CASE	CLUSTER	CLUSTER NAME	OUTLIER	PERIM PATH	CLLI	LONG	LAT	WT	CBG	TYPE	GRPPATH	GRPPATHLAB
1	2	c001	1	1	HYDNCOMA	-107.302076	40.280345	1.000000000	08107954600373	S	5	c001.001001002001
2	2	c001	1	2	HYDNCOMA	-107.299135	40.288716	1.000000000	08107954600373	S	5	c001.001001002001
3	4	c001	1	1	HYDNCOMA	-107.263694	40.297016	1.000000000	08107954600364	S	6	c001.001001002001001
4	3	c001	1	1	HYDNCOMA	-107.416888	40.297466	1.000000000	08107954600381	S	7	c001.001001002002001002
5	5	c001	1	4	HYDNCOMA	-107.300597	40.302741	1.000000000	08107954600382	S	4	c001.001001002
6	5	c001	1	3	HYDNCOMA	-107.297920	40.307827	1.000000000	08107954600386	S	4	c001.001001002
7	5	c001	1	2	HYDNCOMA	-107.306851	40.311325	1.000000000	08107954600373	S	4	c001.001001002
8	5	c001	1	1	HYDNCOMA	-107.316675	40.314522	1.000000000	08107954600373	S	4	c001.001001002
9	7	c001	1	2	HYDNCOMA	-107.348681	40.318126	1.000000000	08107954600373	S	5	c001.001001002002
10	6	c001	1	1	HYDNCOMA	-107.412042	40.318713	1.000000000	08107954600382	S	6	c001.001001002002001
11	6	c001	1	2	HYDNCOMA	-107.411111	40.320061	1.000000000	08107954600376	S	6	c001.001001002002001
12	8	c001	1	1	HYDNCOMA	-107.437231	40.321414	1.000000000	08107954600379	S	7	c001.001001002002001001
13	7	c001	1	1	HYDNCOMA	-107.359833	40.321998	1.000000000	08107954600373	S	5	c001.001001002002
14	9	c001	1	3	HYDNCOMA	-107.421569	40.339277	1.000000000	08107954600376	S	7	c001.001001002002001003
15	9	c001	1	2	HYDNCOMA	-107.418231	40.341659	1.000000000	08107954600333	S	7	c001.001001002002001003
16	9	c001	1	1	HYDNCOMA	-107.422765	40.344995	1.000000000	08107954600376	S	7	c001.001001002002001003
17	10	c001	1	1	HYDNCOMA	-107.312148	40.353151	1.000000000	08107954600367	R	3	c001.001001
18	11	c001	1	1	HYDNCOMA	-107.259345	40.354205	1.000000000	08107954600364	S	2	c001.001
19	12	c001	1	1	HYDNCOMA	-107.332045	40.354644	1.000000000	08107954600333	S	4	c001.001001001
20	10	c001	1	2	HYDNCOMA	-107.304082	40.360108	1.000000000	08107954600367	R	3	c001.001001
21	1	c001	0	1	HYDNCOMA	-107.284715	40.375536	1.000000000	08107954600336	R	1	c001.
22	1	c001	0	0	HYDNCOMA	-107.284426	40.375933	1.000000000	08107954600336	R	1	c001.
23	1	c001	0	0	HYDNCOMA	-107.284129	40.376240	1.000000000	08107954600336	R	1	c001.
24	1	c001	0	0	HYDNCOMA	-107.283429	40.376487	1.000000000	08107954600367	R	1	c001.
25	1	c001	0	3	HYDNCOMA	-107.267750	40.387417	1.000000000	08107954600365	S	1	c001.
26	1	c001	0	2	HYDNCOMA	-107.272922	40.392565	1.000000000	08107954600337	S	1	c001.
27	13	c001	1	1	HYDNCOMA	-107.300834	40.412719	1.000000000	08107954600335	S	2	c001.002
28	14	c001	1	1	HYDNCOMA	-107.316300	40.426321	1.000000000	08107954600333	S	3	c001.002001

