Frontier Search

- In spaces that grow exponentially, the frontier is much larger than the rest of the search space
- In polynomial spaces, the frontier is smaller than the rest of the search space
- Keep only frontier in memory
- Memory Savings (grid example)

Example Problem

- Sequence alignment
  - k sequences, each length n
  - Cost based on whether characters in sequences align
  - total problem memory $O(n^k)$
  - If we only keep frontier, only need $O(n^{k-1})$
Frontier Search

- How can we reconstruct a solution
  - Normally need back pointers at each node
  - In this case, would only give us 1 node from the optimal path
- If we repeat, we do $N^2$ work
- Solution; remember nodes earlier in the space

Divide solution space (Approach 1)

- In a grid, for each node, remember the parent on the optimal path with $x==y$ (depends on axis coordinates)
  - Finds a node roughly half way between the start and the goal
  - Recursively solve each sub-problem

Divide solution space (Approach 2)

- Bidirectional search
  - Perform a bi-direction search
  - When frontiers match, you have a single node on an optimal path from the start to the goal
  - Recursively solve each sub-problem
- At some point you can just use another algorithm to find complete paths

Divide Solution Space (Approach 3)

- On each path, remember the node for which $g==h$
  - Will be approx. half-way between the start and the goal
How much savings?

- On grid-based problems
  - Save a factor of $n$ -- length of each axis
- What if we want to do a BFS through an entire state space?

<table>
<thead>
<tr>
<th>Size</th>
<th>Tiles</th>
<th>Moves</th>
<th>Total States</th>
<th>Max Width</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x2</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>2</td>
<td>6.000</td>
</tr>
<tr>
<td>2x3</td>
<td>5</td>
<td>21</td>
<td>360</td>
<td>44</td>
<td>8.182</td>
</tr>
<tr>
<td>2x4</td>
<td>7</td>
<td>37</td>
<td>20,160</td>
<td>1,999</td>
<td>10.085</td>
</tr>
<tr>
<td>3x3</td>
<td>8</td>
<td>31</td>
<td>181,440</td>
<td>24,047</td>
<td>7.545</td>
</tr>
<tr>
<td>2x5</td>
<td>9</td>
<td>55</td>
<td>1,814,400</td>
<td>133,107</td>
<td>13.361</td>
</tr>
<tr>
<td>2x6</td>
<td>11</td>
<td>80</td>
<td>239,500,800</td>
<td>13,002,649</td>
<td>18.419</td>
</tr>
<tr>
<td>3x4</td>
<td>11</td>
<td>53</td>
<td>239,500,800</td>
<td>21,841,159</td>
<td>10.966</td>
</tr>
<tr>
<td>2x7</td>
<td>13</td>
<td>108</td>
<td>43,589,145,600</td>
<td>1,862,320,864</td>
<td>23.406</td>
</tr>
<tr>
<td>3x5</td>
<td>14</td>
<td>84</td>
<td>653,837,184,000</td>
<td>45,136,428,198</td>
<td>14.486</td>
</tr>
<tr>
<td>2x8</td>
<td>15</td>
<td>140</td>
<td>10,461,394,944,000</td>
<td>367,084,684,402</td>
<td>28.499</td>
</tr>
<tr>
<td>4x4</td>
<td>15</td>
<td>80</td>
<td>10,461,394,944,000</td>
<td>784,195,801,886</td>
<td>13.340</td>
</tr>
</tbody>
</table>

Parallel Algorithms

- For many years it looked like our problems would be solved by increases in processor speed
- Moore’s law predicts the # of transistors, not the speed of the chip
  - The number of transistors continues to grow
  - Processor speed hasn’t

Parallel Search - Simple approaches

- IDA*
  - One processor for each iteration
  - Total speed up? (linear - at most 2x)
- DFBnB
  - Bounds shared between processors
  - Can be much faster, especially if better bounds are found