CSE 6311 : Advanced Computational Models and Algorithms Mar 22, 2011

Linear and Integer Programming

Professor: Dr.Gautam Das *Lecture by: Saravanan*

Introduction

Linear and Integer programming play an important role in algorithm design. Lot of combinatorial optimization problems can be formulated as Integer Programming problems. But as we will see later, Integer Programming is NP-Complete. One potential solution to solve the Integer Programming problems in polynomial time using Linear Programming with the technique of LP relaxation. Thus, Linear Programming and the concepts of rounding and duality are very useful tools in the design of approximation algorithms for many NP-Complete problems. In this lecture, we will discuss the theory of Linear and Integer Programming

Background

Optimization

In an optimization problem, we are given a real valued function f whose domain is some set A and we want to find the parameter which maximizes or minimizes the value of f. This function is also called as *objective* function. An optimization problem can either be a maximization or a minimization problem. In a maximization (minimization) problem, the aim is to find the parameter which maximizes (minimizes) the value of f. In this lecture, we will primarily discuss about maximization problems because maximizing $f(x)$ is same as minimizing $-f(x)$.

Optimization problems can be broadly classified into two types : Unconstrained and constrained optimization problems. In an unconstrained optimization problem , we are given a function $f(x)$ and our aim is to find the value of x which maximizes/minimizes $f(x)$. There are no constraints on the parameter x. In constrained optimization, we are given a function $f(x)$ which we want to maximize/minimize subject to the constraints $g(x)$. We will focus more on constrained optimization because most real world problems can be formulated as constrained optimization problems.

The value x that optimizes $f(x)$ is called the *optima* of the function. So, a *maxima* maximizes the function while the minima *minimizes* the function. A function can potentially

have many optima. A *local* optimum x of a function is a value that is optimal within the neighborhood of x. Alternatively, a *global* optimum is the optimal value for the entire domain of the function. In general, the problem of finding the *global optima* is intractable. However, there are certain subclasses of optimization problems which are tractable.

Convex Optimization

One such special subclass is that convex optimization where there are lot of good algorithms exist to find the optima efficiently. Linear programming happens to be a special case of convex optimization.

Convex sets
A set C is *convex* if for $x, y \in C$ and any $\alpha \in [0, 1]$,

$$\alpha x + (1 - \alpha)y \in C$$

Informally, in a geometric sense, for any two points x,y in a convex region C, all the points in the line joining x and y are also in C. For eg the set of all real numbers , $\mathbb{R}^n$ is a convex set. One of the interesting properties of such sets is that the intersection of two convex sets is again a convex set. We will exploit this feature when discussing the geometric interpretation of Linear Programming.

Convex Function
A real valued function $f(x)$ is convex if for any two points $x, y \in domain(f)$, and $\alpha \in [0, 1]$
,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

Linear/affine functions are, for example, convex. One very neat property of convex functions is that any local optimum is also a global optimum. Additionally, the maximum of a convex function on a compact convex set is attained only on the boundary . We will be exploiting these properties when discussing algorithms that solve linear programming.

Linear Programming

Linear Programming is one of the simplest examples of constrained optimization where both the objective function and constraints are linear. Our aim is to find an assignment that maximizes or minimizes the objective while also satisfying the constraints. Typically, the constraints are given in the form of inequalities. For eg, the standard (or canonical) form of a maximization linear programming instance looks like :

$$
\begin{array}{ll}
\text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{subject to} & \\
& a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\
& \ldots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\
& \forall i, x_i \geq 0
\end{array}
$$

A minimization linear programming instance looks like :

$$
\begin{array}{ll}
\text{minimize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{subject to} & \\
& a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \\
& \ldots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq b_m \\
& \forall i, x_i \geq 0
\end{array}
$$

Any assignment of values for variables $x_1 \ldots x_n$ which also satisfies the constraints is called as a *feasible solution*. The challenge in Linear Programming is to select a feasible solution that maximizes the objective function. Of course, real world linear programming instances might have different inequalities - for eg $\geq$ instead of $\leq$ or the value of x_i is bound between $[p_i, q_i]$. But in almost all such cases, we can convert that linear programming instance to the standard form in a finite time. For the rest of the lecture, we will focus on linear programming problems in standard form as this simplifies the exposition.

