CS 491I Approximation algorithms
Lecture notes
Jan.30,2001
Dejan Desovski

1. Obtaining starting basis for the simplex method

If all relations of a linear program are in ≤ form then by adding the slack variables we will have basis which has the form of identity matrix *I*.

Example: $$\mathrm{A} = \begin{bmatrix} x & y & 1 & 0 & 0 \\ z & w & 0 & 1 & 0 \\ v & q & 0 & 0 & 1 \end{bmatrix}.$$

But in the case when we are not sure about the basis, we can use two methods for obtaining starting basis:

1) Two phase method
2) Big *M* method – this method is numerically unstable.

Example:

$$\begin{aligned} & z = \min 8x_1 + 10x_2 \\ & x_1 - x_2 = 1 \\ & x_1 + x_2 \le 9 \\ & x_1 + \tfrac{1}{2}x_2 \ge 4 \\ & x_1, x_2 \ge 0 \end{aligned} \quad \rightarrow \quad \begin{aligned} & x_1 - x_2 = 1 \\ & x_1 + x_2 + x_3 = 9 \\ & -x_1 - \tfrac{1}{2}x_2 + x_4 = 4 \\ & x_1, x_2, x_3, x_4 \ge 0 \end{aligned} \quad \rightarrow \quad \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & \tfrac{1}{2} & 0 & -1 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 9 \\ 4 \end{bmatrix}$$

As we can see, there is no obvious basis for the A matrix. We proceed by adding additional variable to the equality relation, let it be x_{25}. We now have new linear optimization problem that has an obvious basis but we should prove that it is equivalent with the starting problem, that is – the optimal solution of this problem sets the x_{25} variable to zero. So we must first solve the new problem with the new optimization function $z_2 = \min x_{25}$. If the minimum of x_{25} is zero we can proceed solving the problem by using the same basis. If x_{25} is greater than zero then the starting problem is infeasible and doesn't have solution. This represents the two-phase method.

$$\begin{aligned} & x_1 - x_2 + x_{25} = 1 \\ & x_1 + x_2 + x_3 = 9 \\ & -x_1 - \tfrac{1}{2}x_2 + x_4 = 4 \\ & x_1, x_2, x_3, x_4, x_{25} \ge 0 \end{aligned} \quad \rightarrow \quad \begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & \tfrac{1}{2} & 0 & -1 & 0 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 9 \\ 4 \end{bmatrix}$$

Basis for the
new problem

Instead of solving the same problem two times with different objective functions we can introduce modification of the original objective function by adding the x_{25} variable and giving it some big cost *M* so that it should be phased out of the optimal solution. The new optimization function for the problem will be $z = \min 8x_1 + 10x_2 + Mx_{25}$ where M should be some very large number. If x_{25} is greater than zero the starting system is infeasible. But this method is proved to be numerically unstable because of division rounding in computer hardware (the M could not be appropriately chosen so that we don't have rounding errors).

2. Unboundednes

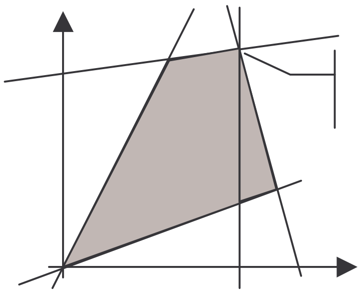
How do we know that the linear program is unbounded by using the simplex method?
In the calculation of which variable to enter the basis we have:

$$\overrightarrow{x_B} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} - x_k \begin{pmatrix} \alpha_{1,k} \\ \vdots \\ \alpha_{m,k} \end{pmatrix} \geq 0,\ x_k = \min\left\{ \frac{b_j}{\alpha_{j,k}}, \alpha_{j,k} > 0 \right\}$$

If all $\alpha_{i,k}$ are less than zero, then the objective function is unbounded.

3. Degeneracy

When we are solving the linear program we have m variables that represent the basis and have potentially nonzero value, the other n – m non-basis variables are set to zero. But the possibility exist some of the basis has some element that is equal to zero in this case we have degenerative basic solution. Degenerative solutions lead to cycling in the simplex method – we think that we are moving from one solution to another but instead we stay at the same optimal point. This happens when an extreme point lies in the intersection of more than *n* linearly independent hyperplanes.



Intersection of 3 lines, we have degenerative solution at this point

4. Equivalence of feasibility and optimization question.

Having an oracle that can answer the feasibility question: Does $\exists \vec{x}, A\vec{x} \leq \vec{b}$ is sufficient to find an optimal solution to the linear program: $opt \max \vec{c}\vec{x}, s.t. A\vec{x} \leq \vec{b}$. We can change the A matrix so that it will include the objective function: $A \cup \{\vec{c}\vec{x} \geq k\}$ and by having an oracle that solves the feasibility question we can find the optimal solution by using binary search. For $k = 0$ the optimization condition is obviously true, then if we have some $k = z$, for which the oracle returns false we know that the optimal answer lies between $k = 0$ and $k = z$. We proceed by asking the oracle about feasibility of $k = z/2$. If the problem is feasible then the optimum lies between $k = z/2$ and $k = z$, otherwise the optimum lies between $k = 0$ and $k = z/2$. Continuing this process we will find the optimal solution z* in n ≈log z steps. The algorithm is presented on the following page. $\Omega(A, \vec{b})$ is an oracle that returns **true** if the linear program $A\vec{x} \leq \vec{b}, \vec{x} \geq 0$ is feasible, and **false** if the linear program is infeasible; *l* is the value of the parameter *k* for which the linear program is feasible (e.g., $l = 0$), and *r* represents the valued of the parameter *k* for which the system is infeasible (e.g., $r = z$). The function calls the oracle to check the feasibility of the linear program for the value of the parameter $k = (l + r) / 2$, and recursively calls itself to find the optimum in the appropriate interval. We can assume rational convergence of this procedure and at the optimal point the variable *l* and *r* will meet.

```
Function Find-Optimal$\left(A, \vec{b}, \vec{c}, l, r\right)$
if $(l = r)$ then return $l$;
if $\left(\Omega\left(A \cup \left\{\vec{c}\vec{x} \ge \frac{l+r}{2}\right\}, \vec{b}\right) = \text{true}\right)$ then Find-Optimal$\left(A, \vec{b}, \vec{c}, \frac{l+r}{2}, r\right)$;
    else Find-Optimal$\left(A, \vec{b}, \vec{c}, l, \frac{l+r}{2}\right)$;
endif
```

Algorithm 1: Find-Optimal Function.

With each call of this recursive algorithm the search state is cut in half, so we have the following observation: $r - l = \frac{z}{2^n}$, where *n* is the number of iterations. The total number of iterations that the algorithm will perform is $n = \log\left(\frac{z}{r-l}\right) = \log z - \log(r - l)$.