Optimization (168)

Lecture 1-2

Jesús De Loera

UC Davis, Mathematics

Wednesday, April 2, 2012

The Brewery Problem

A brewery produces barrels of ale or beer, each needs fixed amount of corn, hops, and malt. They only have 320 lbs. of corn, 130 ozs. of hops and 1120 lbs. of malt. Each barrel of ale yields a profit of $16 and each barrel of beer yields a profit of $12.

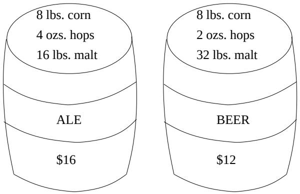


Figure: **How many barrels of ale beer should be produced to maximize total profit given limited resources?**

A diet problem

WISH Minimize the ”cost of the menu” subject to the nutrition and preference requirements:

- Eat enough but not too much of Vitamin A
- Eat enough but not too much Vitamin C
- Eat at least a certain minimum number of servings of beef but not more than the maximum number of servings of beef you want.
- Eat at least a certain minimum number of servings of carrots but not more than the maximum number of servings of carrots you want.

To create a **mathematical model**, we need to define some **variables** and **constraints** :

- $x(\text{beef}) =$ servings of beef in the menu,
- $x(\text{carrots}) =$ servings of carrots in the menu, etc.
- But each variable has some restrictions!
    - cost(beef) = cost per serving of beef
    - min (beef) = minimum number of servings to eat
    - max (beef) = maximum number of servings to eat
    - A(beef) = amount of Vitamin A in one serving of beef
    - C(beef) = amount of Vitamin C in one serving of beef, etc.
- Finally we obtain

Minimize cost(beef) * x(beef) + . . . + cost(carrots) * x(carrots)
Subject to:
min(C) $\leq$ C(beef) * x(beef) + . . . + C(carrots) * x(carrots) $\leq$ max(C),
min(A) $\leq$ A(beef) * x(beef) + . . . + A(carrots) * x(carrots) $\leq$ max(A), etc.
min(beef) $\leq$ x(beef) $\leq$ max(beef),
min(carrots) $\leq$ x(carrots) $\leq$ max(carrots), etc.

Another example

… Company sells two models of … five-leg tables. The basic version uses a wood top, requires 0.6 hours to assemble, and sells for a profit of \$200. The deluxe model takes 1.5 hours to assemble and sells for a profit of \$350. Over the next week … 300 legs, 50 wood tops, 35 glass tops, and 63 hours of assembly available. …determine a maximum profit production plan …

We developed a **mathematical model** for this problem.

We are using two variables to represent the decision on the production quantities:

x_1 – *number of basic tables,* x_2 – *number of deluxe tables*

$$
\begin{array}{rll}
\max & 200x_1 + 350x_2 & \text{(Total profit)} \\
\text{s.t.} & x_1 \leq 50 & \text{(Wood tops available)} \\
& x_2 \leq 35 & \text{(Glass tops available)} \\
& 5x_1 + 5x_2 \leq 300 & \text{(Legs available)} \\
& 0.6x_1 + 1.5x_2 \leq 63 & \text{(Hours of assembly time available)} \\
& x_1, x_2 \geq 0 & \text{(Non-negative quantities to be produced)} \\
& x_1, x_2 \in \mathbf{Z} & \text{(Integer quantities to be produced)}
\end{array}
$$

Data Analysis, Regression

- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.

Data Analysis, Regression

- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.
- At different times t_i we measure a quantity b_i. E.g, Say a police man is interested on clocking the speed of a vehicle by using measurements of its relative distance. Assuming the vehicle is traveling at constant speed, so we know linear formula, but errors exist.

Data Analysis, Regression

- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.
- At different times t_i we measure a quantity b_i. E.g, Say a police man is interested on clocking the speed of a vehicle by using measurements of its relative distance. Assuming the vehicle is traveling at constant speed, so we know linear formula, but errors exist.
- At $t = t_i$, the error between the measured value b_i and the value predicted by the function is $e_i = b_i - (\alpha + \beta t_i)$.

