With the network Ids linked to each supply and demand point,

The next task is to develop a list of ‘candidate’ demands around each supply

These are all demands that are in straight-line distance of each supply

This can be done with SQL along the lines of

Create Table Candidates As

Select supply.gid as supid, supply.snapid as supsnp, demand.gid as demid, demand.snapid as demsnp, demand.pop, ST\_Distance(supply.geom, demand.geom)

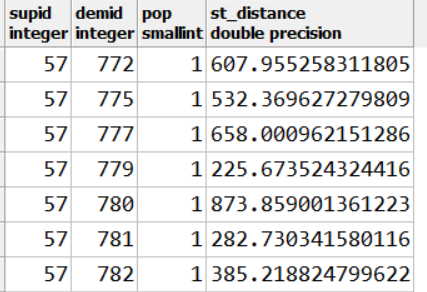
From crowflies.lsoasupplys As supply

Join crowflies.oademands As demand

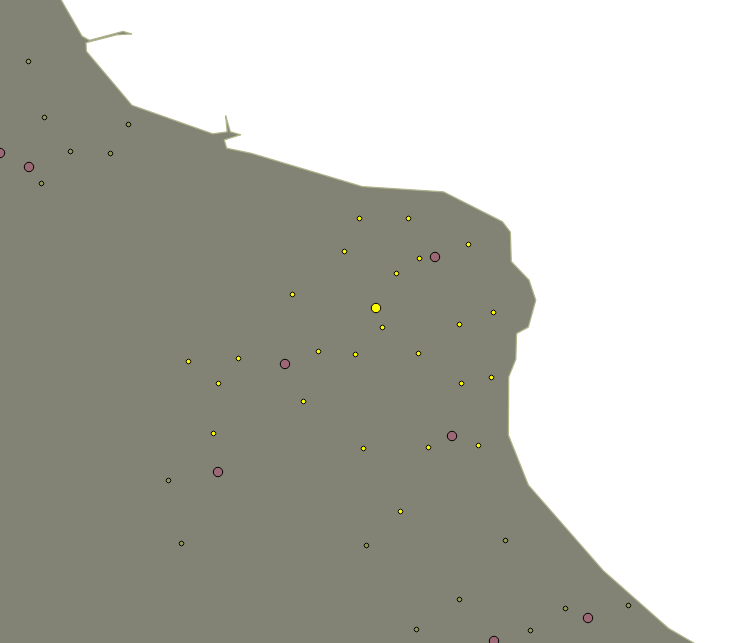
On ST\_DWithin(supply.geom, demand.geom, 1000)

Order By supid, demid;

This creates a table along the lines of…

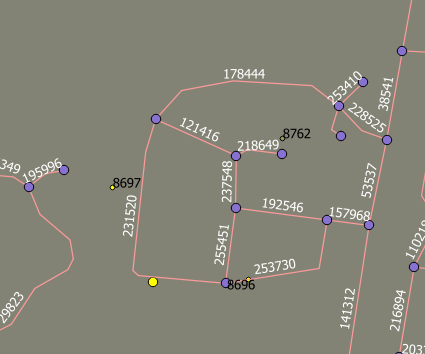


Which, visually, tracks all demands within distance of a given supply…

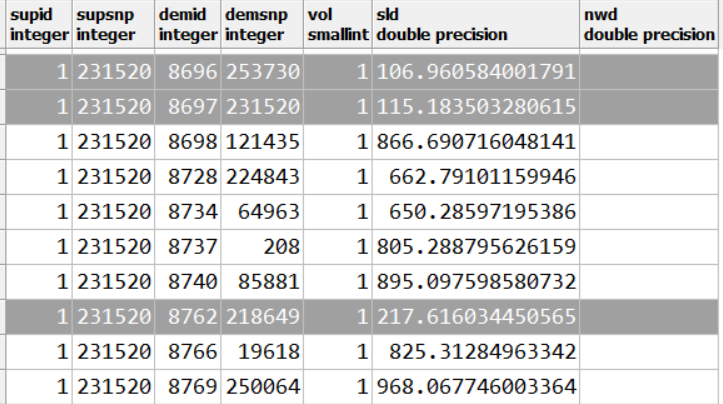


Then use the snapids to compute an actual network distance, and add this to the table

