In this lecture we will see another application of graph embedding. We will see that certain problems (e.g. maximum independent set, MIS) can be solved fast for a graph if it has a low congestion embedding in any tree. Later we will see a different kind of embedding in which vertices of the guest graph are mapped to not just one node but a set of nodes. This will lead to the notion of *treewidth*, and we will also see an application to solving linear systems defined on graphs which arise in engineering calculations.

1 Maximum Independent Set

In a graph $G = (V, E)$, we call $V' \subseteq V$ *independent* if $u, v \in V' \Rightarrow (u, v) \notin E$. In the maximum independent set problem, the requirement is to find an independent set V' of maximum cardinality. In the natural weighted version, for each vertex u we are given a weight w_u and the weight w_U of a set of vertices U is defined as the sum of the weights of the vertices in U. The goal is to find an independent set of maximum total weight. The maximum independent set problem, weighted or unweighted, is NP-complete, and hence unlikely to have a polynomial time algorithm in general.

1.1 A divide and conquer strategy

The basic idea of the algorithm we discuss is divide and conquer. Of course, just dividing the graph into two parts, and taking the union of the MIS of the two parts will not work. This is because the union may not be independent: a vertex u in the right MIS may have an edge to a vertex v in the left independent set.

Instead we consider the following stronger *boundary constrained MIS* problem to solve in the recursion. Let G' be any subgraph of G, and let $\partial G'$ denote those nodes of G' that have edges outside G'. Let s denote a subset of $\partial G'$. Then for each G' and s, we must find $S^s_{G'}$, the maximum weight independent set in G' which must contain the vertices in s, and not contain any vertex in $\partial G' - s$. We will use $A^s_{G'}$ to denote the weight of this set. In fact, we will only worry about finding $A^s_{G'}$, finding the set itself is a minor extension which we will not discuss. If the vertices in s are not independent in G', then $S^s_{G'}$ is not defined, and it will be expected that $A^s_{G'} = -\infty$ will be returned. The solution to a boundary constrained MIS problem for a subgraph G' is simply the collection $A_{G'}$ of the values $A^s_{G'}$ for the different s.

At the leaf level, $A_{G'}$ is evaluated by a brute force strategy, We discuss this for the general case in Section 1.3, where we also show how to combine solutions. Before that, we design an algorithm for trees based on this idea.

1.2 Algorithm for binary trees

We describe the algorithm directly, and then later show how it fits into the general scheme.

1. Pick an arbitrary node and designate it as the root r. For each vertex define the notion of parents and children with respect to r.

2. For each vertex u we will compute P_u, the weight of the MIS for the subtree below u constrained to contain u, and Q_u, the weight of MIS for the subtree under u constrained to not contain u.

   (a) For each leaf u of the tree $P_u = w_u$, $Q_u = 0$.

   (b) Pick any node u such that P, Q have been computed for all its children, say v, w. Then $P_u = w_u + Q_v + Q_w$. $Q_u = \max(P_v, Q_v) + \max(P_w, Q_w)$. In this way we can calculate P_u, Q_u for all non-leaf nodes.

3. Return $\max(P_r, Q_r)$.

It will be seen that the algorithm runs in linear time.

Suppose $T(u)$ denotes the subtree under node u. Then note that $A_{T(u)} = \{P_u, Q_u\}$, noting that the boundary of $T(u)$ is simply the vertex u. The base case for the recursion is simply when u is itself a leaf, step 2a above. Step 2b shows how smaller solutions are combined together; this combination is quite simple for trees.

1.3 General scheme

To solve our problem for a subgraph G' which we have decided is a base case for the recursion, we will evaluate $A_{G'}$ by brute force. Consider the computation of $A^s_{G'}$ for a certain s. By fixing s we have determined which nodes of $\partial G'$ will be in the independent set. That only leaves nodes in $V(G') - \partial G'$. A simple algorithm for this is:

1. For each $t \subseteq V(G') - \partial G'$: Check if $s \cup t$ is independent. If so calculate its weight, and keep track of the maximum weight seen.

2. $A^s_{G'} =$ max weight seen, $= -\infty$ if $s \cup t$ is not independent.

There are $2^{|\partial G'|}$ choices for s, and $2^{|V(G')|-|\partial G'|}$ choices for t. Thus the for loop runs $2^{|V(G')|}$ times. In each loop, checking for independence and computing the weight may take $O(|E(G')|)$ time, which should multiply the time estimate. We will use the notation $\tilde{O}(f)$ to mean $O(fg)$ where f involves exponentials and g only involves polynomial terms.[1] Thus our time taken is $\tilde{O}(2^{|V(G')|})$.

For the divide step, it is conceivable that we may divide simply by removing edges, however, (as for trees) it is more convenient to remove a "middle" region which breaks the problem into "left" and "right" regions. Thus in the combine step, we will typically combine 3 regions L, R, M into a single region U which might be smaller than G.

