CS 312: Algorithm Design & Analysis

Lecture #33: Branch and Bound, Job Assignment

Announcements

- Project #6: Due: Today
- Screencast #38 and Quiz #9 Due: Thursday
- Homework #23 Due: Friday
- Project #7: TSP
  - Today and in the Thursday Screencast: possible approaches
  - ASAP: Read Project #7 instructions
  - Budget about 15 hours (mean)
  - Required: Read the helpful “B&B for TSP Notes” linked from the schedule in the reading column

Objectives

- Apply state-space search to optimization
- Learn how to prune unprofitable parts of the state space
- Develop a “branch and bound” algorithm for the (Job) Assignment Problem

State-space Search

<table>
<thead>
<tr>
<th>Back-tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose: Existence, Enumeration</td>
</tr>
<tr>
<td>Answer: Yes/No, Count, Set</td>
</tr>
<tr>
<td>Idea: Avoid searching the entire state-space</td>
</tr>
<tr>
<td>Main tool: Feasibility function</td>
</tr>
</tbody>
</table>

State-space Search

<table>
<thead>
<tr>
<th>Branch and Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose: Optimization</td>
</tr>
<tr>
<td>Answer: (Good, better, best)</td>
</tr>
<tr>
<td>Idea: “From this state s, I can do no better than B(s).”</td>
</tr>
<tr>
<td>Main tool: Bounding function</td>
</tr>
</tbody>
</table>

Bounding Function

Given some state s in the search space, compute a bound B(s) on the cost/goodness of all solutions that descend from that state.

Better: cost of actual solution is somewhere in here

Worse

bound B(s)
**Job Assignment Problem**

- **Given** $n$ tasks and $n$ agents.
- Each agent has a cost to complete each task.
- Assign one agent to each task / one task to each agent
  - A one-to-one and onto mapping
- **Minimize** cost

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
</tbody>
</table>

Could solve with LP, but it makes a good example for B&B.

**Design a State**

- **How to represent a state?**
  - Remember to include any commitments made so far.
  - A 4-tuple of assignments (could be empty)
- **How about an initial state?**
  - An empty 4-tuple

**Getting Oriented: Minimization**

- **We need to design:**
  1. State
  2. **BSSF**
     - The cost of the BSSF is an Upper Bound on the cost of the optimal solution
  3. **Bounding function $B(s)$**
     - For evaluating any state $s$:
       - $B(s)$ is a Lower Bound on the cost of potential solutions reachable from $s$.
       - Usually involves solving a relaxed form of the problem
  4. **Initial state $s_0$**
     - $LB = B(s_0)$ is a Lower Bound on cost of all potential solutions reachable from the initial state

**Design a State**

- **How to represent a state?**
  - Remember to include any commitments made so far.
  - A 4-tuple of assignments (could be empty)
- **How about an initial state?**
  - An empty 4-tuple
Design a BSSF

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
</tbody>
</table>

Generate a solution (not necessarily optimal) and call that your best solution so far (BSSF). How?

1. Case 1: $A1 - B1 - C1 - D1$ with $C1 = 56$
2. Case 2: $A1 - B1 - C1 - D1$ with $C1 = 65$
3. Diagonal: $A1 - B1 - C1 - D1$ with $C1 = 73$
4. Bounding: $A1 - B1 - C1 - D1$ (worst first) with $C1 = 85$

Tight Upper Bound

- Why would you want the upper bound to be tight (why start lower)?

![Graph showing Cost of BSSF and Cost of BSSF']

Tight Lower Bound

- Why would you want a bounding function that gives a tighter (higher) lower bound on each state?

![Graph showing Cost of BSSF and Cost of BSSF']

Design a Bounding Function

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
</tbody>
</table>

Bounding function:
- Easy to evaluate
- True bound

Next, what should we use as our bounding function?

1. Sum of lowest in each row: 49
2. Sum of lowest in each col: 55
3. MAX(51, 49)

We'll go with: Add the smallest entry in each column.

Design a Bounding Function

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
</tbody>
</table>

Bounding function:
- Easy to evaluate
- True bound
Next, apply the bounding function to the initial state to compute the initial lower bound on all solutions.

**Lower Bound**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>14</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td>17</td>
<td>21</td>
<td>84</td>
</tr>
</tbody>
</table>

Lower Bound

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>14</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td>17</td>
<td>21</td>
<td>84</td>
</tr>
</tbody>
</table>

**BSSF: 73**

Next, apply the bounding function to the initial state to compute the lower bound on all solutions.

**LB: 58**

Initial questions:
- What if the LB == (Cost of) BSSF?
- What if the assignment underlying the LB is a solution?
- What if LB > BSSF?