BlockGroup	Cluster	ClusterName	Outlier	SplitGroup	Area	ClusLong	ClusLat	ServLong	ServLat	Height	Width
81079546003	1	c001	0	HYDNCOMA	3.565559	-107.331	40.30117	-107.331	40.30117	2.875941	3.261201
81079546003	2	c002	0	HYDNCOMA	2.377004	-107.4198	40.32123	-107.4198	40.32123	3.281651	1.375494
81079546003	3	c003	0	HYDNCOMA	8.879631	-107.3142	40.38974	-107.3142	40.38974	5.052039	3.38302
81079546003	4	c001	1	HYDNCOMA	0	-107.2637	40.29702	-107.331	40.30117	0	0
81079546003	5	c003	1	HYDNCOMA	0	-107.2593	40.35421	-107.3142	40.38974	0	0

WtCases	Firms	BusLines	Res	ResLines	SLB	Employees	CTX	RealRate	RealRes	RealFirm
8	0	0	8	8	0	0	0	0	0	0
7	0	0	7	7	0	0	0	0	0	0
11	0	0	11	11	0	0	0	0.545455	6	0
1	0	0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0

GrpPath	GrpPathLabel	GrpDist	GrpMaxDist
1	c001.	3.828918	3.828918
1	c002.	0	0
1	c003.	5.340778	5.340778
2	c001.001		
2	c003.001		

Split	Cluster	ClusterNam	Outlier	BlockGroup	Count	Bus	BusLines	Res	ResLines	SLB	Employees
HYDNCOM	2	c002	0	8.11E+10	7	0	0	7	7	0	0
HYDNCOM	3	c003	0	8.11E+10	11	0	0	11	11	0	0
HYDNCOM	4	c001	1	8.11E+10	1	0	0	1	1	0	0
HYDNCOM	5	c003	1	8.11E+10	1	0	0	1	1	0	0

CTX	RealRes	RealFirm	GrpPathLa	1	c001	0	8.11E+10	8	0	0	8
0	0	0	c002.								
0	6	0	c003.								
0	0	0	c001.001								
0	0	0	c003.001								

8	0	0	0	0	0	c001.

```
	m_Norm = m_Sum / m_nCases;
	return;
}

double CDistance::ComputeDistance(double x1, double y1, double x2, double y2)
{
	// Compute linear euclidian distance
	double distance = 0.0;
	double radius = 3956.0; // Approximate radius of the earth in miles
	double pi = acos(-1.0);
	double radians = pi / 180.0;

	x1 *= radians;
	x2 *= radians;
	y1 *= radians;
	y2 *= radians;

	if (x1 != x2 || y1 != y2)
	{
		distance = pow(sin((y2 - y1) / 2.0), 2.0) + cos(y1) * cos(y2) * pow(sin((x2 - x1) / 2.0), 2.0);
		distance = (2 * asin(min(1, pow(distance, 0.5)))) * radius;
	}

	return distance;
}

unsigned long CDistance::GetLTIndex(long n1, long n2)
{
	if (n1 < n2)
		return ((n2 + 1) * n2 / 2 + (n1 + 1)) - 1;
	else
		return ((n1 + 1) * n1 / 2 + (n2 + 1)) - 1;
}

//////////////////////////////////////////////////////////////////////////////////////////////
//
//  Normalization functions
//
//////////////////////////////////////////////////////////////////////////////////////////////

BOOL CDistance::IsNormalized()
{
	return m_bNormalized;
}
```

