Inmas 2021: Modeling and Optimization

Session 4: Miscellaneous topics in optimization

Alexander Estes

Constraint Generation I

Recall that in general, we want strong formulations.

- Sometimes, to get a strong formulation, we need a very large number of constraints.
- Can we get away with only adding some of the constraints?

Constraint Generation II

Travelling salesman problem (TSP):

- n cities.
- Distance between city i and j is d_{ij}.
- Find shortest path that passes through all cities exactly once and returns to starting city.
- We'll assume this is a symmetric TSP, so $d_{ij} = d_{ji}$.

Constraint Generation III

Decision variables:

- x_{ij}: binary; 1 if we include the edge between city i and city j in our route.
- We are using undirected edges; don't need both x_{ij} and x_{ji}.
- We will only include variables x_{ij} with $i < j$

Objective:

$$\sum_{i<j} d_{ij} x_{ij}$$

Constraint Generation IV

Constraints:

- Every city should have exactly two adjacent arcs selected.

$$\sum_{j \text{ s.t. } j<i} x_{ji} + \sum_{j \text{ s.t. } i<j} x_{ij} = 2 \text{ for each } i$$

  Note: the two summations together sum all variables corresponding to edges adjacent to the node i.

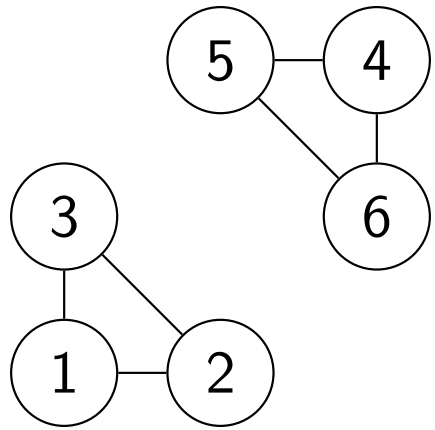
- Variables are binary:

$$x_{ij} \in \{0, 1\}$$

These constraints are valid; are they sufficient?

Constraint Generation V

Problem: subtours



Solution: subtour elimination constraints.

$$\sum_{(i,j)\in S\times S} x_{ij} \leq |S| - 1 \text{ for each subset } S \subsetneq \{1,\ldots,n\}$$

- There are $2^n - n - 1$ of these constraints.
- However, for most instances, most of these constraints aren't needed.

Constraint Generation VI

We can modify a branch-and-bound procedure or a branch-and-cut procedure so that the subtour constraints are added only when needed:

- After solving a node:
  - Check if any subtour elimination constraints are violated.
  - If constraints are violated, add them to model, resolve the node.
  - Repeat until no more constraints are violated.
  - Treat the resulting solution as the solution of the node.

Alternatively: only check for subtour elimination constraints once we have reached an integer solution. This makes it easier to check for the subtour elimination constraints.

- Either approach will lead to an optimal solution.

Constraint Generation VII

Major consideration: the separation problem

- The *separation problem* is the problem of identifying a constraint that has been violated.
- If constraint generation is applied to integer solutions only, we can identify violated constraints efficiently:
  - Choose any starting node
  - Follow the route from this node
  - If we get back to the starting node without reaching all the nodes, we have identified a subtour.
- If constraint generation is applied to fractional solutions, separation is more complicated but can be done efficiently.
- In general, constraints may not be easy to separate.

Constraint Generation in Gurobi

How can we do this in Gurobi?

- We have to use a *callback*
- Gurobi allows you to call a function that it will execute for you at certain points in the branch-and-cut procedure (e.g. whenever an integer solution is found)
- In your callback function, you can pass a limited set of instructions to Gurobi (e.g. you can tell Gurobi to add more constraints).

Gurobi demonstration

See “tsp.py”

Heuristics

What do I do if Gurobi can't solve my problem?