We will assemble A^u_U from A^l_L, A^r_R, A^m_M. Consider S^u_U. It must be the union of sets S^l_L, S^r_R, S^m_M for some l, r, m that are independent, and u is consistent with l, r, m. Since the boundary of U is a subset of the boundaries of L, R, M we must have $u = (l \cup r \cup m) \cap \partial U$. Defining $f(l, r, m) = (l \cup r \cup m) \cap \partial U$, and using $I(x)$ to assert that x is independent, we can write

$$A^u_U = \max_{l,r,m}\{A^l_L + A^r_R + A^m_M \mid I(l \cup r \cup m), u = f(l, r, m)\} \qquad (1)$$

It can be seen that A^u_U for all u together can be computed in time $\tilde{O}(2^{|\partial L|+|\partial R|+|\partial M|})$ as follows:

1. Fix $u \subseteq \partial U$.

2. Fix $t \subseteq (\partial L \cup \partial R \partial M) - \partial U$. Notice that s, t together fix l, r, m.

3. Use equation 1 to find A^u_U.

Since s, t can together be fixed in time $\tilde{O}(2^{|\partial L|+|\partial R|+|\partial M|})$ the bound follows.

[1]This notation hides polynomial factors, just as O notation hides constant factors.

2 Graphs with good tree embedding

Suppose G is a graph with a load L, congestion C embedding on a tree H. The tree H will determine the structure of our recursion. The algorithm is as follows.

1. Pick an arbitrary node of H and designate it as the root r. For each node define the notion of parents and children with respect to r.

2. For each node u let $\mathcal{T}(u)$ denote the graph induced in G by the vertices embedded in the nodes of the subtree of H rooted at u. We will calculate $A^s_{\mathcal{T}(u)}$ for all s recursively starting from the leaves of H.

   (a) For each leaf u of H, we calculate $A^s_{\mathcal{T}(u)}$ for all s together by brute force.

   (b) Let u be a node of H such that the A values have been calculated for its children v, w. We show how the A values can be calculated for u; in this way we can calculate the values for all nodes of H in the order backwards from the leaves towards the root. Let $G(u)$ denote the subgraph induced in G by the vertices embedded in node u of H. The key idea is to use equation 1, noting that $\mathcal{T}(u)$ has been decomposed into subgraphs $\mathcal{T}(v), \mathcal{T}(w), G(u)$.

   $$A^k_{\mathcal{T}(u)} = \max_{l,r,m}\{A^l_{\mathcal{T}(v)} + A^r_{\mathcal{T}(w)} + A^m_{G(u)} \mid I(l \cup r \cup m), k = f(l, r, m)\}$$

3. Return $A^{\phi}_{\mathcal{T}(r)}$.

For step 2a, because the load is L, we have $|V(\mathcal{T}(u))| \leq L$. Thus in time $\tilde{O}(2^L)$ we can find the complete $A_{\mathcal{T}(u)}$ for any fixed u.

In step 2b, we calculate $A_{G(u)}$ by brute force. This takes time $\tilde{O}(2^L)$ as for the leaves in step 2a. Each border node of $\mathcal{T}(u)$ must have at least one edge going outside of $\mathcal{T}(u)$, and this edge must be embedded in the edge from u to its parent. But we know the congestion is C, and hence for all u, $|\partial\mathcal{T}(u)| \leq C$. Further $|\partial G(u)| \leq |V(G(u)| \leq L$. Thus by the analysis of the previous section, computing $A_{\mathcal{T}(u)}$ is done in time $\tilde{O}(2^{L+2C})$ at each node of H, and hence $\tilde{O}(2^{L+2C})$ overall (note that $\tilde{O}$ will hide the $V(H)$ factor). This can in general be much less than the $\tilde{O}(2^{V(G)})$ using the brute force algorithm for all of G.

2.1 Remarks

Our analysis is conservative. Suppose a single vertex x is embedded at tree node v. Suppose x has edges to vertices y, z both embedded at the parent u of v. The edge (u, v) has congestion 2, whereas $|\partial v|$ is just 1.

The algorithm presented here makes it important to find low congestion and low load embedding into arbitrary trees. For example, what embedding would you use for the hypercube Q_n?

Exercises

1. Consider the maximum independent set problem on the tree. For the unweighted case prove that there must exist an optimal solution which contains all the leaf nodes. Use this idea to give an algorithm to find the optimal solution.

2. Show that there may exist optimal solutions which need not contain all the leaf nodes. For the weighted case, show that the optimal solution may not contain a single leaf.

3. Extend the algorithm for finding the weight of the maximum weight independent set to find the set itself.

4. Design an algorithm to find an MIS for $G = P_c \Box P_r$, $r \leq c$, by embedding it in P_c. If your tree has unary nodes, derive a formula for unary nodes and use it. Only estimate the exponential terms in the running time, i.e. give an $\tilde{O}$ answer.