8.1 AN OPTIMIZED IMPLEMENTATION OF OBJECTIVE ANALYSIS

Thomas A. Cannon*
J. Michael Wolfinbarger, Justin R. Greenfield, Thomas B. Stanley, and William G. McPherson, Jr.
Oklahoma Climatological Survey
University of Oklahoma
Norman, OK 73019

1. INTRODUCTION

Objective analysis is the process of interpolating data. In weather visualization software like WeatherScope (Wolfinbarger et al. 2001), a common use is to interpolate data from an irregular map of data gathering stations to a regular grid for the purpose of drawing products such as temperature gradients. Figure 1 shows the locations of Oklahoma Mesonet stations as large red dots, and an arbitrary grid of plotting points as small blue dots. The task for objective analysis is to determine the relative data value of each grid point by interpolating the values measured by the stations within the grid point's radius of influence.

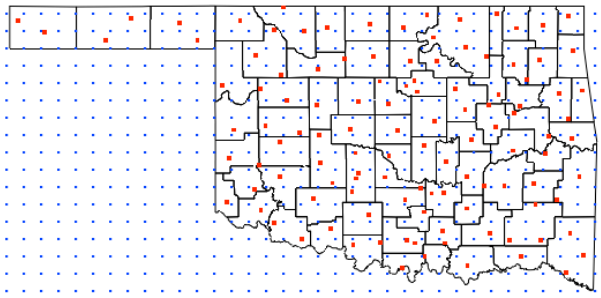

Figure 1

The nature of objective analysis suggests an obvious but inefficient way to implement it in a program. As a result, when plotting large datasets such as those from the National Weather Service, the user experience is detrimentally affected as the user waits for the objective analysis to complete. In truly interactive software such as WeatherScope this effect is problematic, as the objective analysis must be recalculated every time the user changes a variable or a new dataset is downloaded in response to the current time.

The purpose of this paper is to describe the steps taken to optimize WeatherScope's implementation of objective analysis by reducing the workload by two orders of magnitude, with a correlating reduction in time expended.

* *Corresponding author address*: Thomas A. Cannon, Oklahoma Climatological Survey, 100 E Boyd Street, Suite 1210, Norman, OK 73019-1012; e-mail: tom@mesonet.org

2. THE "BRUTE FORCE" METHOD

Given an arbitrary grid point and its radius of influence, an arbitrary station can be tested for inclusion by calculating the distance from the grid point to the station and comparing this distance to the radius of influence (see Figure 2).

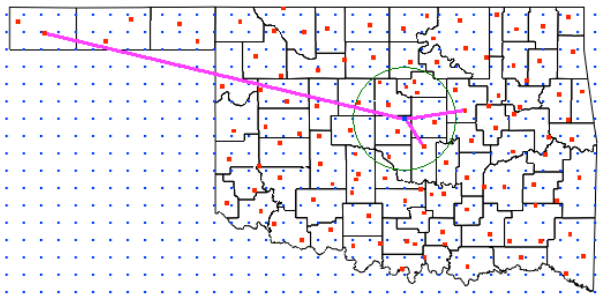

Figure 2

The natural solution is therefore to loop through the grid points and for each grid point loop through the stations to test them for inclusion:

```
for each Grid.Y
  for each Grid.X
    loop stations, testing distance
  next Grid.X
next Grid.Y
```

Unfortunately, this nested loop causes the workload to increase geometrically with the addition of a single station or grid point.

In Figure 2, the grid is 17 points high by 35 points wide, yielding a total of 595 grid points. If the Mesonet has 100 stations reporting data for a given time, the entire interpolation process will require 59,500 distance calculations, and the addition of a single station will cause that figure to rise to 60,095. On modern computers, a few hundred calculations more or less is an insignificant cost, so there is little incentive to improve the implementation.

This would be acceptable if all of the calculations were actually necessary. However, it should be obvious that data collected in New York City has little to do with the value of a grid point in Los Angeles. Figure 2 demonstrates this as well, in that a station in the panhandle should not affect a grid point in the center of Oklahoma. While this is obvious to the human observer looking at a map, it is not obvious to the computer, which is why the distance calculations are used. The inclusion test was discovered to succeed

less than 1% of the time, meaning that 99% of these calculations were wasted effort.

With modern personal computers, this level of performance is acceptable on state-level data, but national-level data is intolerably slow to analyze on all but the latest models using this method. The number of stations grows into the thousands, and the height and width of the grid into the hundreds of points. All of this is multiplied together, and the wasted workload becomes a serious problem.

The approach taken to solving this problem was to essentially draw a bounding square around the circle described by a grid point's radius of influence. The square was then used to cull the list of stations so that only the ones most likely to affect the grid point (those inside the square) were tested for inclusion in the radius of influence.

