Contents Lecture 10

- Linear programming
- The simplex algorithm
- Integer linear programming
- Branch-and-bound
- Examples
Linear program: maximize a linear function in a region

- a region defined by lines including $x_i \geq 0$, i.e. linear constraints, and
- an objective function such as $\max z = x_0 + 2x_1$
Solving a linear program

- Find \( x_i \in \mathbb{R} \) which maximizes \( z \)
- Here \( x_i \) are called decision variables

\[
\begin{align*}
3x_0 + x_1 &= 18 \\
-0.5x_0 + x_1 &= 4
\end{align*}
\]
Objective function and linear constraints using $\leq$

$$\begin{align*}
\text{max} \quad z &= x_0 + 2x_1 \\
-0.5x_0 + x_1 &\leq 4 \\
3x_0 + x_1 &\leq 18.
\end{align*}$$

Also: implicitly $x_i \geq 0$

We can use e.g. $x_i \geq 4$ or $x_i = 5$ but they can be rewritten to use $\leq$
Linear programs

\[
\begin{align*}
\text{max} \quad z &= c_0 x_0 + c_1 x_1 + \ldots + c_{n-1} x_{n-1} \\
\quad a_{0,0} x_0 + a_{0,1} x_1 + \ldots + a_{0,n-1} &\leq b_0 \\
\quad a_{1,0} x_0 + a_{1,1} x_1 + \ldots + a_{1,n-1} &\leq b_1 \\
\quad \vdots \\
\quad a_{m-1,0} x_0 + a_{m-1,1} x_1 + \ldots + a_{m-1,n-1} &\leq b_{m-1} \\
\quad x_0, x_1, \ldots, x_{n-1} &\geq 0
\end{align*}
\]

or simpler as

\[
\begin{align*}
\text{max} \quad z &= cx \\
Ax &\leq b \\
x &\geq 0.
\end{align*}
\]
Each constraint defines a halfplane in \( n \) dimensions.

The intersection of these halfplanes defines the **feasible region**, \( P \), with **feasible solutions** \( x \in P \).

The feasible region is convex, and a point where halfplanes intersect is called a **vertex**.

A linear program is either:

- **infeasible** when \( P \) is empty,
- **unbounded** when no finite solution exists, or
- **feasible**, in which case we search for an optimal solution \( x^* \in P \) which maximizes \( z \).

There may exist more than one optimal solution.
We denote by $z(x)$ the value of the objective function $z$ at point $x$.

A solution $x$ is \textbf{local optimum} for $z(x)$ if there is an $\epsilon > 0$ such that $z(x) \geq z(y)$ for all $y \in P$ with $||x - y|| \leq \epsilon$.

\textbf{Theorem}

A \textit{local optimum} of a linear program is also a \textit{global optimum}.

\textbf{Theorem}

For a bounded feasible linear program with feasible region $P$, at least one vertex is an optimal solution.

So we only need to check $z$ in the vertices and not the inner part of the region.
Slack form

\[ \text{max } c x \]

\[ x_{n+0} = b_0 - \sum_{j=0}^{n-1} a_{0,j} x_j \]

\[ x_{n+1} = b_1 - \sum_{j=0}^{n-1} a_{1,j} x_j \]

\[ \ldots \]

\[ x_{n+m-1} = b_{m-1} - \sum_{j=0}^{n-1} a_{m-1,j} x_j \]

\[ x_i \geq 0 \quad 0 \leq i \leq n + m - 1 \]

- The variables on the left hand side are called **basic variable** and occur only once, i.e. neither in any sum on the right hand side, nor in the objective function.
- The other variables are called **nonbasic variables**.
Slack form of our example

- We start with
  \[
  \max \quad z = x_0 + 2x_1 \\
  -0.5x_0 + x_1 \leq 4 \\
  3x_0 + x_1 \leq 18
  \]

