1 Notation

- The vector space is denoted as $\mathbb{R}, \mathbb{R}^{n}, \mathbb{R}^{m \times n}, V, W$
- Matricies are denoted by upper case, italic, and boldface letters: $A_{m \times n}$
- Vectors are column vectors denoted by boldface and lower case letters: $x \in \mathbb{R}^{n \times 1}$
- $1_n \in \mathbb{R}^n$ is a $n \times 1$ vector of all ones
- $I_n$ is $n \times n$ identity matrix.
- $e_i$ is the unit vector where only the $i$-th element is 1 and the rest are 0.
2 Optimization Terms

- General Form

\[
\begin{align*}
\min_x & \quad f(x) \\
\text{s.t.} & \quad g_i(x) \leq b_i, \quad 1 \leq i \leq m \\
& \quad x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}
\end{align*}
\]

- Details:
  - \( x \) is a vector of \( n = n_1 + n_2 \) variables
  - \( g_i \) are called constraint functions
  - \( f \) is called objective function
- The feasible region is:

\[
F = \{ x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} | g_i(x) \leq b_i \}
\]

- A solution is an assignment of values to variable
- An optimal solution \( x^* \) has smallest value of \( f \) among all feasible solutions.
- term optimization vs. term programming
3 Linear Programming
3.1 General Form

- General form:

$$\min_{x} \ c^T x \quad \text{subject to} \quad Ax \leq b$$

- $x \in \mathbb{R}^n$ is a vector of variables
- $c \in \mathbb{R}^n$ is a vector of known coefficients (weights)
- $A \in \mathbb{R}^{m \times n}$ is a matrix. Each of the $m$ rows of the matrix defines the coefficients of a linear inequality.
- $b \in \mathbb{R}^m$ is a vector. Each entry $b_i$ is on the right hand side of inequality $i$. 
3.2 Example

- Example with two variables and two constraints:

\[
\begin{align*}
\text{min } & \quad c_1 x_1 + c_2 x_2 \\
& \quad a_{11} x_1 + a_{12} x_2 \leq b_1 \\
& \quad a_{21} x_1 + a_{22} x_2 \leq b_2
\end{align*}
\]

- More specific example with two variables and two constraints:

\[
\begin{align*}
\text{min } & \quad -4 x_1 - 2 x_2 \\
& \quad x_1 + 2.4 x_2 \leq 12.1 \\
& \quad 7 x_1 \leq 22
\end{align*}
\]

- Graphical Example:

\[
\begin{align*}
\text{max } & \quad 100 x_1 + 64 x_2 \\
& \quad 50 x_1 + 31 x_2 \leq 250 \\
& \quad 3 x_1 - 2 x_2 \geq -4 \\
& \quad x_1 \geq 0 \\
& \quad x_2 \geq 0
\end{align*}
\]

Optimum at \((376/193, 950/193)\)
3.3 How to solve linear programming problems?

- No analytic formula for the solution
- Reliable and efficient algorithms and software, e.g.
  - Simplex algorithm
  - Interior point algorithms
- Computation time proportional to $n^2 m$ if $m \geq n$; less with structure
- Formulating a problem as linear programming problem is already non-trivial
3.4 From linear programming to linear integer programming

- Optimization problem:
  \[
  \min_{x} \quad c^T x \\
  \text{s.t.} \quad A x \leq b
  \]

- floating point variables
  - \( x \in \mathbb{R}^n \)
  - linear program (LP)

- integer variables
  - \( x \in \mathbb{Z}^n \)
  - (linear) integer program (IP)

- binary variables
  - \( x \in \{0, 1\}^n \)

- float and integer variables
  - \( x \) is split into two groups of variables, \( x_F \) and \( x_I \)
  - \( x_F \in \mathbb{R}^{n_1} \) and \( x_I \in \mathbb{Z}^{n_2} \)
  - mixed integer program (MIP)
3.5 Variations of the standard form

- Optimization problem:

\[
\begin{align*}
\min_x & \quad c^T x \\
Ax & \leq b
\end{align*}
\]

- switch min and max
- switch \(\leq\) and \(\geq\)
- include constraints with \(=\) as separate category
- require all variables to be positive (\(\geq 0\))

- Example Optimization problem:

\[
\begin{align*}
\max_x & \quad c^T x \\
Ax & \leq b \\
x & \geq 0
\end{align*}
\]
3.6 Comments about formulations

**Definition 1.** A polyhedron $P$ is a subset of $\mathbb{R}^n$ described by a finite set of linear constraints. $P = \{ x \in \mathbb{R}^n : Ax \leq b \}$

**Definition 2.** A polyhedron $P \subseteq \mathbb{R}^{n_1+n_2}$ is a formulation for a set $X \subseteq \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}$ if and only if $X = P \cap (\mathbb{Z}^{n_1} \times \mathbb{R}^{n_2})$.