Given a circle of radius R, and bounding square whose sides are 2R long, the circle's area will be 78.5% of the square's area. Assuming random distribution of stations, it is highly unlikely that optimization using this bounding box approach will eliminate all 99% of the wasted calculations. However for this approach to be considered successful, the inclusion test should approach a 78.5% success rate.

3. THE BAND SORT

To begin the process of building a bounding box, the stations were sorted in order of their Y coordinates and stored in an ordered list. The program then loops through the grid points by their X and Y coordinates, simultaneously iterating through the station list to produce a subset of all the stations that is limited by vertical proximity to the current grid point:

```
sort stationlist
iHigh = stationlist.begin
iLow = stationlist.begin

for each Grid.Y
  yHigh = Grid.Y + radius
  yLow = Grid.Y - radius

  while iHigh < yHigh, iHigh++
  while iLow < yLow, iLow++

  for each Grid.X
    loop Station = from iLow to iHigh
      test Station
    end loop
  next Grid.X
next Grid.Y
```

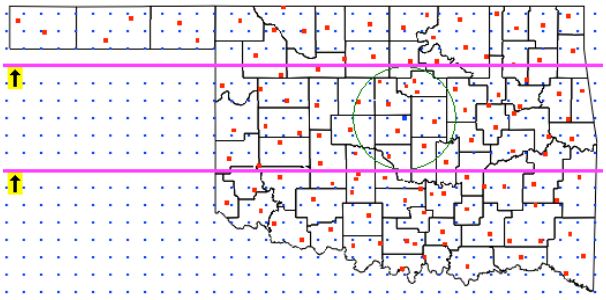

Figure 3

By eliminating a majority of the stations that would ordinarily be used in unnecessary distance calculations on a given grid point, this method produced workload savings of one order of magnitude. A small fraction of this savings is spent sorting the stations beforehand, but the use of an efficient sorting algorithm makes this cost negligible. The inclusion test was successful 7% of the time.

4. BAND SORT WITH LIMIT TEST

In an attempt to further reduce the number of stations being used in distance calculations with a given grid point, stations in the band were further tested according to their X coordinates. If the station's X coordinate fell outside the grid point's X coordinate plus or minus the radius of influence, the station was not included in distance calculations:

```
sort stationlist
iHigh = stationlist.begin
iLow = stationlist.begin

for each Grid.Y
  yHigh = Grid.Y + radius
  yLow = Grid.Y - radius

  while iHigh < yHigh, iHigh++
  while iLow < yLow, iLow++

  for each Grid.X
    xHigh = Grid.X + radius
    xLow = Grid.X - radius

    loop Station = from iLow to iHigh
      if xLow < Station.X < xHigh then
        test Station
    end loop
  next Grid.X
next Grid.Y
```

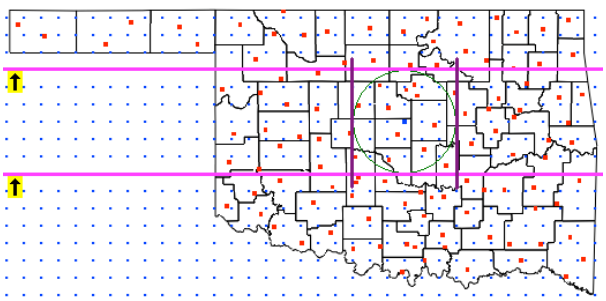

Figure 4

The inclusion test reached a 77% success rate at this point, in accordance with the expectations outlined above. While this method did further reduce the workload by another order of magnitude, time expenditure was not significantly reduced because distance calculations were replaced by comparisons. Comparing two floating-point numbers is only slightly faster than multiplying them, so timesavings are negligible.

5. OPTIMAL BAND WITH LIMIT TEST

To reduce the number of comparisons being made, the initial band sort was made in the longer of the X and Y dimensions of the map. Assuming a random distribution of stations, a map wider than it is tall should produce fewer stations per band if the stations are sorted by X-coordinates instead of Y-coordinates. This will in turn reduce the number of comparisons being done, and the time expended so doing. The code looks the same as in the Band Sort with Limit Test, except X and Y are swapped.

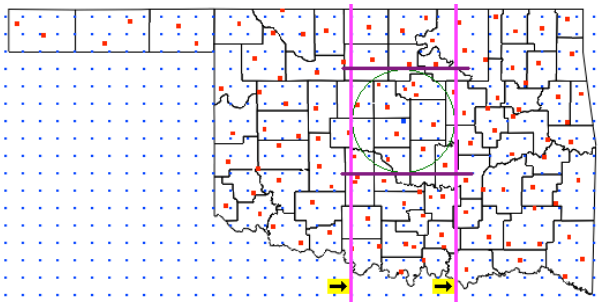

Figure 5

This change produced some savings in workload and time, though the effect is roughly proportional to the disparity between the map's two dimensions. Testing showed that this disparity is usually not great enough to make the savings exceptional.

