Finding Neighbors in Time and Space: An Illustrative Application of Partial Ranking Techniques

Many types of data have both spatial and temporal aspects. Typically, observations close together in both time and space show more similarities than randomly selected data. In other words, such data exhibit spatial-temporal dependence. To model such dependence, one might wish to find all the neighbors close together in space which occurred previously in time. Essentially, Pace *et al*. followed this approach in modeling real estate prices. In real estate, knowing the prices of houses which sold nearby in the past provides crucial information for the prediction of a house's price.

Finding such neighbors can prove difficult. Some of the techniques used in space, such as the use of Delaunay triangles, do not generalize immediately to this context. The use of a three dimensional Delaunay triangle program would ignore the unidirectional character of time. Use of incremental Delaunay triangles can give the very nearest previously sold neighbors, but finding other more distant previously sold neighbors can prove difficult. For example, suppose the previously sold neighbors for a particular observation all sold many years ago. Going to the neighbors of these old observations would only give the previously sold neighbors to these old observations. Hence, using this approach does not lead to recent nearby observations which are not immediate neighbors.

Other more advanced computational geometry approaches could work, but these require careful, non-trivial programming. The use of partial ranking techniques, such as implemented in Orderpack (www.fortran-2000.com), provides an easy way of finding such neighbors using simple programs such as attached to the appendix. To give an idea of the efficacy of these programs, we contrast partial versus unconditional ranking for this problem. The unconditional ranking approach computes the relative rank of each observation which previously occurred while the partial ranking approach computes only the desired number of closest ranked observations.

In Table 1, each approach ignores the first 1,000 observations and finds the nearest, previously occurring observations to each observation from observations 1,000 to the *n*th observation. As one can see, the unconditional ranking times quickly become large, taking around three minutes for only 7,000 observations. In contrast, it takes less than 1.5 minutes to find the 11 closest previously occurring neighbors for 50,000 observations when using the partial ranking approach. The partial ranking approach can handle 100,000 observations in under 10 minutes on a PC (600 Mhz Pentium

III with 1 gigabyte of RAM on NT 4.0 using CVF 6.5). Even if the times grow with the cube of the number of observations, it should be possible to handle 400,000 observations using an overnight run.

In reality, one may wish to limit the extent in time one uses to find neighbors. If the time dimension is augmented which keeping the spatial aspects the same (such as for a single area), one could use a constant window over time and the problem would become approximately linear in the number of observations.

In conclusion, partial ranking and sorting routines can provide straightforward, computationally efficient ways of working with complicated or multidimensional data.

Table 1 — Unconditional versus Partial Ranking Times in Seconds for Finding Spatial-Temporal Nearest Neighbors (ISTART=1,000 and *m*=11)

n	Unconditional Ranking Times	Partial Ranking Times	Ratio of Unconditional to Partial Times
3,000	9.48	0.14	67.44
5,000	67.94	0.42	161.03
7,000	179.53	0.92	194.75
15,000	N/A	4.13	N/A
50,000	N/A	73.14	N/A
100,000	N/A	448.69	N/A

Using the Executable File

One needs to create two auxiliary files for the executable file to process. The first is neighbor_parameters.txt and this should contain two scalars. The first is the number of neighbors and the second is the first observation the program should use to find the neighbors. This should be greater than the number of neighbors at the very least. The second file is coordinates_in.txt and this contains the x_coordinate side-by-side to the y_coordinate (e.g., longitude and latitude). This should be sorted by time with the first row representing the oldest observation and so forth. The program creates the file nnmat.txt which contains the indices of the nearest neighbors subject to the temporal ordering. A negative 1 in nnmat.txt indicates that it is before the first designated observation.

References

Pace, R. Kelley and Ronald Barry, O.W. Gilley, C.F. Sirmans, "A Method for Spatial-temporal Forecasting with an Application to Real Estate Prices," *International Journal of Forecasting*, Volume 16, Number 2, April-June 2000, p. 229-246.

