Optimization Methods for NDP

Hongwei Zhang
http://www.cs.wayne.edu/~hzhang

Acknowledgment: the slides are based on those from Drs. Yong Liu, Deep Medhi, and Michał Pióro.
Optimization Methods

- optimization -- choose the “best”.
  - what “best” means -- objective function
  - what choices you have -- feasible set

\[
\min_X f(X)
\]
subject to \( X \in A \)

- solution methods
  - brute-force, analytical, and heuristic solutions
  - linear/integer/convex programming
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Matlab optimization toolbox
- Stochastic heuristics
Linear Programming - a problem and its solution

- maximize \( z = x_1 + 3x_2 \)
- subject to \(-x_1 + x_2 \leq 1\)
  \( x_1 + x_2 \leq 2\)
  \( x_1 \geq 0, x_2 \geq 0 \)

Extreme point (vertex): a feasible point that cannot be expressed as a convex linear combination of other feasible points.
Linear Program in Standard Form

- **indices**
  - \( j=1,2,\ldots,n \) variables
  - \( i=1,2,\ldots,m \) equality constraints

- **constants**
  - \( c = (c_1,c_2,\ldots,c_n) \) cost coefficients
  - \( b = (b_1,b_2,\ldots,b_m) \) constraint right-hand-sides
  - \( A = (a_{ij}) \) \( m \times n \) matrix of constraint coefficients

- **variables**
  - \( x = (x_1, x_2,\ldots,x_n) \)

**Linear program**
- maximize
  \[ z = \sum_{j=1,2,\ldots,n} c_j x_j \]
- subject to
  \[ \sum_{j=1,2,\ldots,m} a_{ij} x_j = b_i , \quad i=1,2,\ldots,m \]
  \[ x_j \geq 0 , \quad j=1,2,\ldots,n \]

**Linear program (matrix form)**
- maximize
  \[ cx \]
- subject to
  \[ Ax = b \]
  \[ x \geq 0 \]
Transformation of LPs to the standard form

- **Inequalities**: slack variables
  - $\sum_{j=1,2,...,m} a_{ij}x_j \leq b_i$ to $\sum_{j=1,2,...,m} a_{ij}x_j + x_{n+i} = b_i$, $x_{n+i} \geq 0$
  - $\sum_{j=1,2,...,m} a_{ij}x_j \geq b_i$ to $\sum_{j=1,2,...,m} a_{ij}x_j - x_{n+i} = b_i$, $x_{n+i} \geq 0$

- **Unconstrained sign**: diff. between two nonnegative variables
  - $x_k$ with unconstrained sign: $x_k = x_k^' - x_k^''$, $x_k^' \geq 0$, $x_k^'' \geq 0$

**Exercise**: transform the following LP to the standard form

- maximize $z = x_1 + x_2$
- subject to $2x_1 + 3x_2 \leq 6$
  - $x_1 + 7x_2 \geq 4$
  - $x_1 + x_2 = 3$
  - $x_1 \geq 0$, $x_2$ unconstrained in sign
Basic facts of Linear Programming

- **feasible solution** - satisfying constraints
- **basis matrix** - a non-singular $m \times m$ submatrix of $A$
- **basic solution** to a LP - the unique vector determined by a basis matrix: $n-m$ variables associated with columns of $A$ not in the basis matrix are set to 0, and the remaining $m$ variables result from the square system of equations
- **basic feasible solution** - basic solution with all variables nonnegative (at most $m$ variables can be positive)

**Theorem 1.**

The objective function, $z$, assumes its maximum at an extreme point of the constraint set.

**Theorem 2.**

A vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is an extreme point of the constraint set if and only if $\mathbf{x}$ is a basic feasible solution.
Capacitated flow allocation problem
- LP formulation

- **variables**
  - $x_{dp}$ flow realizing demand $d$ on path $p$

- **constraints**
  - $\sum_p x_{dp} = h_d \quad d=1,2,\ldots,D$
  - $\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e \quad e=1,2,\ldots,E$
  - flow variables are continuous and non-negative

- **Property:**
  - $D+E$ non-zero flows at most
    - depending on the number of saturated links
    - if all links unsaturated: $D$ flows only!
Solution Methods for Linear Programs (1)

- Simplex Method
  - Optimum must be at the intersection of constraints
  - Intersections are easy to find, change inequalities to equalities
  - Jump from one vertex to another
  - Efficient solution for most problems, exponential time worst case.
Solution Methods for Linear Programs (2)

- **Interior Point Methods (IPM)**
  - Instead of considering only vertices of the solution polytope by moving along its edges, IPM follow a path through the interior of the polytope

- **Benefits**
  - Scales Better than Simplex
  - Certificate of Optimality
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
## IPs and MIPs

### Integer Program (IP)

**maximize** \( z = cx \)

**subject to** \( Ax \leq b, \ x \geq 0 \) (linear constraints)

**x integer** (integrality constraint)

### Mixed Integer Program (MIP)

**maximize** \( z = cx + dy \)

**subject to** \( Ax + Dy \leq b, \ x, y \geq 0 \) (linear constraints)

**x integer** (integrality constraint)
Complexity: NP-Complete Problems

- Problem Size $n$: variables, constraints, value bounds.
- Time Complexity: asymptotics when $n$ large.
  - polynomial: $n^k$
  - exponential: $k^n$

