CSE 521: Design and Analysis of Algorithms I **Spring 2016**

Lecture 15: Introduction to Linear Programming

Lecturer: Shayan Oveis Gharan *May 16th* *Scribe: Daniel Dueri*

Linear systems are easiest class of optimization problems. Roughly speaking, any linear system of equations can be solved efficiently. The easiest examples are linear systems of equalities. This systems can be solved by matrix inversion. Given a (full rank) matrix A, to solve $Ax = b$, it is enough to compute $A^{-1}b$. In linear programming we study system of linear inequalities.

In the first few lectures we study linear systems of inequalities, also known as Linear Programming (LP). We also allow for a linear cost (or objective) function. LPs can be used to solve problems in a wide range of disciplines from engineering to nutrition and the stock market, At some point it was estimated that half of all computational tasks in the world correspond to solving linear programs.

A general LP can be formalized as follows:

$$\begin{aligned} &\min_x \langle c, x\rangle \\ &\text{s.t. } Ax \leq b, \end{aligned} \tag{15.1}$$

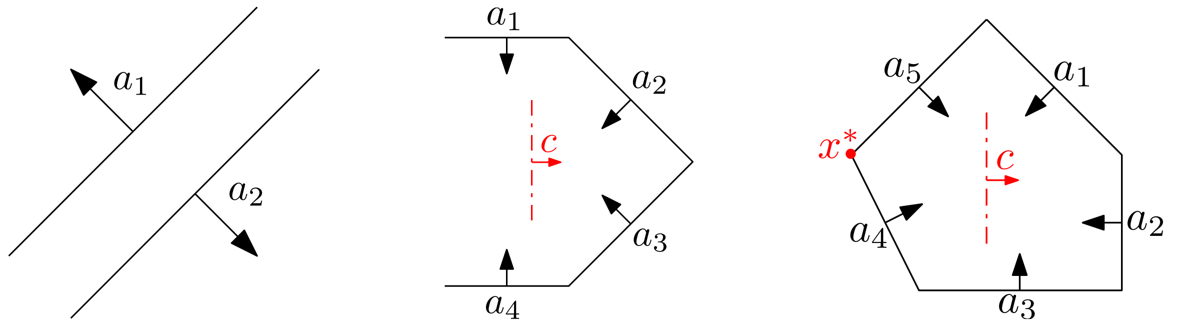
where $x \in \mathbb{R}^n$ is represents the variables, $c \in \mathbb{R}^n$ defines the objective function, and $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ define the constraints. The above form is fairly general; one can model various types of constraints in this form. For example, a constraint $\langle a_1, x\rangle \geq b_1$ can be written as $\langle -a_1, x\rangle \leq -b_1$. Or, a constraint $\langle a_1, x\rangle = b_1$ can be written as $\langle a_1, x\rangle \leq b_1$ and $\langle -a_1, x\rangle \leq -b_1$.

The objective function can be viewed as a hyperplane in $\mathbb{R}^n$ with normal vector c. Further, one can express the constraint matrix A as a series of row vectors:

$$A = \begin{bmatrix} a_1^{\mathrm{T}} \\ a_2^{\mathrm{T}} \\ \vdots \\ a_m^{\mathrm{T}} \end{bmatrix}.$$

When viewed from this perspective, it is easy to see the constraint set $a_i^{\mathrm{T}} x \leq b_i,\ i \in \{1, 2, \ldots, m\}$ as the intersection of finitely many half-spaces. The feasible set an LP is called a polyhedron. Thus, the solution will always be one of the following three cases shown in Figure 15.1.

There are many well known ways of solving LPs, but two of the most well known and used are Interior Point Methods (IPMs) and the simplex method. The simplex method is based on a very geometrically intuitive idea: start at a vertex of the domain of x and move to an adjacent vertex with lower objective - continue until an optimal solution is reached. Theoretically, the worst case time complexity of the simplex method is exponential; however, the simplex method is very fast in practice, and toolboxes such as CPLEX and Gurobi can solve very large LPs (millions of solution variables) in a few seconds. Interior point methods have better theoretical bounds; roughly speaking, using an interior point method, one can solve a general linear program with n variables by solving $\tilde{O}(\sqrt{n})$ many systems of linear equalities.



(a) Infeasible domain of x: No solution.

(b) Unbounded domain of x: $x^* = \pm\infty$

(c) Closed, bounded and compact domain of x: at least one vertex of the domain of x is in the solution set x^*.

