SMA 6304 / MIT 2.853 / MIT 2.854
Manufacturing Systems
Lecture 12: Optimization

Lecturer: Stanley B. Gershwin

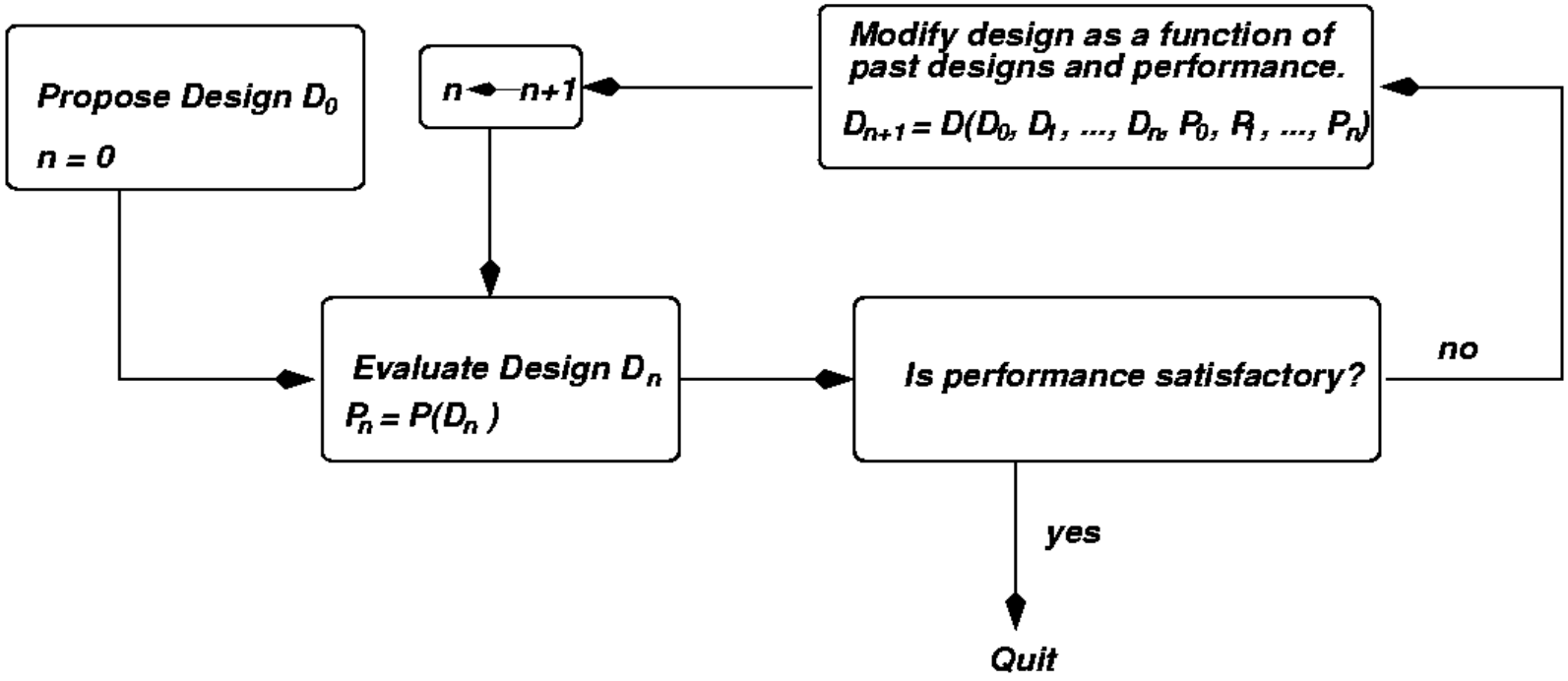
Purpose of Optimization

Choosing the *best* of a set of alternatives.

Applications:

- investment, scheduling, system design
  - ⋆ Design the least expensive production system that produces at a given rate.
  - ⋆ Design the manufacturing system that produces at the maximum rate within financial limits.

Purpose of Optimization



Typically, many designs are tested.

Purpose of Optimization

Issues

- For this to be practical, total computation time must be limited. Therefore, we must control both *computation time per iteration* and *the number of iterations* .

- Computation time per iteration includes evaluation time and the time to determine the next design to be evaluated.

- The technical literature is generally focused on limiting the number of iterations by proposing designs efficiently.

Problem Statement

$\boldsymbol{X}$ is a set of possible choices. $\boldsymbol{J}$ is a scalar function defined on $\boldsymbol{X}$. $\boldsymbol{h}$ and $\boldsymbol{g}$ are vector functions defined on $\boldsymbol{X}$.

Problem: Find $\boldsymbol{x} \in \boldsymbol{X}$ that satisfies

$\boldsymbol{J}(\boldsymbol{x})$ is maximized *(or minimized)* — the *objective*

subject to

$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ — *equality constraints*

$\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}$ — *inequality constraints*

Taxonomy

- static/dynamic
- deterministic/stochastic
- X set: continuous/discrete/mixed

Continuous Variables and Objective

$\boldsymbol{X = R^n}$. $\boldsymbol{J}$ is a scalar function defined on $\boldsymbol{R^n}$. $\boldsymbol{h(\in R^m)}$ and $\boldsymbol{g(\in R^k)}$ are vector functions defined on $\boldsymbol{R^n}$.