- Gurobi is very good at a wide variety of problems
- However, integer programs are hard, and you may run into a problem that off-the-shelf solvers cannot solve fast enough.
- There are some optimization methods that can help (decomposition techniques, custom cutting planes, custom branching schemes, better formulations)
- You can settle for a good solution rather than an optimal solution.
- Heuristics can help

Constructing a starting solution.

The heuristics that we discuss will require an initial feasible solution. One potential approach is a greedy strategy.

- We build a solution iteratively out of components.
- In each iteration, we choose the component that looks best, ignoring interactions between components.
- May require creativity to define criteria for choosing component.
- May require creativity to define method for incorporating new component into current solution.

Example: traveling salesman problem. I

A greedy algorithm for the TSP:

- Choose an initial city.
- In each iteration, move to a city that has not already been visited.
- Choose the city that is closest to the current city.

Example: traveling salesman problem. II

A different greedy algorithm:

- Start with a route between only two cities, specifically the two cities that are farthest apart.
- In each iteration, insert a city into the route.
- Choose the city that is farthest from any city in the current route.
- Insert the city in the position that results in the shortest route.

Local neighborhood search

Once we have an initial solution, we can look for improvements:

1. Start with some initial solution.
2. Find the best solution within a "neighborhood" of the current solution.
3. If the current solution is better than all solutions in neighborhood, then return solution.
4. Otherwise, update current solution to the best solution; go back to step 2.

What is a neighborhood? You get to decide.

Example: traveling salesman problem. I

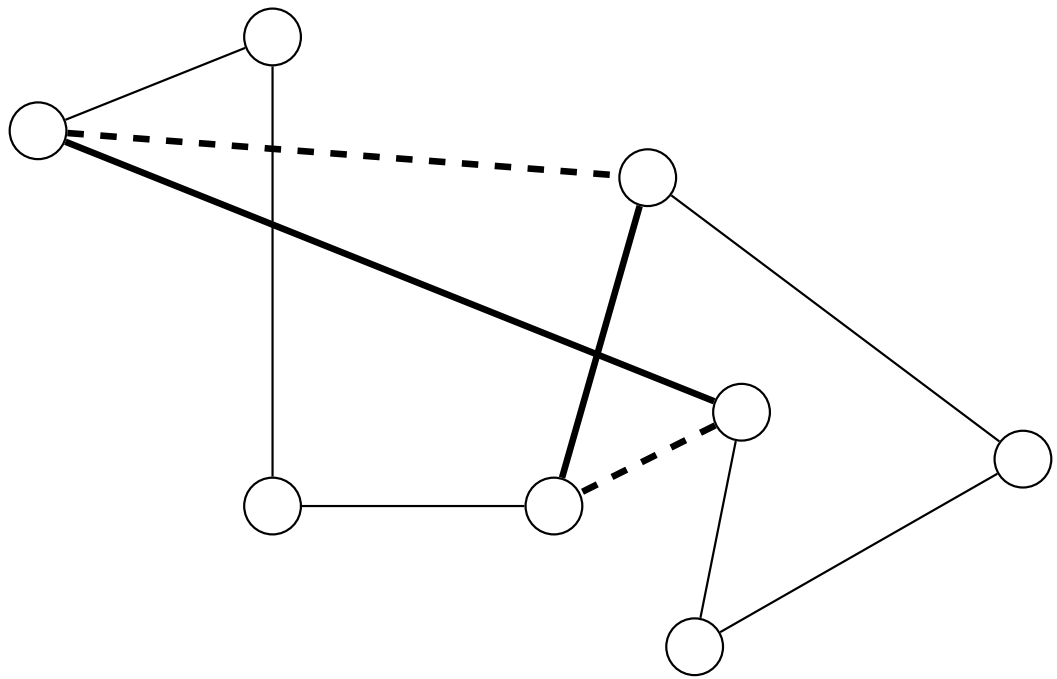
We could define a "city-swap" neighborhood (not a standard term):