**Definition 3.** A convex combination of points from a set $S$, $x_1, x_2, ..., x_k \in S$, is any point of form $\theta_1 x_1 + \theta_2 x_2 + ... + \theta_k x_k$, where $\theta_i \geq 0, i = 1,...,k, \sum_{i=1}^{k} \theta_i = 1$. A set $S$ is convex iff any convex combination of points in $S$ is in $S$.

**Definition 4.** The convex hull $\text{conv } S$ is the set of all convex combinations of points in $S$

- The formulation has to enclose all feasible integer points, but no infeasible integer points
- Runtime depends on
  - number of variables
  - number of constraints
  - tightness of fit
- Formulation $A$ is at least as strong as $B$ if $A \subseteq B$
- Formulation $A$ is stronger than $B$ if $A \subset B$
- A formulation $A$ is ideal if $\text{conv}(\text{feasible solutions}) = A$
3.7 Graphical Example

\[
\begin{align*}
\text{max } & \quad 100x_1 + 64x_2 \\
\text{s.t. } & \quad 50x_1 + 31x_2 \leq 250 \\
& \quad 3x_1 - 2x_2 \geq -4 \\
& \quad x_1 \geq 0 \\
& \quad x_2 \geq 0 \\
& \quad x_1, x_2 \in \mathbb{Z}
\end{align*}
\]

- Rounded solution might not be feasible
- Rounded solution might be far from optimal solution
3.8 Different Components of Optimization in the literature

- Modeling:
  - How to formulate an application problem as a standard optimization problem?
- Algorithm Development:
  - How to derive new optimization algorithms for standard optimization problems?
  - How to derive new optimization algorithms for specialized optimization problems?
- Optimization Theory:
  - Finding convergence guarantees, bounds, ... of optimization algorithms
3.9 Different Components of Optimization in Visual Computing

- **Modeling:**
  - propose an interesting problem formulation for a new or an existing problem in visual computing?

- **Algorithm Development:**
  - propose a new algorithm for a specific optimization problem in visual computing

- **Modeling + Algorithm Development**

- **Theory**
  - typically not done in visual computing, but in optimization and machine learning
3.10 How to solve an IP Problem?

- use a standard solver such as Matlab, Gurobi, Mosek, ... and see what happens
- create a new heuristic solver
3.11 Branch and Bound

- How to create upper and lower bounds for (the objective value of) the solution?
  - The LP relaxation is a lower bound for the optimal solution
  - Any particular feasible solution is an upper bound for the optimal solution
- If we solve the LP relaxation of an MILP problem we distinguish 3 cases:
  - LP is infeasible → MILP is infeasible
  - Optimal LP solution is feasible solution for MILP problem → optimal solution
  - LP is feasible and optimal LP solution is not feasible for MILP → lower bound
- First two cases we are finished, third case we branch (recursively)
- The most common way to branch is to do the following
  - Select a variable $i$ whose value $\hat{x}_i$ is fractional in the LP solution
  - Create two subproblems:
    - Add constraint $x_i \leq \lfloor \hat{x}_i \rfloor$
    - Add constraint $x_i \geq \lceil \hat{x}_i \rceil$
4 Example Problems
4.1 Knapsack Problem

- **Input:**
  - a set of items $i$ with values $v_i$ and weights $w_i$
  - a knapsack with maximum capacity $c$

- **Goal:** pack a subset of items into the knapsack, such that
  - the sum of weights does not exceed the capacity $C$
  - the sum of the values is maximized

- **Example**

```
C = 10
```

```
w_1 = 5, v_1 = 3
```

```
w_2 = 8, v_2 = 7
```

```
w_3 = 3, v_3 = 5
```

