**Module 6. Assembling the complete bipartite D-DP graph and the DDSN-C**

/\* The final aim is to assemble the definitive D-DP bipartite graph and the DDSN-C in which two D are joined (rule 1) if they share the same DP or (rule 2) if two of their DP share the same father or (rule 3) if one DP of the two D is the father of a DP in another D. As explained (in Module 4), we follow a bottom-top strategy starting from the hpo\_i/hpo\_j pairs (hpo\_pairs) up to the corresponding D (and finally to the corresponding PS). \*/

/\* First, by joining hpo\_pairs (from Module 4) with bipartite (from Module 3), assemble the D (and also the PS) around the heterologous hpo\_i/hpo\_j pairs (where hpo\_id\_i **≠** hpo\_id\_j). \*/

CREATE TABLE **hpo\_pairs\_hetero**(hpo\_id\_i TEXT, hpo\_id\_j TEXT, common\_hpo\_id TEXT, ps\_id\_i TEXT, mim\_id\_i INT, ps\_id\_j TEXT, mim\_id\_j INT);

INSERT INTO hpo\_pairs\_hetero

SELECT DISTINCT \* FROM(

SELECT l.hpo\_id\_i AS hpo\_id\_i, l.hpo\_id\_j AS hpo\_id\_j, l.common\_id AS common\_hpo\_id,

r1.ps\_id AS ps\_id\_i, r1.mim\_id AS mim\_id\_i,

r2.ps\_id AS ps\_id\_j, r2.mim\_id AS mim\_id\_j

FROM hpo\_pairs l

JOIN bipartite r1 ON l.hpo\_id\_i = r1.hpo\_id

JOIN bipartite r2 ON l.hpo\_id\_j = r2.hpo\_id

) WHERE mim\_id\_i != mim\_id\_j;

/\* Second, by self-joining bipartite (from Module 3), assemble the D (and also the PS) around the homologous hpo\_i/hpo\_j pairs (where hpo\_id\_i **=** hpo\_id\_j). \*/

CREATE TABLE **hpo\_pairs\_homo**(hpo\_id\_i TEXT, hpo\_id\_j TEXT, common\_hpo\_id TEXT, ps\_id\_i TEXT, mim\_id\_i INT, ps\_id\_j TEXT, mim\_id\_j INT);

INSERT INTO hpo\_pairs\_homo

SELECT DISTINCT \* FROM(

SELECT i.hpo\_id AS hpo\_id\_i, j.hpo\_id AS hpo\_id\_j, i.hpo\_id AS common\_id,

i.ps\_id AS ps\_id\_i, i.mim\_id AS mim\_id\_i,

j.ps\_id AS ps\_id\_j, j.mim\_id AS mim\_id\_j

FROM bipartite i

INNER JOIN bipartite j

ON (i.mim\_id < j.mim\_id) AND (i.hpo\_id = j.hpo\_id)

);

**6.2 The ‘redundant’ DDSN-C (more edges for the same node pair)**

/\* First, assemble all the (homologous and heterologous DP pairs) from above in table **tmp1\_ddsn**, which thus contains the 7-tuples [PSi>Di>DPi>DP(i,j)>DPj>Dj>PSj]. The resulting DDSN-C here is defined ‘redundant’, because there is often more than one edge (more than one shared DP) linking the same pair of D nodes Di and Dj. \*/

CREATE TABLE **tmp1\_ddsn**(ps\_id\_i TEXT, mim\_id\_i INT, hpo\_id\_i TEXT, common\_hpo\_id TEXT, hpo\_id\_j TEXT, mim\_id\_j INT, ps\_id\_j TEXT);

INSERT INTO tmp1\_ddsn

SELECT \* FROM(

SELECT ps\_id\_i, mim\_id\_i, hpo\_id\_i, common\_hpo\_id, hpo\_id\_j, mim\_id\_j, ps\_id\_j

FROM hpo\_pairs\_homo)

**UNION**

SELECT \* FROM(

SELECT ps\_id\_i, mim\_id\_i, hpo\_id\_i, common\_hpo\_id, hpo\_id\_j, mim\_id\_j, ps\_id\_j

FROM hpo\_pairs\_hetero WHERE mim\_id\_i < mim\_id\_j

**UNION**

SELECT ps\_id\_j, mim\_id\_j, hpo\_id\_j, common\_hpo\_id, hpo\_id\_i, mim\_id\_i, ps\_id\_i

