Dynamic Programming on Graphs of Bounded Tree Width

1. Preprocessing

We assume that the vertices of G are labeled with the integers 1 to n, where n=|V(G)|, so that every edge is simply a pair of integers. If the vertices of G are labeled with arbitrary labels from an alphabet, we can sort these labels, and then replace each vertex name with its position in the ordered list. To replace the pair of old vertex names in an edge with the appropriate new vertex names we simply need to find each of these two names in our ordered list which takes O(log n) time. So, in total this relabeling process takes O((|V(G)|+|E(G)|)log(|V(G)|) time.

We will also do some preprocessing after which we can test if two vertices are joined by an edge in O(1) time. We could simply build an array Adjacent[1..n,1..n] where A[i,j] is 1 if ij is and edge of G and 0 otherwise. This would require zeroing the array and then going through the edge list to insert the 2 entries of Adjacent which are 1 corresponding to each edge. However, zeroing the array takes $O(n^2)$ time and we want to do things a bit more quickly. So, we will not zero the array. Instead we will use an auxiliary integer variable number_of_edges and an auxiliary array Edge whose entries are pairs of integers and which is indexed from $[1..n^2]$. To begin we process the edge list of G, updating these variables, so that at the end, number_of_edges is the value it claims to be and for I between 1 and number_of_edges Edge[i] is the endpoints of the i^{th} edge of G. Also if the i^{th} edge is (j,l) then we set Adjacent[j,l]=i. We do this as follows:

```
number_of_edges:=0
while there is an unread edge of G, read the next edge (j,l)
  number_of_edges:= number_of_edges+1
  E[number_of_edges]:=(j,l)
  Adjacent[j,l]=number_of_edges
  Adjacent[l,j]=number_of_edges
endwhile
```

Given this preprocessing which takes O(|E(G)|) time, we can test if two vertices j and l of G are adjacent as follows.

```
If Adjacent[j,l]<1 or Adjacent[j,l]> number_of_edges then return nonadjacent
Else
    If E[Adjacent[j,l]]=(j,l) or (l,j)] then return adjacent else return nonadajacent.
```

For any w', given a graph G we can either determine that it has tree width exceeding w' or find a (rooted) tree decomposition of it of width at most w=4w'+4 with O(|V(G)|) nodes and such that each internal node has 2 children in O(|E(G)|) time (the algorithm presented in class and the text runs in O(|V(G)||E(G)|) time but there is a faster more complicated algorithm to solve the problem).

So. we assume we are given a tree decomposition of a graph G of width some constant w , consisting of (i) a rooted tree T with root r and O(|V(G)|) nodes such that each internal node has 2 children, and (ii) for every node t of T, a subset W_t of V(G). We consider the nodes via a post-order transversal. For each node t, T(t) is the subtree of T consisting of t and its descendants while $G_t \subseteq G$ has $V(G_t) = \bigcup_{s \text{ a node of } T(t)} Ws$ and $E(G_t) = \{xy \in E(G) | x,y \in V(G_t)\}$.

2. The Paradigm

We want to solve various optimization programs using dynamic programming on this tree decomposition. Our paradigm for doing so is as follows:

We construct an array F indexed by the nodes of t, and for each node t, the possible intersections of the restriction of a solution to our problem to G_t with W_t. For each t and "intersection pattern" X, F(t,X) stores the best possible solution cost/value for a restriction of a solution to G_t which has this intersection pattern. We may also record the cost/value as infinity/- infinity to indicate there is no partial solution with this intersection pattern.

To design/analyze such an algorithm we first need to understand what the restrictions of solutions to G_t look like and specify what the "intersection patterns" with W_t should be. We then need to describe how to compute the table for a leaf, and for a non-leaf given tables for its children. The time complexity of our algorithm depends on the size of the table entries for a node, and how long each entry takes to compute.

3. Maximum Stable Set

For the Maximum Stable Set problem considered in the lecture(the text considers a very similar algorithm for Maximum Weight Stable Set), the restriction of a solution to G_t is simply a stable set of G_t and the intersection pattern of this solution with W_t is simply a subset of W_t. Thus, we have a table entry F(t,X) for every subset X of W_t which records the largest stable set of G_t whose intersection with W_t is X (where if X is not a stable set then F(t,X) is – infinity). We return the maximum of F(r,X) over all subsets X of W_r. It remains to describe how to compute the F(t,X).

