Uniform Cost Search

Joe Steinmeyer

Adam Hartz
Find a Path

• Please find a path from Start to Goal, and make a note of how you did it.
Last Time: Graph Search

- Find path between 2 points in an arbitrary graph.

- Represent all possible paths from A with a tree:
Plan/Outlook:

• Last Week: Robots in known mazes
• This Week:
  • Robot in known maze with “tolls”
  • Mapquest/Googlemaps/OpenStreetMap Direction Finding
Graph Search Algorithm

• Basic Algorithm:
  • Initialize **agenda** (list of nodes to consider)
  • Repeat the following **until goal is found or agenda is empty**:
    • Remove one node from the agenda
    • Add its children to the agenda
  • Return resulting path
All in the Order!

- Remove first node in agenda and add its successors to the end
Find the path \( A \rightarrow I \), BFS with Nothing Else!

- A
- AB, AD
- AD, ABA, ABC, ABE
- ABA, ABC, ABE, ADA, ADE, ADG
- ABC, ABE, ADA, ADE, ADG, ABAB, ABAD
- ABE, ADA, ADE, ADG, ABAB, ABAD, ABCB, ABCF
- ...
- ...
- ...
- ...
- ...

- Shortest path eventually found ABCFI
All in the Order!

- Replace first node in agenda by its successors

DFS
Find the path $A \rightarrow I$, DFS with Nothing Else!

- A
- $AB$, $AD$
- $AB$, $ADA$, $ADE$, $ADG$
- $AB$, $ADA$, $ADE$, $ADG$, $ADGD$, $ADGH$
- $AB$, $ADA$, $ADE$, $ADG$, $ADGD$, $ADGHE$, $ADGHG$, Found!

- A path was found...

- We pushed to agenda in alphabetical order here.
Find the path $A \rightarrow I$, DFS with Nothing Else!

- A
- AD, AB
- AD, ABE, ABC, ABA
- AD, ABE, ABC, ABAD, ABAB
- AD, ABE, ABC, ABAD, ABAB, ABABE, ABABC, ABABA
- ...
- ...
- Uh oh...
- With no pruning...DFS can potentially get stuck in infinite loops/never return
Pruning Rule 1

• Don’t consider any path that visits the same state twice
Pruning Rule 2

• If multiple actions lead to the same state, consider only one of them.
Pruning Rule 3

- Don't consider any path that visits a state that you have already visited via some other path
Dynamic Programming (Pruning Rule 3)

• As applies to search: (Depends slightly on which algorithm we’re using)
• BFS: The shortest path $S \to X \to G$ is made up of the shortest path $S \to X$ and the shortest path $X \to G$
• DFS: A path $S \to X \to G$ is made up of a path $S \to X$ and a path $X \to G$

• The moral: once we have found a path $S \to X$, we don’t need to spend time looking for other paths through $X$.

• Anything found later in the search will be worse, given what we’re looking for
With Dynamic Programming

• Algorithm (including dynamic programming):
  • Initialize agenda (list of nodes to consider)
  • Initialize visited set (set of states visited)
  • Repeat the following until goal is found or agenda is empty:
    • Remove one node from the agenda
    • For each of that node’s children:
      • If its state is in the visited list, skip it
      • Otherwise, add it to agenda and add its state to visited list
  • Return resulting path

*Blue text includes additions to our base code coming from DFS and BFS
Python Framework

- **SearchNode class:**
  - **Attributes:**
    - state (arbitrary)
    - parent (instance of SearchNode, or None)
  - **Methods:**
    - path (returns list of states representing path from root)