If you use the program, this would be the appropriate article to cite.

```
!  spatiotemporal neighbors
!
!***************************************************************************
!
!  PROGRAM: spatiotemporal neighbors
!
!  PURPOSE: given n points ordered by increasing date, find the m nearest
!                          neighbors to each observation from the past observations
!
!***************************************************************************

        program spatiotemporalnn

        implicit none

        integer n,i,m_less1,m,istart,jj
        integer, allocatable::irngt(:),nnmat(:,:)
        real(kind=4), allocatable::xcoord(:),ycoord(:),d2(:),xyc2(:)
        real tbeg,tend,xcoordi,ycoordi,xcoord_sum,ycoord_sum


! ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
! Inputting m, the number of neighbors, and istart, the observation to use as
! beginning observation.

OPEN(UNIT=600,FILE='neighbor_parameters.txt')
READ(600,*) m_less1,istart
CLOSE(600)

m=m_less1+1

! ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        !n is the number of observations, m is the number of neighbors needed
        !istart is the earliest observation for which neighbors are desired
        !xcoord, ycoord represent the locational coordinates of an observation
        !and these are ordered by time. The first row has the oldest observation (earliest date) and
the
        !last observation has the newest observation (latest date). We simulate the coordinates
here to demonstrate
        !the computations. For actual data, care should be taken to scale the coordinates to
        !avoid underflow or overflow. If one scales the coordinates, one should do so by the
same factor
        !for both xcoord, ycoord.
```

```
        !@@@@@@@@@@@@@@@@@@@@@@@ FIRST PASS THROUGH DATA
@@@@@@@@@@@@@@@@@@@@@@@
!This part of the program finds out how many observations and unique
!areas there are in the data as well as collecting input statistics
!which could aid in verifying input data



OPEN(UNIT=500,FILE='coordinates_in.txt')
n=0
do while (.true.)
n=n+1
read(500,*,end=555) xcoordi,ycoordi


xcoord_sum=xcoord_sum+xcoordi
ycoord_sum=ycoord_sum+ycoordi


end do
555 print *,'Successful First Pass through the input file coordinates_in.txt'
n=n-1
CLOSE(500)

        !@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

        print *,' '
        print *,'n (total number of records),m (number of nearest neighbors),'
        print *,' '
        print *, n,m_less1
        print *,' '
        print *,'To check on validity of input data, the following contains the mean of:'
        print *,'the geographic id, the repeat sales id, x coordinates, y coordinates'
        print *,'and the x and y locational coordinates'
        print *,' '
        print *, (xcoord_sum/dfloat(n)),(ycoord_sum/dfloat(n))
        print *,' '


!$$$$$$$$$$$$$$$$$$$$$ ACTUAL DATA INPUT $$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!This part actually inputs the data and constructs the indices associated
```

```
!with each area


          allocate(xcoord(n),ycoord(n),d2(n),nnmat(m,n),xyc2(n),irngt(n))



OPEN(UNIT=500,FILE='coordinates_in.txt')
do i=1,n
READ(500,*) xcoord(i),ycoord(i)
end do
CLOSE(500)


!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

          call cpu_time(tbeg)

          call tnn(xcoord,ycoord,m,istart,nnmat)

          call cpu_time(tend)


          print *, 'time secs',tend-tbeg
          print *,' '
          print *,'n,m,istart ',n,m_less1,istart
          print *,' '

          print *,'nnmat(:,n) ',nnmat(:,n)
          print *,' '
          print *,'nnmat(:,istart)',nnmat(:,istart)



          print *,' '

!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!writes results to the files