- **Formulation:**
  - variables: $x_i = 1$ means we pack item $i$

```
\min_x \ v^T x
\text{subject to} \ w^T x \leq c
\quad x_i \in \{0, 1\}
```

- **Difficulty:**
  - NP-hard
  - (pseudo-polynomial) Dynamic Programming solution exists for integer weights and capacity.
4.2 Matlab Code

C = 750
weights = [70; 73; 77; 80; 82; 87; 90; 94; 98; 106; 110; 113; 115; 118; 120];
values = [135; 139; 149; 150; 156; 163; 173; 184; 192; 201; 210; 214; 221; 229; 240];
LZero = zeros(length(weights),1);
LOne = ones(length(weights),1);  
LCount = 1:length(weights);
tic;
tic;
intlinprog( -values, LCount, weights’, C, [], [], LZero, LOne)
toc;
4.3 Map Labeling

- **Input:**
  - a set of map objects $i$ where each object has a discrete set of possible label positions $j$
  - costs $c$ for each label placement
- **Goal:** place at least one label per object without overlap
- **Illustration:** two cities one river

- **Variables**
  - $x_{ij} = 1$ if label for object $i$ is placed at position $j$
- **Constraints:**
  - Binary constraints:
    \[ x_{ij} \in \{0, 1\} \]
  - Coverage constraint - each element is labeled exactly once:
    \[ \forall i \sum_j x_{ij} = 1 \]
  - Non-overlap for conflicting placements:
    - for each pair of overlapping placements $ij$ and $lm$
      \[ x_{ij} + x_{lm} \leq 1 \]
- **Objective:** \[ \min \sum_i \sum_j c_{ij} x_{ij} \]
4.4 Assignment Problem

- **Input:**
  - $n$ people to carry out $n$ jobs
  - $c_{ij}$: cost of assigning person $i$ to job $j$

- **Goal:** assign each person to exactly one job, so that each job has one person assigned to it.

- **Illustration:**

- **Variables**
  - $x_{ij} = 1$ if person $i$ is assigned to job $j$

- **Objective:**

\[
\min \sum_i \sum_j c_{ij} x_{ij}
\]

- **Constraints:**
  - Binary constraints:
    \[
x_{ij} \in \{0, 1\}
    \]
  - Limited work: each person $i$ does exactly one job
    \[
    \forall i \sum_j x_{ij} = 1
    \]
  - Coverage constraint - each job is done by one person:
    \[
    \forall j \sum_i x_{ij} = 1
    \]

- **Difficulty:**
  - Hungarian Method (Kuhn–Munkres algorithm or Munkres assignment algorithm)
  - Auction algorithm
4.5 Tourist Map Layout

- Input:
  - overview map with Points of Interest (POIs)
  - detail maps for each POI
  - positions for detail maps
  - costs $c_{ij}$ for assigning POI $i$ detail map position $j$
- Goal: assign each detail map to one position.
- Illustration:

![Illustration of POI detail map assignment](image)

- Variables
  - $x_{ij} = 1$ if map $i$ is assigned to position $j$
- Objective:
  $$\min \sum_i \sum_j c_{ij}x_{ij}$$
- Constraints:
  - Binary constraints:
    $$x_{ij} \in \{0, 1\}$$
  - Each map $i$ is assigned once
    $$\forall i \sum_j x_{ij} = 1$$
  - No overlap between maps:
    $$\forall j \sum_{(i,j) \in O_j} x_{ij} = 1$$
    ○ $O_j$ is the set of all placements that overlap position $j$
4.6 Tiling

- Input:
  - a set of tiles $i$
  - a domain consisting of positions $j$
  - costs $c_{ij}$ for assigning tile $i$ to position $j$
  - minimum and maximum number of times tile $i$ is allowed to be used ($min_i, max_i$)

- Goal: cover the domain with the given tiles

- Illustration:

- Variables
  - $x_{ij} = 1$ if leftmost square of tile $i$ is assigned to position $j$