- **search function:**
  - **Arguments:**
    - successor function (function state→list of states)
    - starting state
    - goal_test (function state→bool)
    - dfs (True for DFS, False for BFS)
def search(successors, start_state, goal_test, dfs = False):
    if goal_test(start_state):
        return [start_state]
    else:
        agenda = [SearchNode(start_state, None)]
        visited = {start_state}
        while len(agenda) > 0:
            parent = agenda.pop(-1 if dfs else 0)
            for child_state in successors(parent.state):
                child = SearchNode(child_state, parent)
                if goal_test(child_state):
                    return child.path()
                if child_state not in visited:
                    agenda.append(child)
                    visited.add(child_state)
        return None
def search(successors, start_state, goal_test, dfs = False):
    if goal_test(start_state):
        return [start_state]
    else:
        agenda = [SearchNode(start_state, None)]
        visited = {start_state}
        while len(agenda) > 0:
            parent = agenda.pop(-1 if dfs else 0)
            for child_state in successors(parent.state):
                child = SearchNode(child_state, parent)
                if goal_test(child_state):
                    return child.path()
                if child_state not in visited:
                    agenda.append(child)
                    visited.add(child_state)
        return None
def search(successors, start_state, goal_test, dfs = False):
    if goal_test(start_state):
        return [start_state]
    else:
        agenda = [SearchNode(start_state, None)]
        visited = {start_state}
        while len(agenda) > 0:
            parent = agenda.pop(-1 if dfs else 0)
            for child_state in successors(parent.state):
                child = SearchNode(child_state, parent)
                if goal_test(child_state):
                    return child.path()
                if child_state not in visited:
                    agenda.append(child)
                    visited.add(child_state)
        return None
visited holds solutions to solved problems(!)

• The visited list allows us to store and/remember what smaller problems we’ve already solved

• If in DFS, it says what states we’ve already found a path to

• If in BFS, it says what states we’ve already found the/a shortest path to
Find the path $A \rightarrow I$, BFS with Dynamic Programming!

*Push to agenda in alphabetical order*

- A
- AB, AD
- AD, ABC, ABE
- ABC, ABE, ADG
- ABE, ADG, ABCF
- ADG, ABCF, ABEH
- ABCF, ABEH
- ABEH...ABCFI analyzed and found!
Find the path $A \rightarrow I$, DFS with Dynamic Programming!

*Push to agenda in alphabetical order*

- $A$
- $AB$, $AD$
- $AB$, $ADE$, $ADG$
- $AB$, $ADE$, $ADGH$
- $AB$, $ADE$, ...$ADGHI$ and found!
Order of successors!

• The order in which we add successors can be extremely impactful.

• It won’t affect final result of BFS (shortest path will still be returned).

• DFS can be crazily affected, though.
Find the path $G \rightarrow I$, DFS with Dynamic Programming! (create successors in forward alphabetical order)

- G
- GD, GH
- GD...goal found found (GHI)
Find the path $G \rightarrow I$, DFS with Dynamic Programming! (create successors in reverse alphabetical order)

- $G$
- $GH, GD$
- $GH, GE, GDA$
- $GH, GE, GDAB$
- $GH, GE, GDABC$
- $GH, GE, GDABCF$
- $GH, GE$ goal found/returned: $GDABCFI$
Find the path $G \to I$, BFS with Dynamic Programming! (create successors in reverse alphabetical order)

- G
- GH, GD
- GD (found GHI)
Returning to Our Solving of the Maze

• How did you solve?
Our Robot

*DL10 Simulation, with Robot knowing its size and successor function returning randomized successors*

**BFS**

**DFS**

Which one are you (a human) like?
Depth First Search in Human Processing

• DFS will in general have a smaller agenda than BFS and therefore will need less storage

• Human Search is usually best approximated with DFS...and this goes for most stuff that we search through...maps, locating Waldo, etc...

• We do not have the working memory to perform BFS-like searches

• Human language-decoding as well is thought to use something similar to a depth-first search (with variations)

• We lack the memory to simultaneously consider all possible meanings of what is being conveyed
You can Tune a Piano, but you Can’t Tuna Fish

• Time Flies Like an Arrow, Fruit Flies Like a Banana.

• If I am reading this graph correctly, I would be very surprised.

• Etc...
Paraprosdokian

• A paraprosdokian is a figure of speech in which the latter part of a sentence or phrase is surprising or unexpected in a way that causes the reader or listener to re-frame or re-interpret the first part.

• Our brain tends to select meanings based off of previous cases, context. As a sentence is said, we are Depth-First Searching through meaning since we’re incapable of simultaneously tracking all possible meanings.