FROM hpo\_pairs\_hetero WHERE mim\_id\_i > mim\_id\_j);

/\* Second, in table **tmp2\_ddsn**, add the IC values to the shared common\_hpo\_id of tmp1\_ddsn. \*/

CREATE TABLE **tmp2\_ddsn**(ps\_id\_i TEXT, mim\_id\_i INT, hpo\_id\_i TEXT, common\_hpo\_id TEXT, ic REAL, hpo\_id\_j TEXT, mim\_id\_j INT, ps\_id\_j TEXT);

INSERT INTO tmp2\_ddsn

SELECT l.ps\_id\_i, l.mim\_id\_i, l.hpo\_id\_i, l.common\_hpo\_id, r.ic, l.hpo\_id\_j, l.mim\_id\_j, l.ps\_id\_j

FROM tmp1\_ddsn l

JOIN hpo2ic r ON l.common\_hpo\_id = r.hpo\_id;

/\* Third, in table **d\_dp**, assemble the definitive bipartite graph as d\_dp. Note that table bipartite (in Module 3) is the bipartite graph in which each D node is paired with the DP node(s) that directly annotate that D in HPO. In contrast, d\_dp (In the present Module 6) is the bipartite graph in which each D node is paired with the DP (common\_hpo\_id) that the D shares with another D in the provisional DDSN (tmp2\_ddsn). \*/

CREATE TABLE **d\_dp**(mim\_id INT, hpo\_id TEXT, ic REAL);

INSERT INTO d\_dp

SELECT mim\_id\_i AS mim\_id, common\_hpo\_id AS hpo\_id, ic FROM tmp2\_ddsn

UNION

SELECT mim\_id\_j AS mim\_id, common\_hpo\_id AS hpo\_id, ic FROM tmp2\_ddsn;

DROP TABLE tmp; DROP TABLE tmp;

.once c:/SQLITE/LHPS/outputs/d\_dp.txt

SELECT \* FROM d\_dp;

/\* Fourth, in table **tmp3\_ddsn**, assemble the ‘preliminary’ version of the **DDSN-C**. The aim is achieved by assembling all the [Di>Dj] 2-tuples from tmp2\_ddsn, which are the nodes of the DDSN-C, while the [DP(i,j)] are the edge(s) linking the two D nodes from the [Di>DPi>DP(i,j)>DPj>Dj] 5-tuples of tmp2\_ddsn. It is preliminary, because each pair of D can be linked by more than one DP (one DP per edge), given that most of the pairs share more than one DP. \*/

CREATE TABLE **tmp3\_ddsn**(mim\_id\_i INT, common\_hpo\_id TEXT, ic REAL, mim\_id\_j INT);

INSERT INTO tmp3\_ddsn

SELECT DISTINCT \* FROM(

SELECT mim\_id\_i, common\_hpo\_id, ic, mim\_id\_j FROM tmp2\_ddsn);

**6.3 The definitive ‘non-redundant’ DDSN-C (one edge for the same node pair)**

/\* Fifth, in table **ddsn**, assemble the definitive (‘compact’) version of the DDSN-C, by grouping the [Di>DP(i,j)>Dj] 3-tuples of tmp3\_ddsn. Note that there cannot be the reverse pair (mim\_j and mim\_i), because the tmp1\_ddsn table was assembled with the criteria that mim\_ids are integers and that mim\_id\_i < mim\_id\_j. Also, as there are often more hpo terms annotating the same mim pair, calculate hpo term number (count\_hpo) and average of IC (ic\_mean) for all the shared hpo terms. \*/

CREATE TABLE **ddsn**(mim\_id\_i INT, mim\_id\_j INT, count\_hpo INT, ic\_mean REAL);

INSERT INTO ddsn

SELECT mim\_id\_i, mim\_id\_j, COUNT(DISTINCT common\_hpo\_id) AS count\_hpo, ROUND(AVG(ic),3) AS ic\_mean

FROM tmp3\_ddsn

GROUP BY mim\_id\_i, mim\_id\_j;

.once c:/SQLITE/LHPS/outputs/ddsn\_c.txt

SELECT mim\_id\_i, ic\_mean, mim\_id\_j FROM ddsn;