COMP 4752
Computational Intelligence

Lecture 9

Heuristic Search Improvements

Depth-First Branch and Bound

Starcraft AI

A* Search – Memory

- A* search stores nodes in the search tree, so the memory required may far exceed the number of states in the problem
- Worst-Case Memory Required: $O(b^m)$
  - Heuristic may guide us to bad parts of tree
- A* not used in practice for large problems
- How can we reduce memory usage?

Iterative Deepening A* (IDA*)

- Iterative deepening used in DFS to ensure the search was complete
- Instead of using a depth limit like DFS, IDA* uses an increasing f-value cutoff
- But memory / CPU usage still high due to storing and sorting the entire open list

Recursive Best-First Search (RBFS)

- Recall DFS can be implemented using the Tree-Search algorithm with a fringe stack
- However, a better implementation of DFS was using a recursive function
- Similarly, we can implement Best-First Search algorithms as recursive functions
- Mimics the operations of standard BeFS, but only uses linear space

Recursive Best-First Search (RBFS)

- Keeps track of the f-value of the best alternative path from any ancestor of the current node
- If the current node exceeds this limit, recursion unwinds back to the alternative path
- As recursion unwinds, RBFS replaces the f value of each node along the path with the best f-value of its children
- RBGS retains the f-value of the best leaf in the forgotten subtree and can decide whether it should reexpand that subtree at a later time

Recursive Best-First Search (RBFS)

```
1.  Function RBFS(problem, node, f_limit)
2.      if (node.state == goal) return node
3.      successors = Expand(node, problem)
4.      if (successors.empty()) return (failure, infinity)
5.      for s in successors:
6.          s.f = max(s.g + h(s), node.f)
7.      repeat:
8.          best_node = min_f_node(successors)
9.          if (best_node.f > f_limit) return (falure, best.f)
10.         alt_node = second_lowest_f(successors)
11.         result, best_node.f = RBFS(problem, best_node, min(f_limit, alt_node))
12.         if (result != failure) return result
```

RBFS Performance

- Like A*, RBFS is optimal if the heuristic function h(n) is admissible
- Memory usage is linear in search depth O(d)
- Time complexity is difficult to analyze
  - Accuracy of heuristics?
  - How often does the best path change?

Memory-Bounded A*

- Simple Memory-Bounded A* (SMA*)
- Use all available system memory
- When all memory used and we need to add a node, erase the highest f-value node
- SMA* then backs up the value of the forgotten node to its parent
- The ancestor of a forgotten subtree knows the quality of the best path in that subtree
- In case of ties, SMA* selects the newest node generated, and deletes the oldest node generated
- SMA* is complete/optimal only if depth of shallowest goal < memory

Branch and Bound

Branch & Bound Algorithms

- “Any-time” search algorithm
  - Will continue to find increasingly better solutions until the you stop the algorithm
  - If run to completion, will find optimal solution
- Choose an initial search cutoff bound and search all solutions up to that bound, when a solution is found, update the bound to the cost of that solution

Depth-First Branch and Bound (DFBB)

```
best_path = []
bound = infinity
Function DFBB(problem, node, bound)
        if (node.state == goal)
            best_path = reconstruct_path(node)
            bound = path_cost(best_path)
        children = Expand(node, problem)
        sort(children, h())                          # optional move-ordering
        for child in Expand(node, problem):
            if (child.g + h(child) > bound) continue   # h must be admissible
            DFBB(problem, child, bound)
```

DFBB Example: Starcraft

Real-Time Strategy

- War-like Simulation
- Single / Multiplayer Games

- Most RTS Games:
  - Gather Resources
  - Build Town / Army
  - Combat With Enemies

Properties of RTS

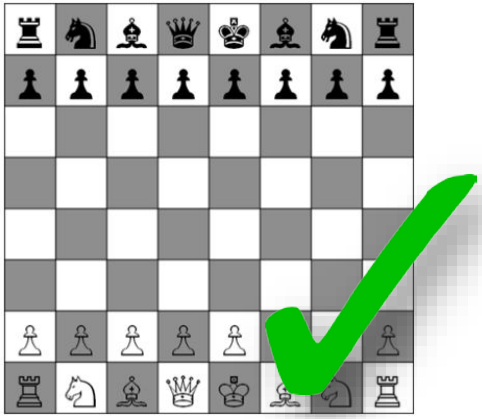
- Real-Time
- Simultaneous Move
- Non-Deterministic
- Imperfect Information
- Multi-Unit Control
- Unknown Game Engine
- Action / State Space