OPEN(UNIT=666,FILE='nnmat.txt',buffered='yes',buffercount=127,blocksize=65536)
do i=1,n
WRITE(666,100) (nnmat(jj,i),jj=1,m)
end do
```

```
100 format(<m>I9)

!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        print *,'finished '


        contains

        Subroutine tnn(xcoord,ycoord,m,istart,nnmat)
        real(kind=4), intent(in):: xcoord(:),ycoord(:)
        real(kind=4) xyc2(size(xcoord,1)),d2(size(xcoord,1))
        integer, intent(out)::nnmat(:,:)
        integer, intent(in)::m,istart
        integer irngt(size(xcoord,1))
        integer i,j

        nnmat=-1 !if a -1 shows up, something is wrong or it is before istart

!       We precompute this for use below
        xyc2=0.5*(xcoord*xcoord+ycoord*ycoord)


        do i=istart,n


!       The interest centers in the correct ordinal distance or correct ranks of some function
!       of distance to the observation i. Clearly, squaring the distances preserves the ranks.
!       The equation below gives squared spatial distance of observations in the past to the
observation.
!       Note, the omission of the terms xcoord(i)*xcoord(i) and ycoord(i)*ycoord(i)
!       do not affect the ranks of the other distances relative to observation i as these are scalars
!       which are the same for all observations. One can test this by uncommenting the
alternative
!       statement of squared-distance. The use of the straightfoward squared-distance costs
!       2 substractions, 2 multiplications, and one addition. The one used here costs 2
subtractions and
!       2 multiplications and so saves a little time.

!       d2(1:i)=(xcoord(i)-xcoord(1:i))**2+(ycoord(i)-ycoord(1:i))**2

        d2(1:i)=xyc2(1:i)-xcoord(i)*xcoord(1:i)-ycoord(i)*ycoord(1:i)
```

```
!This call shows the effect on the times of sorting all i distances (an unconditional ranking)
!         call RNKPAR (d2(1:i), IRNGT, i)


!This call shows the effect of just performing the partial rankings of distances (just need m).
          call RNKPAR (d2(1:i), IRNGT, m)

!Stores the m ranks in the nearest neighbor matrix
          nnmat(:,i)=irngt(1:m)


          end do

          end Subroutine tnn




Subroutine RNKPAR (XDONT, IRNGT, NORD)
!  Ranks partially XDONT by IRNGT, up to order NORD
! __________________________________________________________
!  This routine uses a pivoting strategy such as the one of
!  finding the median based on the quicksort algorithm, but
!  we skew the pivot choice to try to bring it to NORD as
!  fast as possible. It uses 2 temporary arrays, where it
!  stores the indices of the values smaller than the pivot
!  (ILOWT), and the indices of values larger than the pivot
!  that we might still need later on (IHIGT). It iterates
!  until it can bring the number of values in ILOWT to
!  exactly NORD, and then uses an insertion sort to rank
!  this set, since it is supposedly small.
!  Michel Olagnon - Feb. 2000
! __________________________________________________________
      Real, Dimension (:), Intent (In) :: XDONT
      Integer, Dimension (:), Intent (Out) :: IRNGT
      Integer, Intent (In) :: NORD
! __________________________________________________________
      Integer, Dimension (SIZE(XDONT)) :: ILOWT, IHIGT
      Integer :: NDON, JHIG, JLOW, IHIG, IWRK, IWRK1, IWRK2, IWRK3
      Integer :: IDEB, JDEB, IMIL, IFIN, NWRK, ICRS, IDCR
      Real (Kind(XDONT)) :: XPIV, XPIV0, XWRK, XWRK1, XMIN
```

```
!
    NDON = SIZE (XDONT)
!
!   First loop is used to fill-in ILOWT, IHIGT at the same time
!
    If (NDON < 2) Then
      If (NORD >= 1) IRNGT (1) = 1
      Return
    End If
!
!  One chooses a pivot, best estimate possible to put fractile near
!  mid-point of the set of low values.
!
    If (XDONT(2) < XDONT(1)) Then
      ILOWT (1) = 2