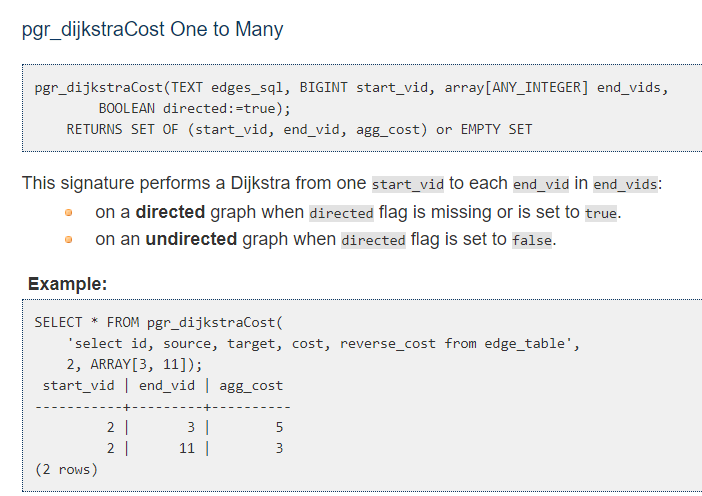
Example close-up of tracing



Supply is close to demands 869, 8697, and 8762  
These have straight-line distances of: 106m, 115m and 217m respectively.



Now compute the network distance from **supsnp** to **demsnp**

**May need to use a one-to-many computation to speed things up, hopefilly** 

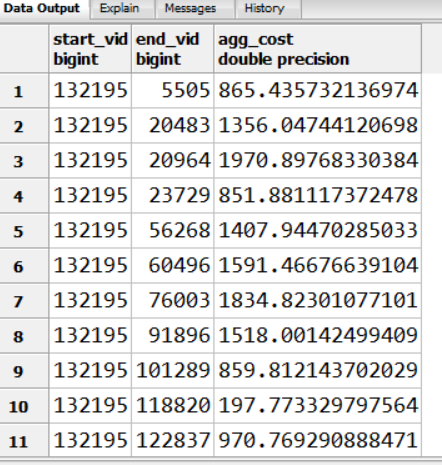
**Select \* from pgr\_dijkstraCost('Select id, source, target, cost\_len as cost From or\_apr17.jst\_wales', 141100, 99511, false)**

**Select Sum(cost) from pgr\_dijkstra('Select id, source, target, cost\_len as Cost From or\_apr17.jst\_wales',141100,99511, false, false)**

**Select \* from pgr\_dijkstraCost(**

**'select id, source, target, cost\_len as cost From or\_apr17.jst\_wales',**

**132195, array(select demsnp from candidates where supsnp = 132195), false)**



**select distinct supsnp from candidates limit 20**

**Select \* from pgr\_dijkstraCost(**

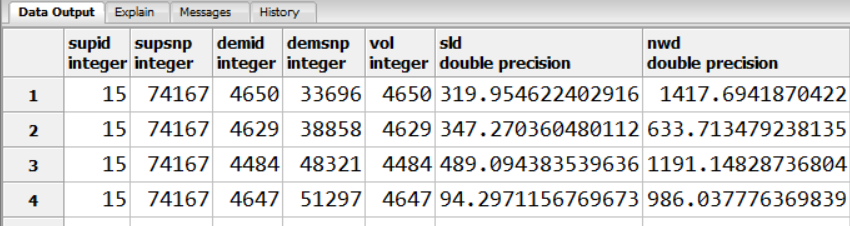
**'select id, source, target, cost\_len as cost From or\_apr17.jst\_wales',**

**132195, array(select demsnp from candidates where supsnp = 132195), false)**

**Select \* from pgr\_dijkstraCost(**

**'select id, source, target, cost\_len as cost From or\_apr17.jst\_wales',**

**96484, array(select demsnp from candidates where supsnp = 96484), false)**



**This is interesting…**

[**https://postgis.net/docs/ST\_ClosestPoint.html**](https://postgis.net/docs/ST_ClosestPoint.html)

[**https://gis.stackexchange.com/questions/49639/qgis-or-postgis-how-to-split-polylines-with-a-point-layer**](https://gis.stackexchange.com/questions/49639/qgis-or-postgis-how-to-split-polylines-with-a-point-layer)

You can see solutions using GRASS [here](http://osgeo-org.1560.n6.nabble.com/How-to-split-a-line-using-points-tp4981464.html)

There's also another option with Spatialite. The newest version 4.0 has a [function](http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.0.0.html" \l "p14b)LinesCutAtNodes() which does what you want. You feed it a linestring geometry and a point geometry, and it returns a linestring split wherever a point exactly intersect a vertex on the line. The points have to intersect vertices on the line, so you should do snap() first to get the points to exactly intersect the line vertices.

For PostGIS [ST\_Split](http://postgis.net/docs/ST_Split.html) is the maching function. You can use [ST\_Snap](http://postgis.net/docs/ST_Snap.html) to add some tolerance into the calculation

https://postgis.net/docs/ST\_Split.html

**Note – check sums: 1809 supplies done but 279 without any demands?**

**Also note that if ndw=0 then snapped to same node – but could use sld instead  
Also check how some nwd are less than the sld !!**