Problem: Find $\boldsymbol{x \in R^n}$ that satisfies

$\boldsymbol{J(x)}$ is maximized *(or minimized)*

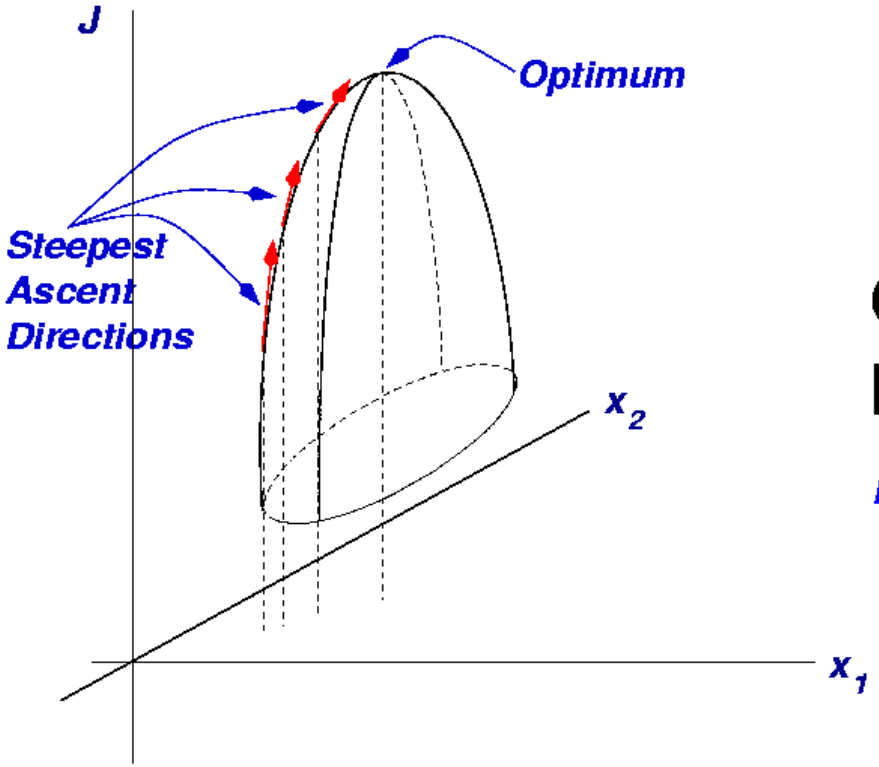
subject to

$$\boldsymbol{h(x) = 0}$$

$$\boldsymbol{g(x) \leq 0}$$

Continuous Variables and Objective

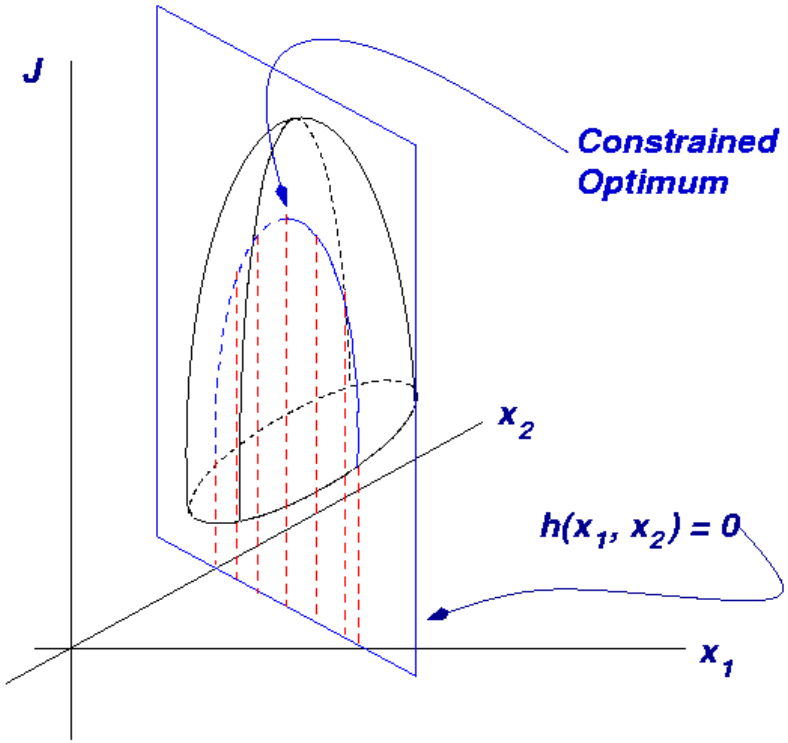
Unconstrained



Optimum often found by *steepest ascent* or *hill-climbing* methods.

Continuous Variables and Objective

Constrained

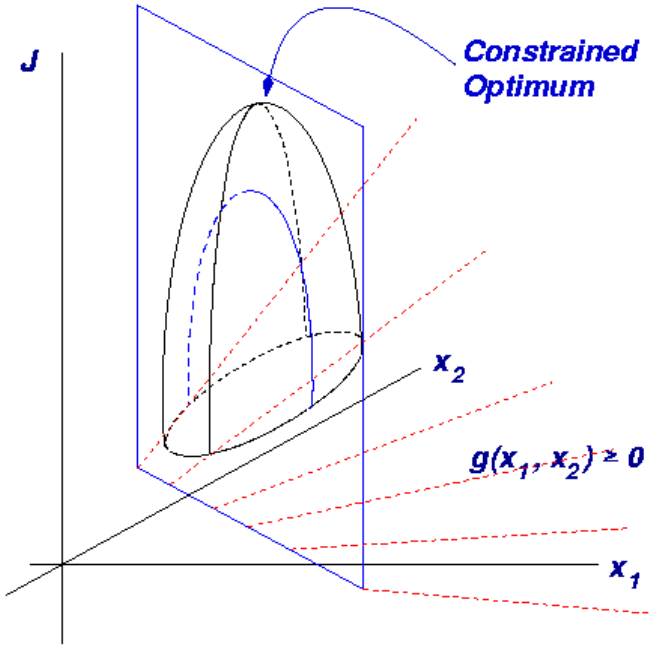


Equality constrained: solution is *on* the constraint surface.

Problems are much easier when constraint is linear, ie, when the surface is a plane.

Continuous Variables and Objective

Constrained



Inequality constrained: solution is on *one side of* the plane.