• Paraprosdokians and many jokes rely on the fact that we have a significant and sudden collapse of a potential solution, causing a DFS branch to be abandoned.
Cognitive Science/Humor Theory

• The kind of mind-blanking that happens when meaning is suddenly redefined with new information is us having a significant DFS branch collapse.

• If we were BFS-like individuals we might not experience this since the collapse of a particular solution branch wouldn’t be as major.

• Is the robot laughing when it realizes it can’t go down that one path?
• Do we care?
What’s a Graph?

• Set $V$ of vertices
• Set $E$ of edges connecting vertices
• Set $W$ of edge costs (“weights”)

• So far ignored weights, assumed every edge was equivalent, but what if the cost going from $D$ to $E$ was not the same as going from $D$ to $G$?
Consider searching for **least-cost** paths instead of shortest paths. Instead of popping from agenda based on when nodes were added, pop based on the cost of the paths they represent.

Slight change to framework:

- **SearchTree** class:
  - Attributes:
    - state (arbitrary)
    - parent (instance of SearchNode, or None)
    - **cost of whole path from start**
  - Methods:
    - path (returns list of states representing path from root)

- **uniform_cost_search** function:
  - Arguments:
    - successor function (state→list of (state,cost) tuples)
    - starting state
    - goal test (function state→bool)
def uniform cost search(successors, start_state, goal_test):
    if goal test(start_state):
        return [start_state]
    agenda = [(0, SearchNode(start_state, None, cost=0))]
    expanded = set()
    while len(agenda) > 0:
        agenda.sort()
        priority, parent = agenda.pop(0)
        if parent.state not in expanded:
            expanded.add(parent.state)
        if goal test(parent.state):
            return parent.path()
        for child_state, cost in successors(parent.state):
            child = SearchNode(child_state, parent, parent.cost+cost)
            if child_state not in expanded:
                agenda.append((child.cost, child))
    return None

- Testing for goal condition must be done at expansion time, not at visit time. Similarly for dynamic programming!!!
Find the path $E \rightarrow I$, Uniform Cost Search

*Blue* is new children, $\rightarrow$ means call to .sort() on agenda,
Initial push in alphabetical order, cost in trailing parentheses

- E
- $\text{EB}(1), \text{ED}(0.5), \text{EF}(1), \text{EH}(1) \rightarrow \text{ED}(0.5)$, EB(1), EF(1), EH(1)
- $\text{EB}(1), \text{EF}(1), \text{EH}(1), \text{EDA}(1.5), \text{EDG}(1.5) \rightarrow \text{EB}(1)$, EDF(1), EH(1), EDA(1.5), EDG(1.5)
- $\text{EF}(1), \text{EH}(1), \text{EDA}(1.5), \text{EDG}(1.5), \text{EBA}(2), \text{EBC}(6) \rightarrow \text{EF}(1)$, EH(1), EDA(1.5), EDG(1.5), EBA(2), EBC(6)
- $\text{EH}(1), \text{EDA}(1.5), \text{EDG}(1.5), \text{EBA}(2), \text{EBC}(6), \text{EFC}(2), \text{EFI}(2) \rightarrow \text{EH}(1)$, EDA(1.5), EDG(1.5), EBA(2), EFC(2), EFI(2)
- $\text{EDA}(1.5), \text{EDG}(1.5), \text{EBA}(2), \text{EFC}(2), \text{EFI}(2) \text{ EBC}(6)$
- $\text{EDG}(1.5)$, $\text{EHI}(1.5)$, $\text{EBA}(2), \text{EFC}(2), \text{EFI}(2), \text{EHG}(2), \text{EBC}(6)$
- $\text{EHI}(1.5), \text{EBA}(2), \text{EFC}(2), \text{EFI}(2), \text{EHG}(2), \text{EBC}(6) \rightarrow \text{EHI}(1.5), \text{EBA}(2), \text{EFC}(2), \text{EFI}(2), \text{EHG}(2), \text{EBC}(6)$
- $\text{EBA}(2), \text{EFC}(2), \text{EFI}(2), \text{EHG}(2), \text{EBC}(6) \rightarrow \text{SOLUTION FOUND!...EHI, cost 1.5!}$
Take-Aways

• Must call `sort()` on our agenda on each iteration (has a cost!)
• `goal_test` analyzed upon expansion of state, not when visiting the state.
• For dynamic programming: No longer remembering what we visited, only what we expanded
  • Ensures we find and remember the cheapest path to each state!
  • We want to be cost-savvy...so we never take the first offer! Only once all offers are in!
Cost?