Data Analysis, Regression

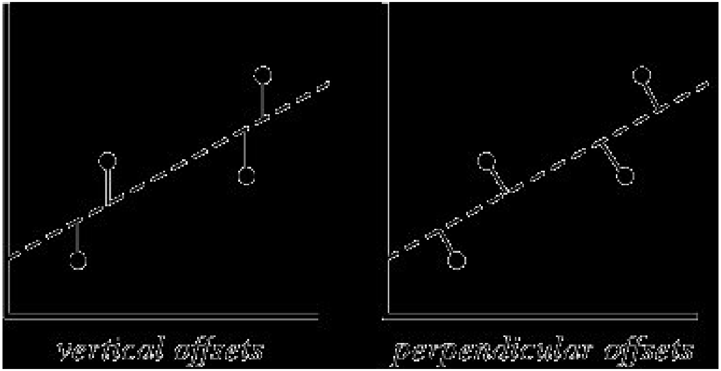
- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.
- At different times t_i we measure a quantity b_i. E.g, Say a police man is interested on clocking the speed of a vehicle by using measurements of its relative distance. Assuming the vehicle is traveling at constant speed, so we know linear formula, but errors exist.
- At $t = t_i$, the error between the measured value b_i and the value predicted by the function is $e_i = b_i - (\alpha + \beta t_i)$.
- We can write it as $e = b - Ax$ where $x = (\alpha, \beta)$. e is the **error vector**, b is the **data vector**. A is an $m \times 2$ matrix.

Data Analysis, Regression

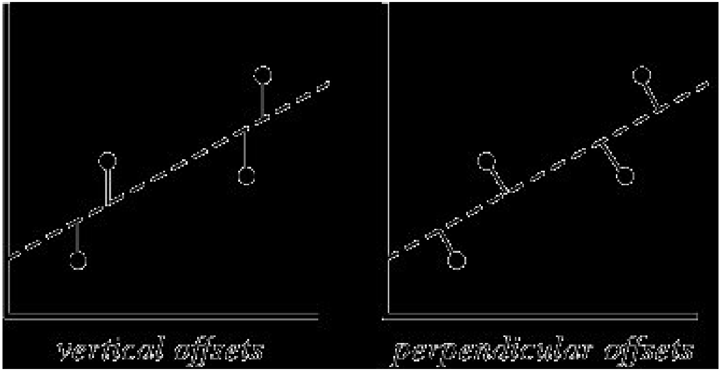
- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.
- At different times t_i we measure a quantity b_i. E.g, Say a police man is interested on clocking the speed of a vehicle by using measurements of its relative distance. Assuming the vehicle is traveling at constant speed, so we know linear formula, but errors exist.
- At $t = t_i$, the error between the measured value b_i and the value predicted by the function is $e_i = b_i - (\alpha + \beta t_i)$.
- We can write it as $e = b - Ax$ where $x = (\alpha, \beta)$. e is the **error vector**, b is the **data vector**. A is an $m \times 2$ matrix.
- We seek the line that minimizes the ERROR!!!

Data Analysis, Regression

- Suppose we do a series of experiments, collecting data. Suppose we expect the output b to be a linear function of the input t $b = \alpha + t\beta$, but we need to **determine** α, β.
- At different times t_i we measure a quantity b_i. E.g, Say a police man is interested on clocking the speed of a vehicle by using measurements of its relative distance. Assuming the vehicle is traveling at constant speed, so we know linear formula, but errors exist.
- At $t = t_i$, the error between the measured value b_i and the value predicted by the function is $e_i = b_i - (\alpha + \beta t_i)$.
- We can write it as $e = b - Ax$ where $x = (\alpha, \beta)$. e is the **error vector**, b is the **data vector**. A is an $m \times 2$ matrix.
- We seek the line that minimizes the ERROR!!!
- BUT WHAT DO WE MEAN BY ERROR??