- Given a solution, the neighborhood is the set of solutions that can be reached by swapping the positions of two cities in a route.
- E.g. suppose there are five cities, one of which is always the starting city.
- Suppose that the current solution is $1 \to 2 \to 5 \to 3 \to 4$.
- Then the city-swap neighborhood would be:
  $(1 \to 5 \to 2 \to 3 \to 4)$, $(1 \to 3 \to 5 \to 2 \to 4)$,
  $(1 \to 4 \to 5 \to 3 \to 2)$, $(1 \to 2 \to 3 \to 5 \to 4)$,
  $(1 \to 2 \to 4 \to 3 \to 5)$, $(1 \to 2 \to 5 \to 4 \to 3)$.

Example: traveling salesman problem. II

Alternative: 2-opt or 2-exchange neighborhood.

- Suppose that we remove two arcs.
- There is only one way to add 2 more arcs that will result in a valid, different tour.
- Define the 2-opt neighborhood to be the set of tours that can be reached by removing two arcs and replacing them.



Tabu Search I

A problem with neighborhood search: you get stuck.

- Once you reach a local minima, you won't improve.
- One idea: move to the best solution in the neighborhood even if its worse.
- New problem: cycling.
- Usually, if solution B is in neighborhood of A, the reverse is also true.
- We might move back and forth between solutions $A \rightarrow B \rightarrow A \rightarrow B \rightarrow \ldots$.
- New solution: keep a "tabu list" that bans certain solutions or modifications.

Tabu Search II

Tabu search algorithm:

1. Initialize a tabu list to empty.
2. Construct an initial solution x
3. Set $x^* = x$ (the value x^* will store the best found solution)
4. Find the best solution x' in the neighborhood of the current solution, excluding those prohibited by the tabu list.
5. If x' is better than x^*, then set $x^* = x'$
6. Set $x = x'$ and update tabu list.
7. Check termination criteria; if satisfied stop, otherwise go back to Step 3.

Tabu Search III

The tabu list:

- The tabu list could simply be the most recent t solutions. Then, we would not be allowed to move to these solutions. In this case, $t = 7$ has been suggested as a good choice.
- If we are using the city-swap neighborhood, the tabu list could include the most recent t swaps that we performed. Then, we would not be allowed to perform those swaps again.
- Similarly, If we are using the 2-opt neighborhood heuristic, the tabu list could include the most recent t pairs of edges.
- In general, requires some creativity/judgment.

Tabu Search IV

Stopping criteria:

- Stop after a certain number of iterations
- Stop after a certain number of iterations without improvement in the objective.

Simulated Annealing I

Simulated annealing provides another way of escaping local minima

- Some random chance of selecting a neighbor that worsens the solution.

Simulated Annealing II

Simulated annealing parameters:

- Initial temperature T
- Reduction factor r
- Loop length L

Simulated Annealing III

Algorithm:

1. Construct an initial solution x.
2. Set $x^* = x$
3. Perform the following loop L times:
   - 3.1 Pick a random neighbor x' of current solution x.
   - 3.2 Let $\Delta = f(x) - f(x')$ where f is the objective function.
   - 3.3 If $\Delta \leq 0$ (i.e. neighbor is better) set $x = x'$.
   - 3.4 If $\Delta > 0$ set $x = x'$ with probability $e^{-\Delta/T}$
4. Set $T = rT$.
5. Check stopping criteria; if criteria is satisfied then return x^*. Otherwise, go back to step 3.

Simulated Annealing IV

Note:

- The probability that we accept a worse solution $e^{-\Delta/T}$ is increasing in T.
- So, as we decrease T, this probability will decrease.

Dynamic optimization under uncertainty I

So far, the optimization problems we have discussed are static and deterministic

- All information is known ahead of time
- All decisions are chosen ahead of time.

However, many optimization problems are dynamic and uncertain:

- Actions take place over time
- Problem parameters are unknown and are revealed over time.
- Decisions can react to revealed random parameters.

Dynamic optimization under uncertainty II

One approach: rolling horizon with a deterministic approximation.