• Cost can now take into account many factors!
• Search of US can be based off of distances not arbitrary number of junctions between points
• Cost can express preferences beyond distance:
  • Highways (weight less than the normal roads)
  • Toll roads (avoid them)
  • Projected fuel economy/speed
  • Traffic!
  • Scenic routes vs. ones that go through Ohio
• Will route take you by where your ex lives that you’d prefer not to run into?
With our Robot...

*It can now go diagonal!*
Things to Watch out for (Inaccurate Costs)

• Costs that reflect true cost incorrectly or give free lunches

• Negative Cost...this can be a reasonable thing to implement, **but** be careful...the system will try to minimize it

If I let it go diagonal, but don’t properly reflect that in costs (have every move be the same), the robot doesn’t care and will gladly move diagonal

Cost of diagonals became negative 😄
A Problem

• So far, searches have radiated outward from the starting point.
• We only notice the goal when the system stumbles upon it.

Too much time spent searching on the wrong side of the goal.
Returning to Our Solving of the Maze

• How did you solve?
Heuristics

• So far, our searches only consider start-to-current. We can add heuristics to consider an estimate of current-to-goal as well.

• $h(x)$ estimate of cost of lowest-cost path $X \rightarrow \text{goal}$

  • SearchNode class:
    • Attributes:
      • state (arbitrary)
      • parent (instance of SearchNode, or None)
      • cost of whole path from start
    • Methods:
      • path (returns list of states representing path from root)
  
  • uniform_cost_search function:
    • Arguments:
      • successor function (state→list of (state,cost) tuples)
      • starting state
      • goal test (function state→bool)
    • heuristic (function state→estimated cost to goal)
Heuristics

• Ideally we’d like a heuristic to perfectly estimate the remaining cost from any state to the goal
• This is often hard to do
Admissible Heuristic

• An **admissible heuristic** does not overestimate the actual cost of the shortest cost path.

• If the heuristic $h(s)$ is larger than the actual cost from $s$ to goal, then the “best” solution may be missed!

• If the heuristic is an underestimate, the search space will be larger than necessary, but we are guaranteed the shortest path.

• The ideal heuristic should be:
  • as close as possible to actual cost (without overestimating)
  • easy to calculate

• **A* (without DP) is guaranteed to find least-cost path if heuristic is admissible.**

• With DP, heuristic must also be consistent.
def uniform_cost_search(successors, start_state, goal_test, heuristic=lambda s: 0):
    if goal_test(start_state):
        return [start_state]
    agenda = [(heuristic(start_state), SearchNode(start_state, None, cost=0))]
    expanded = set()
    while len(agenda) > 0:
        agenda.sort()
        priority, parent = agenda.pop(0)
        if parent.state not in expanded:
            expanded.add(parent.state)
            if goal_test(parent.state):
                return parent.path()
        for child_state, cost in successors(parent.state):
            child = SearchNode(child_state, parent, parent.cost+cost)
            if child_state not in expanded:
                agenda.append((child.cost + heuristic(child_state), child))
    return None
Find the path $E \rightarrow I$, Uniform Cost Search, heuristic: $h(s) = M(s, I)/2$

Blue is new children, $\rightarrow$ means call to .sort() on agenda, Initial push in alphabetical order, cost in trailing parentheses

- E
- $\text{EB}(2.5), \text{ED}(2.0), \text{EF}(1.5), \text{EH}(1.5) \rightarrow \text{EF}(1.5), \text{EH}(1.5), \text{ED}(2.0), \text{EB}(2.5)$
- $\text{EH}(1.5), \text{ED}(2.0), \text{EB}(2.5), \text{EFC}(3), \text{EFI}(2) \rightarrow \text{EH}(1.5), \text{ED}(2.0), \text{EFIT}(2), \text{EB}(2.5), \text{EFC}(3)$
- $\text{ED}(2.0), \text{EFIT}(2), \text{EB}(2.5), \text{EFC}(3), \text{EHG}(3), \text{EHI}(1.5) \rightarrow \text{EHI}(1.5), \text{ED}(2.0), \text{EFIT}(2), \text{EB}(2.5), \text{EFC}(3), \text{EHG}(3)$
- $\text{ED}(2.0), \text{EFIT}(2), \text{EB}(2.5), \text{EFC}(3), \text{EHG}(3).... (\text{SOLUTION FOUND}!... \text{EHI, cost 1.5!})$
Question!