- Objective:
  $$\min \sum_i \sum_j c_{ij} x_{ij}$$

- Constraints:
  - Binary constraints:
    $$x_{ij} \in \{0, 1\}$$
  - Each tile $i$ is assigned between its within its allowed limits
    $$\forall i \quad min_i \leq \sum_j x_{ij} \leq max_i$$
  - No overlap between squares in the domain:
    $$\forall j \quad \sum_{(i,j) \in O_j} x_{ij} = 1$$
    - $O_j$ is the set of all tile placements that overlap position $j$
4.7 Shape Matching

- Input:
  - two shapes where each shape has $n$ vertices.
  - a cost $c_{ij}$ for assigning vertex $i$ from shape 1 to vertex $j$ on shape 2,
- Goal: assign each vertex on shape 1 to exactly one vertex on shape 2
- Formulation: identical to the assignment problem
- Literature:
### 4.8 Camera Placement

- **Input:**
  - a domain sampled into positions $p$
  - a set of possible camera positions $i$
- **Goal:** select a minimal set of cameras that cover the domain
- **Illustration:**

  ![Illustration of camera placement](image)

- **Variables**
  - $x_i = 1$ if camera position $i$ is selected
- **Objective:**
  $$\min \sum_i x_i$$
- **Constraints:**
  - Binary constraints:
    $$x_i \in \{0, 1\}$$
  - Position conflict constraints
    $$\forall i \sum_{j \in N_i} x_j \leq 1$$
  - $N_i$ is the set of locations that conflict with location $i$
  - Visibility constraint:
    $$V x \geq 1$$
    - the $i^{th}$ column of $V$ is a binary mask that encodes what positions are seen by camera $i$
4.9 Graph Review

- Graph \((V, E)\)
  - \(V\) is a set of nodes
  - \(E\) is a set of edges
- \(E(S) = \{e = (i, j) : i, j \in S\}\)
- \(\delta(S) = \{e = (i, j) : i \in S\text{ and } j \in V \setminus S\}\)
- \(\delta(i)\) are all edges incident to node \(i\).
- A tree is a connected graph with \(|V| - 1\) edges.

4.10 Minimum Spanning Tree

- Input:
  - a graph \((V, E)\)
  - the cost \(c_e\) for selecting edge \(e \in E\).
- Goal: find a minimum cost spanning tree
- Variables
  - \(x_e = 1\) if edge \(e\) is selected
- Binary constraints:
  \[x_e \in \{0, 1\}\]
- Number of edges constraint:
  \[\sum_{e \in E} x_e = n - 1\]
- Cut constraint:
  \[\forall S \subset V, S \neq \emptyset, V \sum_{e \in \delta(S)} x_e \geq 1\]
- Objective function:
  \[\min \sum_{e \in E} c_e x_e\]
- We call the linear relaxation of this formulation \(P_{\text{cut}}\)
- Alternative constraint: subtour elimination constraint
  \[\forall S \subset V, S \neq \emptyset, V \sum_{e \in E(S)} x_e \leq |S| - 1\]
- We call the resulting linear relaxation of the formulation \(P_{\text{sub}}\)
- Notes:
  - \(P_{\text{sub}}\) is the convex hull of the set of feasible solutions.
  - \(P_{\text{sub}}\) is a strictly better formulation than \(P_{\text{cut}}\).
4.11 Traveling Salesman

- Input:
  - a graph \((V, E)\)
  - the cost \(c_e\) for selecting edge \(e \in E\).
- Goal: find a minimum cost tour
- Variables
  - \(x_e = 1\) if edge \(e\) is selected
- Binary constraints:
  \[ x_e \in \{0, 1\} \]
- Number of incident edges constraint:
  \[ \forall i \sum_{e \in \delta(i)} x_e = 2 \]
- Cut constraint:
  \[ \forall S \subset V, S \neq \emptyset, \sum_{e \in \delta(S)} x_e \geq 1 \]
- Objective function:
  \[ \min \sum_{e \in E} c_e x_e \]
- Alternative constraint: subtour elimination constraint
  \[ \forall S \subset V, 2 \leq |S| \leq |V| - 1 \sum_{e \in E(S)} x_e \leq |S| - 1 \]
- Similarly, we call the resulting linear relaxations \(P_{cut}\) and \(P_{sub}\)
  - \(P_{cut} = P_{sub}\)
  - Neither is the convex hull of the feasible points
4.12 City Exploration

- Input:
  - a city map as graph \((V, E)\)
  - \(c \in \mathbb{R}^{|E|}\) - the attractiveness of each edge
  - \(t \in \mathbb{R}^{|E|}\) - time it takes to walk along an edge
  - \(T\) - maximum time for the walk
  - a designated start node \(s\) and end node \(e\)
- Goal: find a walk through the city from start node to end node that explores the most attractive edges but stays under the time limit.