Now, make an assignment to agent A

A: 1

Then apply the bounding function:
- start with the cost of commitments
- add the smallest values in each column.

This is a **lower bound on the cost of any solution** with job 1 assigned to A.

It's not a solution!
Example

A:1 (60)
A:2 (58)
A:3 (65)
A:4 (78)

How to proceed from here?

A:2, B:1 (68)
A:2 (58)
A:3 (65)
A:4 (78)

Example

A:1 (60)
A:2 (58)
A:3 (65)
A:4 (78)

What can we say about the last option?

How to proceed from here?

Many options:
- Breadth-first
- Depth-first
- Most promising first – we’ll try this one today
- Other possibilities
Example

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 73

A:1 (60)
A:2, B:1 (68)
A:3 (65)
A:4 (78)

A:2, B:3 (59)

A:2, B:4 (64)

Example

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 73

A:1 (60)
A:2 (58)
A:3 (65)
A:4 (78)

A:2, B:1 (68)
A:2, B:3 (59)
A:2, B:4 (64)

Example

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 64

A:1 (60)
A:2, B:1 (68)
A:2, B:3 (59)
A:2, B:4 (64)
A:2, B:3, C:1, D:4 (64)
A:2, B:3, C:4, D:1 (65)

A:2, B:3 (59)
A:2, B:4 (64)
A:2, B:3, C:1, D:4 (64)
A:2, B:3, C:4, D:1 (65)

Example

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 64

A:1, B:2 (66)
A:2, B:3 (61)
A:1, B:4 (66)
A:2, B:3 (61)
A:2, B:4 (66)
A:2, B:3, C:1, D:4 (64)
A:2, B:3, C:4, D:1 (65)

Example

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 64

A:1, B:2 (66)
A:2, B:3 (61)
A:1, B:4 (66)
A:2, B:3 (61)
A:2, B:4 (66)
A:2, B:3, C:1, D:4 (64)
A:2, B:3, C:4, D:1 (65)

When you reach a solution, update the BSSF if better.

When you reach a solution, update the BSSF if better. Then what?

Prune!

Finish

A 1 2 3 4
B 14 15 13 22
C 11 17 19 23
D 17 14 20 28

BSSF: 64

A:1, B:2, C:3, D:4 (64)
A:1, B:3, C:4, D:2 (64)
A:1, B:3, C:2, D:4 (64)
A:2, B:3, C:1, D:4 (64)
A:2, B:3, C:4, D:1 (65)
A:2, B:4, C:1, D:2 (64)
A:2, B:4, C:2, D:1 (65)
That's the basic idea. Many details have been left out.

Let's add a few …

Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>17</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
</tbody>
</table>

To Recap: States

- What should each state contain?
  - Enough information to:
    - represent a partial / complete solution
    - commitments made so far
    - generate its children in the state space
    - compute the bound
    - The value of the bound on this state
    - A mark or an explicit ordering
      - indicating where the next step should be taken
  - How should state expansion work?
    - For each possible option at the mark, copy the parent and take one more step
    - The next, incremental step is specified by the mark
    - Enforce the feasibility test / partial criterion function

Frontier of the Search

- How to store the set of states visited but remaining to be explored (i.e., “frontier of the search”)?
  - A set, sometimes called the “agenda” or “open list”
    - DFS: use a stack
    - BFS: use a queue
    - Most Promising First: use a priority queue
    - Other possibilities (e.g., hybrids)
  - Sound familiar?

Summary: Critical Elements of a B&B Algorithm

1. State definition
2. Initial state
3. State expansion strategy
4. Bounding function
5. BSSF
6. Agenda
7. Solution Test / Criterion function

Algorithm: Basic Control Flow

- Repeat until the agenda is empty or time is up:
  - Select state from the agenda
  - Generate children
  - Calculate the bound
  - Identify children that are solutions and update the BSSF if possible
  - Place promising children on the agenda
    - Promising means “better than the BSSF”
  - Prune the losers

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 not 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 not 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)
            end if
        end for
    end while
    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.empy() and time remains) and BSSF.cost != Agenda.first().bound do
        u ← Agenda.first()
        Agenda.remove_first()
        if (u.bound is better than BSSF.cost) then
            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
                    else
                        Agenda.add(w, w.bound)
            return BSSF
    return BSSF

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

B&B for Minimization

Minimization

Low

B(s) is lower bound

BSSF

Maximization

High

B(s) is upper bound

B(s) must always be optimistic.

B&B for Maximization

Minimization

Low

B(s) is lower bound

BSSF

Maximization

High

B(s) is upper bound

B(s) must always be optimistic.

Assignment

- Homework #23
- Due Friday
- Read the Project #7 instructions!
- Come with questions next time
- Required: Read B&B Notes

B(s) must always be optimistic.