• Consider searching in a four-action grid (up, down, left, right), where all actions have cost 1. Let \((r_0, c_0)\) represent the current location, and \((r_1, c_1)\) represent the goal.

• Which of the following heuristics are admissible?
  1. \(\text{abs}(r_0-r_1) + \text{abs}(c_0-c_1)\)
  2. \(\min(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))\)
  3. \(\max(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))\)
  4. \(2\times\min(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))\)
  5. \(2\times\max(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))\)

Or in other words...which of these will never overestimate the true remaining cost from any given node to the goal?
Question!

• Which of the admissible heuristics minimizes the number of nodes expanded?

1. $\text{abs}(r_0-r_1) + \text{abs}(c_0-c_1)$
2. $\text{min}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$
3. $\text{max}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$
4. $2\times\text{min}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$
5. $2\times\text{max}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$

Or in other words...which one is as close to an ideal estimator as possible?
$$\text{abs}(r_0 - r_1) + \text{abs}(c_0 - c_1)$$
\[ \min(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1)) \]
\( \max(\text{abs}(r_0 - r_1), \text{abs}(c_0 - c_1)) \)
2*\min(\abs(r_0-r_1), \abs(c_0-c_1))
## Results

### The Manhattan Distance

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>States Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{abs}(r_0-r_1) + \text{abs}(c_0-c_1)$</td>
<td>42</td>
</tr>
<tr>
<td>$\text{min}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$</td>
<td>114</td>
</tr>
<tr>
<td>$\text{max}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$</td>
<td>60</td>
</tr>
<tr>
<td>$2\times\text{min}(\text{abs}(r_0-r_1), \text{abs}(c_0-c_1))$</td>
<td>72</td>
</tr>
</tbody>
</table>
Word Ladder with A*

- **UC**
  - States expanded: 396
  - ['quiz', 'quit', 'suit', 'slit', 'slat', 'seat', 'beat', 'best']

- What could a good, admissible heuristic for the word ladder be?
- Optimizing over $f(x) = g(x) + h(x)$ where $g(x)$ is cost so far and $h(x)$ is estimated remaining cost to goal

- Make $h(x)$ be letters different from goal text e.g. $h(‘slat’) = 3$, $h(‘seat’) = 2$

- **A**
  - States expanded: 28
  - ['quiz', 'quit', 'suit', 'slit', 'slat', 'seat', 'beat', 'best']

- quid
Example: Eight-Puzzle

- Large number of board configurations (states):
  - $9! = 362,880$ (if you count all)
  - $9!/2 = 181,440$ accessible from start state
- Almost half of accessible states ($84,516$) are expanded by UC.
Example: Eight-Puzzle
Question:

• Consider three heuristics for the “eight puzzle”:
  1. 0
  2. number of tiles out of place
  3. sum over tiles of Manhattan distances to their goals

• Let $M_i =$ num. moves in the best solution when using heuristic $i$ (from above)
• Let $E_i =$ num. states expanded when using heuristic $i$ (from above)

• Which of the following are true?
  1. $M_1 = M_2 = M_3$
  2. $M_1 > M_2 > M_3$
  3. $E_1 = E_2 = E_3$
  4. $E_1 \geq E_2 \geq E_3$
  5. the same “best” solution results from all three heuristics
Results

• Heuristics:
  1. 0
  2. number of tiles out of place
  3. sum over tiles of Manhattan distances to their goals

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Expanded</th>
<th>Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>84,516</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>8,329</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>1,348</td>
<td>22</td>
</tr>
</tbody>
</table>
In Summary...