- The **NP-Complete** problems are an interesting class of problems whose status is unknown
  - no polynomial-time algorithm has been discovered for an NP-Complete problem
  - no supra-polynomial lower bound has been proved for any NP-Complete problem, either
  - All NP-Complete problems “equivalent”.
Prove NP-Completeness

- Why?
  - most people accept that it is probably intractable
  - don’t need to come up with an efficient algorithm
  - can instead work on approximation algorithms

- How?
  - reduce (transform) a well-known NP-Complete problem P into your own problem Q
  - if P reduces to Q, P is “no harder to solve” than Q
IP (and MIP) is NP-Complete

- SATISFIABILITY PROBLEM (SAT) can be expressed as IP
- even as a binary program (all integer variables are binary)
SATISFIABILITY PROBLEM

SAT

\( U = \{u_1, u_2, \ldots, u_m\} \) - Boolean variables; \( t : U \rightarrow \{\text{true, false}\} \) - truth assignment

a clause - \( \{u_1, u_2, u_4\} \) represents conjunction of its elements \( (u_1 \lor u_2 \lor u_4) \)

a clause is satisfied by a truth assignment \( t \) if and only if one of its elements is true under assignment \( t \)

\( C \) - finite collection of \( n \) clauses

\textbf{SAT:} given: a set \( U \) of variables and a collection \( C \) of clauses

question: is there a truth assignment satisfying all clauses in \( C \)?

\textbf{SAT} is NP-complete \hspace{1cm} (Cook’s theorem)

So far there are several thousands of known NP problems, (including Travelling Salesman, Clique, Steiner Problem, Graph Colourability, Knapsack) to which SAT can be reduced
Integer Programming is NP-Complete

X - set of vectors \( \mathbf{x} = (x_1, x_2, \ldots, x_n) \)
\( \mathbf{x} \in X \) iff \( A\mathbf{x} \leq \mathbf{b} \) and \( \mathbf{x} \) are integers

**Decision problem:**
Instance: given \( n, A, b, C \), and linear function \( f(\mathbf{x}) \).
Question: is there \( \mathbf{x} \in X \) such that \( f(\mathbf{x}) \leq C \)?

The SAT problem is directly reducible to a binary IP problem.
- assign binary variables \( x_i \) and \( \overline{x_i} \) with each Boolean variables \( u_i \) and \( \overline{u_i} \)
- an inequality for each clause of the instance of SAT \( (x_1 + x_2 + x_4 \geq 1) \)
- add inequalities: \( 0 \leq x_i \leq 1, 0 \leq \overline{x_i} \leq 1, 1 \leq x_i + \overline{x_i} \leq 1, i=1,2,\ldots,n \)
Optimization Methods for MIP and LP

- no hope for efficient (polynomial time) exact general methods

- main stream for achieving exact solutions: branch-and-bound
  - based on LP
    - can be enhanced with Lagrangian relaxation
  - a variant: branch-and-cut

- stochastic heuristics
  - evolutionary algorithms, simulated annealing, etc.
Why LPs, MIPs, and IPs are so Important?

- in practice only LP guarantees efficient solutions
- decomposition methods are available for LPs

- MIPs and IPs can be solved by general solvers using the branch-and-cut method, based on LP
  - sometimes very efficiently

- otherwise, we have to use (frequently) unreliable stochastic meta-heuristics (sometimes specialized heuristics)
Solution Methods for Integer Programs

- Enumeration – Tree Search, Dynamic Programming etc.

- Guaranteed to find a feasible solution (only consider integers, can check feasibility (P))
- But, exponential growth in computation time
How about solving LP Relaxation followed by rounding?
Integer Programs

- LP solution provides lower bound on IP
- But, rounding can be arbitrarily far away from integer solution
Why not combine both approaches!
- Solve LP Relaxation to get fractional solutions
- Create two sub-branches by adding constraints
Solution Methods for Integer Programs

- Known as Branch and Bound
  - Branch as above
  - For minimizing problem, LP give lower bound, feasible solutions give upper bound

```
LP
J* = J0

LP + x1 ≥ 4
J* = J2

LP + x2 ≥ 3

LP + x1 ≤ 3, x2 ≤ 2
J* = J3

LP + x1 ≤ 3, x2 ≥ 3
J* = J4

LP + x1 ≥ 4, x2 ≤ 3
J* = J5

LP + x1 ≥ 4, x2 ≥ 4
J* = J6
```

- $x_1 = 3.4, x_2 = 2.3$
- $x_1 = 3, x_2 = 2.6$
- $x_1 = 4, x_2 = 3.7$
Branch and Bound Method for Integer Programs

Branch and Bound Algorithm

1. Solve LP relaxation for lower bound on cost for current branch
   • If solution exceeds upper bound, branch is terminated
   • If solution is integer, replace upper bound on cost

2. Create two branched problems by adding constraints to original problem
   • Select integer variable with fractional LP solution
   • Add integer constraints to the original LP

3. Repeat until no branches remain, return optimal solution.
Additional Refinements – Cutting Planes

- Idea stems from adding additional constraints to LP to improve tightness of relaxation
- Combine constraints to eliminate non-integer solutions

- All feasible integer solutions remain feasible
- Current LP solution is not feasible

### Diagram

- x₁, x₂ axes
- Added cut
- Feasible integer solutions remain feasible
- Current LP solution is not feasible
General B&B algorithm for the binary case

- **Problem** $P$
  - minimize $z = cx$
  - subject to $Ax \leq b$
    - $x_i \in \{0,1\}$, $i=1,2,...,k$
    - $x_i \geq 0$, $i=k+1,k+2,...,n$