© PNR and Associates, All rights reserved

```
double CDistance::Normalize(double num)
{
	return num / m_Norm;
}

double CDistance::UnNormalize(double num)
{
	return num * m_Norm;
}

BOOL CDistance::Normalize()
{
	if (!m_bNormalized)
	{
		for (long i = 0; i < m_nMatSize; i++)
			m_Matrix[i] /= m_Norm;

		m_bNormalized = TRUE;
		return TRUE;
	}
	else
		return FALSE;
}

BOOL CDistance::UnNormalize()
{
	if (m_bNormalized)
	{
		for (long i = 0; i < m_nMatSize; i++)
			m_Matrix[i] *= m_Norm;

		m_bNormalized = TRUE;
		return TRUE;
	}
	else
		return FALSE;
}

double CDistance::LatFromDist(double Lat, double Distance)
{
	double Radius = 3956.0;
	double pi = acos(-1.0);
	double Radians = pi / 180.0;
```

© PNR and Associates, All rights reserved

```
	return (Distance / Radius + Lat * Radians) / Radians;
}

double CDistance::LongFromDist(double x1, double y1, double Distance)
{
	double Radius = 3956.0;
	double pi = acos(-1.0);
	double Radians = pi / 180.0;

	x1 *= Radians;
	y1 *= Radians;

	return (x1 + 2 * asin(pow( pow(sin(Distance/(2*Radius)), 2.) / (cos(y1)*cos(y1)), .5))) / Radians;
}

//////////////////////////////////////////////////////////////////////////////////////////////
//
//  Attribute functions
//
//////////////////////////////////////////////////////////////////////////////////////////////

double CDistance::FindNext(double value, BOOL higher /*TRUE*/)
{
	double temp, next = -1.0;
	for (long i = 1; i < m_nCases; i++)
		for (long j = 0; j < i; j++)
		{
			temp = Get(i, j);
			if ((higher == TRUE && temp > value && (temp < next || next == -1.0)) ||
			   (higher == FALSE && temp < value && (temp > next || next == -1.0)))
				next = temp;
		}

	return next;
}

double CDistance::Get(long n1, long n2)
{
	if (m_Matrix)
		return m_Matrix[GetLTIndex(n1,n2)];
	else
	{
		double dist;
```

© PNR and Associates, All rights reserved

```
	long row1 = (long)floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = (long)floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = (float)(pow(pow(row2-row1,2) + pow(col2-col1,2), .5) * m_dFeet);

	return (long)distance;
}

long CClustGrid::GetRLDistance(long Cell1, long Cell2)
{
	long row1 = (long)floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = (long)floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = (float)(labs(row2-row1) + labs(col2-col1)) * m_dFeet;


	return (long)distance;
}

void CClustGrid::Initialize(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long
nPoints, CDblArray *pLong, CDblArray *pLat, float *pWeight, CDWordArray* pPtMask /*NULL*/)
{
	if (!nPoints || !feet)
		return;

	// Determine width and height of grid
	double dist1 = Distance(MinLong, MinLat, MaxLong, MinLat);
	double dist2 = Distance(MinLong, MaxLat, MaxLong, MaxLat);
	m_Width = (dist1 > dist2) ? dist1 : dist2;
	m_Height = Distance(MinLong, MinLat, MinLong, MaxLat);

	// Deallocate
	delete[] m_Lines;
	delete[] m_Member;
	delete[] m_pRow;
	delete[] m_pCol;

	// Initialize member variables
	m_dFeet = feet;
	m_nPoints = nPoints;
```