Illustration

- Variables
  - \(x_i = 1\) if edge \(i\) is selected
  - \(v_j = 1\) if vertex \(j\) is selected
- Binary constraints:
  - \(x_i, v_j \in 0, 1\)
- Time constraint:
  - \(t^T x \leq T\)
- Connection constraint:
  - \(\sum_{i \in N_j} x_i = v_j\)
  - \(\sum_{i \in N_s} x_i = 1\)
  - \(\sum_{i \in N_e} x_i = 1\)
- \(N_j\) is the set of edges incident to vertex \(j\)
- Objective function:
  - \(\max c^T x\)
• Cycles:
  – the formulation can create closed cycles
  – solution 1: lazy constraint adding
  – solution 2: add constraints that forbid cycles (similar to MST and TS formulations)
5 MIP Modeling Techniques

5.1 AND of variables

• "y is true if all elements in x are true. y is false otherwise."

\[ y = x_0 \land x_1 \land \ldots \land x_{N-1} \]

• y and x are Boolean variables. \( x_0, x_1, \ldots, x_{N-1} \) are the elements in x. N is the size of x.
• Trivial way to model:

\[ y = x_0 x_1 \ldots x_{N-1} \]

It is not going to work!
• As linear inequalities:

\[ 0 \leq \sum x - N y \leq N - 1 \]

• Example:
  – Vertex configurations in a 2D triangle-quad hybrid mesh:

\[ C_{j,m} \]

\[ E_0 \quad E_1 \quad E_2 \quad E_3 \quad E_4 \quad E_5 \quad E_6 \quad E_7 \quad E_8 \quad E_9 \quad E_{10} \quad E_{11} \]

\[ C_{j,m} \text{ is the } m\text{-th configuration for vertex } v_j. \ C_{j,m} \text{ contains } E_1, E_4, E_6, E_9, \text{ and } E_{11} \text{ out of } v_j' \text{'s twelve adjacent edges:} \]

\[ C_{j,m} = \overline{E_0} \land E_1 \land \overline{E_2} \land \overline{E_3} \land E_4 \land \overline{E_5} \land E_6 \land \overline{E_7} \land \overline{E_8} \land E_9 \land \overline{E_{10}} \land E_{11} \]

As linear inequalities:

\[ 0 \leq (1-E_0)+E_1+(1-E_2)+(1-E_3)+E_4+(1-E_5)+E_6+(1-E_7)+(1-E_8)+E_9+(1-E_{10})+E_{11}-12y \leq 11 \]
5.2 OR of variables

- "$y$ is true if any element in $x$ is true. $y$ is false otherwise."
  
  $$y = x_0 \lor x_1 \lor ... \lor x_{N-1}$$

- As linear inequalities:
  
  $$-N + 1 \leq \sum x - Ny \leq 0$$

- Example:
  
  - Converge constraint: a vertex is "covered" if and only if at least one of the edges that are within a close proximity is selected.
    
    $$v_i = e_0 \lor e_1 \lor ... \lor e_{N-1}$$

    $v_i$ is the Boolean variable indicating if the vertex is covered. $e_0, e_1, ..., e_{n-1}$ are Boolean variables of edges within a close proximity to the vertex.

    - For a minimal-vertex cover problem, we may require that the coverage variables of all vertices are true while minimizing the number of selected edges.
5.3 XOR of variables

- "y is true if elements in \( x \) sum to odd. y is false if elements in \( x \) sum to even."

\[
y = x_0 \oplus x_1 \oplus ... \oplus x_{N-1}
\]

- As linear inequalities:

\[
y = x_0 + x_1 + ... + x_{N-1} - 2t
\]

\( t \) is an integer slack variable. \( 0 \leq t \leq N - 1 \).

- Alternatively, model it as a sequence of 2-inputs XORs (the \( t \) variables become Booleans).
5.4 Special order set (SOS)

- Special Ordered Sets of type 1 (SOS1):
  - Given an ordered set of variables, \(q\), at most one element in \(q\) can be non-zero.