- $N_U, N_0, N_1 \subseteq \{1,2,...,k\}$ partition of $\{1,2,...,k\}$

- $P(N_U,N_0,N_1)$ – relaxed problem in continuous variables $x_i$, $i \in N_U \cup \{k+1,k+2,...,n\}$
  - $0 \leq x_i \leq 1$, $i \in N_U$
  - $x_i \geq 0$, $i=k+1,k+2,...,n$
  - $x_i = 0$, $i \in N_0$
  - $x_i = 1$, $i \in N_1$

- $z_{\text{best}} = +\infty$
procedure BBB($N_U, N_0, N_1$)
begin

  solution($N_U, N_0, N_1, x, z$);  \{ solve $P(N_U, N_0, N_1)$ \}

  if $N_U = \emptyset$ or for all $i \in N_U$ $x_i$ are binary then
    if $z < z^{\text{best}}$ then begin $z^{\text{best}} := z$; $x^{\text{best}} := x$ end
  else
    if $z \geq z^{\text{best}}$ then
      return \{ bounding \}
    else
      begin \{ branching \}
          choose $i \in N_U$ such that $x_i$ is fractional;
          BBB($N_U \setminus \{ i \}, N_0 \cup \{ i \}, N_1$); BBB($N_U \setminus \{ i \}, N_0, N_1 \cup \{ i \}$)
      end
end \{ procedure \}
original problem:
(IP) maximize $cx$
subject to $Ax \leq b$
$x \geq 0$ and integer

linear relaxation:
(LR) maximize $cx$
subject to $Ax \leq b$
$x \geq 0$

- The optimal objective value for (LR) is greater than or equal to the optimal objective for (IP).
- If (LR) is infeasible then so is (IP).
- If (LR) is optimised by integer variables, then that solution is feasible and optimal for (IP).
- If the cost coefficients $c$ are integer, then the optimal objective for (IP) is less than or equal to the “round down” of the optimal objective for (LR).
B&B - knapsack problem

- maximize \( 8x_1 + 11x_2 + 6x_3 + 4x_4 \)
- subject to \( 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \)
  \( x_j \in \{0,1\}, j=1,2,3,4 \)
- (LR) solution: \( x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0, z = 22 \)
  - no integer solution will have value greater than 22

Fractional \( z = 22 \)

- \( x_3 = 0 \) Fractional \( z = 21.65 \)
- \( x_3 = 1 \) Fractional \( z = 21.85 \)

\( x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0.667 \)
\( x_1 = 1, x_2 = 0.714, x_3 = 1, x_4 = 0 \)
we know that the optimal integer solution is not greater than 21.85 (21 in fact)
we will take a subproblem and branch on one of its variables
- we choose an active subproblem (here: not chosen before)
- we choose a subproblem with highest solution value

\[ \text{Fractional } \quad z = 22 \]

\[ \begin{align*} x_3 &= 0 \\ \text{Fractional } \quad z &= 21.65 \end{align*} \]

\[ \begin{align*} x_3 &= 1 \\ \text{Fractional } \quad z &= 21.85 \end{align*} \]

\[ \begin{align*} x_3 &= 1, x_2 &= 0 \\ \text{Integer } \quad z &= 18 \\ \text{INTEGER} \end{align*} \]

\[ \begin{align*} x_3 &= 1, x_2 &= 1 \\ \text{Fractional } \quad z &= 21.8 \end{align*} \]

no further branching, not active

\[ x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1 \]

\[ x_1 = 0.6, x_2 = 1, x_3 = 1, x_4 = 0 \]
B&B example cntd.

Fractional
z = 22

x₃ = 0
Fractional
z = 21.65

x₃ = 1
Fractional
z = 21.85

x₁ = 0, x₂ = 1, x₃ = 1, x₄ = 1

x₁ = 1, x₂ = 1, x₃ = 1, x₄ = ?

there is no better solution
than 21: bounding

x₃ = 1, x₂ = 0
Integer
z = 18

INTEGER

x₃ = 1, x₂ = 1
Fractional
z = 21.8

x₃ = 1, x₂ = 1, x₁ = 0
Integer
z = 21

INFEASIBLE

x₃ = 1, x₂ = 1, x₁ = 1
Infeasible

optimal
Solve the linear relaxation of the problem. If the solution is integer, then we are done. Otherwise create two new subproblems by branching on a fractional variable.

A subproblem is not active when any of the following occurs:
- you have already used the subproblem to branch on
- all variables in the solution are integer
- the subproblem is infeasible
- you can bound the subproblem by a bounding argument.

Choose an active subproblem and branch on a fractional variable. Repeat until there are no active subproblems.

**Remarks**
- If $x$ is restricted to integer (but not necessarily to 0 or 1), then if $x = 4.27$ you would branch with the constraints $x \leq 4$ and $x \geq 5$.
- If some variables are not restricted to integer you do not branch on them.
Also, integer MIP can always be converted into binary MIP transformation: \( x_j = 2^0u_{j0} + 2^1u_{j1} + \ldots + 2^qu_{jq} \) \( (x_j \leq 2^{q+1} - 1) \)

Lagrangian relaxation can also be used for finding lower bounds (instead of linear relaxation).

