# 15-451 Homework 6B

Arthur O'Dwyer
K. Elliott Fleming

April 19, 2005

## 3. Finding a separating hyperplane.

*Suppose we are given $n$ green points $G = (P_i \mid i \in 1..n)$ and $m$ red points $R = (Q_i \mid i \in 1..m)$ in $d$ dimensions. Show how to determine a separating hyperplane if one exists, using linear programming.*

We have $mn$ linear constraints, of the form "hyperplane $H$ separates points $P_i$ and $Q_j$" for each $P_i \in G$ and $Q_j \in R$. If we represent the hyperplane as three $d$-dimensional vectors $X$, $Y$, $Z$, then our linear programming constraints become

$$a_{ij}X + b_{ij}Y + c_{ij}Z = x_{ij}P_i + (1 - x_{ij})Q_j$$
$$0 \leq x_{ij} \leq 1$$

Thus we have an LP problem in $4mn$ variables and $4mn$ simple constraints. Solve the system. If it is feasible, the resulting $X$, $Y$, $Z$ will define the hyperplane we're looking for.

## 4. 3D Linear Programming.

*Give a randomized expected-linear-time algorithm for solving three-dimensional linear programming problems.*

This is a direct generalization of the 2D case. The bounding box gives us an initial "feasible" polytope in three dimensions (a polyhedron). We pick the $i$th constraint at random. It corresponds to a plane cutting across the current feasible polytope, intersecting up to $i - 1$ previous constraint planes in up to $i - 1$ lines. We project the objective function onto the $i$th plane, and that gets us a two-dimensional LP problem: maximize the objective function over the polygon bounded by the up to $i - 1$ intersection lines. We solve that problem in linear time by the previously given algorithm. Repeat for $i \in 1..n$.

The running time is expected to be $O(n)$ for the same reasons that the 2D case's running time is expected to be $O(n)$: backwards analysis. Each $i$th constraint we remove, working backwards, has a $3/i$ chance of actually affecting the current solution. Affecting the current solution means doing $O(i)$ work to solve a 2D LP problem with $i$ constraints. Not affecting the current solution means doing some constant work, $c$.

So the expected running time of this algorithm is $\sum_{i \in 1..n} O(i) \cdot \frac{3}{i} + c = \sum_{i \in 1..n} O(1) = O(n)$. *Q.E.D.*

# 5. Perfect matching.

*Let $G = (V, E)$ be an undirected graph on $n$ vertices. A* perfect matching *of $G$ is a subset $E' \subset E$ such that each vertex is the endpoint of exactly one edge of $E'$.*

*Consider the following expression of perfect matching as an LP problem. For each edge $e \in E$ we have a variable $x_e$. The constraints are:*

$$0 \leq x_e \leq 1 \quad \textit{for each } e \in E$$
$$\sum_{e \in E | v \in \varphi(e)} x_e = 1 \quad \textit{for each } v \in V$$

**(a.)** *Show that if $G$ is bipartite* and *the system above is feasible then there is a perfect matching in the graph $G$.*

This is a direct application of Hall's Marriage Theorem. We choose a bipartition $(X, Y)$ of $G$. Let the subset $S \subseteq X$ be given, and define $E(S)$ the set of edges with one endpoint in $S$. Then $|N(S)| \geq |S|$, because $\sum_{e \in E(S)} x_e = |S|$ by the constraint that each vertex's adjoining edge weights must sum to 1. Note that $E(S) \subseteq E(N(S))$. Then $\sum_{e \in E(S)} x_e \leq \sum_{e \in E(N(S))} x_e = |N(S)|$, and we have the result we wanted.

**(b.)** *Give an example of a graph with no perfect matching, but a feasible edge-weighting.*

Take $G = K_3$, with each of the three edges weighted 1/2.