GPU Accelerated Bipartite Graph Collaborative Filtering

Qianbin Xia, xiaq2@vcu.edu

In the mathematical field of graph theory, a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V, and vertices in U and V are connected through edges. There is no connection within the same group. Vertex set U and V are always denoted as partite sets. Figure1 gives an example of a bipartite graph structure.

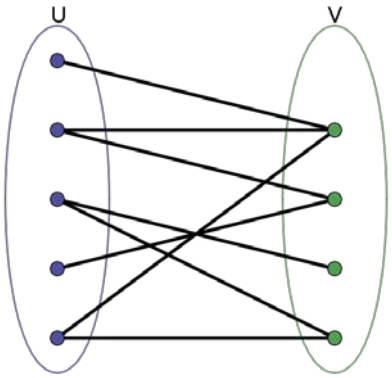

Fig. 1 An example of bipartite graph structure

Bipartite Graphs have many applications in real life, especially modelling relations between two different objects. For instance, a graph of peoples and books, the peoples form the partite set U, while the different books constitute the partite set V, different people have read different books. So there are connections between vertices in set U and V. Another example, a graph of workers and skills, different workers have different skills: C++, java, python .etc. This kind of bipartite graph gives the manager a clear and detail information about how to assign different jobs to different workers.

There are two graph algorithms which can be very helpful for us to mine some useful information from the bipartite graph: naive graph collaborative filtering and full graph collaborative filtering. Definition 1.1 and Definition 1.2 depict the naive graph collaborative filtering and collaborative filtering algorithms in detail. The naive graph collaborative filtering will find out the k most related vertices to u_start from U. There are many real application cases which can benefit from the naive graph filtering. For example, if you are the boss of a company and one of you employee retired. You need to find a substitution from the LinkedIn. If we build a bipartite graph with the members and skills, then we can use the retired workers as the u_start, and find the k most related members from the LinkedIn as our candidate.

Definition 1.1. (Naive Graph Collaborative Filtering): Given a bipartite graph $G((U,V),E)$ and a vertex $u \in U$, find top k relevant vertices to u, denoted by $u'_1, u'_2, \cdots, u'_k$ according to:

$$u'_i = \arg \max_{x \in U \setminus \{u'_1, \cdots, u'_{i-1}\}} |\Gamma_u \cup \Gamma_x| \quad (1)$$

where $\Gamma_u = \{v | v \in V, (u,v) \in E\}$.

Definition 1.2. (Collaborative Filtering for Visualization): Given a bipartite graph $G((U,V),E)$ and a vertex $u \in U$, find top k relevant vertices to u, denoted by $U' = \{u'_1, u'_2, \cdots, u'_k\}$; then, for each $u'_i \in U'$, find the top k' vertices relevant to u'_i, denoted by $U''_i = \{u''_1, \cdots, u''_{k'}\}$. Return $U' \cup \{U''_i\}_{\forall i=1,\cdots,k}$.

The full collaborative filtering algorithm is based on the naive collaborative filtering algorithm. When the naive collaborative filtering algorithm find the k most related vertices to u_start, the full collaborative filtering algorithm will search the k most related vertices to each of the k most related vertices in naive collaborative filtering stage. One possible application of full collaborative filtering algorithm is in the social networks like Facebook. For friend recommendation systems in social networks, if we can find out k best friends from all Eric's friends, then we search k best friends for each of Eric's k best friends respectively. The meaning behind the full collaborative filtering algorithm is that, if two people are best friends, there is a high possibility that they will know each other's best friends through birthday party or some other activities. So if they are not friends on Facebook, we can recommend them to each other.

In both of these algorithms, the relation counting is the bottleneck of the performance. In our project, we plan to leverage GPU to accelerate the relation counting part. GPU can provide high parallel computing power, which is suitable with the relation counting part since the relation counting for each individual vertices in part U are independent of each other.