**Branch-and-Cut (B&C)**
- combination of B&B with the *cutting plane method*
  - the most effective exact approach to NP-complete MIPs
- *idea*: add ”valid inequalities” which define the facets of the integer polyhedron
  - the valid inequalities generation is problem-dependent, and not based on general “formulas” as for the cutting plane method (e.g., Gomory fractional cuts)
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
LP decomposition methods

- Lagrangian Relaxation (LR)
- Column Generation technique for Candidate Path List Augmentation (CPLA)
  - Based on LR
  - Need-based, incremental addition of candidate paths for link-path formulation
- Bender’s decomposition
  - Master problem + feasibility test
    - When feasibility test fails, add a new (linear) inequality to the master problem
Lagrangian Relaxation Method to solve large problems (especially with integer variables having special structure)

Consider the 3-node capacity design problem:

\[
\begin{align*}
\min_{x,y} \quad & F(x, y) = \xi_1 y_1 + \xi_2 y_2 + \xi_3 y_3 \\
\text{subject to} \quad & x_{11} + x_{12} = h_1 \\
& x_{21} + x_{22} = h_2 \\
& x_{31} + x_{32} = h_3 \\
& x_{11} + x_{22} + x_{32} \leq M y_1 \\
& x_{12} + x_{21} + x_{32} \leq M y_2 \\
& x_{12} + x_{22} + x_{31} \leq M y_3 \\
& x \geq 0 \\
& y \geq 0 \text{ integer}
\end{align*}
\]

(1)

Use a ‘dual’ multiplier \( \pi_e \geq 0 \) with each of the capacity constraints. The Lagrangian relaxation is

\[
L(x, y, \pi) = \xi_1 y_1 + \xi_2 y_2 + \xi_3 y_3 \\
+ \pi_1 (x_{11} + x_{22} + x_{32} - M y_1) \\
+ \pi_2 (x_{12} + x_{21} + x_{32} - M y_2) \\
+ \pi_3 (x_{12} + x_{22} + x_{31} - M y_3)
\]

(2)

You can verify that

\[
F(x, y) \geq L(x, y, \pi)
\]
LR method (cont’d)

Now consider the parametric problem (“relaxed problem”):

\[ W(\pi) = \min_{\{x,y\}} \quad L(x, y, \pi) = \xi_1 y_1 + \xi_2 y_2 + \xi_3 y_3 \]
\[ + \pi_1(x_{11} + x_{22} + x_{32} - My_1) \]
\[ + \pi_2(x_{12} + x_{21} + x_{32} - My_2) \]
\[ + \pi_3(x_{12} + x_{22} + x_{31} - My_3) \]

subject to
\[ x_{11} + x_{12} = h_1 \]
\[ x_{21} + x_{22} = h_2 \]
\[ x_{31} + x_{32} = h_3 \]
\[ x \geq 0 \]
\[ y \geq 0 \text{ integer} \]

Since \( W(\pi) \) is minimum of \( L(x, y, \pi) \) over \( x, y \), the following holds

\[ F(x, y) \geq L(x, y, \pi) \geq W(\pi) \]

Thus, the goal is to MAXIMIZE \( W(\pi) \) over \( \pi \geq 0 \).
Now re-write (3) as:

\[
W(\pi) = \min_{\{x, y\}} L(x, y, \pi) = (\xi_1 - M\pi_1)y_1 + (\xi_2 - M\pi_2)y_2 + (\xi_3 - M\pi_3)y_3
+ \pi_1 x_{11} + (\pi_2 + \pi_3)x_{12}
+ \pi_2 x_{21} + (\pi_1 + \pi_3)x_{22}
+ \pi_3 x_{31} + (\pi_2 + \pi_2)x_{32}
\]

subject to

\[
\begin{align*}
x_{11} + x_{12} &= h_1 \\
x_{21} + x_{22} &= h_2 \\
x_{31} + x_{32} &= h_3 \\
x &\geq 0 \\
y &\geq 0 \text{ integer}
\end{align*}
\]  

The above can be decoupled to \(y\) and \(x\) separately! For example, for each link \(\ell = 1, 2, 3\), we have:

\[
\begin{align*}
\min_{y_1} &\{(\xi_1 - M\pi_1)y_1 \mid y_1 \geq 0\} \\
\min_{y_2} &\{(\xi_2 - M\pi_2)y_2 \mid y_2 \geq 0\} \\
\min_{y_3} &\{(\xi_3 - M\pi_3)y_3 \mid y_3 \geq 0\}
\end{align*}
\]

\[
\begin{align*}
\min_{x_{11}, x_{12}} &\{\pi_1 x_{11} + (\pi_2 + \pi_3)x_{12} \mid x_{11} + x_{12} = h_1, x_{11} \geq 0, x_{12} \geq 0\} \\
\min_{x_{21}, x_{22}} &\{\pi_2 x_{21} + (\pi_1 + \pi_3)x_{22} \mid x_{21} + x_{22} = h_2, x_{21} \geq 0, x_{22} \geq 0\} \\
\min_{x_{31}, x_{32}} &\{\pi_3 x_{31} + (\pi_1 + \pi_2)x_{32} \mid x_{31} + x_{32} = h_1, x_{31} \geq 0, x_{32} \geq 0\}
\end{align*}
\]

Each of them is easy to solve; for the first three, put an article upper bound on the value \(y_{\ell}\) takes (to account for the case when \(\xi_{\ell} - M\pi_{\ell} < 0\), \(\ell = 1, 2, 3\)). Note, that for each iteration of \(\pi\), the above simple subproblems are solved. Need a way to update \(\pi\) iteratively.
Comment: \( W(\pi) \) is not a differential function in general; rather it has a sub-gradient.