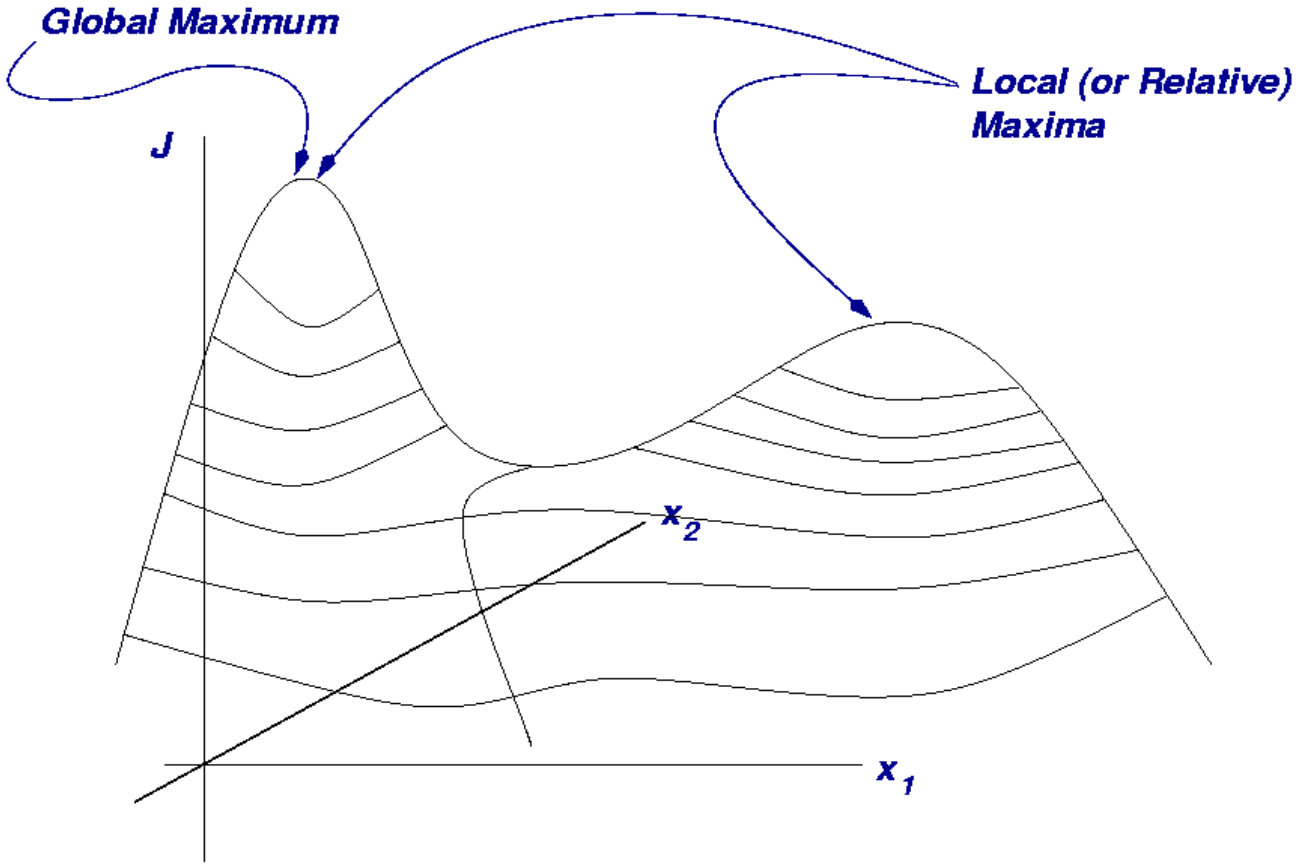
Inequality constraints that are satisfied with equality are called *effective* or *active* constraints.

Nonlinear Programming

Optimization problems with continuous variables, objective, and constraints are called *nonlinear programming* problems, especially when at least one of J, h, g are not linear.

Nonlinear Programming

Multiple Optima



Nonlinear Programming

Kuhn-Tucker Conditions

- Let x^* be a local minimum.
- Assume all gradient vectors $\partial h_i/\partial x$, $\partial g_j/\partial x$, (where g_j is effective) are linearly independent.
- Then there exist vectors λ and μ of appropriate dimension ($\mu \geq 0$ component-wise) such that at $x = x^*$,

$$\frac{\partial J}{\partial x} + \lambda^T \frac{\partial h}{\partial x} + \mu^T \frac{\partial g}{\partial x} = 0$$

$$\mu^T g = 0$$

Nonlinear Programming

Kuhn-Tucker Conditions

Subscript notation

There exist vectors $\lambda \in R^m$ and $\mu \in R^k$ ($\mu_j \geq 0$) such that at $x = x^*$,

$$\frac{\partial J}{\partial x_i} + \sum_{q=1}^{m} \lambda_q \frac{\partial h_q}{\partial x_i} + \sum_{j=1}^{k} \mu_j \frac{\partial g_j}{\partial x_i} = 0, \quad \text{for all } i = 1, \ldots, n,$$

$$\sum_{j=1}^{k} \mu_j g_j = 0$$

Kuhn-Tucker Conditions

Subscript notation

The last constraint implies that

$$g_j(x^*) > 0 \rightarrow \mu_j = 0$$

$$\mu_j > 0 \rightarrow g_j(x^*) = 0.$$

Nonlinear Programming

Numerical methods

No general method guaranteed to always work because "nonlinear" is too broad a category.

- *Feasible directions:* Take steps in a feasible direction that will reduce the cost.
  - ⋆ Issue: hard to get the feasible direction when constraints are not linear.
- *Gradient Projection:* project gradient onto the plane tangent to the constraint set. Move in that direction a short distance and then move back to the constraint surface.
  - ⋆ Issue: how short a distance? And how do you get back to the constraint surface.

Numerical methods

- *Penalty Methods:*
  1. Transform problem into unconstrained problem such as
     $\min \bar{J}(x) = J(x) + KF(h(x), g(x))$
     where $F(h(x), g(x))$ is positive if $h(x) \neq 0$ or any component of $g(x)$ is negative.
  2. Solve the problem with small positive K and then increase K. The solution for each K is a starting guess for the problem with the next K.