© PNR and Associates, All rights reserved

```
	m_nRows = (long)(m_Height / ((double)m_dFeet/5280.) + 1);
	m_nCols = (long)(m_Width / ((double)m_dFeet/5280.) + 1);

	// Allcoate member variables
	m_Lines = new float[m_nRows * m_nCols];
	m_Member = new long[m_nRows * m_nCols];
	for (long Point = 0; Point < long(m_nRows * m_nCols); Point++) {
		m_Lines[Point] = 0;
		m_Member[Point] = 0;
	}

	m_pRow = new long[m_nPoints];
	m_pCol = new long[m_nPoints];
	for (Point = 0; Point < m_nPoints; Point++)
		m_pRow[Point] = m_pCol[Point] = 0;

	// Assign point cell locations
	BOOL bFound;
	long CurPoint = 0, nMaskIndex = 0;
	double miles = (double)m_dFeet/5280.;
	double BegLat, BegLong, EndLat, EndLong;
	for (Point = 0; Point < m_nPoints; Point++) {		// Loop works because data sorted by Lat then
Long

		bFound = FALSE;
		if (pPtMask) {
			if (pPtMask->GetAt(nMaskIndex) == (DWORD)Point)
				++nMaskIndex;
			else
				bFound = TRUE;
		}

		if (!bFound) {
			BegLat = EndLat = MinLat;
			for (long i = 0; i < m_nRows; i++) {
				BegLat = EndLat;
				EndLat = LatFromDist(BegLat, miles);

				if (!(pLat->GetAt(Point) < EndLat))
					continue;

				BegLong = EndLong = MinLong;
				for (long j = 0; j < m_nCols; j++) {
					BegLong = EndLong;
```

© PNR and Associates, All rights reserved

```
                                EndLong = LongFromDist(BegLong, (BegLat + EndLat) / 2.0, miles);
                                if (pLong->GetAt(Point) < EndLong) {
                                      m_pRow[Point] = i;
                                      m_pCol[Point] = j;
                                      bFound = TRUE;
                                      break;
                                }
                          }

                          if (bFound)
                                break;
                    }
              }
              if (pPtMask && nMaskIndex == pPtMask->GetSize())
                    break;
        }

        for (long i = 0; i < m_nPoints; i++) {
              m_Lines[Index(GetPCol(i), GetPRow(i))] += pWeight[i];
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustSet:  Storage for cluster information
//
////////////////////////////////////////////////////////////////////////////////////////////////

IMPLEMENT_DYNCREATE(CClustSet, CObject)

CClustSet::CClustSet(CClustSet *CSet)
{
      m_Feet = CSet->m_Feet;

      m_nMaxClusters = CSet->m_nMaxClusters;

      m_nPoints = CSet->m_nPoints;
      m_pCluster = new int[m_nPoints];
      for (long i=0; i<m_nPoints; i++)
            m_pCluster[i] = CSet->m_pCluster[i];

      m_OutlStart = CSet->m_OutlStart;
      m_OutlMemStart = CSet->m_OutlMemStart;
      m_nClusters = CSet->m_nClusters;
```

© PNR and Associates, All rights reserved

```
//          m_Lines.SetValue(Index(col, row), m_ClGrid.GetLines(Index(col, row)) + m_pWeight[i]);
//    }

      return TRUE;
}

BOOL CClustComp::WithinMaxDist(CDWordArray &CellList, long CurCell, CWordArray &WithinClusts, CWordArray
&RejectClusts)
{
      long i, j, n;

      int NewClust = m_ClGrid.GetMember(CurCell);

      // Check if Current Cell is a member of the approved cluster list
      if (NewClust) {
            for (i=0; i<WithinClusts.GetSize(); i++)
                  if (WithinClusts[i] == NewClust)
                        return TRUE;
      }

      // If New Cell is not yet Clustered, Compare with all other Unclustered Cells in CellList and with
Current clusts
      if (!NewClust) {
            for (i=0; i<CellList.GetSize(); i++) {
                  if (m_ClGrid.GetRLDistance(CellList[i], CurCell) > m_MaxClustDist) {
                        return FALSE;
                  }
            }

            for (n=0; n<WithinClusts.GetSize(); n++) {
                  CDWordArray *CurClustList = m_CurClust->GetClustList(WithinClusts[n]);

                  for (i=0; i<CurClustList->GetSize(); i++) {
                        if (m_ClGrid.GetRLDistance(CurClustList->GetAt(i), CurCell) > m_MaxClustDist) {
                              return FALSE;
                        }
                  }
            }
      } else {
            // check if new cluster already has failed merge with current cluster
static BOOL Test=TRUE;
            for (i=0; i<RejectClusts.GetSize(); i++) {
                  if (RejectClusts[i] == NewClust)
                        return FALSE;
```