\textit{A general solution strategy for LR approach:}

- Pick a starting \( \pi \)
- Solve (4) to obtain \( W(\pi) \) (means the subproblems)
- Find a direction from sub-gradient of \( W \) at this point and take a step along this direction
- Obtain a new \( \pi \) and iterate back until some criteria are satisfied
LR method: generalization from the 3-node example

- **Using matrix notation:**

  \[
  \begin{align*}
  \text{minimize} & \quad F = \xi y \\
  \text{subject to} & \quad Ex = h \\
  & \quad Dx \leq My \\
  & \quad x \geq 0; \ y \text{ non-negative integer.}
  \end{align*}
  \]

- **Lagrangian**

  \[
  L(x, y; \pi) = \xi y + \pi (Dx - My).
  \]

- **Rearranging**

  \[
  L(x, y; \pi) = (\pi D)x + (\xi - M \pi)y.
  \]
Now the relaxed problem is:

\[
\begin{align*}
\text{minimize} & \quad L(x, y; \pi) = \xi y + \pi(Dx - My) \\
\text{subject to} & \quad Ex = h, \quad x \geq 0, \quad y \text{ non-negative integer}.
\end{align*}
\]

Dual function \( W(\pi) \)

\[
W(\pi) = \min_{x,y} \{ L(x, y; \pi) : Ex = h, \ x \geq 0, y \geq 0 \text{ and integer} \} = \min_x \{ (\pi D)x : Ex = h, \ x \geq 0 \} + \min_y \{ (\xi - M \pi)y : y \geq 0 \text{ and integer} \}.
\]

Subproblems to solve (given \( \pi \)):

\[
W(\pi) = \min_{x,y} \{ L(x, y; \pi) : Ex = h, \ x \geq 0, \ y \geq 0 \} = \min_x \{ (\pi D)x : Ex = h, \ x \geq 0 \} + \min_y \{ (\xi - M \pi)y : y \geq 0 \}
\]

Now, sub-gradient \( \frac{\partial W(\pi)}{\partial \pi} = Dx^*(\pi) - My^*(\pi) \).
Iterative process (5.4.4b)

\[ \pi^{k+1} = \max \left\{ \pi^k + t_k \frac{\partial W(\pi)}{\partial \pi^k}, 0 \right\}. \]

where update “step-size” by (5.4.4c)

\[ t_k = \rho \left( W - W(\pi^k) \right) / \left\| \frac{\partial W(\pi)}{\partial \pi^k} \right\|^2 \]

\[ 0 \leq \rho \leq 2. \]
Algorithm 5.8 Lagrangian relaxation (LR) based Dual Algorithm

Step 0 Choose an initial $\pi^0$, $k_{max}$ and $\rho_{maxiter}$. Set $\rho = 2$, $\rho_{min} = 0.005$, $k = 0$, $\rho_{iter} = 0$, $F^{best} = \infty$

Step 1 $k := k + 1$, $\rho_{iter} := \rho_{iter} + 1$
Given $\pi^k$, solve (5.4.2) as decoupled subproblems in $x$ and $y$ to obtain solutions $x^k$, and $\bar{y}^k$

Step 2 Use $x^k$ to compute feasible $y^k$ (integer valued) that satisfies $Dx^k \leq My^k$
Use $x^k$ and $y^k$ to computer primal objective $F$;
If ($F < F^{best}$) then begin $F^{best} = F$; $x^{best} = x^k$; $y^{best} = y^k$; $W = F^{best}$ end
If ($\rho_{iter} > \rho_{maxiter}$) then begin $\rho = \max\{\rho/2, \rho_{min}\}$; $\rho_{iter} = 0$ end

Step 3 Use decoupled solutions $x^k$ and $\bar{y}^k$, compute
subgradient: $\frac{\partial W}{\partial \pi^k}$ (refer to (5.4.4a))
dual objective: $W(\pi^k) = \xi \bar{y}^k + \pi^k (Dx^k - My^k)$
step size: $t_k$ (refer to (5.4.4c))
dual variable: $\pi^{k+1}$ (refer to (5.4.4b))

Step 4 If $k > k_{max}$, stop; otherwise go to step 1
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
Stochastic heuristics

- Local Search
- Simulated Annealing
- Evolutionary Algorithms
- Simulated Allocation
- Tabu Search
- Others: greedy randomized adaptive search
Local Search: steepest descent

- minimize $f(x)$
  - starting from initial point $x_c = x_0$
  - iteratively minimize value $f(x_c)$ of current state $x_c$, by replacing it by point in its neighborhood that has lowest value.
  - stop if improvement no longer possible

- “hill climbing” when maximizing
Problem with Local Search

- may get stuck in local minima

Question: How to avoid local minima?
What about Occasional Ascents?

desired effect
Help escaping the local optima.

adverse effect
Might pass global optima after reaching it (easy to avoid by keeping track of best-ever state)
Simulated annealing (SAN): basic idea

- From current state, pick a random successor state;
- If it has better value than current state, then “accept the transition,” that is, use successor state as current state;
- Otherwise, do not give up, but instead flip a coin and accept the transition with a given probability (that is lower as the successor is worse).
- So we accept to sometimes “un-optimize” the value function a little with a non-zero probability.
Simulated Annealing

Kirkpatrick et al. 1983:

- Simulated annealing is a general method for making likely the escape from local minima by allowing jumps to higher value states.