Figure 15.1: The possible domains of the solution variable x for an LP, along with the solution.

15.1 Linear Modeling

In the rest of this lecture we study several applications of linear programming in the real world. In many of the real world applications one models a given problem by a linear system and the uses linear programming solvers to find an optimal solution of the linear system. The optimal solution can be seen as an approximate solution of the problem at hand.

15.1.1 Example 1: Diet Problem

Consider the problem of finding an optimal combination of meat, vegetables, fruits, and dairy products that has less than 1,500 calories and costs less than \$20 a day subject to maximizing our daily happiness. Let c_m, c_v, c_f, c_d be the calories per gram of each food group, p_m, p_v, p_f, p_d be the price per gram of each food group in cents, and h_p, h_v, h_f, h_d be the happiness derived from eating a gram of each food group. Assume that any happiness derived from eating a specific food group is completely independent of other food groups, that is, the happiness derived from one group does not affect the happiness derived from any other food group. Then, the following optimization can be solved to obtain the optimal meal combination that maximizes our daily happiness:

$$\begin{aligned} \max \quad & h_m x_m + h_v x_v + h_f x_f + h_d x_d \\ \text{s.t.} \quad & c_m x_m + c_v x_v + c_f x_f + c_d x_d \leq 1500, \\ & p_m x_m + p_v x_v + p_f x_f + p_d x_d \leq 2000 \\ & x_m, x_v, x_f, x_d \geq 0. \end{aligned} \tag{15.2}$$

where x_m, x_v, x_f, and x_d are the quantities of each food group in grams that we need to consume in a day.

15.1.2 Example 2: Support Vector Machines

Suppose one has images $x^i \in \mathbb{R}^n$ and wishes to separate those which contain a car from those without a car. One possible solution is to find a separating hyperplane such that all images with a car lie on one side of the hyperplane and those without a car lie on the other side of the hyperplane. One can find such a hyperplane (if it exists) by solving an LP. Let $v \in \mathbb{R}^n$ be the unknown normal vector corresponding to the hyperplane, then the following LP finds the minimum error separating hyperplane:

$$\begin{aligned} &\min_{\epsilon} \sum_i |\epsilon_i| \\ &\text{s.t. } \langle v, x^i \rangle \geq \frac{1}{2} + \epsilon_i, \ \forall\ i \text{ w/ a car}, \\ &\qquad \langle v, x^i \rangle \leq -\frac{1}{2} + \epsilon_i, \ \forall\ i \text{ w/o a car}. \end{aligned} \tag{15.3}$$

Note that the objective function is not a linear function, but one can use linear programming to minimize the objective. See the first problem of PS6.

15.2 Decision Theory

In the field of decision theory, an agent is faced with choosing an action (or a sequences of actions) aimed at optimizing the final reward. In many cases, the reward is an output of a stochastic process; so an agent tries to maximize the expected from his actions. A general model for modeling decisions in a stochastic enviroment is a Markov Decision Process (MDP).

MDPs are one of the basic tools in many areas of computer science including artificial intelligence and robotics. Let us start by a motivating example: Consider a cleaning robot; at each time step the robot is located at a specific position in the room and it has a set of possible actions. It can move to a neighboring location, it can clean the current spot, it can test if the current spot is clean, or it can do nothing. Each of these actions has a reward (or a cost). So, the goal is to choose a sequence of actions to maximize the expected reward.

Definition 15.1. *A Markov Decision Process can be seen as a generalization of a Markov Chain where in each state an action is taken, and the next state is chosen random as a function of the chosen action. There is In particular, if we are at state i, and we take action a, we move to state j with probability $P_a(i,j)$. In such a case the agent immediately receives a reward $R_a(i,j)$*

Most importantly, this process is memory-less, i.e., similar to Markov Chain, the evolution of the system is only a function of the current state, and it has nothing to do with the past.

The objective of an MDP is to design a policy that for each state i and any time t gives an action $a_t(i)$ such that the expected reward over a given time finite horizon T is maximized. We use Dynamic programming to find the optimal policy corresponding to a given MDP with finite horizon T.