The standard form of linear programming also enables us to concisely represent the linear programming instance using linear algebra. For eg, the maximization problem can be written as ,

$$
\begin{array}{ll}
\text{maximize} & \mathbf{c}^T\mathbf{x} \\
\text{subject to} & \\
& A\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{array}
$$

In this representation, c is a column vector which represents the co-efficient of the objective function . x is also a column vector of the variables involved in the Linear Programming . Hence the equation c^Tx represents the objective function. $x \geq 0$ represent the nonnegativity constraints. A is matrix of dimensions $n \times m$ and each of its row represents an inequality. Each of the column represent a variable. So the inequality $a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kn}x_n \leq b_k$ becomes the k^{th} row of whose elements are $a_{k1}, a_{k2}, \ldots, a_{kn}$. The column vector b represents the RHS of each inequality and the k^{th} entry has the value b_k .

The linear algebra representation of minimization problem is :

$$
\begin{array}{ll}
\text{minimize} & \mathbf{c}^T\mathbf{x} \\
\text{subject to} & \\
& A\mathbf{x} \geq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{array}
$$

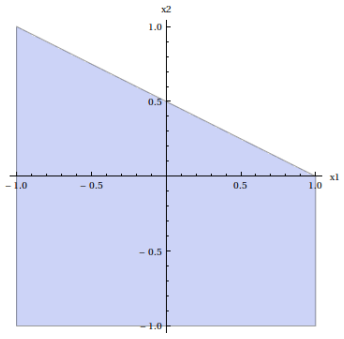
Geometric Interpretation of Linear Programming

For the ease of visualization, let us take a Linear Programming instance with just 2 variables. Any assignment for the variables can be easily visualized as a 2D point in a plane.

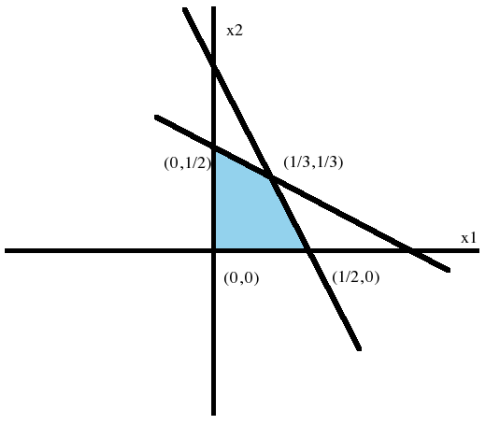
Consider a simple Linear Programming instance with 2 variables and 4 constraints :

$$
\begin{array}{ll}
\text{maximize} & x_1 + x_2 \\
\text{subject to} & \\
& x_1 + 2x_2 \leq 1 \\
& 2x_1 + x_2 \leq 1 \\
& x_1 \geq 0 \\
& x_2 \geq 0
\end{array}
$$

Each of the 4 inequalities divide the plane into two regions - one where the inequality is satisfied and one where it does not. For eg, the inequality $x_1 + 2x_2 \leq 1$ divides the plane into two regions: the set of points *(x1,x2)* such that $x_1 + 2x_2 > 1$ which are not feasible for our problem and the points such that $x_1 + 2x_2 \leq 1$ where the inequality is satisfied. The first set does not contain any feasible solution but the second set could contain points which are feasible solution of the problem provided they also satisfy the other inequalities. The line equation $x_1 + 2x_2 = 1$ is the *boundary* which separates these regions. In the figure below, the shaded region contains the set of points which satisfy the inequality.



