Announcements

- Homework #25 due now
  - Questions about bounding functions?
- Screencast and Quiz
  - Due: end of day Thursday
- Project #7: TSP
  - We continue discussing design options today
  - Help sessions: Thursday (3/26)
  - Whiteboard experience: Monday (3/30)
  - Early: Wednesday 4/8
  - Due: Friday 4/10

Objectives

- Consider an alternate way to structure the state space.
  - i.e., an alternate method for state expansion
- Provide more design options to complete the project.

Review the Ingredients of B&B

Review: A Bound on TSP Tour

1. Reduce Rows
2. Reduce Columns

- Reduce Columns

```
999 1 999 0 999
999 999 2 999 0
999 0 999 1 999
999 0 1 999 6
0 999 999 9 999
```

Review: Feasibility

- Part of our state expansion function
- The feasibility test may include any condition you might think of:
  - Vertex doubly entered
  - Vertex doubly exited
  - Cycle among sub-set of cities
  - No options remaining – dead end.

Review: Pruning

- Under what conditions can a search state be pruned?
  - Just one condition:
    - the (lower) bound for state ≥ BSSF (current upper bound)
  - When?
    - Right after the state is created
    - Don’t even add it to the agenda
    - Either / or:
      - Eager pruning: When the BSSF is updated
      - Lazy pruning: When the state comes off the agenda

B&B Pseudo-Code: Eager Pruning

```
function BandB()
    s ← init_state()
    BSSF ← quick-solution(s)  // BSSF.cost holds cost
    Agenda.clear()
    Agenda.add(s, s.bound)
    while !Agenda.empty() and time remains and BSSF.cost != Agenda.first().bound do
        u ← Agenda.first()
        Agenda.remove_first()
        children = successors(u)
        for each w in children do
            if ! time remains then break
            if (w.bound is better than BSSF.cost) then
                if criterion(w) then
                    BSSF ← w
                    Agenda.prune(BSSF.cost)
                else
                    Agenda.add(w, w.bound)
    return BSSF
```

Proj. #7 option:
- Use this pseudo-code for eager B&B.

B&B Pseudo-Code: Lazy Pruning

```
function BandB()
    s ← init_state()
    BSSF ← quick-solution(s)  // BSSF.cost holds cost
    Agenda.clear()
    Agenda.add(s, s.bound)
    while !Agenda.empty() and time remains and BSSF.cost != Agenda.first().bound do
        u ← Agenda.first()
        Agenda.remove_first()
        children = successors(u)
        for each w in children do
            if ! time remains then break
            if (w.bound is better than BSSF.cost) then
                if criterion(w) then
                    BSSF ← w
                    Agenda.prune(BSSF.cost)
                else
                    Agenda.add(w, w.bound)
    return BSSF
```

Proj. #7 option:
- Use this pseudo-code for lazy B&B.

Structure of State Space

- Recall the structure of the state space from last time.
Review: State Space #1

- Generate all children one step away
  - Each child state is generated by including an edge from city j, to a previously unvisited city (where j is the destination of the last edge in the parent state)
- Pro: Does not include premature cycles. Why not?
- Pro: Simple algorithm: pick all neighbors that remain from vertex j, (leave the starting vertex for the end)
- Con: On many instances, while searching with B&B, will result in excessively deep (possibly exponential) agendas. Why?
- Is B&B in this state space optimal, given enough time and space?

Structure of State Space

- Could we structure the space another way?
- How else could we generate children states?

State Space #2

*Use same representation for states & same bounding function.*

New method for generating children:

- Include edge \( (i,j) \)
- Exclude edge \( (i,j) \)

Why is branch and bound in this state space optimal, given enough time?

How: Excluding an Edge

What happens to a state’s cost matrix when an edge is excluded?

Excluding edge 4 → 2

Size of the State Space

- Start with Complete Graph

\[ |E| = n(n-1) = n^2 - n \]
Which edge to select?

- What strategies for edge selection can you think of?
- Idea: try to reduce the size of the part of the state-space you actually need to explore!
- Avoid going into the deep (exclude) side, if possible.

Try #1: Consider edge (3,2)

Try #2: Consider edge (2,3)

Try #3: Consider edge (5,1)

State Space so far
Consider edge (2,5)

Parent:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{include} (2,5) & \text{exclude} (2,5) & \text{bound} = 28 & \text{bound} = 21 \\
\end{array}
\]

Again, right child more likely to be pruned.

Search Space so far

\[
\begin{array}{cccc}
\text{include} (5,1) & \text{exclude} (5,1) & 21 & 30 \\
\text{include} (2,5) & \text{exclude} (2,5) & 21 & 28 \\
\text{include} (3,2) & \text{exclude} (3,2) & 21 & 22 \\
\end{array}
\]

Consider edge (3,2)

Parent:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{include} (3,2) & \text{exclude} (3,2) & \text{bound} = 21 & \text{bound} = 22 \\
\end{array}
\]

Search Space so far

\[
\begin{array}{cccc}
\text{include} (5,1) & \text{exclude} (5,1) & 21 & 30 \\
\text{include} (2,5) & \text{exclude} (2,5) & 21 & 28 \\
\text{include} (3,2) & \text{exclude} (3,2) & 21 & 22 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{include} (4,3),(1,4) : 21 \\
\end{array}
\]

Summarize: State Space #2

- Pick an edge \(e = (i,j)\) and generate two children:
  1. Include edge \(e\)
  2. Exclude edge \(e\)

- One might pick an edge by:
  - maximizing the difference between the included edge child and the excluded edge child
  - maximizing the bound on the excluded edge child
  - minimizing the bound on the included edge child

- One must be careful not to create a cycle prematurely
  - How to prevent?
Conclusions

- You have options!
- Having some control without sacrificing optimality is important.
- You may be able to prune more aggressively without sacrificing optimality.

Assignment

- HW #26
  - Due Friday
- Read the project instructions and lecture notes
- Have your “whiteboard experience” for Project #7
  - Consider your options
  - Make your choice
  - Design your algorithm
  - Work and talk through an example