For an integer $0 \leq t \leq T$, let $V_t(i)$ denote the expected reward of the optimal policy if it starts at time t from state i. Note that $V_T(i) = 0$ for every state i, since no further action can be taken, so no more rewards can be received. For any state i and time $0 \leq t \leq T$, $V_t(i)$ can be described as:

$$V_t(i) = \max_a \left(\sum_j P_a(i,j) \Big(R_a(i,j) + V_{t+1}(j) \Big) \right). \tag{15.4}$$

The latter is known as the Bellman operator. The Bellman operator F receives a vector V and returns a vector $F(V)$ where for each i,

$$F(V)_i = \max_a \left(\sum_j P_a(i,j) \Big(R_a(i,j) + V_{t+1}(j) \Big) \right).$$

To obtain $V_0(i)$ all we need to do is to apply the Bellman operator T times to the all zeros vector:

$$V_0 = F(V_1) = F(F(V_2)) = \cdots = F^T(\mathbf{0}).$$

The optimal policy can be determined for each state and time by choosing the action corresponding to the maximum value. In particular,

$$a_t(i) = \text{argmax}_a \left(\sum_j P_a(i,j)\Big(R_a(i,j) + V_{t+1}(j)\Big)\right)$$

Observe that the running time of the above algorithm is $O(T \cdot n^2)$ because it takes $O(n^2)$ to compute $F(V)$ for a given vector V. Unfortunately, as T gets larger the running time of the algorithm gets larger and larger, in particular, we cannot run the above algorithm for $T = \infty$ which is a specially important case. To motivate this case recall that the cleaning robot indeed never turns of.

To study an MDP over infinite horizon first we need to make sure that the sum of the rewards are bounded; otherwise any arbitrary policy would have an infinite reward. To model that we use a discount factor $0 < \gamma < 1$. We assume that the value of reward at time t is scaled by γ^t. This ensures that the expectation of the reward is bounded even as $t \to \infty$. For example, you can think of γ as the infliation rate if you are modeling the stock market.

With this notation, our goal is to find the optimal policy with the largest infinite horizon discounted reward. As we will see, the optimal policy is time-homogeneous, i.e., assigns a fixed action $a(i)$ to any state i and $a(i)$ is independent of t.

Now, we can use linear programming to find the optimal policy.

$$\begin{aligned} &\min \sum_i V(i) \\ &\text{s.t. } V(i) \geq \max_a \left(\sum_j P_a(i,j)\Big(R_a(i,j) + \gamma V(j)\Big)\right), \ \forall\, i. \end{aligned} \tag{15.5}$$

Theorem 15.2. *The optimal expected reward V^* corresponds the optimal solution of* (15.5).

Proof. Let $F_\gamma(V)$ be the Bellman operator corresponding to the MDP with discounted reward γ. In particular, for any i,

$$F(V)_i = \max_a \left(\sum_j P_a(i,j)\Big(R_a(i,j) + \gamma V(j)\Big)\right).$$

Note that since we multiply $V(j)$ with γ any reward received after the transition to j will be scaled by γ at the current time step. So, the above operator indeed computes the discounted reward

To prove the theorem we use the following two facts:

Fact 1. $V^* \geq F_\gamma(V^*)$, i.e. V^* is feasible of the LP. This is because for any state i, we must have$V^*(i) = F_\gamma(V^*)_i$; otherwise, we can choose a better action at state i to increase V^*. But that means that V^* is not the optimal reward which is a contradiction.

Fact 2. For any vectors U, V where $U \geq V$, we have $F_\gamma(U) \geq F_\gamma(V)$. This follows from the observation that $F_\gamma(V)$ is a monotone function.

Let V be the optimal solution of the LP. Since V is feasible, we have $V \geq F_\gamma(V)$. Using Fact 2, we can write

$$F_\gamma(V) \geq F_\gamma(F_\gamma(V)).$$

Continuing this we have

$$V \geq F_\gamma(V) \geq F_\gamma(F_\gamma(V)) \cdots \geq F_\gamma^\infty(V) = V^*.$$

So, V^* must be the optimal solution to (15.5). $\square$

We emphasize that the above linear program is not the fastest way that we know to solve a given MDP. There are iterative methods that outperform linear programming. However, the reason that we focus on linear programming on these lectures is that for many optimization tasks linear programming can be used as a central tool to find an optimal or an approximate solution. Since very fast linear programming solvers are available in the world, one can always use this tool as a first order approximation to a sophisticated optimization problem. If the solvers find a good solution, we can try to optimize our approach using more sophisticated techniques.