "scale_0", "Track": 2, "Speed": 9.483470682614094, "Age": 0.0, "Longitude": -122.33, "StartTime": "20140501_000215", "OldTrack": 2, "MotionEast": 8.094001, "scale": "0.000000", "MaxRef-10C": -99903.0, "Latitude": 49.54, "LatRadius": 6.333373, "Size": 60.912094}

How it works:

The main algorithm used in BTRT is largely derived from the WDSS-II besttrack algorithm described in Lakshmanan et al. 2015. We highly suggest reading their paper for all the mathematical details, but we will cover the basics of our implementation here:

Initial Processing

The first time BTRT is run on a dataset (both for real-time and post-processing applications), the program loads the file for the current timestep as well as any available files dated prior to the current timestep that fall within the user-specified history time. For example, if the current time in a real-time run is 0100 UTC and the history time is set to 30 minutes, then BTRT will load files dated from 0030 UTC to 0100 UTC if they exist within the input directory. This allows the BTRT algorithm to learn the current objects' histories for the next step of the algorithm.

Once BTRT has processed at least one time step, a history json file, which contains the relevant information of all cells processed during previous time steps, will be generated in the designated location. These cells will remain in the history file up to the number of timesteps designated in the initial parameters (historyTime). By maintaining this history file, we are able to drastically improve run time compared to continuously reading in and processing files from previous timesteps.

After the necessary files have been loaded, the cells contained within each file are parsed and converted into dictionary structures. Cells that are not valid at the current timestep (old cells) are sorted into a different dictionary than the currently valid (new) cells, and the old cells are then matched into tracks using each object's track / ID / RowName number. The track number of each new cell is then compared to these tracks, and any new cells that match an existing track are automatically added to that track. The new cell is then removed from the list of cells needing to

be processed, and the assigned track number is retired such that no other new cells can be assigned to it for the current timestep. By the end of this step, only new cells that don't already match any existing track should remain.

Main Track Comparison

The main step of the BTRT algorithm compares each remaining new cell with every remaining track until a match is found. First, the time of the last object of a track is compared with the time of the new cell. If the last cell of the track is older than the user-specified buffer time, then the track is skipped. For example, if the current time is 1200 UTC and the buffer time is 5 minutes, tracks that ended before 1155 UTC will not be considered. Also, if the last object in the track somehow has the same time as the new cell, then the track is skipped. Otherwise, the new cell and all old cells within the track being compared are internally plotted using each object's centroid as a point location as shown in Fig. 1a. BTRT then calculates the Theil-Sen fit (Theil 1950, Sen 1968, Lakshmanan et al. 2015) for the old cells in order to derive a mean motion and speed for the physical phenomenon represented by the track (i.e. a storm cell; Fig. 1b). The algorithm then extrapolates the most recent old cell in the track forward in time along the Theil-Sen trajectory until the cell is temporally co-located with the new cell. If the centroid of the extrapolated cell falls within the buffer distance to the centroid of the new cell (Fig. 1c), then the track is designated as a potential match and the algorithm moves on to the next track. If a new cell is matched with multiple tracks, then the track with the smallest extrapolated distance to the new cell is considered the best match and the cell is added to that track (Fig. 1d). The new cell and matched track are then removed from any additional processing for this timestep.