      IHIGT (1) = 1
    Else
      ILOWT (1) = 1
      IHIGT (1) = 2
    End If
!
    If (NDON < 3) Then
      If (NORD >= 1) IRNGT (1) = ILOWT (1)
      If (NORD >= 2) IRNGT (2) = IHIGT (1)
      Return
    End If
!
    If (XDONT(3) < XDONT(IHIGT(1))) Then
      IHIGT (2) = IHIGT (1)
      If (XDONT(3) < XDONT(ILOWT(1))) Then
        IHIGT (1) = ILOWT (1)
        ILOWT (1) = 3
      Else
        IHIGT (1) = 3
      End If
    Else
      IHIGT (2) = 3
    End If
!
    If (NDON < 4) Then
      If (NORD >= 1) IRNGT (1) = ILOWT (1)
      If (NORD >= 2) IRNGT (2) = IHIGT (1)
      If (NORD >= 3) IRNGT (3) = IHIGT (2)
      Return
    End If
```

```
!
    If (XDONT(NDON) < XDONT(IHIGT(1))) Then
      IHIGT (3) = IHIGT (2)
      IHIGT (2) = IHIGT (1)
      If (XDONT(NDON) < XDONT(ILOWT(1))) Then
        IHIGT (1) = ILOWT (1)
        ILOWT (1) = NDON
      Else
        IHIGT (1) = NDON
      End If
    Else
      IHIGT (3) = NDON
    End If
!
    If (NDON < 5) Then
      If (NORD >= 1) IRNGT (1) = ILOWT (1)
      If (NORD >= 2) IRNGT (2) = IHIGT (1)
      If (NORD >= 3) IRNGT (3) = IHIGT (2)
      If (NORD >= 4) IRNGT (4) = IHIGT (3)
      Return
    End If
!
    JDEB = 0
    IDEB = JDEB + 1
    JLOW = IDEB
    JHIG = 3
    XPIV = XDONT (ILOWT(IDEB)) + REAL(2*NORD)/REAL(NDON+NORD) * &
                          (XDONT(IHIGT(3))-XDONT(ILOWT(IDEB)))
    If (XPIV >= XDONT(IHIGT(1))) Then
      XPIV = XDONT (ILOWT(IDEB)) + REAL(2*NORD)/REAL(NDON+NORD) * &
                          (XDONT(IHIGT(2))-XDONT(ILOWT(IDEB)))
      If (XPIV >= XDONT(IHIGT(1))) &
         XPIV = XDONT (ILOWT(IDEB)) + REAL (2*NORD) / REAL (NDON+NORD) * &
                          (XDONT(IHIGT(1))-XDONT(ILOWT(IDEB)))
    End If
    XPIV0 = XPIV
!
!  One puts values > pivot in the end and those <= pivot
!  at the beginning. This is split in 2 cases, so that
!  we can skip the loop test a number of times.
!  As we are also filling in the work arrays at the same time
!  we stop filling in the IHIGT array as soon as we have more
!  than enough values in ILOWT.
!
!
```

```
    If (XDONT(NDON) > XPIV) Then
      ICRS = 3
      Do
        ICRS = ICRS + 1
        If (XDONT(ICRS) > XPIV) Then
          If (ICRS >= NDON) Exit
          JHIG = JHIG + 1
          IHIGT (JHIG) = ICRS
        Else
          JLOW = JLOW + 1
          ILOWT (JLOW) = ICRS
          If (JLOW >= NORD) Exit
        End If
      End Do
!
!  One restricts further processing because it is no use
!  to store more high values
!
      If (ICRS < NDON-1) Then
        Do
          ICRS = ICRS + 1
          If (XDONT(ICRS) <= XPIV) Then
            JLOW = JLOW + 1
            ILOWT (JLOW) = ICRS
          Else If (ICRS >= NDON) Then
            Exit
          End If
        End Do
      End If
!
!
    Else
!
!  Same as above, but this is not as easy to optimize, so the
!  DO-loop is kept
!
      Do ICRS = 4, NDON - 1
        If (XDONT(ICRS) > XPIV) Then
          JHIG = JHIG + 1
          IHIGT (JHIG) = ICRS
        Else
          JLOW = JLOW + 1
          ILOWT (JLOW) = ICRS
          If (JLOW >= NORD) Exit
        End If
```

```
      End Do
!
      If (ICRS < NDON-1) Then
        Do
          ICRS = ICRS + 1
          If (XDONT(ICRS) <= XPIV) Then
            If (ICRS >= NDON) Exit
            JLOW = JLOW + 1
            ILOWT (JLOW) = ICRS
          End If
        End Do
      End If
    End If
!
    Do
!
!   We try to bring the number of values in the low values set
!   closer to NORD.
!
      Select Case (NORD-JLOW)
      Case (2:)
!
!   Not enough values in low part, at least 2 are missing
!
        Select Case (JHIG)
!!!!!          CASE DEFAULT
!!!!!            write (*,*) "Assertion failed"
!!!!!            STOP
!
!   We make a special case when we have so few values in
