Winter term 2014/15
Professor Dr. Stephan Held
Jannik Silvanus

Research Institute for Discrete Mathematics
University of Bonn

# Combinatorial Optimization

## Exercise Sheet 8

**Exercise 8.1:** Given an undirected graph, an *odd cycle cover* is defined to be a subset of the edges containing at least one edge of each odd circuit. Show how to find in polynomial time a minimum weight odd cycle cover in a planar graph with nonnegative weights on the edges. Can you also solve the problem for general weights?

(4 Points)

**Exercise 8.2:** Let $G$ be a planar 2-connected graph with a fixed embedding, let $C$ be the circuit bounding the outer face, and let $T$ be an even cardinality subset of $V(C)$. Prove that the minimum cardinality of a $T$-join equals the maximum number of pairwise edge-disjoint $T$-cuts.
Hint: Color the edges of $C$ red and blue such that, when traversing $C$, colors change precisely at the vertices of $T$. Consider the planar dual graph, split the vertex representing the outer face into a red and a blue vertex, and apply Menger's Theorem.

(4 Points)

**Exercise 8.3:** Let $G$ be an undirected graph and $T \subseteq V(G)$ with $|T| = 2k$ even. Prove that the minimum cardinality of a $T$-cut in $G$ equals the maximum of $\min_{i=1}^{k} \lambda_{s_i,t_i}$ over all pairings $T = \{s_1, t_1, \dots, s_k, t_k\}$, where $\lambda_{s,t}$ denotes the maximum number of pairwise edge-disjoint $s$-$t$-paths.
Hint: Use the polynomial time algorithm for computing minimum capacity $T$-cuts.

(4 Points)

**Deadline:** Tuesday, December 2, 2014, before the lecture.
**Information:** Submissions by groups of one or two students are allowed.

**Programming Exercise 2:**

Implement the Minimum Mean Cycle Algorithm from exercise 7.4.

*Program Specification:* Your program must accept a filename as a command-line parameter (i.e. it must be called with `myprogram input.dmx`). The command-line parameter contains the filename of the file that encodes the graph.

*Input:* The input file is a DIMACS file that encodes a weighted undirected graph. That means, one line has the format p edge $n$ $m$

where $n$ is the number of vertices of the graph and $m$ is the number of edges. After this line, $m$ lines have the format

    e $i$ $j$ $c$

where $i$ and $j$ are the indices of the vertices connected by this edge and $c$ is the weight of the edge. The vertices are indexed from 1 to $n$. Lines starting with a `c` are comments and should be ignored. For a more complete definition of the DIMACS format, see `http://www.or.uni-bonn.de/lectures/ss12/praktikum/ccformat.pdf`.

*Output:* Your program must write the mean weight of a minimum mean cycle to the standard output, followed by a minimum mean cycle (in the DIMACS format). More precisely, the first line of the output must consist of exactly one integer, which has to be the cost of an optimum solution. The rest of the output, starting from the second line, must encode the minimum mean cycle as a graph in DIMACS format, including the first line of the format `p edge` $n$ $m$.

*Perfect Matchings:* To compute minimum-weight perfect matchings, you may use existing graph libraries like Lemon (`http://lemon.cs.elte.hu/trac/lemon`) or Blossom V (`http://pub.ist.ac.at/~vnk/software.html`). We provide source code for an example program that reads a DIMACS graph and computes a minimum-weight perfect matching with Blossom V at `http://www.or.uni-bonn.de/lectures/ws14/co_uebung_ws14.html`. Note that Blossom V crashes when the input graph does not have a perfect matching. However, you may also use your own implementation of a MWPM algorithm.

*Instances:* Test instances are provided together with the example code.

*Programming Languages:* Your program must be written in C or C++ and compile with a GNU compiler on a current Linux machine.

*Criteria:* The following criteria are relevant for the number of points you will be awarded: Correctness, speed, code documentation, number of compiler warnings, overall elegance, standard compliance. Note that we use the `-Wall`, `-Wextra` and `-pedantic` compiler flags and use Valgrind to check your program for memory corruptions.

*Submission:* Send your program to `silvanus@or.uni-bonn.de`. Make sure to attach a makefile or at least a shell script compiling your program into a binary.

(20 Points)

**Information:** Note that participation criteria require *both* at least 50% of the achievable points in the theoretical exercises as well as 50% of the achievable points in the programming exercises.
**Deadline:** Tuesday, December 23, 2014, 2 pm.