# Ollscoil na hÉireann
# The National University of Ireland

# Coláiste na hOllscoile, Corcaigh
# University College, Cork

Mid-Term Examination 2018

**CS6323 Complex Networks and Systems**

M.Sc. Software and Systems for Mobile Networks
M.Sc. Computer Science

Dr. W. Scanlon, Extern
Professor C. Sreenan, HoD
Prof. G. Provan

*Attempt all questions*

*Total marks: 100*

*60 minutes*

# CS6323 Complex Networks and Systems

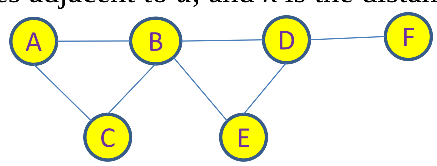
Mid-Term Examination 2018

**Please answer all questions**
**Marks for each question are indicated by [xx]**
**Total Marks: 100**

1. **[35]** *Social Network weighting*: Consider a social network in which we want to assign weights to people based on their friend network. We define a social network as an undirected graph, G(V,E), where an edge joins *u* and *v* if they are direct friends. Indirect friends are called *k*-hop away if there are *k* edges in the shortest path between them. Each person *u* is assigned a weight *w* as follows:

$$w(u) = \sum_{v \in \Delta(u)} [1/k] w(v)$$

where $\Delta(u)$ are the nodes adjacent to *u*, and *k* is the distance of node *v* from *u*.




w(A)= 1/1[w(B) + w(C)]+ ½[w(D) + w(D)]+ 1/3 w(F)
= [2] + ½ [2] + 1/3 [1]
= 3.333

**Figure 1: Friend network with computation of weight of A**

Figure 1 shows the (centralized) computation of *w*(A), where nodes B and C are of distance 1, nodes D and E of distance 2, and node F of distance 3, and we count the weight of a neighbour as 1.

We want to use MapReduce to compute the weights assigned to each vertex in G based on friend relations.

a. **[10]** Draw the MapReduce architecture for computing the friend relations weighting function when *w(v)*=1, i.e., we weight only presence of friends.
b. **[10]** Write down the pseudo-code for computing the weighting function.
c. **[5]** Derive an upper bound for the number of cycles of MapReduce that are required, justifying your answer.
d. **[10]** Illustrate the operation of your algorithm using the example of Figure 1. Show the number of cycles of MapReduce, and the final weighting for nodes B, C, D and E.

2. [**35**] Consider a store that wants to compute the impact of having a set S={$S_1,\ldots,S_k$} of items on sale. Given the set G={$G_1,\ldots,G_n$} of goods at regular prices, the store wants to calculate the probability that a customer will buy good $G_i$ given that they have bought sale item $S_j$, i.e., $P(G_i|S_j)$, over all *i* goods and *j* sale items. We call this our sale distribution. We have as input a very large collection or orders, where an order has data for sale-price and regular-price items purchased together, e.g,

$\{S_1,\ldots,S_k, G_1,\ldots,G_n\}$. Figure 2 shows an example of this data; in order $O_1$, for example, a customer purchased $\{S_1, S_2, G_1\}$.

| | S1 | S2 | G1 | G2 | G3 |
|---|---|---|---|---|---|
| $O_1$ | 1 | 1 | 1 | | |
| O2 | | 1 | | 1 | 1 |
| O3 | 1 | 1 | 1 | 1 | |
| O4 | 1 | | | 1 | |
| O5 | 1 | 1 | | 1 | 1 |
| O6 | | 1 | | 1 | 1 |
| O7 | 1 | | 1 | | 1 |
| O8 | 1 | 1 | 1 | | 1 |

**Figure 2: Data for 8 orders, with 1 indicating a purchase**

a. **[10]** Draw the MapReduce architecture for computing the sale distribution, assuming that we partition a set O of *m* orders into a collection of sub-orders with *q* orders per sub-order.
b. [**10**] Show the pseudo-code for a MapReduce algorithm to compute the sale distribution.
c. [**5**] How many MapReduce rounds will it take to compute the sale distribution?
d. [**10**] Apply your algorithm for MapReduce to the data shown in Figure 2, assuming we partition the data into 2 pieces.