17 Combinatorics

17.3 Hamiltonian cycles (5 units)

This project is based on the material found in the Part II Graph Theory course.

In this project you will need to be able to generate graphs from $\mathcal{G}(n,p)$, the space of graphs with n labelled vertices, edges appearing independently and at random with probability p.

A *Hamiltonian cycle* in a graph is a cycle which contains every vertex.

Question 1 Describe a simple algorithm to check whether a graph has a Hamiltonian cycle, and implement it. Test your program on a few particular graphs; and then use it on a selection of graphs from $\mathcal{G}(n,p)$ with n up to 21 and p firstly varying from 0.1 to 0.9 and then varying from $0.1 \ln n/n$ to $1.9 \ln n/n$. Tabulate your results, showing for each n and p the number of graphs from your selection which had a Hamiltonian cycle.

Question 2 Estimate the theoretical running time of your algorithm as best you can. Compare the answers for the worst case and an average case.

Question 3 Find a simple property possessed by many of your non-Hamiltonian examples that is sufficient (though maybe not necessary) to force a graph to be non-Hamiltonian. Why do you think the second range of values of p was chosen?

You will notice that your algorithm rapidly becomes prohibitively expensive as the order of the graph increases. In this case an "approximation algorithm" can be useful. An approximation algorithm for the Hamiltonian cycle problem would seek to make a very good attempt at finding a cycle in a short space of time. If it succeeds, well and good. If it fails, there may have been a cycle it missed, but it is hoped that the probability of this will be small.

Here is a simple algorithm to search for a Hamiltonian cycle. Construct a sequence of paths $P_1, P_2, \ldots$, where P_1 is just a single vertex v_0. Given a path P_j from v_0 to v_k, proceed as follows:

1. If P_j has length $n-1$ and $v_0v_k \in E(G)$, output a Hamiltonian cycle;
2. if P_j has length less than $n-1$ and v_k is joined to a vertex not in P_j, extend the path P_j to a path P_{j+1}. If there are several neighbours not in P_j, pick one of them at random;
3. otherwise construct a new path of the same length as P_j in this way: select a neighbour v_i of v_k in P_j at random. Then P_{j+1} is the path $v_0 \ldots v_{i-1}v_iv_kv_{k-1}\ldots v_{i+1}$.

Question 4 Implement this algorithm, and try it on your earlier examples. You should set a stopping time $T$for the procedure so that, if it has constructed P_T and still found no cycle, it quits. What functions work well in practice (i.e. fairly reliably find a cycle but aren't too expensive)?

Question 5 In general, the stopping time $T = T(n,p)$ should be a function of both n and p. In the case that p is fixed and n is large, what do you think would be a good choice of T? How do you think the running time might vary with p.

References

[1] Bollobas, B., Modern Graph Theory, Springer 1998.

