Bipartite Matching

Brian Page

1 Introduction

Graph matching seeks to determine a set of edges within the graph such that there are no vertices in common among the edges selected. As its name implies, bipartite matching is a matching performed on a bipartite graph [4] in which the vertices of said graph can be divided into two disjoint sets.

Bipartite matching has many real world applications, many of which resemble some form of assignment or grouping [6]. One such example would b that of job positions vs job applicants. Each applicant has a subset of jobs they have applied for, yet each position can filled by at most one applicant. A matching of this graph would be performed in an attempt to find the maximum number of applicants that can be placed into the job openings. This of course is but one example of bipartite matching.

Bipartite matching while useful in its own right, is often used as an intermediate algorithm to prepare data for subsequent computation. Because of this, efficient computation of bipartite matching has become an interesting topic among HPC researchers as scalability and performance continue to increase in importance.

The Problem as a Graph Before we can dive into bipartite matching, we must first understand the different types of graph matching [5]. Considering a graph $G = (V, E)$ where V is the set of vertices and E the set of all edges a vertex is considered to have been *matched* when an edge has the vertex as one its endpoints.

Maximal matching is a matching M in which the the inclusion of an edge not currently in M would make M no longer a valid matching. This occurs when an edge is added, that has had at least one it its vertices matched previously. Fig. 1 illustrates three sample graphs and their corresponding maximal matching.

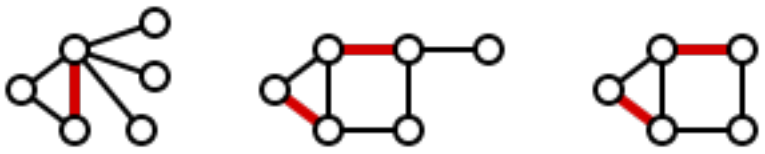

Figure 1: Maximal Matching

The *maximum matching* of a graph is a matching consisting of the largest possible independent edge set. The maximum matching does not have to be a unique solution, as often there may be multiple maximum matchings for a given graph. Fig. 2 illustrates the maximum matching for the previously shown sample graphs.

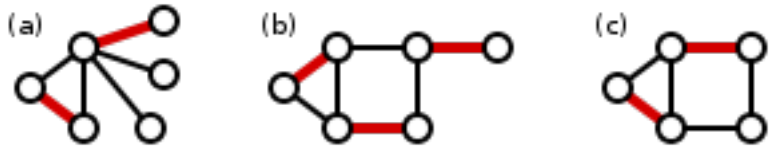

Figure 2: Maximum Matching

One important thing to observe is that each *maximum matching* is a *maximal matching*, however not all *maximal matching* are the *maximum matching* for a particular graph. This can be seen in Fig. 2 where for graphs (a) and (b) the maximum matching is different from the maximal matchings shown in Fig. 1. Additionally we see that for graph (c) the previous maximal matching is in fact the maximum matching.

There are other forms of matching that can be discussed, however the most widely used is that of determining the *maximum matching* of a graph and is the focus of this topic.

One of the most common methods for solving bipartite matching is to treat the graph $G = (V, E) = ((u, v), E)$ as a flow network in which a connection or edge between vertices u and v may or may not be selected in the final matching. The Ford-Fulkerson algorithm determines the maximum flow through just such a graph/network and in the case of bipartite matching, is used to determine the maximum matching on G. Ford-Fulkerson [7] works by adding and removing edges while checking the matching with the changed edge state (included or excluded) until it has determined the optimal edge set or *matching*.

There are of course other methods such as Hopcroft-Karp which performs a localized randomization of edge inclusion/exclusion, as well as the well known Bellman-Ford algorithm. Fig. 3 shows a graph which has been reduced to a flow network. Reduction of G to that of a flow network problem is performed by many of the most common methods. This is due to the improved time complexity achieved, $\mathcal{O}(nlog^3n)$, that arises from the the simplification of G to a planar graph.