  ⋆ Issues: Intermediate solutions are usually not feasible; and problem gets hard to solve as K increases.

Linear Programming

Example

Two machines are available 24 hours per day. They are both required to make each of two part types. No time is lost for changeover. The times (in hours) required are:

Part	Machine	
	1	2
1	1	2
2	3	4

What is the maximum number of Type 1's we can make in 1000 hours given that the parts are produced in a ratio of 2:1?

Linear Programming

Example

Formulation

Let U_1 be the number of Type 1's produced and let U_2 be the number of Type 2's. Then the number of hours required of Machine 1 is

$$U_1 + 3U_2$$

and the number of hours required of Machine 2 is

$$2U_1 + 4U_2$$

and both of these quantities must be less than 1000.

Also,

$$U_1 = 2U_2.$$

Or,

$$\max U_1$$

subject to

$$U_1 + 3U_2 \leq 1000$$

$$2U_1 + 4U_2 \leq 1000$$

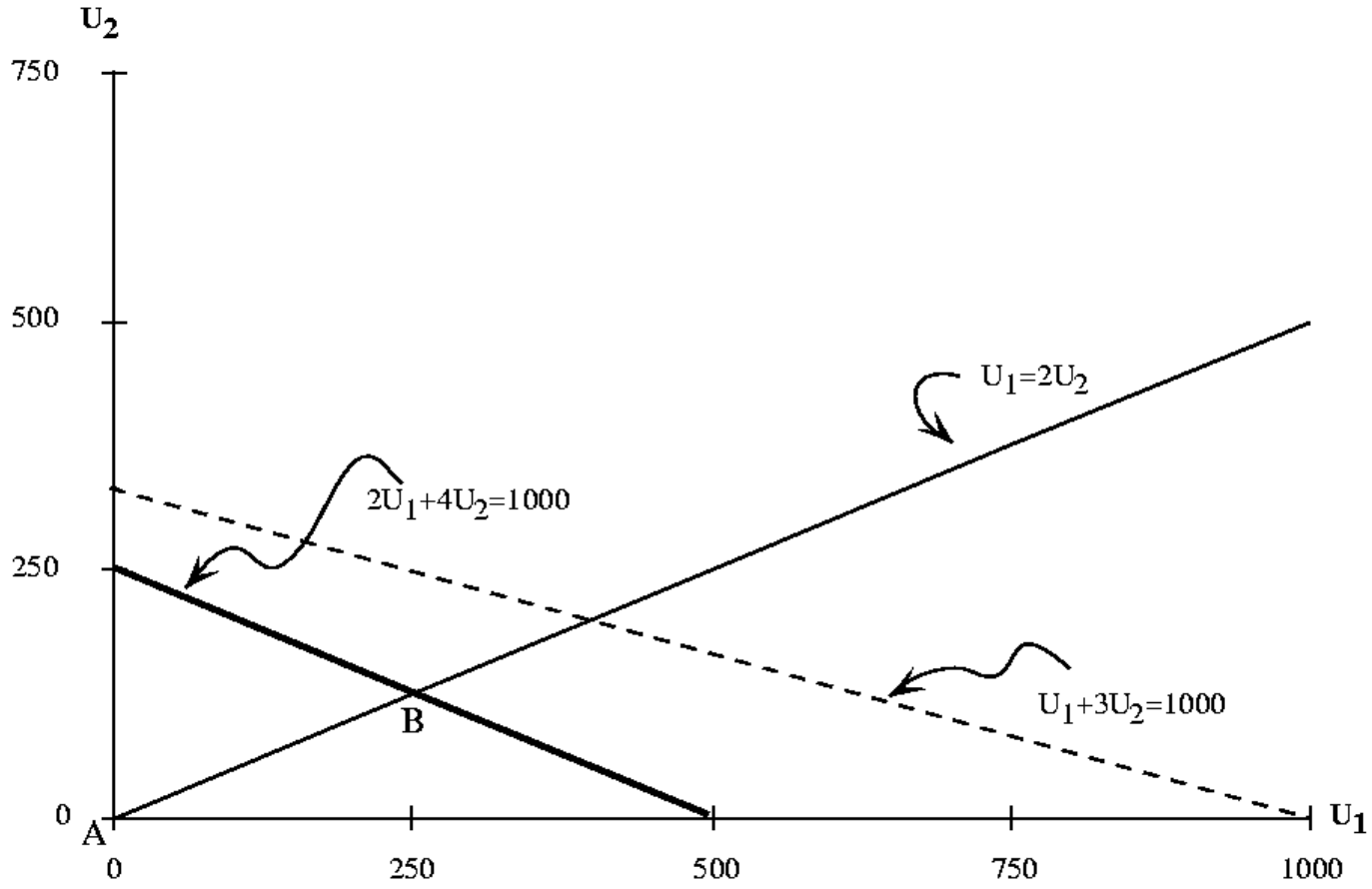
$$2U_1 = U_2$$

$$U_1 \geq 0; \quad U_2 \geq 0$$

Linear Programming

Example

Formulation



Linear Programming

General formulation

Let $x \in R^n, A \in R^{m \times n}, b \in R^m, c \in R^n$.

$$\min_x \sum_{j=1}^{n} c_j x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, i = 1, \ldots, m$$

$$x_j \geq 0, j = 1, \ldots, n$$

General formulation

Or,

$$\min_{x} c^T x$$

subject to

$$Ax = b$$

$$x \geq 0$$

Here, $\geq$ is interpreted *component-wise* .

This is the standard or canonical form of the LP.

Linear Programming

General formulation

All LPs can be expressed in this form. The example can be written

$$\mathbf{min}(-1)U_1$$

subject to

$$\begin{aligned}
&U_1 + 3U_2 + U_3 = 1000\\
&2U_1 + 4U_2 + U_4 = 1000\\
&U_1 = 2U_2\\
&U_1 \geq 0, U_2 \geq 0, U_3 \geq 0, U_4 \geq 0
\end{aligned}$$

in which U_3 and U_4 are *slack variables* . Here, they represent the idle times of Machine 1 and Machine 2.