- Special Ordered Sets of type 2 (SOS2):
  - Given an ordered set of variables, \(q\), at most two elements in \(q\) can be non-zero. And if two elements are non-zero, they must be consecutive in their ordering.

- Supported by popular MIP solvers such as Gurobi and IBM CPLEX. These solvers use special branching strategies to take advantage of SOSs.

- Examples:
  - A SOS1 set, \(x\), of Boolean variables \(x_0, x_1, ..., x_{N-1}\), means that:
    \[
    x_0 + x_1 + ... + x_{N-1} \leq 1
    \]
  - SOS2: "knight8" template for translational symmetry in urban layout design:

5.5 Exhaustive enumeration of all feasible solutions of a (Boolean) IP problem

- Let $Z$ denote a feasible solution of an IP problem with only Boolean variables. We can forbid $Z$ to be feasible, that is,

$$Z \land F = \emptyset$$

where $F$ is the feasible region of the problem, by adding the following constraint:

$$\sum_{0 \leq i \leq N-1} (x_i \text{ if } Z_i \text{ is true, or } (1 - x_i) \text{ if } Z_i \text{ is false}) \leq N - 1$$

to the IP formulation. $x$ denotes the variables. $N$ is the number of variables.

- An enumeration of unique feasible solutions can be done by repeatedly solving the IP problem with all previously retrieved solutions forbidden.

- An exhaustive enumeration proceeds until the problem becomes infeasible.

- Examples:
  - Given a IP with three Boolean variables, $x_0$, $x_1$, and $x_2$, adding the following constraint would forbid $(0, 1, 0)$ as a feasible solution:
    $$(1 - x_0) + x_1 + (1 - x_2) \leq 2$$
    - Exhaustive enumeration of triangle-quad tilings in a 12-gon with side length 2.
5.6 **Big-$M$ method**

- Use Boolean slack variables with sufficiently large coefficients to allow constraints to be "de-activated".
- That is, rewriting a linear constraint:

\[ a^T x \leq b \]

...to be:

\[ a^T x \leq b + M y \]

...would allow it to be violated. $M$ is a sufficiently large positive constant and $y$ is a Boolean slack variable. When it is violated, $y$ is true.
- Optionally, add $y$ to the objective function (to minimize) to introduce penalty for the constraints to be violated.
• Example:
  - "Constrain the union of two (mutually exclusive) constraints to be true":

\[ a_0^T x_0 \leq b_0 \quad \text{or} \quad a_1^T x_1 \geq b_1 \]

○ As linear inequalities:

\[ a_0^T x_0 \leq b_0 + M(1 - y) \]
\[ a_1^T x_1 \geq b_1 - My \]

where \( M \) is a sufficiently big positive constant and \( y \) is a Boolean slack variable.

○ Example:

\[ x \leq 2 \quad \text{or} \quad x \geq 6 \]

is reformulated as:

\[ x \leq 2 + M(1 - y), \quad x \geq 6 - My \]

• Discussions
  - Many modeling techniques in MIP are variations of the big-\( M \) method.
  - In general, big-\( M \) methods are more preferable than the equivalent non-linear formulations.
  - \( M \) should be kept as small as possible. Very big \( M \) impacts performance.

• Literature:
6 Quadratic Programming
6.1 General Form

- General form:

\[
\min_x \frac{1}{2} x^T Q x + c^T x \\
A x \leq b
\]

- \( x \in \mathbb{R}^n \) is a vector of variables
- \( c \in \mathbb{R}^n \) is a vector with known entries
- \( Q \in \mathbb{R}^{n \times n} \) is a symmetric matrix with known entries
- \( A \in \mathbb{R}^{m \times n} \) is a matrix. Each of the \( m \) rows of the matrix define the coefficients of a linear inequality.
- \( b \in \mathbb{R}^m \) is a vector. Each entry \( b_i \) is on the right hand side of inequality \( i \).