1. Produce your best prediction of what will happen in the next T time units (hours/seconds/steps in a process etc.)
2. Formulate an optimization problem for the next T time units, assuming that your prediction will come true.
3. Solve the optimization problem
4. Start implementing your solution.
5. Periodically repeat steps 1-3 and adjust solution based on results.

There are more sophisticated approaches, but this type of approach is frequently and successfully used.

Example: routing under uncertainty. I

Goal: route a truck through a road network with minimal travel time.

1. The travel times along each road segment are random, depending on traffic, weather, etc.
2. For simplicity, ignore waiting time at intersections.
3. Can represent the road network as a directed graph, where nodes represent intersections, and edges of the graph represent road segments between intersections.
4. Let E be the set of edges in the graph.
5. For a node v, let $E^-(v)$ and $E^+(v)$ denote the incoming and outgoing edges of v respectively.
6. Let s denote the starting location of the truck, and let d denote the destination.

Example: routing under uncertainty. II

First, we will form an optimization problem that we can solve before the truck departs.

- Let $\hat{t}_{ij}$ be our prediction of the travel time of the road segment from i to j

Variables: for each edge $(i, j) \in E$, let x_{ij} be a binary variable,

$$x_{ij} = \begin{cases} 1 & \text{route uses road segment from } i \text{ to } j \\ 0 & \text{route doesn't use road segment from } i \text{ to } j \end{cases}$$

Objective:

$$\min_x \sum_{i,j \in E} \hat{t}_{ij} x_{ij}$$

Example: routing under uncertainty. III

Constraints:

$$\sum_{j \in E^+(s)} x_{si} = 1 \qquad \text{we must leave starting point}$$

$$\sum_{i \in E^-(d)} x_{id} = 1 \qquad \text{we must reach destination}$$

$$\sum_{i \in E^+(v)} x_{vi} = \sum_{i \in E^-(v)} x_{iv} \quad \text{for each } v \notin \{s, d\} x_{vi} \qquad \in \{0, 1\}$$

The last constraint essential states that we can enter an intersection if and only if we exit the intersection (unless the intersection is the start or destination)

Example: routing under uncertainty. IV

- Periodically, we receive updates of the estimates $\hat{t}_{ij}$.
- When we receive an update, we solve a new routing problem.
- The new optimization problem will be exactly the same, except the starting location will be our current location and we use the updated values of $\hat{t}_{ij}$.

Example: routing under uncertainty. V

Potential improvements:

- As currently described, our method might prescribe route changes with little to no warning.
- Alternative: let s^* be the location that we will be in 5 minutes under the current routing.
- Solve a routing problem from s^* to the destination.

Example: routing under uncertainty. VI

Another potential improvement:

- If there are two routes that are very similar in estimated time and we update, the suggested route might "jitter" between the two routes.
- Improvement: only change the suggested route if the time savings are significant
- Alternatively (or in addition): don't resolve the problem immediately after every update; wait a certain amount of time since the last optimization.

Some further resources with more sophisticated approaches:

- *Introduction to Stochastic Programming* by Birge and Louveaux.
- *Reinforcement Learning* by Sutton and Barto.
- *Reinforcement Learning and Stochastic Optimization: a unified framework for sequential decisions* by Warren Powell.

General Modeling Advice I

Some general advice for modeling:

- Start with the simplest reasonable model.
- Make sure you capture the most important decisions.
- Some objectives can be constraints, and some constraints can be objectives.

General Modeling Advice II

Look for opportunities to break up problem. For example, consider the problem of running an airline's operations. This can be split into several steps:

1. Flight scheduling: Choose which flights that will be offered (e.g. New York LaGuardia to Chicago O'Hare every Monday at 9:00 a.m.)
2. Tail assignment: decide which aircraft will be used for each flight (e.g. Aircraft N7732A will fly from New York to Los Angelos to Chicago)
3. Crew assignment: decide which crew will be used on each flight