Linear Programming

General formulation

Slack variables

In general, for every constraint of the form

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

define a new variable x_k and replace this constraint with

$$\sum_{j=1}^{n} a_{ij} x_j + x_k = b_i$$

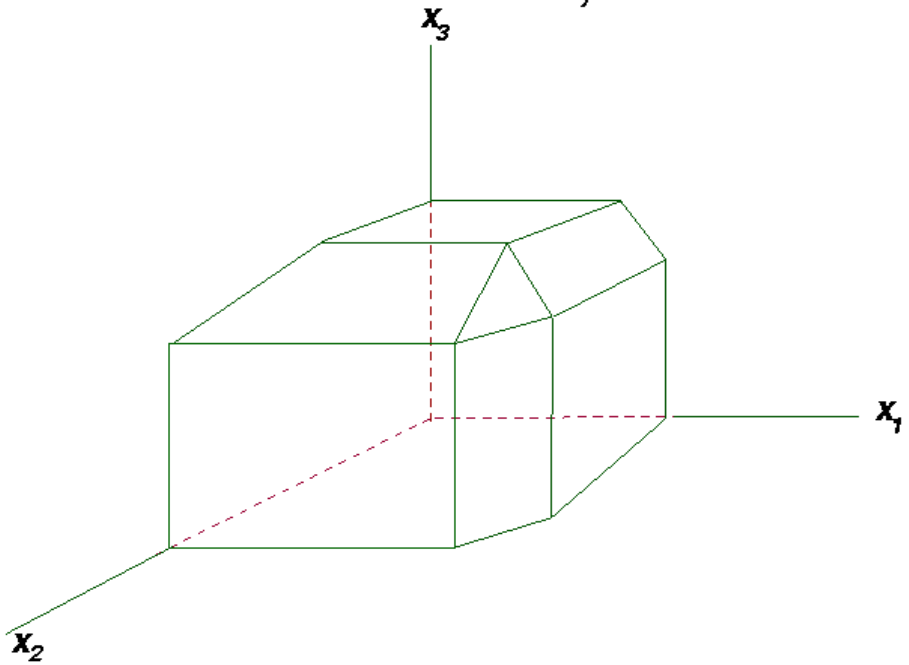
$$x_k \geq 0$$

Linear Programming

General formulation

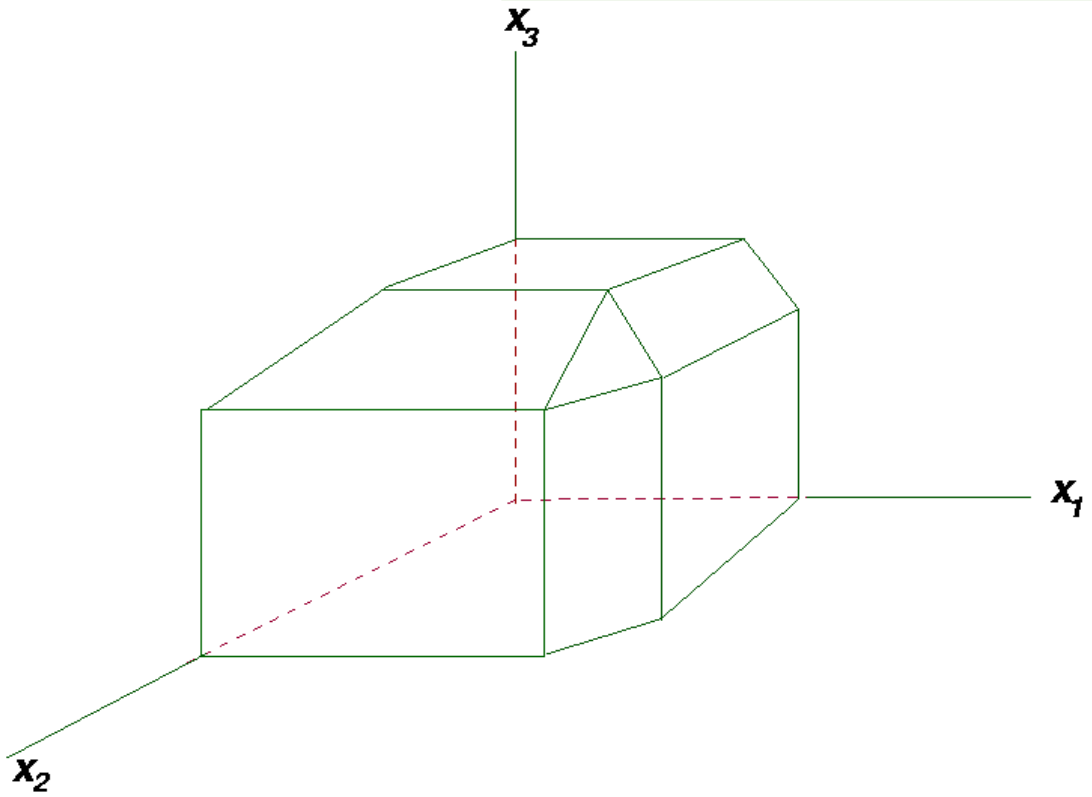
Slack variables

For this constraint set,



there are 3 variables, no equality constraints, and (at least) 7 inequality constraints (not counting $x_i \geq 0$).

The LP can be transformed into one with 10 variables, (at least) 7 equality constraints, and no inequalities (except for $x_i \geq 0$).



