Moydenskall (Lloyd's k-means)

Project Moydenskall

This project will solve instances of the facility location problem as stated in http://www.or.uni-bonn.de/~held/praktikum/facility/aufgabe.pdf This will include the following approaches:
1. enumeration algorithm for finding perfect solutions
2. Lloyd's k-means algorithm (approximative solutions)
3. diverse approaches to find starting seeds for Lloyd's algorithm

Current status:
1. finished
2. not yet started
3. not yet started

Compile hints

developed and tested with Microsoft Visual C++ Compiler
tested with g++ and clang++3.5 (Ubuntu 14.04)

compile command for linux systems:
`g++ ./*.cpp ./*.hpp -o moydenskall -O3 -std=c++11`

`clang++-3.5 ./*.cpp ./*.hpp -std=c++11 -O3`

Execution and command line parameter

sample execution for both linux and windows systems:
`./moydenskall instance.tsp -f 50 -u 5`

Synopsis

`./programname <filename> [options]` filename is an instance in the tsplib format

Options

- `-f <int>` double indicating fixed site costs (optional parameter, default: 0)
- `-u <int>` facility capacity, max number of customers per facility (optional parameter, default: 0)
- `-time <bool>` turn time measurement on and off (optional parameter, default: false)
- `-svg <bool>` create svg visualization in result.svg (optional parameter, default: false)

Code Analysis

- Clang analyzer states some warnings, which is a bug in clang (see: https://llvm.org/bugs/show_bug.cgi?id=16686)
- Valgrind: `All heap blocks were freed -- no leaks are possible`

Source Overview

- `main.cpp` provides command line argument interpretation, runtime measurement, and runs the show
- `Point.cpp` representation of points (x,y coordinates, ID)
- `Enumerator.cpp` implements a class which organizes the enumerating process
- `Tools.cpp` collection of useful functions, see below

Tools.cpp

- `Plane readfile(std::string filename);` read a file in tsp format
- `Point centroid(const Plane&);` get centroid of points

- `Plane centroid(const std::vector<Plane>&);` get all centroids for a given partition
- `double eucl2dist(Plane, Point);` sum euclidean square distances from a site to all customers
- `double evaluate_partition(std::vector<Plane>, Plane, double);` get costs for a given partition and sites and fix_costs

Enumerator.cpp

- `void create_partition(std::vector<Plane>& partition, Plane& left)` recursive approach to create all possible partitions
- `void print_result(bool svg)` print best partition to cout in required format (and toggle svg output if svg is true)
- `void svg_output()` best partition visualized in svg (creates a .svg file)