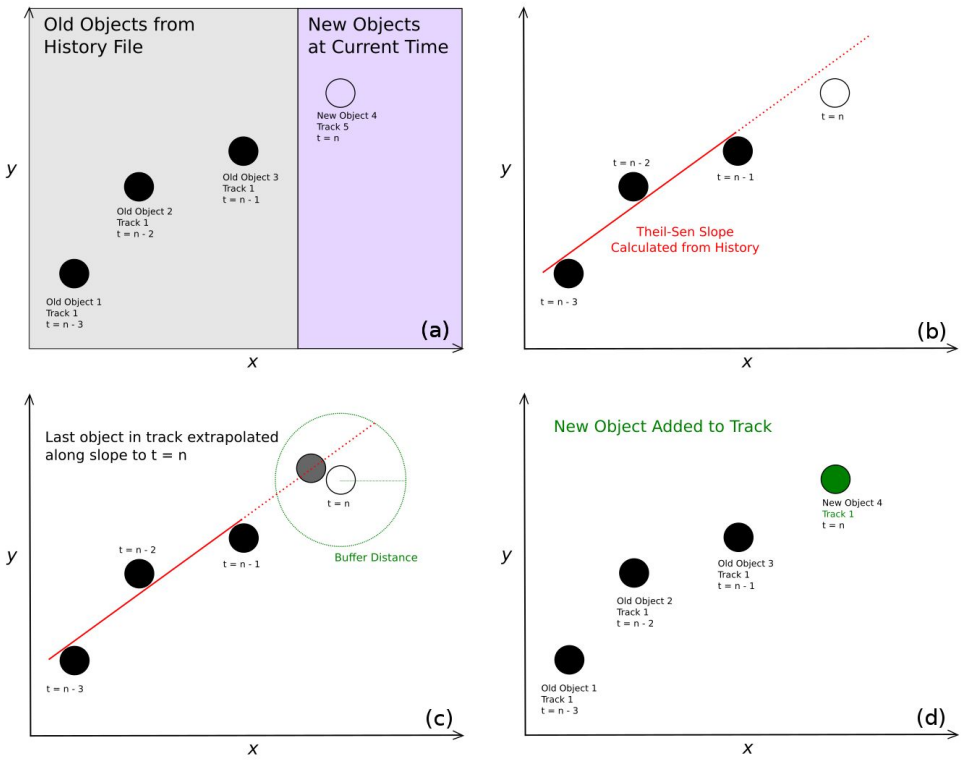

Figure 1: Four step track comparison process: (a) Cells are internally plotted as points. (b) The Theil-Sen trajectory is calculated using the old cells. (c) The last object is extrapolated and the distance to the new cell is calculated. (d) The new cell is added to the track if all conditions are met.

QLCS Algorithm

Any new cells that aren't matched with an existing track are then processed using our new QLCS algorithm. The QLCS algorithm was designed in an attempt to handle large, elongated objects that split or merge together. Because these operations typically result in a large spatial shift in an object's centroid, these types of track breakages may not be identified using the main track comparison method. We implement two slightly different algorithms for splits and merges:

Splits:

When a large object is split apart, it often results in two or more objects that have a different track number than the original (Fig. 2a). To solve this, we use a similar process to the main track comparison, where each remaining new object is compared to each potential QLCS track. Just like before, the Theil-Sen trajectory for the track is calculated and the most recent old object in that track is extrapolated forward to the current time. However, this time a 5 km buffer is added around the perimeter of the extrapolated object (Fig. 2b) in order to account for more varied motions and evolutions of QLCS storm systems. If the centroid of the new cell is contained within the buffered extrapolated object, then the new cell is added to that track and the cell and track are removed from further processing (Fig. 2c). If multiple objects are contained within the

extrapolated object, BTRT will attempt to prioritize the objects by severe probability. However, this does not always work as expected. Because of end user needs, only one object may be assigned to a specific track at a single timestep. Therefore, only one of the objects resulting from a split may be added back to the original track.

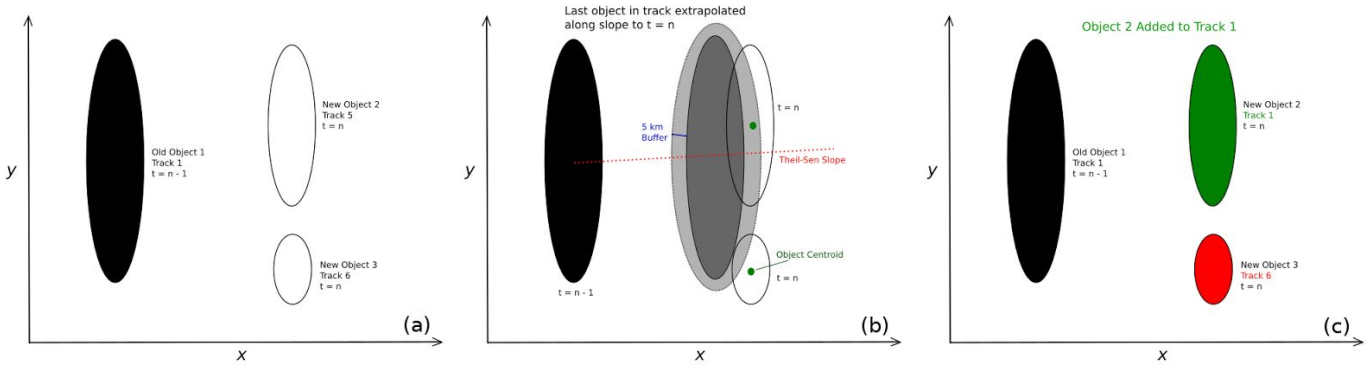

Figure 2: Three step QLCS split algorithm: (a) Cells are internally plotted as polygons. (b) The Theil-Sen trajectory is calculated and used to extrapolate the most recent old cell forward to the current timestep. A 5 km buffer is added around the object. (c) If an new cell's centroid is contained within the buffered, extrapolated object, it is added to that track.