- The analogy here is with the process of annealing used by a craftsman in forging a sword from an alloy.
Real annealing: Sword

- He heats the metal, then slowly cools it as he hammers the blade into shape.
  - if he cools the blade too quickly the metal will form patches of different composition;
  - if the metal is cooled slowly while it is shaped, the constituent metals will form a uniform alloy.
Simulated Annealing - algorithm

- uphill moves are permitted but only with a certain (decreasing) probability ("temperature" dependent) according to the so called Metropolis Test

begin
choose an initial solution \(i \in S\);
select an initial temperature \(T > 0\);
while stopping criterion not true
  count := 0;
  while count < L
    choose randomly a neighbour \(j \in N(i)\);
    \(\Delta F := F(j) - F(i)\);
    if \(\Delta F \leq 0\) then \(i := j\)
    else if \(\text{random}(0,1) < \exp\left(-\frac{\Delta F}{T}\right)\) then \(i := j\);
    count := count + 1
  end while;
reduce temperature (\(T := T \times \alpha\))
end while
end

Metropolis test
Simulated Annealing - limit theorem

- limit theorem: global optimum will be found
- for fixed $T$, after sufficiently number of steps:
  - $\text{Prob} \{ X = i \} = \exp(-F(i)/T) / Z(T)$
  - $Z(T) = \sum_{j \in S} \exp(-F(j)/T)$
- for $T \to 0$, $\text{Prob} \{ X = i \}$ remains greater than 0 only for optimal configurations $i \in S$

This is not a very practical result:
  too many moves (number of states squared) would have to be made to achieve the limit sufficiently closely
Evolution Algorithm: motivation

- A population of individuals exists in an environment with limited resources.
- **Competition** for those resources causes selection of those **fitter** individuals that are better adapted to the environment.
- These individuals act as seeds for the generation of new individuals through **recombination** and **mutation**.
- The new individuals have their fitness evaluated and compete (possibly also with parents) for survival.
- Over time **Natural selection** causes a rise in the fitness of the population.
Evolution Algorithm: general schemes

- EAs fall into the category of “generate and test” algorithms
- They are stochastic, population-based algorithms
- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty
- Selection reduces diversity and acts as a force pushing quality
Evolutionary Algorithm: basic notions

- **population** = a set of $\mu$ chromosomes
  - generation = a consecutive population

- **chromosome** = a sequence of genes
  - individual solution (point of the solution space)
  - genes represent internal structure of a solution
  - fitness function = cost function
Genetic operators

- **mutation**
  - is performed over a chromosome with certain (low) probability
  - it perturbs the values of the chromosome’s genes

- **Crossover/recombination**
  - exchanges genes between two parent chromosomes to produce an offspring
  - in effect the offspring has genes from both parents
  - chromosomes with better fitness function have greater chance to become parents

In general, the operators are problem-dependent.
begin
n:= 0; initialize(P_0);
while stopping criterion not true
O_n:= ∅;
for i:= 1 to L do O_n:= O_n ∪ crossover(P_n);
for ε ∈ O_n do mutate(ε);
n:= n+1,
P_n:= select_best(O_n ∪ P_n);
end while
end
Evolutionary Algorithm for the flow problem

- Chromosome: \( x = (x_1, x_2, \ldots, x_D) \)
- Gene:
  \[ x_d = (x_{d1}, x_{d2}, \ldots, x_{dP_d}) \] - flow pattern for the demand \( d \)

\[
\begin{array}{ccccccc}
5 & 2 & 3 & 3 & 1 & 4 \\
1 & 2 & 0 & 0 & 3 & 5 \\
1 & 0 & 2 & 1 & & \\
2 & 3 & & & & \\
\end{array}
\]

chromosome
Evolutionary Algorithm for the flow problem cntd.

- **crossover of two chromosomes**
  - each gene of the offspring is taken from one of the parents
  - for each $d=1,2,\ldots,D$: $x_d := x_d(1)$ with probability 0.5
  - $x_d := x_d(2)$ with probability 0.5
  - better fitted chromosomes have greater chance to become parents

- **mutation of a chromosome**
  - for each gene shift some flow from one path to another
  - everything at random
Simulated Allocation (SAL)

Modular Links and Modular Flows Dimensioning

**LP: D/ML/MF**

**Modular Links and Modular Flows**

indices

- \( d = 1, 2, \ldots, D \), demands
- \( p = 1, 2, \ldots, P_d \), candidate paths for demand \( d \)
- \( e = 1, 2, \ldots, E \), links

constants

- \( \delta_{edp} \): 1, if link \( e \) belongs to path \( p \) realizing demand \( d \); 0, otherwise
- \( L \): size of the demand flow capacity module
- \( h_d \): volume of demand \( d \) expressed as the number of demand modules
- \( \xi_e \): cost of one capacity module on link \( e \)
- \( M \): size of the link capacity module

variables

- \( x_{dp} \): number of demand modules allocated to path \( p \) of demand \( d \)
- \( y_e \): capacity of link \( e \) expressed in the number of modules

objective

\[ \min F = \sum_e \xi_e y_e \]  

(5.3.6a)

constraints

\[ \sum_p x_{dp} = h_d \quad d = 1, 2, \ldots, D \]  

(5.3.6b)

\[ \sum_d \sum_p \delta_{edp} x_{dp} \leq (M/L) y_e \quad e = 1, 2, \ldots, E \]  

(5.3.6c)