!   the high values set that it is bad performance to choose a pivot
!   and apply the general algorithm.
!
        Case (2)
          If (XDONT(IHIGT(1)) <= XDONT(IHIGT(2))) Then
            JLOW = JLOW + 1
            ILOWT (JLOW) = IHIGT (1)
            JLOW = JLOW + 1
            ILOWT (JLOW) = IHIGT (2)
          Else
            JLOW = JLOW + 1
            ILOWT (JLOW) = IHIGT (2)
            JLOW = JLOW + 1
            ILOWT (JLOW) = IHIGT (1)
          End If
```

```
            Exit
!
          Case (3)
!
!
            IWRK1 = IHIGT (1)
            IWRK2 = IHIGT (2)
            IWRK3 = IHIGT (3)
            If (XDONT(IWRK2) < XDONT(IWRK1)) Then
              IHIGT (1) = IWRK2
              IHIGT (2) = IWRK1
              IWRK2 = IWRK1
            End If
            If (XDONT(IWRK2) > XDONT(IWRK3)) Then
              IHIGT (3) = IWRK2
              IHIGT (2) = IWRK3
              IWRK2 = IWRK3
              If (XDONT(IWRK2) < XDONT(IHIGT(1))) Then
                IHIGT (2) = IHIGT (1)
                IHIGT (1) = IWRK2
              End If
            End If
            JHIG = 0
            Do ICRS = JLOW + 1, NORD
              JHIG = JHIG + 1
              ILOWT (ICRS) = IHIGT (JHIG)
            End Do
            JLOW = NORD
            Exit
!
          Case (4:)
!
!
            XPIV0 = XPIV
            IFIN = JHIG
!
!  One chooses a pivot from the 2 first values and the last one.
!  This should ensure sufficient renewal between iterations to
!  avoid worst case behavior effects.
!
            IWRK1 = IHIGT (1)
            IWRK2 = IHIGT (2)
            IWRK3 = IHIGT (IFIN)
            If (XDONT(IWRK2) < XDONT(IWRK1)) Then
              IHIGT (1) = IWRK2
```

```
        IHIGT (2) = IWRK1
        IWRK2 = IWRK1
      End If
      If (XDONT(IWRK2) > XDONT(IWRK3)) Then
        IHIGT (IFIN) = IWRK2
        IHIGT (2) = IWRK3
        IWRK2 = IWRK3
        If (XDONT(IWRK2) < XDONT(IHIGT(1))) Then
          IHIGT (2) = IHIGT (1)
          IHIGT (1) = IWRK2
        End If
      End If
!
      JDEB = JLOW
      NWRK = NORD - JLOW
      IWRK1 = IHIGT (1)
      JLOW = JLOW + 1
      ILOWT (JLOW) = IWRK1
      XPIV = XDONT (IWRK1) + REAL (NWRK) / REAL (NORD+NWRK) * &
                    (XDONT(IHIGT(IFIN))-XDONT(IWRK1))
!
!  One takes values <= pivot to ILOWT
!  Again, 2 parts, one where we take care of the remaining
!  high values because we might still need them, and the
!  other when we know that we will have more than enough
!  low values in the end.
!
      JHIG = 0
      Do ICRS = 2, IFIN
        If (XDONT(IHIGT(ICRS)) <= XPIV) Then
          JLOW = JLOW + 1
          ILOWT (JLOW) = IHIGT (ICRS)
          If (JLOW >= NORD) Exit
        Else
          JHIG = JHIG + 1
          IHIGT (JHIG) = IHIGT (ICRS)
        End If
      End Do
!
      Do ICRS = ICRS + 1, IFIN
        If (XDONT(IHIGT(ICRS)) <= XPIV) Then
          JLOW = JLOW + 1
          ILOWT (JLOW) = IHIGT (ICRS)
        End If
      End Do
```

```
         End Select
!
!
      Case (1)
!
!  Only 1 value is missing in low part
!
         XMIN = XDONT (IHIGT(1))
         IHIG = 1
         Do ICRS = 2, JHIG
           If (XDONT(IHIGT(ICRS)) < XMIN) Then
             XMIN = XDONT (IHIGT(ICRS))
             IHIG = ICRS
           End If
         End Do
!
         JLOW = JLOW + 1
         ILOWT (JLOW) = IHIGT (IHIG)
         Exit
!
!
      Case (0)
!
!  Low part is exactly what we want
!
         Exit
!
!
      Case (-5:-1)
!
!  Only few values too many in low part
!
         IRNGT (1) = ILOWT (1)
         Do ICRS = 2, NORD
           IWRK = ILOWT (ICRS)
           XWRK = XDONT (IWRK)
           Do IDCR = ICRS - 1, 1, - 1
             If (XWRK < XDONT(IRNGT(IDCR))) Then
               IRNGT (IDCR+1) = IRNGT (IDCR)
             Else