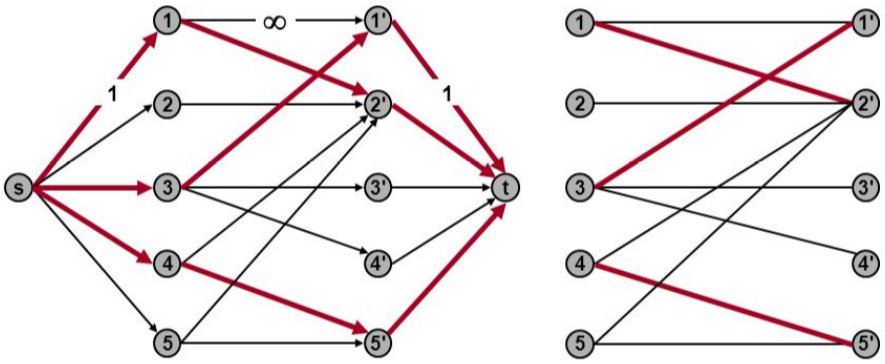

Figure 3: Graph conversion to a flow network for the purpose of determining its maximum matching.

2 Some Realistic Data Sets

A bipartite graph can and often is represented as sparse matrix, therefore there are many sources of bipartite graphs in existence today. The Suite Sparse Matrix Collection [2] contains many real world data sets for different research areas such as fluid dynamics and circuit problems. The matrices have varying characteristics such as row count and total non-zeros ranging from as few as 20 rows with 90 non-zeros to millions of rows with hundreds of millions of non-zeros.

Another source for real world data is the Stanford Large Network Dataset Collection (SNAP) [3] which hosts many large graphs for social media and web based networks.

Lastly there are many tools which are used for the generation of synthetic bipartite graphs. Using such tools, a researcher can create graphs in which they have control over structural characteristics.

Bipartite Matching Kernel Given the myriad of potential implementations for bipartite matching the notional kernel we will discuss uses the Hopcroft-Karp [8] algorithm to find the maximum matching.

Algorithm 4 is based on the push and relabel method for finding maximum flow in which a bipartite graph is given as input. The algorithm then uses breadth first search (BFS) in order to partition the vertices into two sets *matched* and *unmatched*. Edges are then swapped in and out of the matching, with the resulting matching M evaluated against the highest matching achieved thus far.

One key difference between hopcroft-karp and ford-fulkerson is its use of localized path augmentations where an edge incident to the current vertex can be included or excluded without determining the entire path accross G. The results in an overall time complexity of $\mathcal{O}(\mathsf{E}\sqrt{\mathsf{V}})$.

Input: Bipartite graph $G(U \cup V, E)$
Output: Matching $M \subseteq E$
$M \leftarrow \emptyset$
repeat
  $\mathcal{P} \leftarrow \{P_1, P_2, \ldots, P_k\}$ *maximal set of vertex-disjoint shortest augmenting paths*
  $M \leftarrow M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$
until $\mathcal{P} = \emptyset$

Figure 4: Hopcrof-Karp algorithm for finding maximum flow on a bipartite graph.

References

[1] Jeremy Kepner and John Gilbert. Graph algorithms in the language of linear algebra, 2011.

[2] Suite spare matrix collection. https://sparse.tamu.edu.

[3] Stanford large network dataset collection (snap). https://snap.stanford.edu/data/.

[4] Wikipedia. Bipartite graph. https://en.wikipedia.org/wiki/Bipartite$_graph$.

[5] Wikipedia. Matching. https://en.wikipedia.org/wiki/matching$_(graph_theory)$.

[6] Wikipedia. Assignment problem. https://en.wikipedia.org/wiki/assignment$_problem$.

[7] Ford-fulkerson algorithm. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/.

[8] Hopcroft-karp algorithm. https://www.geeksforgeeks.org/hopcroft-karp-algorithm-for-maximum-matching-set-1-introduction/.