- Developed a new class of search algorithms: uniform cost
- Developed a new class of optimizations: heuristics
- Summary of Search Algorithm:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Agenda</th>
<th>Goal Test</th>
<th>DP</th>
<th>Guarantees</th>
</tr>
</thead>
</table>

* Provided a Path exists
* In a finite search domain
This Week

• Will be fun labs this week, but last labs :-(

• Old lab (not doing)...maybe the basis for a final project this year!
Things Due!

• EX11 due After break!
• Exam on Tuesday 11/21/2017 (Location: 6.01 Lab)
• Final Project Abstract Due today (11/14/2017!!)
• Final Project Proposal Due Tuesday 11/21/2017
Extra UC Practice/Example

• These examples were too much so I shoved them at end...but might be good reference
Find the path $A \rightarrow I$, Uniform Cost Search

Blue is new children, $\rightarrow$ means call to .sort() on agenda,
Initial push in alphabetical order, cost in trailing parentheses

- $A$
- $AB(1), AD(1) \rightarrow AB(1), AD(1)$
- $AD(1), ABC(6), ABE(2) \rightarrow AD(1), ABE(2), ABC(6)$
- $ABE(2), ABC(6), ADE(1.5), ADG(2) \rightarrow ADE(1.5), ABE(2), ADG(2), ABC(6)$
- $ABE(2), ADG(2), ABC(6), ADEF(2), ADEH(2) \rightarrow ABE(2), ADG(2), ADEF(2), ADEH(2), ABC(6)$
- $ADG(2), ADEF(2), ADEH(2), ABC(6) \rightarrow ADG(2), ADEF(2), ADEH(2), ABC(6)$
- $ADEF(2), ADEH(2), ABC(6), ADGH(3) \rightarrow ADEF(2), ADEH(2), ADGH(3), ABC(6)$
- $ADEH(2), ADGH(3), ABC(6), ADEF(3.5), ADEFI(3.5) \rightarrow ADEH(2), ADGH(3), ADEFC(3.5), ADEFI(3.5), ABC(6)$
- $ADGH(3), ADEFC(3.5), ADEFI(3.5), ABC(6), ADEHI(3) \rightarrow ADGH(3), ADEHI(3), ADEFC(3.5), ADEFI(3.5), ABC(6)$
- $ADEHI(3), ADEFC(3.5), ADEFI(3.5), ABC(6) \rightarrow ADEHI(3), ADEFC(3.5), ADEFI(3.5), ABC(6)$
- $ADEFC(3.5), ADEFI(3.5), ABC(6),...$ (SOLUTION FOUND!...ADEHI, cost 3.0)
Find the path $A \rightarrow I$, Uniform Cost Search, heuristic: $h(s) = M(s, I)/2$

Blue is new children, $\rightarrow$ means call to .sort() on agenda, Initial push in alphabetical order, cost in trailing parentheses

- A
- $AB(2.5), AD(2.5) \rightarrow AB(2.5), AD(2.5)$
- $AD(2.5), ABC(7), ABE(3) \rightarrow AD(2.5), ABE(3), ABC(7)$
- $ABE(3), ABC(7), ADE(2.5), ADG(3) \rightarrow ADE(2.5), ABE(3), ADG(3), ABC(7)$
- $ABE(3), ADG(3), ABC(7), ADEF(2.5), ADEH(2.5) \rightarrow ADEF(2.5), ADEH(2.5), ABE(3), ADG(3), ABC(7)$
- $ADEH(2.5), ABE(3), ADG(3), ABC(7), ADEFC(4.5), ADEFI(3.5) \rightarrow ADEH(2.5), ABE(3), ADG(3), ADEFI(3.5), ABC(7)$
- $ABE(3), ADG(3), ADEFI(3.5), ADEFC(4.5), ABC(7), ADEHG(4.5), ADEHI(3) \rightarrow ABE(3), ADG(3), ADEHI(3), ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7)$
- $ADG(3), ADEHI(3), ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7) \rightarrow ADG(3), ADEHI(3), ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7)$
- $ADEHI(3), ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7) \rightarrow ADEHI(3), ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7)$
- $ADEFI(3.5), ADEFC(4.5), ADEHG(4.5), ABC(7).... (\text{SOLUTION FOUND!}... ADEHI, \text{ cost 3.0})$