Each of the four inequalities individually form a convex region where the feasible solution for the respective inequality is satisified. The feasible region of the problem is found by intersecting all the individual feasible regions. By the property of convex sets, the problem's feasible region is also a convex region. For this example, the feasible region geometrically looks like :



Since the feasible region is not empty, we say that this linear programming instance is *feasible*. Sometimes, the intersection might result in an empty feasible region and the problem becomes *infeasible*. Yet another possibility occurs when the feasible region extends to infinity and the problem becomes *unbounded*.

The discussion above extends naturally to higher dimensions. If the linear programming instance consists of n variables, then each of the inequalities divide the space $\mathbb{R}^n$ into two regions where in one of them the inequality is satisfied and in another it is not. The *hyperplane* $a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kn}x_n = b_k$ forms the boundary for constraint C_k. The feasible region formed by intersection of all the constraint hyperplane is a *polytope*.

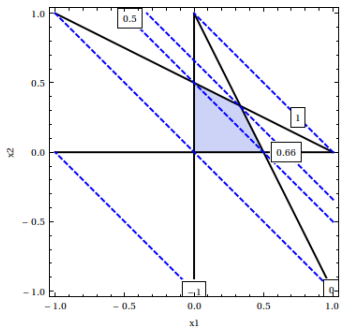
Approaches to solve Linear Programming

Once we have found the feasible region, we have to explore the region to find the point which maximises the objective function. Due to feasible region being convex and the objective function being a convex function (linear function), we know that the optimum lies only in the boundary. Once we find a *local* optima we are done as convex function on a compact convex set implies that this is also the *global* optima of the function.

One thing that simplifies the search is the fact that the optima can only exist in the vertices of the feasible region. This is due to two facts : (1) Optima of convex function on a convex set occurs on the boundary . (2) Due to convexity property any point on the line xy between x and y can be expressed as a linear combination of x and y which implies that $(x\ ,\ y)$ and $(f(x), f(y))$ are indeed the extremal points in their respective spaces for the line xy.

In the 2D example, the objective function is a line that sweeps the 2D space. At any point in the sweep, all the points in the line have the same value of the objective function. In

other words, at each step the line becomes a contour line. The figure below shows the contour values for few instances of the sweep.



Each of the vertices of the feasible region are points that is the solution of the linear equations formed when two constraints are changed to equalities. In other words, each of the vertices are formed by the intersection of the boundary line of two constraints. For eg, the point $(0.5, 0)$ is obtained by converting the inequalities $x_1 \geq 0$ and $2x_1 + x_2 \leq 1$ to equalities and solving for them. So in a two variable linear programming instance with m constraints, there can atmost be $\binom{m}{2}$ vertices. This results in a simple brtue force algorithm (assuming the problem is feasible):

Brute force algorithm for 2D :

1. For every unique pair of constraints (inequalities) C_i, C_j :
   (a) Convert them to equalities and find the point that satisfies them.
   (b) Check if the point satisfies other $m - 2$ inequalities. If not discard the point.
   (c) Else, find the cost of this point in the goal function
2. Return the point with the largest cost for goal function.

There are atmost $\binom{m}{2}$ vertices and it takes $O(m)$ to check if the point is feasible and hence this algorithm takes $O(m^3)$. This naive approach works for higher dimensions also. In a linear programming problem with n variables and m constraints, each vertex in the feasible region are formed by converting some n inequalities out of m to equalities and solving this system. There are now $\binom{m}{n}$ potential vertices and the naive algorithm takes around $O(m^{n+1})$ and results in an exponential time algorithm.

Clearly, we can do better. One approach is to prune the feasible region more aggressively. That is , if our current maximum value of the goal function for some vertex is C then do not even bother to check the vertices that have a lower cost. This makes sense practically and is the intuition behind the popular *Simplex algorithm* developed by *George Dantzig*.

As we discussed above, each of the vertices of the feasible region are points are the solution the linear system where n constraints are changed to equalities. For eg, let vertex V was

obtained by solving some n equations $e_1, e_2 \ldots e_n$. There are potentially, n neighbors for V. To find each of them , we can set equation e_i back to inequality (other equations are unaltered) and find the set of feasible points for the new system (with $n-1$ equations and rest inequalities).