If X is not a stable set then we set F(t,X) to – infinity.
If X is a stable set and t is a leaf, then F(t,X) is |X|.

If X is a stable set and t has two children c(1) and c(2) then we perform the following computation:

F(t,X):=|X|
For each subset X_1 of $W_{c(1)}$ and subset X_2 of $W_{c(2)}$
  If $X_1 \cap W = X \cap W_{c(1)}$ AND $X_2 \cap W_t = X \cap W_{c(2)}$ AND $|X|+F(c_1,X_1)-|X \cap X_1|+F(c_2,X_2)-|X \cap X_2| > F(t,X)$
    $F(t,X) := |X|+F(c_1,X_1)-|X \cap X_1|+F(c_2,X_2)-|X \cap X_2|$

We note that determining if a set X is stable can be done in $O(|X|^2)=O(w^2)$ time using our adjacency testing routine. For X which are stable, computing F(t,X) involves running through all of at most 2^{2w} pairs of table entries, one from the table for c_1 and the other from the table for c_2. For each such pair of entries and each i in {1,2}, it takes $O(w^2)$ time to check if $X_i \cap W_t = X \cap W_{c(i)}$. So we can compute F(t,X) in $O(2^{2w} w^2)$ time. Since there are at most 2^w subsets of W_t it follows that we can compute the table entries for t in $O(2^{3w} w^2)$ time. Since T has O(|V(G)|) nodes and w is a fixed constant, we see that the total time taken by our procedure is O(|V(G)|).

4. Dominating Set

For the Dominating Set problem, the restriction of a solution to G_t is simply a subset of $V(G_t)$ which dominates all of the vertices of G_t-W_t (some vertices of W_t can be dominated by their neighbours in the rest of the graph). The intersection pattern of such a restriction of the solution with W_t consists of two subsets D and C of W_t with $D \subseteq C$ where D is the intersection of the dominating set with W_t and C is a set of vertices of W_t each of which is adjacent to a vertex of the restriction of the dominating set to G_t. Thus, we have a table entry F(t,D,C) for every such pair C and D which records the size of a smallest dominating set of G_t-(W_t-C) whose intersection with W_t is D (where if no such dominating set exists then F(t,D,C) is infinity). We return the maximum of $F(r,D,W_t)$ over all subsets D of W_r. It remains to describe how to compute the F(t,D,C).

If t is a leaf then if every vertex of C-D is adjacent to a vertex of D then F(t,D,C)=|D| and otherwise F(t,D,C) is infinity.

If t has two children c(1) and c(2) then we perform the following computation:

F(t,D,C):=infinity
For each table entry (c_1,D_1,C_1) for c_1 and each table entry (c_2,D_2,C_2) for c_2
If $D_1 \cap W = D \cap W_{c(1)}$ AND $D_2 \cap W_t = D \cap W_{c(2)}$ AND C_1 contains all of $W_{c(1)}$-W_t
AND C_2 contains all of $W_{c(2)}$-W_t AND every vertex of C-D is either in $C_1 \cup C_2$ or is adjacent to a vertex in D AND $|D|+F(c_1,D_1,C_1)-|D \cap D_1|+F(c_2,D_2,C_2)-|D \cap D_2|<F(t,D,C)$
$F(t,D,C):= |D|+F(c_1,D_1,C_1)-|D \cap D_1|+F(c_2,D_2,C_2)-|D \cap D_2|$

As in the previous example, since T has O(|V(G)|) nodes and w is a fixed constant, we see that the total time taken by our procedure is O(|V(G)|).

5. K-DRP

The restriction of a solution to an instance (G,S,T) of k-DRP to G_t consists of a set of paths with their endpoints in $X_t=W_t \cup [(S \cup T) \cap V(G_t)]$. The intersection pattern corresponding to such a solution is a set Z of disjoint paths whose vertex sets form a partition of X_t. For each node t of T and every such set of paths $Z=\{Q_1,\ldots,Q_l\}$. we have a table entry F(t,Z) which is True if there is a set of disjoint paths $\{R_1,..,R_l\}$ of G_t such that the vertices of Q_i appear along R_i in the same order they appear along Q_i, and False otherwise. The answer to our problem is the OR of F(r,Z) over all those Z such that for each I between 1 and k, Z contains a path whose intersection with $S \cup T$ is $\{s_i,t_i\}$. HOW DO WE CONSTRUCT THESE TABLES?