Gurobi fast: (1) written in low-level language, maybe C or C++, (2) takes full advantage of parallelization (multiple cores in CPU, cluster computing, etc.),
(3) cutting-plane procedures: iteratively refines a feasible set or objective fctn by linear ineq. called "cuts",
(4) heuristics.

Did 400-city problem in only 8 minutes

2. Formulate and solve the traveling salesman problem.

(a) Formulate the problem by defining the decision variables, objective, and constraints. As shown in class, the most obvious formulation does not avoid "sub-tours". Think of constraints you can add to avoid sub-tours. If you get tired of thinking, read the attached paper and implement their constraints.

- In my Python implementations of the traveling salesman problem (for both the PuLP and Gurobi packages) I employed the "sub-tour formulation" from the paper by Pataki recommended by Dr. Harris. I will describe this formulation here.
- **DECISION VARIABLES:**
  Define decision variables as $x_{i,j} := \left\{ \begin{array}{ll} 1 & \text{if go city}_i \to \text{city}_j \\ 0 & \text{if don't go city}_i \to \text{city}_j \end{array} \right\} \forall i,j \text{ where } i \neq j, \text{ and } i,j \in \{1,2,\ldots,n\}$
  To make this more familiar, we can think about our decision variables being organized in a matrix $X_{n \times n}$, where the entry $x_{i,j}$ is as described above. For example, if $x_{1,3} = 1$ this would mean that after the salesman travels from city 1 to city 3. (This matrix isn't completely necessary for representation of the variables, but it helps us nail down the concept more concretely. If we did use this, all diagonal entries would necessarily be 0 since we can't go from a city back to itself.)

- **OBJECTIVE:**
  Our objective is: $\min_x \sum_{i,j}^{n} (x_{i,j})(d_{i,j})$ where $d_{i,j}$ is the straight-line distance between city$_i$ and city$_j$, and $x_{i,j}$ indicates whether the salesman departs-from city$_i$ and arrives-to city$_j$. This means that we are minimizing the distance traveled in order to visit each city exactly once. (For sake of completeness in the code, we then go from the final city back to the origin city to 'complete the tour.')

- **CONSTRAINTS:**
  Assignment Constraints:
  - Thought of in terms of graph theory, this would mean that each node has exactly one incoming-directed edge and exactly one outgoing-directed edge.
  - Thought of in terms of cities and salesman, this would mean that each city is arrived-to exactly once by the salesman, and each city is departed-from exactly once by the salesman.
  - Mathematically:
    i. $\left[\sum_{i=1}^{n} x_{i,j} = 1, \forall i\right]$ ("each city departed-from exactly once")
    ii. $\left[\sum_{j=1}^{n} x_{i,j} = 1, \forall i\right]$ ("each city arrived-to exactly once")

  Sub-tour Elimination Constraints:
  - A sub-tour is a 'directed cycle' in a graph. A directed cycle is a non-empty directed trail in which the first and last vertices are repeated (may be one and the same).
  - In the context of this problem, that would involved the salesman coming back through a city which he's already visited. This violates our assignment constraint of departing-from a city only once since he'd have already been there, meaning he's also already departed from there once.
  - To prevent this more strictly than just with our assignment constraints:
    $\left[\sum_{i \in S, j \in S} x_{i,j} \leq |S| - 1\right]$ where $\left[(S \subset V), (|S| > 1), (\forall i \neq 1, \forall j \neq 1)\right]$
  - Here, S is a set of unique cities. V is another (different from S) set of unique cities. $(S \subset V)$ means that S is a proper subset of V (all cities in S are also in V, but there is at least one city in V that isn't in S).
  - Also, here $|S|$ means the cardinality (number of elements) in S rather than absolute value.
  - This constraint effectively says that, for the cities in S, the number of elements in S minus 1 must be at least the sum of the binary variables for all of the cities in S. Since the sum is one less than the number of cities, this prevents the salesman from looping back to the first (or other) cities in S, thus eliminating any sub-tours for the set of cities in S.
  - This constraint does introduce some redundancy since there are many sets S and V that are possible for all cities. This is taken care of by a *separation algorithm* which we didn't talk about, nor does the Pataki paper. (The Python code will take care of this for us though.)

TSP prstn -- Tue, 10/29/2019

- Brief go over of: (1) DEC VARS → put in matrix X (i = depart, j = arrive), (2) OBJ shortest dist while visiting each city & back to org?

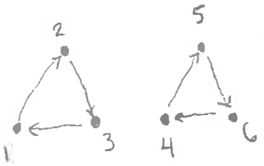
- "Assignment" constraint: don't want to revisit cities, so we specify that we arrive-to & depart-from each city exactly once.

  (a.) $\sum_{i=1}^{n} x_{ij},\ A_j$ → summing for each col X → arrive-to each city once

  $\overset{\text{“}}{=}\ \boxed{1}$

  $\sum_{j=1}^{n} x_{ij},\ A_i$ → summing for each row X → depart-from each city once

- "Sub-tour elimination" constraint:



$$X = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ 1 & & 0 & & & \\ & & & 0 & 1 & \\ & & & & 0 & 1 \\ & & & 1 & & 0 \end{bmatrix}$$

- asmt constraints don't prevent sub-tours, example:
- all cols & rows in X Σ to 1 (e.g. depart-from & arrive-to constraints satisfied) but we don't have a "connected/complete" tour that returns to origin city.
- In graph theory, a sub-tour would be called a directed cycle, where the first vertex is also the last vertex.
- The way we fix this is simple to draw (harder in math): just remove one of the edges.
- Fix in math: $\sum_{i \in S,\, j \in S} x_{i,j} \le |S| - 1$ where $(S \subset V)$, $(|S| > 1)$, $(\forall i \ne 1, \forall j \ne 1)$
- [S = set of unique cities], [V = another set of unique cities], [$S \subset V$ = S proper subset of V] → give example.
- $|S|$ means cardinality of S, just # of cities (elements) in set S.
- What is the constraint saying? Must have at least one less edge than we do nodes for some collection of cities S.

  Ex: Let $S = \{1,2,3\}$ from above, make B matrix sub-matrix of X, $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$. Σ all elements (x_{ij}) in B must be $\le ((|S|-1) = 3-1 = 2)$ But we can see $1+1+1 = 3 \nleq 2$.

- Tons of constraints though: if we have 8 cities we can make constraints for $|S| = 6$, $|S| = 2$, $|S| = 5$, $|S| = 3$, $|S| = 4$ $= (8-2)!$ constraints.
- Tricky to implement. If interested, look up "$\overset{\text{(delayed)}}{\text{column generation}}$" or "Dantzig-Wolfe decomposition"