6. DOUBLE-SORTED BAND

Finally, the stations were sorted twice. The first sort was in the more optimal direction as with the Optimal Band technique. Each band is then treated as a map unto itself, and sorted in its most optimal direction, which is the unsorted one:

```
sort stationlist on Station.X
iHigh = stationlist.begin
iLow = stationlist.begin

for each Grid.X
  xHigh = Grid.X + radius
  xLow = Grid.X - radius

  while iHigh < xHigh, iHigh++
  while iLow < xLow, iLow++

  Band = Set(iLow to iHigh)
  sort Band on Station.Y

  bHigh = Band.begin
  bLow = Band.end

  for each Grid.Y
    yHigh = Grid.Y + radius
    yLow = Grid.Y - radius

    while bHigh < yHigh, bHigh++
    while bLow < yLow, bLow++

    loop Station = from bLow to bHigh
        test Station
    end loop
  next Grid.Y
next Grid.X
```

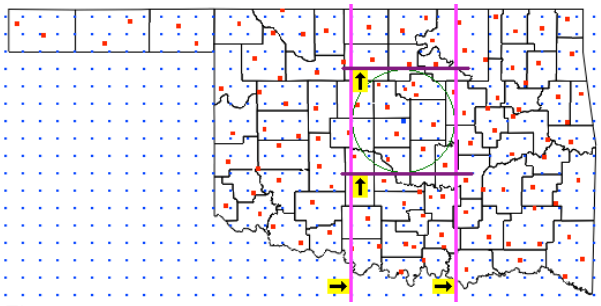

Figure 6

By iterating through the bands as with the Band Sort, then iterating within each band along the other axis, the number of comparisons was minimized along with the number of distance calculations, yielding a second order of magnitude in timesavings.

8. CONCLUSION

The elimination of unnecessary work was the primary force behind the significant timesavings in performing objective analysis. As the workload became more efficient, time spent doing the work dropped dramatically.

The implications of this improvement in objective analysis implementation are significant to the development of interactive software such as WeatherScope. It was previously unthinkable to do multiple pass analysis on national datasets, or to load multiple gridded datasets for simultaneous viewing, since even a single pass on a single dataset was detrimental to the user experience in interactive software. With this optimization, even users of relatively slow or outdated computers can easily make use of national data.

9. ACKNOWLEDGEMENTS

WeatherScope is a registered trademark of the Oklahoma Climatological Survey.

downloaded via the Internet at http://sdg.ocs.ou.edu. The development and deployment of WeatherScope was funded by several projects within OCS, including the Mesonet, OK-FIRST, and ARM Educational Outreach projects.

The optimization technique described in this paper may be used in any software that may benefit from it, so long as proper credit is given to the Oklahoma Climatological Survey for having developed it.

Renee McPherson and Ken Crawford provided much support, assistance, and encouragement. At OCS, they foster an environment where creativity can be exercised and new ideas explored.

10. REFERENCES

Wolfinbarger, J. M., J. R. Greenfield, T. B. Stanley, and R. A. Young, 2001: WeatherScope: Interactive Software for Visualizing Web-Based Meteorological Data Sets. 17th International Conference on IIPS, Amer. Meteor. Soc.

APPENDIX A

Table 1 is a sample of the results from one of a series of tests performed at various stages of optimization. The tests were performed using air temperature observations from the ASOS network, selected from random dates and times. Various computers were tested, and while workload reductions were consistent, time saved for each computer was necessarily dependent on processor speed and available resources.

In this instance, the computer being tested was a Dell Inspiron 8100 with a Pentium III processor and 512MB of RAM. The network reported data from 1,411 stations, and the grid was 158 points high by 148 wide.

Method	Seconds	Distance Calculations	Inclusion Hits	Inclusion Rate	Calculations per Point	Tests	Tests per Point
Brute Force	17.571	32,994,824	275,949	0.8%	1411.0	0	0.0
Band Sort	1.184	3,531,932	275,949	7.8%	151.0	0	0.0
Limit Test	1.057	355,348	275,949	77.6%	15.2	3,531,932	151.0
Optimal Band	1.045	355,348	275,949	77.6%	15.2	3,319,048	141.9
Double Sort	0.381	355,348	275,949	77.6%	15.2	366,151	15.7

Table 1