- and then introduce two new variables, one for each constraint, and write it on slack form:
  \[
  \max \quad z = x_0 + 2x_1 + y \\
  x_2 = 4 - (-0.5x_0 + x_1) \\
  x_3 = 18 - (3x_0 + x_1).
  \]

- All \( x_i \geq 0 \) and \( y \) is initially zero.
- We rewrite the problem until all coefficients in the objective function become negative, and set all nonbasic variables to zero.
Entering and leaving basic variables

- Select a nonbasic variable with positive $c_i$ coefficient
- We take nonbasic variable $x_0$ as the so called entering basic variable
  \[
  \max \quad z = x_0 + 2x_1 + y
  \]
  \[
  x_2 = 4 - (-0.5x_0 + x_1)
  \]
  \[
  x_3 = 18 - (3x_0 + x_1).
  \]
- Since $c_0$ is positive, we want to increase $x_0$ as much as possible
- The basic variables can limit how much $x_0$ may be increased (if there is no restriction, then the linear program is unbounded)
- $x_3$ restricts increasing $x_0$ to at most 6.
- Therefore we select $x_3$ as the so called leaving basic variable.
Rewritten linear program

- We rewrite the linear program by letting the entering and leaving basic variables switch roles.
- This is a tedious but simple algebraic manipulation
- Do this by hand at least once

\[
\begin{align*}
\text{max} \quad z &= -0.333x_3 + 1.667x_1 + 6 \\
\end{align*}
\]

\[
\begin{align*}
x_2 &= 7 - (0.167x_3 + 1.167x_1) \\
x_0 &= 6 - (0.333x_3 + 0.333x_1)
\end{align*}
\]

- Next we must select \(x_1\) as entering basic variable
- \(x_2\) is restricted by \(7 - 1.167x_1 \geq 0\)
- \(x_0\) is restricted by \(6 - 0.333x_1 \geq 0\)
- \(x_2\) is most restricted and becomes the leaving basic variable
Solution

- All $c_i$ are negative so $z$ cannot be increased with positive values of the nonbasic variables.
- By setting the nonbasic variables to zero the maximum becomes 16 in $x = (4, 6)$ which indeed is a vertex.

$$\max z = -0.6x_3 - 1.4x_2 + 16$$

$$x_1 = 6 - (0.1x_3 + 0.9x_2)$$
$$x_0 = 4 - (0.3x_3 - 0.3x_2)$$

- Summary: we start in a vertex and then go to a neighboring vertex until all coefficients are negative, which gives the optimal solution.
- It was an open problem but George Dantzig was late for a lecture at Berkeley and mistook it for a home assignment (he got a PhD for it).
More issues

- The nonbasic variables are always set to zero
- The basic variables are always set to the corresponding $b_i$
- If the point 0 is not in the feasible region we cannot use it as the start vertex.
- This happens if some $b_i$ is negative
- Then we set up a new system to find a feasible vertex to start from.
- When we have found it, or that none exists, we continue solving the original problem.
- The course book explains the details.
Integer programming

- Integer programming is similar to linear programming with the extra condition that \( x_i \in \mathbb{N} \).
- Some problems including this have no efficient algorithms.
- One bad "method" to solve problems is to enumerate all solutions.
- This does not sound good though.
- We will use the algorithm design paradigm branch-and-bound to solve integer programs (not all due to integer programming is NP-complete).
A relaxation makes a problem simpler (by solving another problem)

For integer programming we solve the corresponding linear program, i.e. relaxing the integer requirement on the solution.

Suppose we have an integer program and give it to the Simplex algorithm and $x_k \notin \mathbb{N}$

Assume the Simplex algorithm assigns $x_k = u$

We can then branch by creating two new linear programs:
- one with the additional constraint $x_k \leq \lfloor u \rfloor$, and
- another with the additional constraint $x_k \geq \lceil u \rceil$.

Each new problem is solved directly with the Simplex algorithm

If it has an integer solution we can limit the search tree (bound)

If it has a non-integer solution and it is better than best the integer solution we put it in the queue
If the Simplex algorithm found an integer solution we do the following:

- We check if this is the best integer solution found so far, and remember it in that case.
- We remove from the queue all unexplored linear programs whose optimal value is less than the value of the integer solution we just found.
Consider an undirected graph $G(V,E)$ with $m$ edges and $n$ vertices.