We present three choices!! Remember e_i is the error $e_i = b_i - \sum_{j=1}^{n} a_{ij} x_j$.



We present three choices!! Remember e_i is the error $e_i = b_i - \sum_{j=1}^{n} a_{ij} x_j$.

1. Minimize $\sqrt{\sum_{i=1}^{m} e_i^2}$ This is just linear algebra!!
2. Minimize $\sum_{i=1}^{m} |e_i|$
3. Minimize $\max_i |e_i|$

- If we wish to minimize $\sum_{i=1}^{m} |b_i - \sum_{j=1}^{n} a_{ij}x_j|$.
- Consider the linear program
  Minimize $\sum_i^m e_i$
  subject to: $e_i + \sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad (i = 1 \ldots m)$.
  $e_i - \sum_{j=1}^{n} a_{ij}x_j \geq -b_i \quad (i = 1 \ldots m)$.
- Thus the optimal solution
  $e_i^* \geq max(b_i - \sum_{j=1}^{n} a_{ij}x_j^*, -b_i + \sum_{j=1}^{n} a_{ij}x_j^*) = |b_i - \sum_{j=1}^{n} a_{ij}x_j^*|$
- Suppose now we wish to minimize $max(|b_i - \sum_{j=1}^{n} a_{ij}x_j|)$. Consider the linear program
  Minimize z
  subject to: $z + \sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad (i = 1 \ldots m)$.
  $z - \sum_{j=1}^{n} a_{ij}x_j \geq -b_i \quad (i = 1 \ldots m)$.
- We reach now the conclusion that z^* is bounded below by the maximum, thus x^* must be the best approximation with MinMax error.

THE SIMPLEX METHOD

LINEAR PROGRAMS (LPs)

Optimal decisions by solving **Linear Programming Problems:**

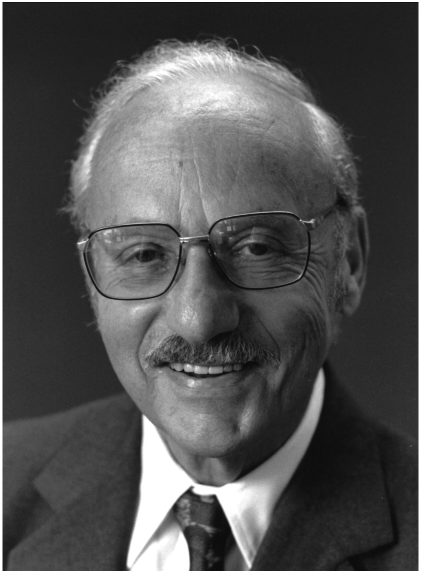
$$\text{maximize } C_1 x_1 + C_2 x_2 + \cdots + C_d x_d$$

$$\text{among all } \; x_1, x_2, \ldots, x_d, \text{satisfying:}$$

$$\begin{gathered}
a_{1,1} x_1 + a_{1,2} x_2 + \cdots + a_{1,d} x_d \leq b_1 \\
a_{2,1} x_1 + a_{2,2} x_2 + \cdots + a_{2,d} x_d \leq b_2 \\
\vdots \\
a_{k,1} x_1 + a_{k,2} x_2 + \cdots + a_{k,d} x_d \leq b_k
\end{gathered}$$

The simplex method

George Dantzig invented the SIMPLEX METHOD when he was working for the Air Force in the 1940's.



Simple form of a Mathematical Optimization Problem

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_1(\mathbf{x}) \leq 0 \\ & \vdots \\ & g_m(\mathbf{x}) \leq 0 \\ & \mathbf{x} = (x_1, \ldots, x_n) \in X \end{aligned}$$

where $f, g_1, \ldots, g_m \colon \mathbf{R}^n \rightarrow \mathbf{R}$ are functions and $X = \mathbf{R}^n$, or $X = \mathbf{Z}^n$, or $X = \mathbf{R}^{n_1} \times \mathbf{Z}^{n_2}$.

