Advanced Algorithms — Problem set 7

Whenever you give an algorithm, also argue for its running time and correctness. Always feel free to ask in the forum or send a mail if you're stuck, need a clarification, or suspect a typo.

1. Inclusion-exclusion algorithms:

   (a) Implement the inclusion-exclusion algorithm for counting perfect matchings and compare it against the formula from statistical physics on grids. Check that your algorithm gives the correct output for all grids it can handle. (Mostly already done in class.)

   (b) Same as before, but use the algorithm for bipartite graphs (also known as Ryser's formula). Note that square grids are bipartite. How much larger can the grids be while still giving an acceptable running time?

   (c) Implement the inclusion-exclusion based algorithm for Hamiltonian cycles from the lecture and compare its running time against the DP algorithm: What is the maximum number of vertices that can be handled with each of the algorithms?

   (d) Show (theoretically) how to extend the inclusion-exclusion based algorithm for Hamiltonian cycles to the TSP problem with polynomially bounded integer edge-weights. (Hint: Introduce a variable x and encode an edge-weight W as x^W. Compute a sum over all Hamiltonian cycles, each weighted by the product of edge-weights occurring on it. To this end, rather than counting walks, sum over the walks such that each walk $(v_1, \ldots, v_k, v_1)$ is weighted by the product of the edge-weights occurring in it. Show that this sum can also be computed via matrix multiplication.)

2. Describe reductions from the following problems to SAT. Recall that it may help to implement an "exactly-one" clause, as discussed in class.

   (a) Vertex cover (input: graph G, number k).

   (b) k-coloring (input: graph G, number k)

   (c) Hamiltonian cycle (input: graph G)

   Consider the JavaScript implementation[1] of the SAT solver *MiniSAT*. Implement the reduction for Hamiltonian cycles as a computer program: Given as input the adjacency matrix of a graph G, your program should output a MiniSAT input file (see specification in link) for a CNF φ such that G has a Hamiltonian cycle iff φ is satisfiable. You may also implement a "reader" that converts the output of MiniSAT back to a Hamiltonian cycle. Check your program on grid graphs.

3. Implement the algorithm for finding a maximum-weight perfect matching in a graph G based on finding cycles in the LP solution (described in Lecture 20) and test it on graphs of your choice.

4. We have encountered the k-clique problem before: The input is a graph G and $k \in \mathbb{N}$, and the output is whether G contains a k-clique. We have seen that this problem can be solved in $O(n^{\omega k/3})$ time, where $\omega < 3$ is the exponent for matrix multiplication. Show the following: If, for every $\epsilon > 0$, there is an $O(n^{\epsilon k})$ time algorithm for the k-clique problem, then for every $\epsilon' > 0$, there is an $O(2^{\epsilon' n})$ time algorithm for the 3-coloring problem. (Such an algorithm would refute the exponential-time hypothesis.) To solve this exercise, use the split-and-list technique to transform an input H to the 3-coloring problem into an input G for the k-clique problem. Vertices of G should correspond to partial colorings of H and edges of G should encode compatibility of colorings.

[1] https://jgalenson.github.io/research.js/demos/minisat.html