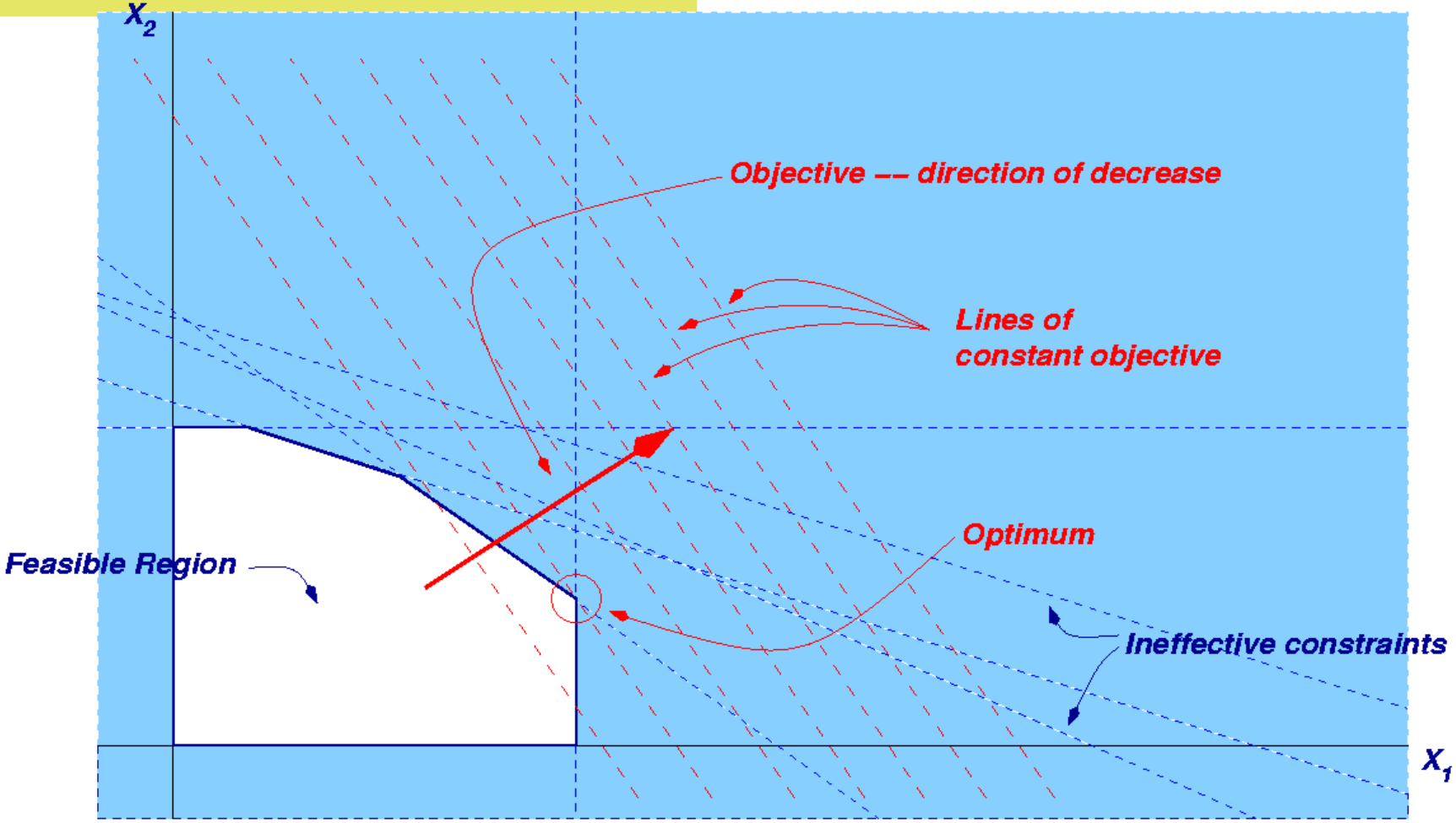
This set is called a *polyhedron* or a *simplex* .

If x satisfies the constraints, it is a *feasible solution* .

If x is feasible and it minimizes $c^T x$, it is an *optimal feasible solution* .