At most $n$ colors are needed and we create $n$ binary decision variables $w_i$ for $0 \leq i < n$, with the meaning $w_i = 0$ if color $i$ is unused and $w_i = 1$ if at least one vertex is assigned color $i$.

For each vertex $v$ we then create $n$ decision variables $x_{vi}$, also binary, with the meaning that $x_{vi} = 1$ if vertex $v$ is assigned color $i$ and otherwise zero. The object function is to minimize the number of used colors, i.e., the sum of all $w_i$. There are two types of constraints:

- each vertex must be assigned a color, so the sum of all $x_{vi}$ for a particular vertex $v$ must be one, and
- two neighbors cannot use the same color, so if $u$ and $v$ are neighbors $x_{ui} + x_{vi} \leq w_i$, for all $i$. 
The model becomes:

\[
\min \quad z = w_0 + w_1 + \ldots + w_{n-1}
\]

\[
\sum_{i=0}^{n-1} x_{vi} = 1 \quad \forall v \in V
\]

\[
x_{ui} + x_{vi} \leq w_i \quad \forall (u, v) \in E
\]

\[
x_{vi}, w_i \in \{0, 1\} \quad \forall v \in V, 0 \leq i < n
\]
Second model

Since we expect the constraints to be on the form $A x \leq b$, we write it as follows instead:

$$
\min \quad z = w_0 + w_1 + \ldots + w_{n-1}
$$

$$
\sum_{i=0}^{n-1} x_{vi} \leq 1 \quad \forall v \in V
$$

$$
\sum_{i=0}^{n-1} -x_{vi} \leq -1 \quad \forall v \in V
$$

$$
x_{ui} + x_{vi} - w_i \leq 0 \quad \forall (u, v) \in E
$$

$$
x_{vi}, w_i \leq 1 \quad \forall v \in V, 0 \leq i < n$$
Instruction scheduling

- This is an important part of all optimizing compilers.
- The purpose is to minimize delays in the pipeline of a CPU. Assume for simplicity the input is a list of instructions, each with two source operands, one destination operand and a number indicating how many clock cycles it takes a CPU to execute the instruction.
- For instance, for a floating point add instruction, this **latency** may be five clock cycles. The input can be described as a weighted directed acyclic graph with nodes being instructions and with an edge from $u$ to $v$ if instruction $u$ computes the value of an operand of instruction $v$. The edge the latency of $u$ as weight.
- The instruction scheduling problem is NP-complete for most realistic CPUs. How can we solve it with integer linear programming? Let our goal be to find an $m$ cycle schedule, and that our CPU can issue ("start executing") $r$ instructions each clock cycle.
Let $x^j_i$ be a decision variable which says instruction $i$ is scheduled in cycle $j$.

Our schedule must satisfy the dependences in the dag, i.e., with an edge from $u$ to $v$, $u$ must not be scheduled after $v$ for correctness, but preferably sufficiently earlier than $v$ to take the latency of $u$ into account.

Denote by $L_{uv}$ this latency. In this example we look for any feasible solution.

With the nodes $V$, with $n = |V|$, and edges $E$, a basic model then becomes:
Instruction scheduling model

\[ \sum_{j=1}^{m} x_{i}^{j} = 1 \quad \forall x_{i} \in V \]

\[ \sum_{i=1}^{n} x_{i}^{j} \leq r \quad \forall 1 \leq j \leq n \]

\[ \sum_{j=1}^{m} j \times x_{k}^{j} + L_{ki} \leq \sum_{j=1}^{m} j \times x_{i}^{j} \quad \forall (k, i) \in E \]

In the last constraint, the expression \( \sum_{j=1}^{m} j \times x_{k}^{j} \) is the cycle in which instruction \( k \) is scheduled.