We classify optimization problems according to the properties of the functions f and g_i and the space X.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at $2.5\,\mathrm{GHz}$ could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at 2.5 GHz could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

High dimension

More impressive examples appear when we have many variables, even if they have low bounds.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at $2.5\,\mathrm{GHz}$ could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

High dimension

More impressive examples appear when we have many variables, even if they have low bounds. In a combinatorial problem with 100 binary variables ($x_i = 0$ or $x_i = 1$), we would have to check 2^{100} solutions.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \le x_i \le 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at 2.5 GHz could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

High dimension

More impressive examples appear when we have many variables, even if they have low bounds. In a combinatorial problem with 100 binary variables ($x_i = 0$ or $x_i = 1$), we would have to check 2^{100} solutions. A current supercomputer, IBM's Blue Gene/P, has 884,736 processors running at 850 MHz. It would take $1.6 \cdot 10^{15}$ years.

Can we solve it via Brute Force?

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

The brute force method is not useful because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at $2.5\,\mathrm{GHz}$ could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

High dimension

More impressive examples appear when we have many variables, even if they have low bounds. In a combinatorial problem with 100 binary variables ($x_i = 0$ or $x_i = 1$), we would have to check 2^{100} solutions. A current supercomputer, IBM's Blue Gene/P, has 884,736 processors running at 850 MHz. It would take $1.6 \cdot 10^{15}$ years. The age of the universe is only $(13.7 \pm 0.1) \cdot 10^9$ years (Wikipedia).

How to solve this problem, III: Using optimization software

We will be using three pieces of software in this lecture. (All of them are free at least for noncommercial and academic use. ZIMPL is free (open source) software.)

SCIP – "Solving Constraint and Integer Programs"

SCIP is a very good solver for **Mixed Integer Linear Optimization Problems**, i.e.,

$$\begin{aligned} \max \quad & f(\mathbf{x}) & \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 & \text{for } i = 1, \dots, m \\ & \mathbf{x} = (x_1, \dots, x_n) \in X & \end{aligned}$$

where f and g_i are linear functions, and $X = \mathbf{R}^n$, or $X = \mathbf{Z}^n$, or $X = \mathbf{R}^{n_1} \times \mathbf{Z}^{n_2}$.

SoPLEX – a solver for linear optimization problems

A super-fast solver for linear optimization problems ($X = \mathbf{R}^n$) is the basic building block of optimization and operations research technology.

ZIMPL – an algebraic modeling language for optimization

ZIMPL provides a convenient way to write down optimization problems in the computer.

How to solve this problem, III: Using optimization software

We will be using three pieces of software in this lecture. (All of them are free at least for noncommercial and academic use. ZIMPL is free (open source) software.)

SCIP – "Solving Constraint and Integer Programs"

SCIP is a very good solver for **Mixed Integer Linear Optimization Problems**, i.e.,

$$\begin{aligned} \max \quad & f(\mathbf{x}) & \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 & \text{for } i = 1, \dots, m \\ & \mathbf{x} = (x_1, \dots, x_n) \in X & \end{aligned}$$

where f and g_i are linear functions, and $X = \mathbf{R}^n$, or $X = \mathbf{Z}^n$, or $X = \mathbf{R}^{n_1} \times \mathbf{Z}^{n_2}$.

SoPLEX – a solver for linear optimization problems

A super-fast solver for linear optimization problems ($X = \mathbf{R}^n$) is the basic building block of optimization and operations research technology.

ZIMPL – an algebraic modeling language for optimization

ZIMPL provides a convenient way to write down optimization problems in the computer.

SCIP uses both SoPLEX and ZIMPL internally.