Linear Programming

Geometry



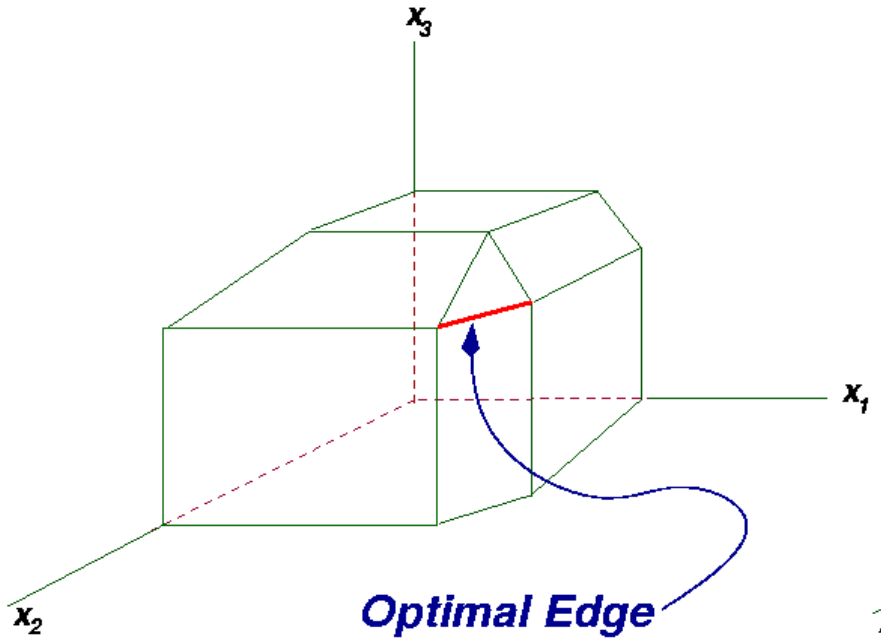
Linear Programming

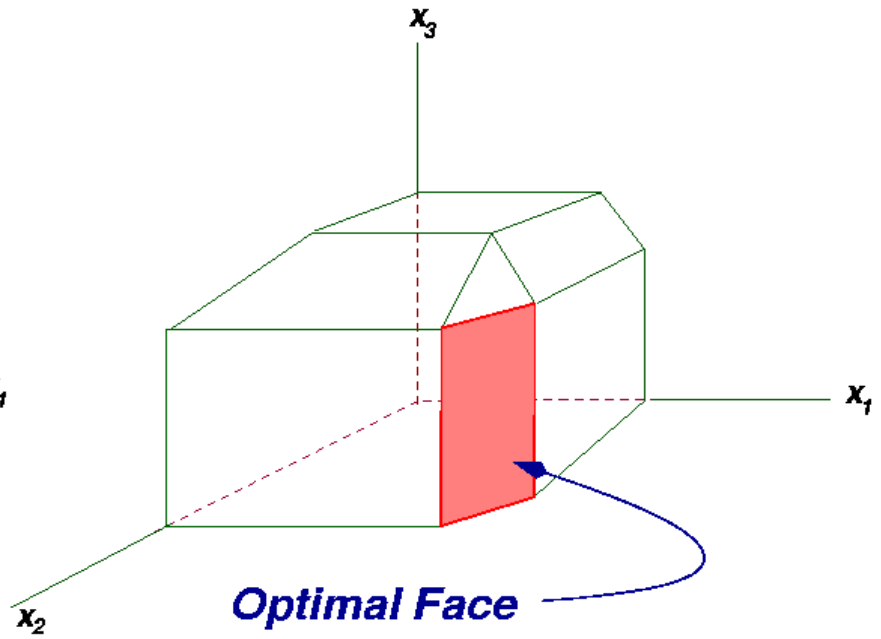
Special Cases

- Problem could be *infeasible* — no feasible set — no solution.
- Feasible set could be unbounded.
  - ⋆ Minimum of objective could be unbounded ($-\infty$) — infinite solution.
- Effective constraints could be non-independent — adds complexity to the solution technique.
- c vector could be orthogonal to the boundary of the feasible region — infinite number of solutions.

Linear Programming

Non-unique solution





Basic Solutions

Assume that there are more variables than equality constraints (that $n > m$) and that matrix A has rank m.

Let A_B be *any* matrix which consists of m columns of A. It is square ($m \times m$) and invertible.

Then A can be written

$$A = (A_B, A_N)$$

in which A_B is the *basic part* of A. The *non-basic part*, A_N, is the rest of A.

Correspondingly, $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$.

Basic Solutions

Then $\boldsymbol{Ax} = \boldsymbol{A}_B\boldsymbol{x}_B + \boldsymbol{A}_N\boldsymbol{x}_N = \boldsymbol{b}$,

or $\boldsymbol{x}_B = \boldsymbol{A}_B^{-1}(\boldsymbol{b} - \boldsymbol{A}_N\boldsymbol{x}_N)$.

If $\boldsymbol{x}_B = \boldsymbol{A}_B^{-1}\boldsymbol{b} \geq \boldsymbol{0}$ then $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{A}_B^{-1}\boldsymbol{b} \\ \boldsymbol{0} \end{pmatrix}$ is feasible and $\boldsymbol{x}$ is a *basic feasible solution* .

- Geometrically: basic feasible solutions are *corners* of the constraint set. Each corner corresponds to a different $\boldsymbol{A}_B$.

Linear Programming

The Fundamental Theorem

- If there is a feasible solution, there is a basic feasible solution.
- If there is an optimal feasible solution, there is an optimal basic feasible solution.

The Simplex Method

- Since there is always a solution at a corner (when the problem is feasible and there is a bounded solution), search for solutions only on corners.

- At each corner, determine which adjacent corner improves the objective function the most. Move there. Repeat until no further improvement is possible.

- Moving to an adjacent corner is equivalent to interchanging one of the columns of A_B with one of the columns of A_N.

Choose a feasible basis. The LP problem can be written

$$\min c_B^T x_B + c_N^T x_N$$

subject to

$$A_B^T x_B + A_N^T x_N = b$$

$$x_B \geq 0, x_N \geq 0$$

If we eliminate x_B, the problem is

$$\min \left(c_N^T - c_B^T A_B^{-1} A_N \right) x_N$$

subject to

$$\begin{aligned} A_B^{-1} A_N x_N &\leq A_B^{-1} b \\ x_N &\geq 0 \end{aligned}$$

This is an LP (although not in standard form).

Define the *reduced cost* $\boldsymbol{c}_R^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T \boldsymbol{A}_B^{-1} \boldsymbol{A}_N$. If all components of $\boldsymbol{c}_R$ are positive, $\boldsymbol{x}_N = \boldsymbol{0}$ is optimal.

If some elements of $\boldsymbol{c}_R$ are 0 and the rest are positive, there are many solutions.

Very simplified explanation of the simplex method:

- Move to an adjacent corner by taking one variable out of the basis and replace it by one not currently in the basis.
- Add to the basis the column corresponding to the most negative element of $\boldsymbol{c_R}$.
- Determine which element of the basis would decrease the cost most if it replaced by the new column.
- Update the basis inverse.
- Stop when no elements of $\boldsymbol{c_R}$ are negative.

Sensitivity Analysis

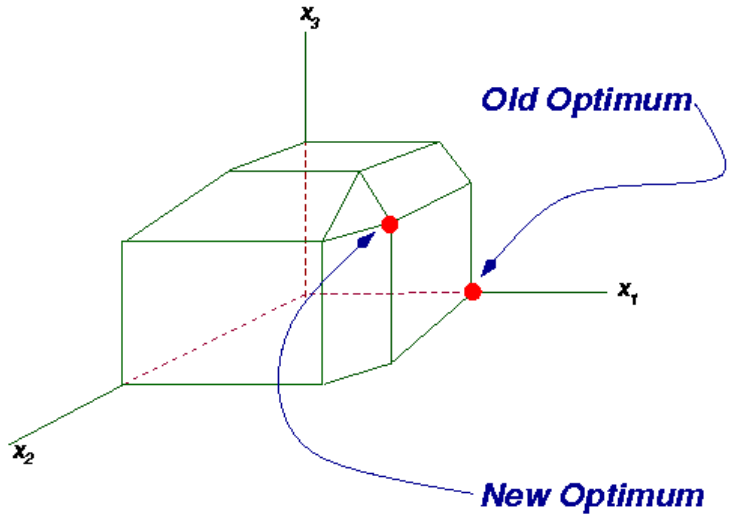
Suppose A, b, or c change by a little bit to A', b', and c'. Then the optimal solution may change. Cases:

- The basic/non-basic partition remains optimal. That is, the reduced cost vector based on the old partition remains all non-negative. The solution changes by a little bit.
- Some elements of the reduced cost go to 0. In that case, there are many solutions.