               Exit
             End If
           End Do
           IRNGT (IDCR+1) = IWRK
         End Do
```

```
!
          XWRK1 = XDONT (IRNGT(NORD))
          Do ICRS = NORD + 1, JLOW
            If (XDONT(ILOWT (ICRS)) < XWRK1) Then
              XWRK = XDONT (ILOWT (ICRS))
              Do IDCR = NORD - 1, 1, - 1
                If (XWRK >= XDONT(IRNGT(IDCR))) Exit
                IRNGT (IDCR+1) = IRNGT (IDCR)
              End Do
              IRNGT (IDCR+1) = ILOWT (ICRS)
              XWRK1 = XDONT (IRNGT(NORD))
            End If
          End Do
!
          Return
!
!
        Case (:-6)
!
! last case: too many values in low part
!
          IDEB = JDEB + 1
          IMIL = (JLOW+IDEB) / 2
          IFIN = JLOW
!
!  One chooses a pivot from 1st, last, and middle values
!
          If (XDONT(ILOWT(IMIL)) < XDONT(ILOWT(IDEB))) Then
            IWRK = ILOWT (IDEB)
            ILOWT (IDEB) = ILOWT (IMIL)
            ILOWT (IMIL) = IWRK
          End If
          If (XDONT(ILOWT(IMIL)) > XDONT(ILOWT(IFIN))) Then
            IWRK = ILOWT (IFIN)
            ILOWT (IFIN) = ILOWT (IMIL)
            ILOWT (IMIL) = IWRK
            If (XDONT(ILOWT(IMIL)) < XDONT(ILOWT(IDEB))) Then
              IWRK = ILOWT (IDEB)
              ILOWT (IDEB) = ILOWT (IMIL)
              ILOWT (IMIL) = IWRK
            End If
          End If
          If (IFIN <= 3) Exit
!
          XPIV = XDONT (ILOWT(1)) + REAL(NORD)/REAL(JLOW+NORD) * &
```

```
                        (XDONT(ILOWT(IFIN))-XDONT(ILOWT(1)))
        If (JDEB > 0) Then
          If (XPIV <= XPIV0) &
             XPIV = XPIV0 + REAL(2*NORD-JDEB)/REAL (JLOW+NORD) * &
                      (XDONT(ILOWT(IFIN))-XPIV0)
        Else
          IDEB = 2
        End If
!
!  One takes values > XPIV to IHIGT
!  However, we do not process the first values if we have been
!  through the case when we did not have enough low values
!
        JHIG = 1
        IHIGT (JHIG) = ILOWT (IFIN)
        JLOW = JDEB
!
        If (XDONT(ILOWT(IFIN)) > XPIV) Then
          ICRS = JDEB
          Do
            ICRS = ICRS + 1
            If (XDONT(ILOWT(ICRS)) > XPIV) Then
              If (ICRS >= IFIN) Exit
              JHIG = JHIG + 1
              IHIGT (JHIG) = ILOWT (ICRS)
            Else
              JLOW = JLOW + 1
              ILOWT (JLOW) = ILOWT (ICRS)
              If (JLOW >= NORD) Exit
            End If
          End Do
!
          If (ICRS < IFIN-1) Then
            Do
              ICRS = ICRS + 1
              If (XDONT(ILOWT(ICRS)) <= XPIV) Then
                JLOW = JLOW + 1
                ILOWT (JLOW) = ILOWT (ICRS)
              Else
                If (ICRS >= IFIN) Exit
              End If
            End Do
          End If
        Else
          Do ICRS = IDEB, IFIN - 1
```

```
          If (XDONT(ILOWT(ICRS)) > XPIV) Then
            JHIG = JHIG + 1
            IHIGT (JHIG) = ILOWT (ICRS)
          Else
            JLOW = JLOW + 1
            ILOWT (JLOW) = ILOWT (ICRS)
            If (JLOW >= NORD) Exit
          End If
        End Do
!
        Do ICRS = ICRS + 1, IFIN - 1
          If (XDONT(ILOWT(ICRS)) <= XPIV) Then
            JLOW = JLOW + 1
            ILOWT (JLOW) = ILOWT (ICRS)
          End If
        End Do
      End If
!
    End Select
!
  End Do
!
!  Now, we only need to complete ranking of the 1:NORD set
!  Assuming NORD is small, we use a simple insertion sort
!
  IRNGT (1) = ILOWT (1)
  Do ICRS = 2, NORD
    IWRK = ILOWT (ICRS)
    XWRK = XDONT (IWRK)
    Do IDCR = ICRS - 1, 1, - 1
      If (XWRK < XDONT(IRNGT(IDCR))) Then
        IRNGT (IDCR+1) = IRNGT (IDCR)
      Else
        Exit
      End If
    End Do
    IRNGT (IDCR+1) = IWRK
  End Do
  Return
!
End Subroutine RNKPAR


    end program spatiotemporalnn
```