Our example problem in ZIMPL

ZIMPL input files are plain text files; use a text editor (such as *Emacs* or *gedit*) to create them, not a word processor. This is file `5legs.zpl`:

```
var basic_tables integer;
var deluxe_tables integer;

maximize profit:
  200 * basic_tables + 350 * deluxe_tables;

subto legs_constraint:
  5 * (basic_tables + deluxe_tables) <= 300;

subto wood_tops_constraint:
  basic_tables <= 50;

subto glass_tops_constraint:
  deluxe_tables <= 35;

subto assembly_hours_constraint:
  0.6 * basic_tables + 1.5 * deluxe_tables <= 63;
```

The ZIMPL language: Variable definitions

A complete description is found in Section 4.4 in ZIMPL User Guide.

Variable type:

`var NAME integer;`	Defines an integer variable
`var NAME binary;`	Defines a binary (0/1) variable
`var NAME real;`	Defines a real variable
`var NAME;`	Also defines a real variable

The ZIMPL language: Variable definitions

A complete description is found in Section 4.4 in ZIMPL User Guide.

Variable type:

`var NAME integer;`	Defines an integer variable
`var NAME binary;`	Defines a binary (0/1) variable
`var NAME real;`	Defines a real variable
`var NAME;`	Also defines a real variable

All variables (except binary) have a lower bound of 0 and no upper bound. If different upper bounds are needed, change the variable definition:

`var NAME integer >= -5 <= 5`	Defines an integer variable that can take values from -5 to 5
`var NAME real >= -infinity <= infinity`	Defines a free variable that can take arbitrary real values

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

(The ZIMPL language has a much greater power than this; we'll dive into its complexity gradually.)

The Traveling Salesperson Problem

Traveling Salesperson Problem (TSP)

A traveling salesperson needs to visit *n* cities, starting from city 1, visiting each other city exactly once, and returning to city 1. Find the shortest possible such tour.

“Distances” between the cities can be measured in different ways:

- If cities are considered as points $(x_i, y_i) \in \mathbf{R}^2$, a useful distance between cities could be the Euclidean distance

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

- Other distances that make sense are: Number of miles on the shortest road connection between cities *i* and *j*, airfares in US dollars, . . .
  Some of these distance functions *d* have some of these properties:
  - Satisfies triangle inequality (Metric TSP):

$$d(i,j) \leq d(i,k) + d(k,j)$$

  - Symmetry (Symmetric vs. Asymmetric TSP):

$$d(i,j) = d(j,i)$$

Some comments on the TSP

The TSP is one of the most famous combinatorial optimization problems.

- It appears in a wide array of applications.
  - Various route planning problems
  - Printed circuit board drilling
  - Application in genome sequencing

Some comments on the TSP

The TSP is one of the most famous combinatorial optimization problems.

- It appears in a wide array of applications.
  - Various route planning problems
  - Printed circuit board drilling
  - Application in genome sequencing
- It is notoriously hard to solve.
  - A symmetric TSP has $(n-1)!/2$ possible tours, so brute force is not feasible.
  - Simple, intuitive heuristics such as the “visit-nearest-neighbor algorithm” fail spectacularly
  - No “efficient” algorithm is known for it today, and it belongs to a huge class (*NP-hard*) of other, seemingly unrelated, difficult problems that we could suddenly solve efficiently, once an efficient algorithm for TSP became available.
- We’ll find out how well our black-box optimization solver works for this model

Excellent web resource, by Prof. William Cook (Georgia Tech):

`http://www.tsp.gatech.edu/`

- Concorde (state-of-the-art software for TSP), world records on the TSP, applications, and illustrations.

Modeling the TSP as a standard optimization problem

Variables?

Constraints (inequalities)?