© PNR and Associates, All rights reserved

```
            }

            // Else For Each member of included clusters, compare to new cluster
            CDWordArray *pNewClustList = m_CurClust->GetClustList(NewClust);

            for (i=0; i<CellList.GetSize(); i++) {
                  for (j=0; j<pNewClustList->GetSize(); j++) {
                        if (m_ClGrid.GetRLDistance(CellList[i], pNewClustList->GetAt(j)) > m_MaxClustDist) {
                              if (NewClust && Test)
                                    RejectClusts.Add(NewClust);
                              return FALSE;
                        }
                  }
            }

            // If existing clusters, For Each member of existing cluster, compare to new cluster
            if (WithinClusts.GetSize()) {
                  for (n=0; n<WithinClusts.GetSize(); n++) {
                        CDWordArray *CurClustList = m_CurClust->GetClustList(WithinClusts[n]);

                        for (i=0; i<CurClustList->GetSize(); i++) {
                              for (j=0; j<pNewClustList->GetSize(); j++) {
                                    if (m_ClGrid.GetRLDistance(CurClustList->GetAt(i), pNewClustList-
>GetAt(j)) > m_MaxClustDist) {
                                          if (NewClust && Test)
                                                RejectClusts.Add(NewClust);
                                          return FALSE;
                                    }
                              }
                        }
                  }
            }
      }

      // If cluster passes the test, insert into Cluster list
      if (NewClust)
            WithinClusts.Add(NewClust);

      return TRUE;
}


void CClustComp::CheckAdjacent(long iStart, long jStart, char *Checked, CDWordArray &CellList, int loop,
BOOL &Mergable, BOOL NoMerge, CWordArray &WithinClusts, CWordArray &RejectClusts)
```

© PNR and Associates, All rights reserved

```
{
      long i, j;
      long ix, jx, iDir, jDir;
      int Direct[2] = {1, -1};

      CellList.Add(Index(jStart,iStart));

static BOOL NewWay = TRUE;
if (NewWay) {
      for (iDir=0; iDir<2; iDir++) {
            for (ix=iDir; ix<=loop; ix++) {
                  i = iStart + Direct[iDir]*ix;
                  if (i<0 || i>=m_ClGrid.GetNRows())
                        break;

//                  if (Checked[Index(jStart, i)] && ix)      // if hit a checked box, finished moving this
direction
//                        continue; //break;

                  for (jDir=0; jDir<2; jDir++) {
                        for (jx=jDir; jx<=loop; jx++) {     // 2nd pass starts at jx=1 since checked 0
already
                              if (!ix && !jx)
                                    continue;

                              j = jStart + Direct[jDir]*jx;
                              if (j<0 || j>=m_ClGrid.GetNCols())
                                    break;

                              if (Checked[Index(j,i)])      // if hit a checked box, finished moving this
direction
                                    continue;   //break;

                              BOOL CurNoMerge = m_CurClust->IsNoMerge(m_ClGrid.GetMember(Index(j,i)),
m_NM_Count);
                              if ((m_ClGrid.GetMember(Index(jStart,iStart)) ==
m_ClGrid.GetMember(Index(j,i))) ||
                                                      (!NoMerge && !CurNoMerge)) {

                                    BOOL CloseEnough = TRUE;

                                    if (m_ClGrid.GetLines(Index(j,i)))
                                          CloseEnough = WithinMaxDist(CellList, Index(j,i), WithinClusts,
RejectClusts);
```

