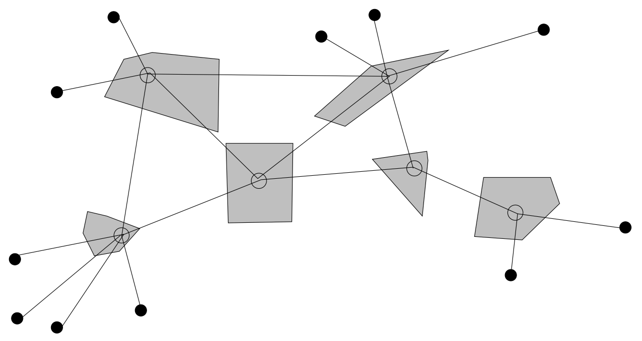

Figure 1: The network and the allowable regions for the hubs. Solid circles are airports, hollow circles are hubs.

Description

The goal of the project is to develop a tool to optimize placement of "hubs" — intersections of the "highways in the sky". Air planes, after a takeoff, follow a network of air lanes. Given the volume of air traffic, even a slightest improvement in the locations of the network nodes, even on a local scale, will lead to enormous savings.

Input Specification

Formally, the input to the problem is specified by a graph $G = (V, E)$ representing the network (Fig. 1). The vertices V of the graph are partitioned into two sets:

1. *Static* vertices, $\{a_1 \dots a_A\}$, representing the airports, and
2. *Hubs*, $\{h_1, \dots, h_H\}$, representing the intersections.

For every pair a_k, a_l of the airports we are given a number, denoted f_{kl}, of flights scheduled to fly between a_k and a_l. The flights follow given paths π_{kl} in G.

Format The network is stored either as a $|V|$-by-$|V|$ adjacency matrix, or as an $|V|$-by-$|E|$ incidence matrix. Each vertex has a point in the plane, and is represented with its coordinates. The demands are just an array of integers. Each path is a sequence of vertices in G.

Solution Approach

The optimization is done via Second-Order Conic Programming (SOCP), which is an extension of Linear Programming (LP). We will have a decision variable $p_i = (p_i^x, p_i^y)$ for each vertex $i \in V$; the variable shows where the ith vertex of G is placed. We will have an auxialry variable t_{ij}, for every edge $(i, j) \in E$. The SOCP will be:

$$\begin{array}{rl} \text{minimize} & \sum w_{ij} t_{ij} \\ \text{subject to:} & \\ p_k = a_k & \text{for every airport } a_k \in A \\ ||p_i - p_j|| \le t_{ij} & \text{for every edge } (i, j) \in E \end{array}$$

The weight w_{ij} of an edge (i, j) is the number of times the edge is used by the paths π_{kl}.

Stages of the Development

Stage I: **Input Visualization.** Draw the network. This should be done quickly.

Stage II: **Choose the SOCP solver.** For instance, `http://www.stanford.edu/~boyd/old_software/SOCP.html` , SeDuMi (for MATLAB) `http://sedumi.mcmaster.ca/` , or NEOS server `http://www-neos.mcs.anl.gov/`.

Stage III: **Output.** Feed the input into the SOCP solver, get the output, and visualize it.

Allowable Regions

If the project runs fast, at the next stage of the development, we may take into account that the locations for the hubs are not completely arbitrary — they need to be supervised by human air traffic controllers (ATC). The developed tool must allow ATC to specify an *allowable region*, R_i, for every hub h_i (see Fig. 1). This is done through the following stages:

Stage I: **Recognizing Vertices by Clicks.** Create interface to recognize which vertex of the drawn network the user has clicked on.

Stage II: **Specifying Allowable Regions.** Create interface for the user to click at the locations of the vertices of an allowable region. Check that the region is convex. In the final tool this should interact with the previous stage as follows: the user clicks on a hub, then specifies the region for that hub.

Stage III: **Adding Linear Constraints to SOCP.** The description will be provided on demand.

Possible Extensions

Later, a user may be given a possibility to input his own set of airports, paths between them and demands, and to define the network. It would be convenient for the user to also be able to mark some nodes of the network as static (not only the airports, but also some hubs); this is equivalent to specifying an allowable region equal to the point, at which the node is situated. For demonstration purposes a set of examples may be created, each including a network, demands and paths, and a set of allowable regions. The next stage would be to add a realistic feel to the tool: use a map in the background, base examples on real airports locations and the existing highways, include weather and no-fly zones, etc.