6.2 Comments

- if \( Q \succ 0 \) (the matrix is positive-definite) the optimization is convex
7 Quadratic Integer Programming Examples

7.1 Quadratic Assignment

- Input:
  - a set of \( n \) facilities \( i \)
  - a set of \( n \) possible facility location \( j \)
  - costs \( c_{ijkl} \) for assigning facility \( i \) to location \( j \) and facility \( k \) to location \( l \)
- Goal: assign facilities to grid cells to minimize costs
- Variations:
  - costs \( c_{ijkl} \) can be modeled arbitrarily
  - costs \( c_{ijkl} \) are modeled as the product \( c_{ijkl} = f_{ik}d_{jl} \), where \( f_{ik} \) is a flow between facility \( i \) and \( k \) and \( d_{jl} \) is a distance between \( j \) and \( l \). This is the classical quadratic assignment problem.
- Variables
  - \( x_{ij} = 1 \) if facility \( i \) is assigned to location \( j \)
- Objective:
  \[
  \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} c_{ijkl}x_{ij}x_{kl}
  \]
- Constraints:
  - Binary constraints:
    \( x_{ij} \in \{0,1\} \)
  - Non-overlap: each facility \( i \) has exactly one position
    \[
    \forall i \sum_{j} x_{ij} = 1
    \]
  - Coverage: each position is covered by exactly one facility
    \[
    \forall j \sum_{i} x_{ij} = 1
    \]
7.2 Quadratic Assignment for Images

- **Input:**
  - a set of $n$ images with image distances $d_{ij}$
  - a set of $n$ possible image positions with distances $g_{kl}$
  - costs $c_{ijkl} = f(d_{ik}, g_{jl})$

- **Goal:** assign images to grid cells to minimize the costs

- **Variables**
  - $x_{ij} = 1$ if image $i$ is assigned to grid cell $j$

- **Objective:**
  $$\min \sum_{i}^{n} \sum_{j}^{n} \sum_{k}^{n} \sum_{l}^{n} c_{ijkl} x_{ij} x_{kl}$$

- **Constraints:**
  - Binary constraints:
    $$x_{ij} \in \{0, 1\}$$
  - Non-overlap: each image $i$ has exactly one position
    $$\forall i \sum_{j} x_{ij} = 1$$
  - Coverage: each position is covered by exactly one image
    $$\forall j \sum_{i} x_{ij} = 1$$

- **Literature:** Fried et al., "IsoMatch: Creating Informative Grid Layouts", Eurographics 2015.
7.3 Quadratic Assignment for Shape Matching

- Literature:
  - Kezurer et al., Tight Relaxation of Quadratic Matching, SGP 2015.

7.4 Joint Segmentation

- Input:
  - Two shapes. Each shape is subdivided into smaller patches $P_1$ and $P_2$, respectively
  - A set of candidate segments for each shape: $S_1$ and $S_2$. Each segment consists of multiple patches.
  - A cost vector $c$ where $c_{ij}$ is the cost selecting a segment $j$ in shape $i$.
  - A cost vector $d$ where $d_{ij}$ encodes the cost of mapping segment $i$ in shape one to segment $j$ in shape two.
  - A cost matrix $Q$ where $q_{ijkl}$ encodes the cost of mapping segment $i$ in shape one to segment $j$ in shape two and segment $k$ in shape one to segment $l$ in shape two.

- Variables:
  - $x_{ij} = 1$ if segment $j$ is selected from shape $i$.
  - $p_{ij} = 1$ if patch $j$ is selected from shape $i$.
  - $m_{ij}$ if segment $i$ in shape one maps to segment $j$ in shape two.

- Literature:
  - Huang et al., Joint-Shape Segmentation with Linear Programming, ACM TOG 2011.
7.5 Fit and Diverse Sampling
8 Quadratically Constrained Quadratic Programming
8.1 General Form

- General form:

$$\min_{x} \frac{1}{2} x^T Q_0 x + c_0^T x$$

$$x^T Q_i x + c_i^T x \leq b_i$$

- $x \in \mathbb{R}^n$ is a vector of variables
- $c_i \in \mathbb{R}^n$ are vectors with known entries
- $Q_i \in \mathbb{R}^{n \times n}$ are symmetric matrices with known entries
- $b \in \mathbb{R}^m$ is a vector. Each entry $b_i$ is on the right hand side of inequality $i$. 
8.2 Mixed Integer Quadratically Constrained Programming

- Can be solved by commercial solvers