So the informal outline of the simplex algorithm (assuming the problem is feasbile) is :

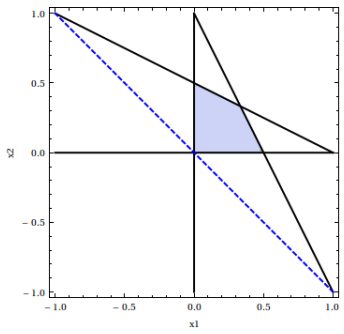
Simplex Algorithm :

1. Find an arbitrary starting vertex V.
2. Find the neighbors of V.
   (a) Find the cost of each neighbour.
   (b) If no neighbour has a cost greater that V, declare it as optimal assignment.
   (c) If some neighbour V_{new} exists whose cost is greater than V, set $V = V_{new}$ and repeat step 2.
3. Return the point with the largest cost for goal function.

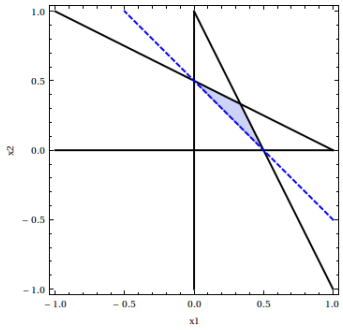
Unfortunately, the worst case cost of simplex algorithm is still exponential and takes $O(m^{n+1})$. This is because of the fact that it has to check all the $\binom{m}{n}$ vertices in the worst case. Even though this does not appear better than the naive algorithm, in practice, it runs very fast.

The simplex algorithm is really a family of algorithms with different set of heuristics for selecting initial vertex, breaking ties when there are two vertices with same cost and avoidig cycles. However, presence of heuristics make the precise analysis of simplex algorithm hard. For eg, the outline above uses a random starting point and the simple greedy heuristic to pick the next vertex.

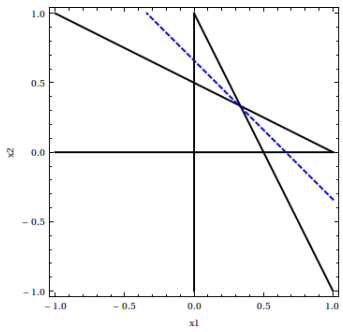
Let us see how simplex fares in our example. Lets say the initial vertex was $(0, 0)$. This vertex was obtained from the equations $x_1 = 0, x_2 = 0$. The cost of this point is 0. So we do not need to check any points with cost of below 0. The figure below shows the feasible region we must still explore and the line $x_1 + x_2 = 0$ which is our initial stab at pruning the region.



Now we check the neighbours of $(0, 0)$. We can obtain the point $(0.5, 0)$ by discarding the equation $x_1 = 0$ and including the equation $2x_1 + x_2 = 1$. Similarly, we can obtain the point $(0, 0.5)$ by discarding the equation $x_2 = 0$ and including the equation $x_1 + 2x_2 = 1$. Both of the vertices have same cost of 0.5 and lets arbitrarily pick the vertex $(0.5, 0)$. So the new lower bound of the objective function becomes 0.5 and we do not need to check any more points in the region where $x_1 + x_2 \leq 0.5$. The figure below shows the new region to explore.



The only neighbour of $(0.5, 0)$ in the new feasible region is $(0.33, 0.33)$ and the cost of this point is 0.66. So the new lower bound of the objective function becomes 0.66 and we do not need to check any more points in the region where $x_1 + x_2 \leq 0.66$. When we try to find the feasible region where $x_1 + x_2 > 0.66$ we do not find any and we output this point as the optimum. The figure below shows the final result.