\( x_{dp} \) and \( y_e \) non-negative integers
SAL: general schemes

- Work with partial flow allocations
  - some solutions NOT implement all demands
- In each step chooses, with probability $q(x)$, between:
  - $allocate(x)$ – adding one demand flow to the current state $x$
  - $disconnect(x)$ – removing one or more demand flows from current $x$
- Choose best out of N full solutions
procedure SAL 
begin 
  \( n := 0; \bar{x} := 0; \ F^{best} := +\infty; \)
repeat 
  if \( random(0, 1) < q(|x|) \) then allocate(\( x \)) else disconnect(\( x \)); 
  if \( |x| = H \) then 
    begin 
      \( n := n + 1; \)
      if \( F(x) < F^{best} \) then 
        begin 
          \( F^{best} := F(x); \)
          \( \bar{x}^{best} := x \)
        end 
    end 
  until \( n = N \) or \( F^{best} \leq cost\_lower\_bound \) 
end \{ procedure \}
SAL: details

- **allocate**($x$)
  - randomly pick one non-allocated demand module
  - allocate demand to the shortest path
    - link weight 0 if unsaturated
    - link weight set to the link price if saturated
  - increase link capacity by 1 on saturated links

- **disconnect**($x$)
  - randomly pick one allocated demand module
  - disconnect it from the path it uses
  - decrease link capacity by 1 for links with empty link modules
Outline

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
Optimization packages

- Matlab optimization toolbox

- CPLEX: can solve large scale LP/IP/MIP;
  AMPL: a standard programming interface for many optimization engines.
  - Student version windows/unix/linux
    - 300 variables
Matlab & optimization toolbox

- Matlab: a powerful technical computing software
  - Script-like language
  - Rich toolboxes: optimization, statistics, symbolic math, simulink, image processing, etc
  - User-friendly online document/help

- Optimization toolbox
  - linprog: solve linear programming problems
  - bintprog: solve binary integer programming problems
  ...
linprog
Solve linear programming problems

Equation
Finds the minimum of a problem specified by
\[
\min_x f^T x \quad \text{such that} \quad A x \leq b \\
A_{eq} x = b_{eq} \\
lb \leq x \leq ub
\]
where \( f, x, b, b_{eq}, lb, \) and \( ub \) are vectors and \( A \) and \( A_{eq} \) are matrices.

Syntax
\[
x = \text{linprog}(f,A,b) \\
x = \text{linprog}(f,A,b,A_{eq},b_{eq}) \\
x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub) \\
x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub,x0) \\
x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub,x0,options) \\
[x,fval] = \text{linprog}(...) \\
[x,\lambda,\text{exitflag}] = \text{linprog}(...) \\
[x,\lambda,\text{exitflag},\text{output}] = \text{linprog}(...) \\
[x,fval,\text{exitflag},\text{output},\lambda] = \text{linprog}(...)
\]

Description
linprog solves linear programming problems.
\( x = \text{linprog}(f,A,b) \) solves \( \min f^T x \) such that \( A x \leq b \).
\( x = \text{linprog}(f,A,b,A_{eq},b_{eq}) \) solves the problem above while additionally satisfying the equality constraints \( A_{eq} x = b_{eq} \). Set \( A=[] \) and \( b=[] \) if no inequalities exist.
\( x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub) \) defines a set of lower and upper bounds on the design variables, \( x \), so that the solution is always in the range \( lb \leq x \leq ub \). Set \( A_{eq}=[] \) and \( b_{eq}=[] \) if no equalities exist.
\( x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub,x0) \) sets the starting point to \( x_0 \). This option is only available with the medium-scale algorithm (the LargeScale option is set to 'off' using optimset). The default large-scale algorithm and the simplex algorithm ignore any starting point.
\( x = \text{linprog}(f,A,b,A_{eq},b_{eq},lb,ub,x0,options) \) minimizes with the optimization options specified in the structure options. Use optimset to set these options.
\( [x,fval] = \text{linprog}(...) \) returns the value of the objective function \( f \) at the solution \( x \): \( fval = f^T x \).
\( [x,\lambda,\text{exitflag}] = \text{linprog}(...) \) returns a value \( \text{exitflag} \) that describes the exit condition.
Find $x$ that minimizes
\[ f(x) = -5x_1 - 4x_2 - 6x_3 \]
subject to
\[
\begin{align*}
x_1 - x_2 + x_3 &\leq 20 \\
3x_1 + 2x_2 + 4x_3 &\leq 42 \\
3x_1 + 2x_2 &\leq 30 \\
0 \leq x_1, 0 \leq x_2, 0 \leq x_3
\end{align*}
\]
First, enter the coefficients
\[
\begin{align*}
f &= [-5; -4; -6] \\
A &= \begin{bmatrix} 1 & -1 & 1 \\ 3 & 2 & 4 \\ 3 & 2 & 0 \end{bmatrix}; \\
b &= [20; 42; 30]; \\
\text{lb} &= \text{zeros}(3,1);
\end{align*}
\]
Next, call a linear programming routine.
\[
[x, fval, exitflag, output, lambda] = \text{linprog}(f, A, b, [], [], \text{lb});
\]
Entering $x$, $\lambda.\text{ineqlin}$, and $\lambda.\text{lower}$ gets
\[
\begin{align*}
x &= \\
&= 0.0000 \\
&\quad 15.0000 \\
&\quad 3.0000 \\
\lambda.\text{ineqlin} &= \\
&= 0 \\
&\quad 1.5000 \\
&\quad 0.5000 \\
\lambda.\text{lower} &= \\
&= 1.0000 \\
&\quad 0 \\
&\quad 0
\end{align*}
\]
Summary