Linear Programming

Sensitivity Analysis

- Some elements of the reduced cost vector (according to the current partition) become negative. In that case, the basis must change and the solution moves to a new corner. This could mean there is a large change in the solution.



Shadow price

If the optimal value of the LP is $J = c^T x^*$, the *shadow price* of constraint j is

$$\frac{\partial J}{\partial b_j}$$

Network Problems

- Let b_i^k be the flow introduced at node i destined for node j.
- Let x_{ij}^k be the flow on link (i, j) destined for node k. $x_{ij}^k = 0$ if there is no direct link from i to j.
- Let c_{ij}^k be the cost per unit of flow on link (i, j) for flow destined for node k. $c_{ij}^k = \infty$ if there is no direct link from i to j.

Network Problems

Conservation of flow

Flow into a node = flow out of the node.

$$\sum_{j \neq i} x_{ji}^k + b_i^k = \sum_{j \neq i} x_{ij}^k \text{ for } i \neq k$$

$$\min \sum_{i,j,k} c_{ij}^k x_{ij}^k$$

$$\sum_{j \neq i} x_{ji}^k + b_i^k = \sum_{j \neq i} x_{ij}^k \text{ for all } j, k; \text{ for all } i \neq k$$

$$x_{ij}^k \geq 0 \text{ for all } i, j, k$$

Dynamic Programming

- Optimization over time.
  - ⋆ Decisions made now can have costs or benefits that appear only later, or might restrict later options.
- Deterministic or stochastic.
- *Examples:* investment, scheduling, aerospace vehicle trajectories.
- *Elements:* state, control, objective, dynamics, constraints.

Dynamic Programming

Discrete time, Deterministic

Special Class of NLPs

Objective:

$$J(x(0)) = \min_{\substack{u(i),\\ 0 \le i \le T-1}} \sum_{i=0}^{T-1} L(x(i), u(i)) + F(x(T))$$

such that

Dynamics:

$$x(i+1) = f(x(i), u(i), i)$$

$x(0)$ specified

Constraints:

$$h(x(i), u(i)) = 0; \quad g(x(i), u(i)) \le 0.$$

Dynamic Programming

Continuous time, Deterministic

Objective: $J(x(0)) = \min_{u(t),\ 0 \le t \le T} \int_0^T g(x(t), u(t))dt + F(x(T))$

such that

Dynamics: $\frac{dx(t)}{dt} = f(x(t), u(t), t)$

$x(0)$ specified

Constraints: $h(x(t), u(t)) = 0; \quad g(x(t), u(t)) \le 0.$

Dynamic Programming

Continuous time, Deterministic

- Continuous time problems can be discretized and approximated as discrete-time problems. They can therefore be approximately solved as NLPs.
- Special purpose algorithms exist for continuous-variable dynamic programming problems that make use of the special structure. However, ...
- *Curse of dimensionality:* the number of variables is typically very large (the $\boldsymbol{x}$ and $\boldsymbol{x}$ vector for *every* time step.) It is worse for discrete-variable problems where there is less structure.

Dynamic Programming

Stochastic

Discrete time

Objective: $J(x(0)) =$

$$E \min_{\substack{u(i), \\ 0 \le i \le T-1}} \sum_{i=0}^{T-1} L(x(i), u(i), w(i)) + F(x(T), w(T))$$

such that

Dynamics: $x(i+1) = f(x(i), u(i), w(i), i)$

$x(0)$ specified

Constraints: $h(x(i), u(i)) = 0; \quad g(x(i), u(i)) \le 0$

$w(i)$ is a random variable with specified distribution.

Dynamic Programming

Stochastic

- Stochastic dynamic programming problems *cannot* easily be transformed into static nonlinear programming problems.

- This is because some of the information $(\boldsymbol{w}(i))$ is not available at the time the problem is formulated.

- The solution must be specified as a *control policy* (or *feedback law*) $\boldsymbol{u}(i) = \boldsymbol{U}(\boldsymbol{x}(i))$.

Integer Programming

- Sometimes also called *combinatorial optimization* .
- Optimization in which all variables take on only discrete values.
- Problems with both discrete and continuous variables are called *mixed* .
- Much harder than problems with continuous variables.
- *Complexity theory* studies how much effort is required to solve IP/CO problems.
- *Hard* problems are those in which computation time increases exponentially or worse with the problem size.

Integer Programming

Solution techniques

- Heuristics — must be designed for specific problems.
- Relaxation — replace discrete variables by continuous and solve with LP or NLP methods. if solution is non-integral, fix up (somehow!).
- Branch and bound — let one variable have two possibilities, and get bounds (somehow) on the results for each case. Eliminate cases where bounds guarantee non-optimality. Repeat for next variable.

Applications

- Scheduling:
  - ⋆ Minimize total time to complete a set of tasks.
  - ⋆ $x_{ijk} = 1$ if task i is performed on resource j during time slot k; $x_{ijk} = 0$ otherwise.
  - ⋆ For each j, k, $\sum_i x_{ijk} = 1$
  - ⋆ precedence constraints

Applications

- Systems design:
  - ⋆ Minimize total cost of system.
  - ⋆ $x_{ij} = 1$ if machine j is used for operation i.
  - ⋆ For each i, $\sum_j x_{ij} = 1$.
  - ⋆ For each j, $\sum_i x_{ij} = 1$.
  - ⋆ Performance constraints, where system performance is expressed as a function of characteristics of the machines.