Ellipsoid Algorithm

Even though simplex algorithm is exponential, there exist other polynomial time algorithm for solving linear programming. One of the earliest is the Ellipsoid algorithm by Leonid Khachiyan. This algorithm applies not just for linear programming but for convex optimization in general. In contrast to simplex algorithm which checks the vertices of the polytope both ellipsoid and interior point algorithm follow a different outline. Ellipsoid algorithm in particular uses an approach very close to *bisection* method used in unconstrained optimization. Assume that the problem is feasible ie has a nonempty feasible region and the optimal solution for Linear Programming is x^*.

Ellipsoid Algorithm :

1. Find an ellipsoid which contains the entire feasible region (including the point x^*).
2. Construct a cutting plane that splits the ellipsoid to two half-ellipsoids.
3. Using an (cutting plane) oracle, find which half-ellisoid contains x^*.
4. Construct the minimal volume ellipsoid that contain the half-ellipsoid with the optimal solution.
5. Repeat steps (2)-(4) till the ellipsoid is "small" enough . Output the point within the ellipsoid as optimal.

Even though the ellipsoid algorithm is polynomial, it is quite expensive. Usually simplex or interior point algorithms are preferred to solve Linear Programming . The complexity of ellipsoid algorithm is approximately $O(Lm^4)$ where L informally, is the number of bits of input to the algorithm. The high cost of ellipsoid algorithm is attributable to the cutting plane oracle which says how the ellisoid must be cut to form half-ellipsoids and also constructing a minimal volume ellipsoid over the half-ellipsoid that contains the optimal point.

Interior Point Algorithms

Interior point methods are a family of algorithms which can solve Linear Programming in polynomial time. This is also a generic algorithm applicable for convex optimization in general. This algorithm starts with an initial point inside the polytope and slowly generates a sequence of points that are successively closer to the optimal point. Barrier methods, which penalize the point for violating constraints (and hence go out of polytope) are used to steer the point towards optimal point. This process is repeated till the current point is arbitrarily close to the optimal vertex.

Principle of Duality

The concept of duality plays an important role in Linear Programming and approximation algorithm design but we discuss it only very briefly. For additional details refer to resources section.

Consider a maximization linear program instance.

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

and let us call it as the primal Linear Programming instance. Then, the dual of this instance is a minimization Linear Programming instance,

$$\begin{array}{ll} \text{minimize} & \mathbf{b}^T\mathbf{y} \\ \text{subject to} & \\ & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{array}$$

Informally, the role of variables and constraints are swapped between primal and dual problem - ie for each constraint in primal there exist one variable in dual . Similarly, for each variable in primal , there exist one constraint in dual. The upper bounds of each of the constraints became the coefficient of the dual problem. Each of the variables in primal became constraints and the coefficients of primal become their respective lower bounds. If the primal is a maximization problem, dual is a minimization problem and vice versa.

Facts :

1. Dual of a linear programming problem in standard form is another linear programming problem in standard form. If the primal problem is a maximization, the dual is a minimization and vice versa.
2. Dual of dual is primal itself.
3. Feasible solutions of dual provide a way to verify if the optimal value of the primal is $\leq C$ for some constant in C. Since we can verify the certificate in polynomial time, Linear programming is in $NP \bigcap Co - NP$.
4. If both primal and dual are feasible, then their optimal values are same. (aka *Strong duality*).
5. Every feasible solution of dual problem gives a lower bound on the optimal value of the primal problem. In other words, the optimum of the dual is an upper bound to the optimum of the primal (aka *Weak duality*).
6. Weak duality is usually used to derive approximation factors in algorithm design.

References and Resources

1. Luca Trevisan's lecture notes on Linear Programming at http://cs.stanford.edu/people/trevisan//cs261/lecture05.pdf . The example and 2D geometric interpretation were inspired from his notes. http://cs.stanford.edu/people/trevisan//cs261/lecture06.pdf contains a discussion of principle of duality.
2. http://www.stanford.edu/class/msande310/lecture07.pdf contains a worst case example of Simplex algorithm and also a discussion of the Ellipsoid algorithm.