Algorithms – handout 2

Dynamic Programming

October 28, 1999

A very useful tool for designing algorithms is dynamic programming. We shall illustrate the use of dynamic programming for solving *fixed parameter* versions of NP-complete optimization graph problems. We shall consider the following problems:

1. k-**bisection:** Does the input graph have a bisection (cutting the graph into two parts of equal number of vertices) with only k edges joining the two parts? Dynamic programming (easily) solves this problem in polynomial time for every fixed k.

2. k-**path:** Does the graph contain a simple path of length k? Augmenting dynamic programming by the method of *color coding*, this problem can be solved in polynomial time when $k = O(\log n)$.

3. k-**bandwidth:** Does the graph has a linear arrangement (numbering the vertices from 1 to n) in which the length of the longest edge is at most k? Dynamic programming (more difficult) solves this problem in polynomial time for every fixed k.

4. k-**clique:** Does the input graph contain a complete subgraph with k vertices? Appears to lack the structure that makes dynamic programming work.

5. k-**coloring:** Can the input graph be legally colored with k-colors? Remains NP-complete for $k \geq 3$.

Homework:

1. For a directed graph with edge lengths, min-TSP is the problem of finding a cyclic tour of minimum length that visits every vertex exactly once. There are $(n-1)!$ possible tours. Show that dynamic programming solves min-TSP in time $O(n^c 2^n)$ for some constant c.

2. Extend the algorithm for k-bandwidth so that it *counts* the number of linear arrangements of bandwidth at most k in the input graph. Assume here that G is connected.

Correction to last week's homework: already corrected in the web version of handout 1. (For extra credit: prove that the expected number of comparisons in the modified algorithm is smaller than that of randomized quicksort, even when taking into account the additional comparisons used in the selection of the splitting elements.)