Modeling the TSP as a standard optimization problem, I

- **Observation:** every tour of the TSP on n cities can be viewed as a subgraph (V, E') of the complete graph $K_n = (V, E)$ on n nodes.
  (We disregard orientation and starting point of the tour by doing so.)
- The edges of the complete graph are the 2-element subsets of n:

$$E = \{\, \{i,j\} : i,j = 1, \ldots, n \,\}$$

  Note that the order of i and j does not play any role:

$$\{i,j\} = \{j,i\}$$

- We can "encode" any subgraph (V, E') with a "set" of 0/1 variables, one for each edge of the complete graph:

$$x_{\{i,j\}} = \begin{cases} 1 & \text{if edge } \{i,j\} \text{ is present, i.e., } \{i,j\} \in E' \\ 0 & \text{if edge } \{i,j\} \text{ is not present} \end{cases}$$

- Thus, each vector $(x_{\{i,j\}})_{\{i,j\} \in E} \in \{0,1\}^E$ "encodes" a subgraph (V, E'). One way of writing this vector is as the upper triangle of a square $n \times n$ matrix – but how we write it, is not essential.
- Note: This is a one-to-one correspondence between the combinatorial objects ("subgraphs") and 0-1-vectors.

Modeling the TSP as a standard optimization problem, II

Next we wish to express the objective function.

- WISH to minimize the total length of the tour T, which is the edge set of a subgraph (V,T) of the complete graph $K_n = (V,E)$.
- Use $d(i,j)$ for the distance from city i to j (or reversely – remember we deal with the symmetric case!), the total length is:

$$length(T) = \sum_{\{i,j\}\in T} d(i,j)$$

  This summation is not “nice” – its domain of summation depends on the solution T. We prefer to sum over fixed domains of summation!

Modeling the TSP as a standard optimization problem, II

Next we wish to express the objective function.

- WISH to minimize the total length of the tour T, which is the edge set of a subgraph (V, T) of the complete graph $K_n = (V, E)$.
- Use $d(i,j)$ for the distance from city i to j (or reversely – remember we deal with the symmetric case!), the total length is:

  $$length(T) = \sum_{\{i,j\} \in T} d(i,j)$$

  This summation is not “nice” – its domain of summation depends on the solution T. We prefer to sum over fixed domains of summation!
- Now we remember that we have 0/1 variables $x_{\{i,j\}}$ that are 1 if $\{i,j\} \in T$, and 0 otherwise. So it does not change anything if we multiply $d(i,j)$ by $x_{\{i,j\}}$:

  $$length(T) = \sum_{\{i,j\} \in T} x_{\{i,j\}} d(i,j) = \sum_{\{i,j\} \in E} x_{\{i,j\}} d(i,j)$$

  In the last step we extended the domain of summation to all edges in E – Nothing happens, since all the added summands are 0.

Modeling the TSP as a standard optimization problem, II

Next we wish to express the objective function.

- WISH to minimize the total length of the tour T, which is the edge set of a subgraph (V, T) of the complete graph $K_n = (V, E)$.
- Use $d(i,j)$ for the distance from city i to j (or reversely – remember we deal with the symmetric case!), the total length is:

  $$length(T) = \sum_{\{i,j\} \in T} d(i,j)$$

  This summation is not “nice” – its domain of summation depends on the solution T. We prefer to sum over fixed domains of summation!
- Now we remember that we have 0/1 variables $x_{\{i,j\}}$ that are 1 if $\{i,j\} \in T$, and 0 otherwise. So it does not change anything if we multiply $d(i,j)$ by $x_{\{i,j\}}$:

  $$length(T) = \sum_{\{i,j\} \in T} x_{\{i,j\}} d(i,j) = \sum_{\{i,j\} \in E} x_{\{i,j\}} d(i,j)$$

  In the last step we extended the domain of summation to all edges in E – Nothing happens, since all the added summands are 0.
- We have expressed the length of a tour as a linear function in our $x_{\{i,j\}}$ variables; note the domain of summation is independent of the tour!

