Lecture 18: Linear Programming

Scriber: Xiaoming Liu

April 2020

1 Complimentary Slackness

If $a^T x \geq b$ is a primal constraint, and y is the corresponding dual variable, then for any pair of optimal solutions, $y(a^T x - b) = 0$.

1. If $a^T x > b$ (primal constrait is not tight), then $y = 0$. (dual variable is 0)
2. If $y > 0$ (dual variable is positive), then $a^T x = b$. (primal constraint is tight)

2 LP Algorithms

Given an LP, how do we find its optimal solution? Many algorithms (fairly complex and people end up using available packages/solvers):

1. Simplex
2. Ellipsoid
3. Interior point

2.1 Simplex Algorithm

Let us revisit the geometric interpretation of LPs with two variables. Here, each constraint can be thought of as a half-space, and the set of feasible solutions is the intersection of all these half-spaces. The objective function works as the direction of gravity. The optimal solution then corresponds to the lowest point in this convex polygon/polytope if we were to rotate the feasible region so that the direction given by the objective, i.e., gravity, points vertically downwards.

Basic Feasible Solution:
A basic feasible solution of a linear program with n variables is a feasible solution equal to the solution of a system of where each equation is a tight constraint. In other words, for a linear program with n variables and m constraints(m¿n), a basic feasible solution is a feasible solution where n out of m constraints are set to equalities. From a geometric perspective, a basic feasible solution is a vertex of the feasible solution.

Idea: Start with a basic feasible solution and follow an edge in the polytope.

Algorithmically: Following an edge is equivalent to swapping a constraint. Many ways to decide which constraint to swap.

Running Time Each move takes polynomial time. How many times do we need to move to get to an optimal solution? In the worst-case with n variables and m constraints, it can take 2n moves to reach a global optimal solution. In practice, the algorithm is very fast. It is one of the most popular algorithms in the packages solving LPs. Explanation/theoretical reasons for this are beyond the scope of the class.

2.2 Ellipsoid Algorithm

Seperation Oracle: Given a candidate solution (an assignment of x), decide if x is feasible or output a constraint which x violates.
Idea: We can add a constraint that specifies a bound on the value that the objective can take. If there are still feasible solutions, we can move this bound further in the direction of gravity until there are no more feasible solution to the systems of linear equations with this new, added constraint. We can perform a binary search in this range of bounds and try to identify the point where the intersection of the (new) half-space and the original set of feasible solutions is exactly one point.

How do we find a feasible solution?
Algorithmically: The algorithm starts with a big ball that contains the entire feasible region inside it. We look at the center of this ball and see if it is a feasible solution. If not, then we can identify the violated con- straint. We know that the actual feasible solution must then lie in the intersection of the half-space specified by the violated constraint and the previous ellipsoid. The algorithm then finds a new ellipsoid that contains this entire intersection. Repeat!
The size/volume of the ellipsoid decreases by at least half in each step. Eventually we either find a feasible solution or the ellipsoid becomes so small that we can conclude that there are no feasible solutions.
Running time: This is faster than the Simplex algorithm. The running time is polynomial in the number of variables and constraints. Actual running time is slow, so this algorithm is often used for low dimensional problems.

2.3 Interior Point Algorithms

Idea: We try to smooth out the very hard threshold for constraints. Build barrier functions for constraints. These are functions that are small for the feasible part of the constraint, but go to infinity at the boundary (for a minimization problem).
Algorithmically: First find the optimal solution for the objective plus all the barrier functions. This will result in a point that cannot be on the boundary on the feasible region by definition of barrier functions. This point will lie in the interior of the feasible region.
Next, gradually decrease the barrier functions to make the point closer to the actual

optimal. Thus, the in- fluence of the objective becomes larger while that of the barrier functions goes down. We skip details here.

Running time: It is polynomial in the number of variables and constraints, and can be implemented very efficiently in practice.