© PNR and Associates, All rights reserved

```
                                    if (CloseEnough) {
                                          Checked[Index(j,i)] = TRUE;

                                          if (m_ClGrid.GetLines(Index(j,i))) {
//                                              while (jx >= jDir) {
//                                                    j = jStart + Direct[jDir]*jx;
//                                                    Checked[Index(j,i)] = 2;
//                                                    jx--;
//                                              }

                                              CheckAdjacent(i, j, Checked, CellList, loop, Mergable,
NoMerge, WithinClusts, RejectClusts);
//                                              break;
                                          }
                                    }
                              }
                        }
                  }
            }
      }
/*
      for (ix=0; ix<loop; ix++) {
            for (jx=0; jx<loop; jx++) {
                  if (!ix && !jx)
                        continue;

                  i = iStart + ix;
                  j = jStart + jx;

                  if (Checked[Index(j,i)])
                        break;
            }

            if (!jx)
                  break;
      }
*/


} else {
      int BegLoop = 0-loop;
      int EndLoop = loop+1;
```

© PNR and Associates, All rights reserved

```
//    for (int n=0; n<loop; n++) {
            for (int ix=BegLoop; ix<EndLoop; ix++) {
                  for (int jx=BegLoop; jx<EndLoop; jx++) {
//                for (int jpos=0; j<2; j++) {
//                      int jx = (j) ? -loop : loop;

                        if (!ix && !jx)
                              continue;

                        i = iStart + ix;
                        j = jStart + jx;

                        if (i<0 || j<0 || i>=m_ClGrid.GetNRows() || j>=m_ClGrid.GetNCols())
                              continue;

                        if (!Checked[Index(j,i)])
                        {
//                            BOOL NewMergable = m_CurClust->IsMergable((int)m_ClGrid.GetMember(Index(j,i)),
m_LM_Count);
                              BOOL CurNoMerge = m_CurClust->IsNoMerge((int)m_ClGrid.GetMember(Index(j,i)),
m_NM_Count);

//                            if ((m_ClGrid.GetMember(Index(jStart,iStart)) ==
m_ClGrid.GetMember(Index(j,i))) ||
//                                      ((Mergable || (!Mergable && NewMergable)) &&
//                                      (!NoMerge || m_ClGrid.GetMember(Index(j,i))==0) && !CurNoMerge)) {

                              if ((m_ClGrid.GetMember(Index(jStart,iStart)) ==
m_ClGrid.GetMember(Index(j,i))) ||
                                                      (!NoMerge && !CurNoMerge)) {

                                    BOOL CloseEnough = TRUE;

                                    if (m_ClGrid.GetLines(Index(j,i)))
                                          CloseEnough = WithinMaxDist(CellList, Index(j,i), WithinClusts,
RejectClusts);

                                    if (CloseEnough) {
                                          Checked[Index(j,i)] = TRUE;

//                                        if (!NewMergable)
//                                              Mergable = NewMergable;
//static BOOL ClusterEmpties = FALSE;
                                          if (m_ClGrid.GetLines(Index(j,i)))
```

© PNR and Associates, All rights reserved

```
                                                    CheckAdjacent(i, j, Checked, CellList, loop, Mergable,
NoMerge, WithinClusts, RejectClusts);
//                                          else if (ClusterEmpties)
//                                                  CellList.Add(Index(j,i));
                                        }
                                    }
                                }
                            }
                        }
//      }
}

}

void CClustComp::ComputeClusters(SPECS* Specs, BOOL Outliers, COutput *OutputFile)
{
        // Assign member variables
        m_NM_Count = Specs->nMaxMerge;
        if (!Outliers)
        {
                m_MinLines = Specs->nMinN;
                m_LM_Count = Specs->nLimitedMerge;
        }
        else
        {
                m_MinLines = 0;
                m_LM_Count = Specs->nMinN;
        }


        // Declare and initialize
        long nRows = m_ClGrid.GetNRows();
        long nCols = m_ClGrid.GetNCols();
        long dFeet = m_ClGrid.GetDistance();

        if (!Outliers)
                delete m_CurClust;
        if (!m_CurClust)
                m_CurClust = new CClustSet(m_nPoints, dFeet);

        m_CurClust->SetOutlier(Outliers, m_LastCluster);
        CClustSet *NextClust = new CClustSet(m_CurClust);
        CClustSet *MoveClust = NextClust;
        NextClust = m_CurClust;
```

© PNR and Associates, All rights reserved