- Linear programming
- Integer/mixed integer programming
  - NP-Completeness
  - Branch-Bound
- LP decomposition methods
- Stochastic heuristics
- Matlab optimization toolbox
Assignment

Exercise #2
- Exercises 5.2 and 5.4
Additional slides: CPLEX/AMPL
Solving LP/IP/MIP with CPLEX-AMPL

- CPLEX is the best LP/IP/MIP optimization engine out there.
- AMPL is a standard programming interface for many optimization engines.
- Student version windows/unix/linux
  - 300 variables
- Maximal Software has a free student version (up to 300 variables): uses CPLEX engine
  - Maximal’s format is slightly different than CPLEX format
Essential Modeling Language Features

- Sets and indexing
  - Simple sets
  - Compound sets
  - Computed sets

- Objectives and constraints
  - Linear, piecewise-linear
  - Nonlinear
  - Integer, network

- . . . and many more features
  Express problems the various way that people do
  Support varied solvers
Consider the following load balancing example

\[
\begin{align*}
\text{minimize} & \quad \max\{x_1/10, x_2/15\} \\
\text{subject to} & \quad x_1 + x_2 = 10 \\
& \quad x_1, x_2 \geq 0.
\end{align*}
\]

Will need to convert to an LP first!

\[
\begin{align*}
\text{minimize} & \quad r \\
\text{subject to} & \quad x_1 + x_2 = 10 \\
& \quad x_1/10 \leq r \\
& \quad x_2/15 \leq r \\
& \quad x_1, x_2 \geq 0.
\end{align*}
\]
In CPLEX notation, type the follow in save in file load-balance.lp

Minimize
   r
Subject to
   demandflow: x1 + x2 = 10
   link1utilization: x1 -10 r <= 0
   link2utilization: x2 -15 r <= 0
End

At CPLEX prompt,

  CPLEX> read load-balance.lp
  CPLEX> optimize
  CPLEX> display solution variables -

  r           0.400000
  x1          4.000000
  x2          6.000000
See Appendix-D (formulation is too big to fit into a slide!)
Each optimization program has 2-3 files

- **optprog.mod**: the model file
  - Defines a class of problems (variables, costs, constraints)

- **optprog.dat**: the data file
  - Defines an instance of the class of problems

- **optprog.run**: optional script file
  - Defines what variables should be saved/displayed, passes options to the solver and issues the solve command
Running AMPL-CPLEX

- Start AMPL by typing `ampl` at the prompt
- Load the model file
  - `ampl: model optprog.mod;` (note semi-colon)
- Load the data file
  - `ampl: data optprog.dat;`
- Issue solve and display commands
  - `ampl: solve;`
  - `ampl: display variable_of_interest;`
- OR, run the run file with all of the above in it
  - `ampl: quit;`
  - `prompt: ~> ampl example.run`
minimizing maximal link utilization

\[
\begin{align*}
\text{minimize} & \quad r \\
\text{subject to} & \quad \sum_{d=1}^{P_d} x_{dp} = h_d \quad d = 1, 2, \ldots, D \\
& \quad \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \leq c_e r \quad e = 1, 2, \ldots, E \\
& \quad x_{dp}, r \text{ continuous, non-negative.}
\end{align*}
\]
AMPL: the model (1)

parameters

\begin{verbatim}
param D > 0 integer;
param E > 0 integer;
param N > 0 integer;
param Pd > 0 integer;
set Nodes := 1..N;
set link_nos := 1..E;
set demand_nos := 1..D;
set route_nos := 1..Pd;
\end{verbatim}

links

\begin{verbatim}
#Generation of links
param link_src {link_nos} within Nodes;
param link_dest {link_nos} within Nodes;
param link_capacity {link_nos} >= 0 integer;
\end{verbatim}

demands

\begin{verbatim}
#Generation of Demands
param demand_src {demand_nos} within Nodes;
param demand_dest {demand_nos} within Nodes;
\end{verbatim}

routes

\begin{verbatim}
#Generation of Routes
set Routes{demand_nos,route_nos} within link_nos;
\end{verbatim}

incidences

\begin{verbatim}
param delta {e in link_nos, d in demand_nos, p in route_nos} = if e in Routes[d,p] then 1 else 0;
\end{verbatim}

flow variables

\begin{verbatim}
var x {d in demand_nos, p in route_nos} >= 0; var r >= 0;
\end{verbatim}
AMPL: the model (II)

**Objective**

```plaintext
#Objective function- Maximize the Throughput
minimize MaxLinkUtil: r;
```

**Constraints**

```plaintext
subj to all_demands {d in demand_nos}:
    sum{p in route_nos} x[d,p] = h[d];

subj to capacity_constraints {e in link_nos}:
    sum{d in demand_nos} ( sum{p in route_nos} (delta[e,d,p] *x[d,p]))
    - link_capacity[e]*r <= 0;
```
AMPL: the data

data;
param D := 2;
param E := 3;
param N := 3;
param Pd := 2;

param: link_src   link_dest   link_capacity :
        1        1        2       20
        2        2        3       10
        3        3        1       10    ;

param: demand_src   demand_dest   h :
        1        1        2       12
        2        2        3       10    ;

set Routes[1,1] := 1;
set Routes[1,2] := 2 3;
set Routes[2,1] := 2;
set Routes[2,2] := 1 3;

derive;