Merges:

During a merge operation, two or more objects are combined into a single object which may have a different track number than any of the original cells (Fig. 3a). We solve this by essentially inverting the process used to handle splits. The Theil-Sen trajectory for each potential QLCS track is calculated and the last objects in each track are extrapolated forward to the current time. No buffer is applied this time. If the centroid of the extrapolated object is contained within the new cell (Fig. 3b), then the new cell is added to the track of that extrapolated object (Fig. 3c). As before, if multiple objects are contained within the new cell, BTRT will attempt to prioritize objects based on the probability of severe value. Because two or more tracks are being merged into one, only one of the original tracks will be preserved.

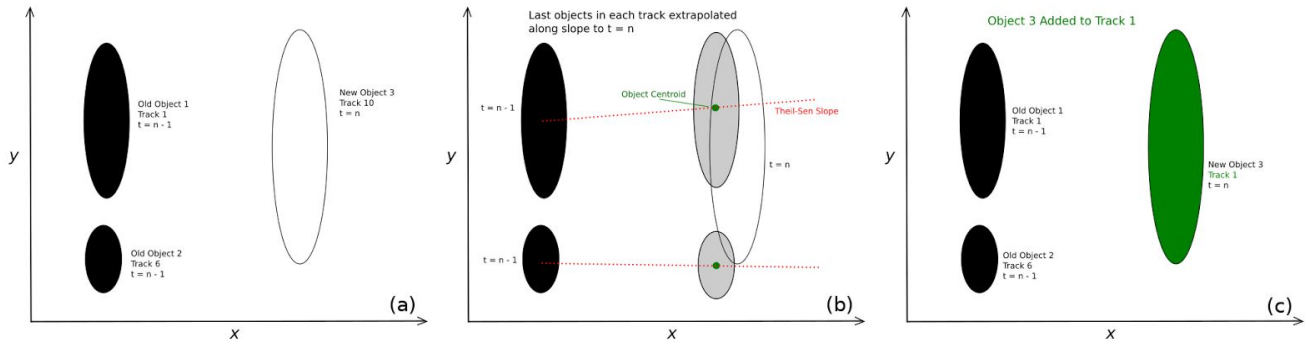

Figure 3: Three step QLCS merge algorithm: (a) Cells are internally plotted as polygons. (b) The Theil-Sen trajectory is calculated and used to extrapolate the most recent old cells forward to the current timestep. (c) If the extrapolated object's centroid is contained within the new cell, the new cell is added to that track.

Note: Due to data limitations, the QLCS algorithm is only performed on ProbSevere filetypes (ascii or json). This process is skipped for Segmotion input (xml).

Performance

BTRT has been optimized and is able to be run on personal computers and supercomputers. The exact run time will depend on the user settings and the data being processed, but for reference, a semi-operational version of BTRT implemented by PHI is able to process over 300 objects at a time in roughly 12 to 15 seconds. We have been able to post-process a full day of data in under 15 minutes using a standard computer, and a year of data in a few hours using a supercomputer.

If you find the memory or disk space usage is becoming excessive while running BTRT, try reducing the history time. If it doesn't appear that BTRT is modifying your data, make sure the buffer time greater than or equal to the timestep size of your data.

License:

References:

Lakshmanan, V., B. Herzog, and D. Kingfield, 2015: A Method for Extracting Postevent Storm Tracks. *J. Appl. Meteor. Climatol.,* **54**, 451 - 462, doi: 10.1175/JAMC-D-14-0132.1.

Sen, P., 1968: Estimates of the regression coefficient based on kendall's tau. *J. Amer. Stat. Assoc.*, **63**, 1379–1389.

Theil, H., 1950: A rank-invariant method of linear and polynomial regression analysis. i, ii, iii. *Nederl. Akad. Wetensch. Proc.*, **53**, 386–392,521–525,1397–1412.