Modeling the TSP as a standard optimization problem, III

- **Since all tours are subgraphs, but not all subgraphs are tours, we need to add constraints on our variables, to make sure that only tours are feasible solutions.**

Modeling the TSP as a standard optimization problem, III

- **Since all tours are subgraphs, but not all subgraphs are tours, we need to add constraints on our variables, to make sure that only tours are feasible solutions.**
- We call a vertex v and an edge e **incident** if $v \in e$, i.e., v is one of the endpoints of the edge e.
- The **degree** of a vertex v is the number of edges incident with it.

Modeling the TSP as a standard optimization problem, III

- **Since all tours are subgraphs, but not all subgraphs are tours, we need to add constraints on our variables, to make sure that only tours are feasible solutions.**
- We call a vertex v and an edge e **incident** if $v \in e$, i.e., v is one of the endpoints of the edge e.
- The **degree** of a vertex v is the number of edges incident with it.
- **Observation:** A tour, viewed as a subgraph of K_n, every vertex has degree 2:
- So let's write down this insight as a constraint, for every vertex i:

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2.$$

  Again, the domain of summation is independent of the tour, so this equation is a linear constraint in our variables $x_{\{i,j\}}$.

Putting the TSP model in the computer

- In the computer, it is convenient to represent edges (2-element sets) $\{i, j\}$ as (ordered) pairs (i, j) with $i < j$.
- Thus, for a 6-city TSP, we would be using variables named `x12`, `x13`, `x14`, `x15`, `x16`, `x23`, `x24`, `x25`, `x26`, `x34`, `x35`, `x36`, `x45`, `x46`, `x56`
- When we write down the constraint

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2,$$

by using the ordered-pair representation, we actually write

$$\sum_{j<i} x_{(j,i)} + \sum_{j>i} x_{(i,j)} = 2.$$

- The resulting optimization model in ZIMPL is found in the file `tsp6-1.zpl`

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

$$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

This does not look like a tour! What are we missing?

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

$$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?

- We are not missing anything; to the contrary, we have **too much**! Our integer program has many feasible solutions that do not correspond to tours!

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

$$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?
- We are not missing anything; to the contrary, we have **too much**! Our integer program has many feasible solutions that do not correspond to tours!
- We need to add more inequalities that “forbid” short cycles.

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

$$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?
- We are not missing anything; to the contrary, we have **too much**! Our integer program has many feasible solutions that do not correspond to tours!
- We need to add more inequalities that “forbid” short cycles.

Lemma

Let $T \subseteq E$ be any TSP tour on K_n.
Let $S \subseteq V$ be a vertex subset of size $3 \leq |S| \leq n-3$. Then

$$|\{\{i,j\} \in T : i,j \in S\}| \leq |S| - 1.$$

Complete TSP Formulation

Theorem (Complete TSP Formulation)

The 0/1 solutions of the system

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2 \qquad \text{for all vertices } i = 1,\dots,n$$

$$\sum_{\substack{\{i,j\}\in E:\\ i,j\in S}} x_{\{i,j\}} \leq |S| - 1 \qquad \text{for all } S \text{ in } K_n \text{ with } 3 \leq |S| \leq n-3$$

are in one-to-one-correspondence with the TSP tours on K_n.

- How many short-cycle inequalities?

$$2^n - 2\binom{n}{0} - 2\binom{n}{1} - 2\binom{n}{2}$$

For $n = 15$: about 32000.

TSP and Math 168

This formulation is the basis for the state-of-the-art approaches (“branch and cut”) to the TSP, which:

- Repeatedly solve a linear relaxation (where the integrality of the variables is ignored) – a **linear optimization problem**, we use the Simplex Method!!!
- Branch on subproblems where some variables are fixed to 0 or 1
- Use linear relaxation bounds to throw away subproblems where no good solution exists
- Add violated inequalities (short-cycle inequalities and “stronger” inequalities)

It motivates us in this class to:

- Learn expressive modeling languages (ZIMPL) that allow us to add a large number of well-structured inequalities without writing each of them by hand
- Study fast algorithms for solving linear optimization problems, even if the number of variables and constraints is very large.