Benefits of RTS AI

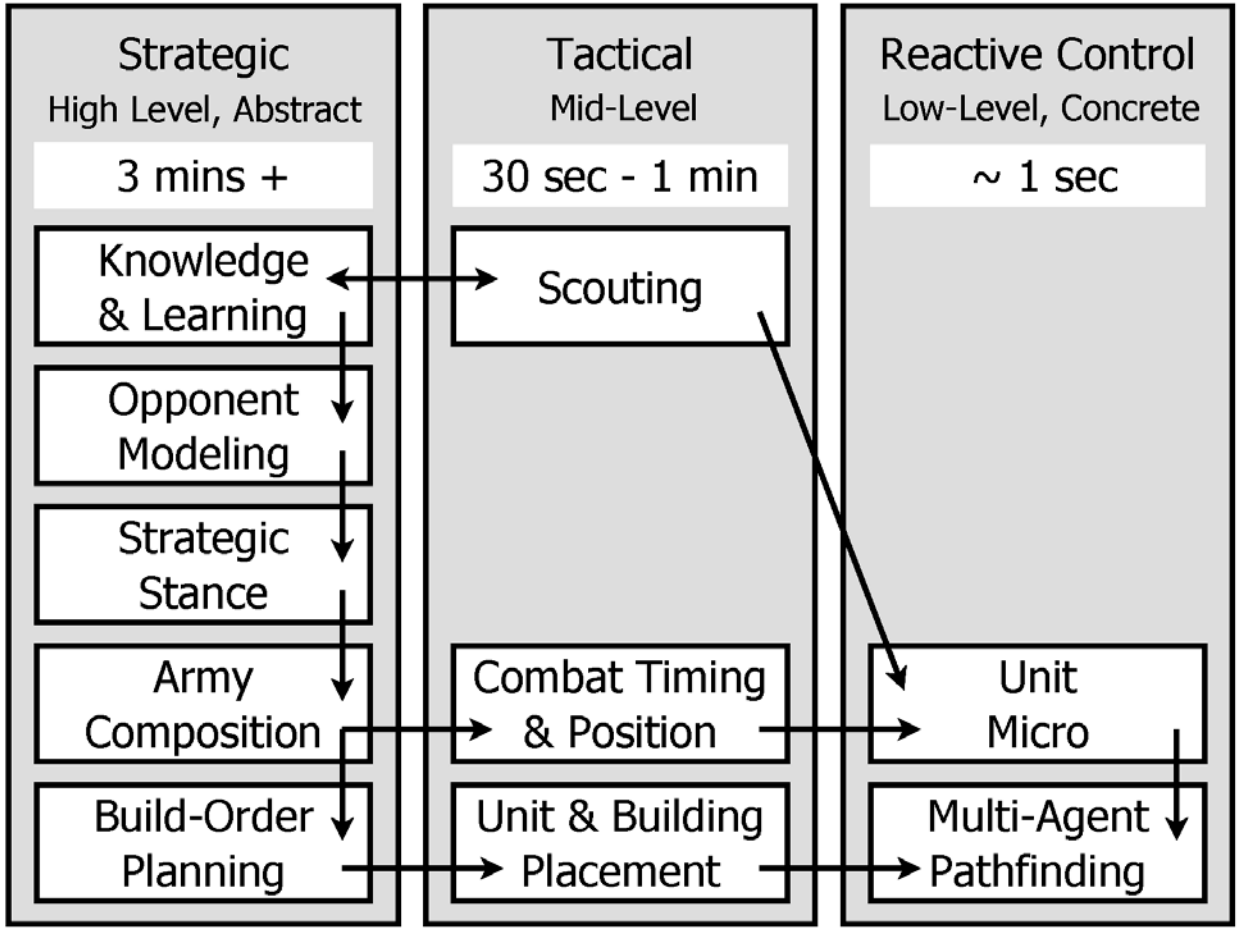
- Better In-Game AI
  - More intelligent NPCs
  - Better single player
- Create Offline Tools
  - Game balancing
  - Reduce human testing
- Apply to any game

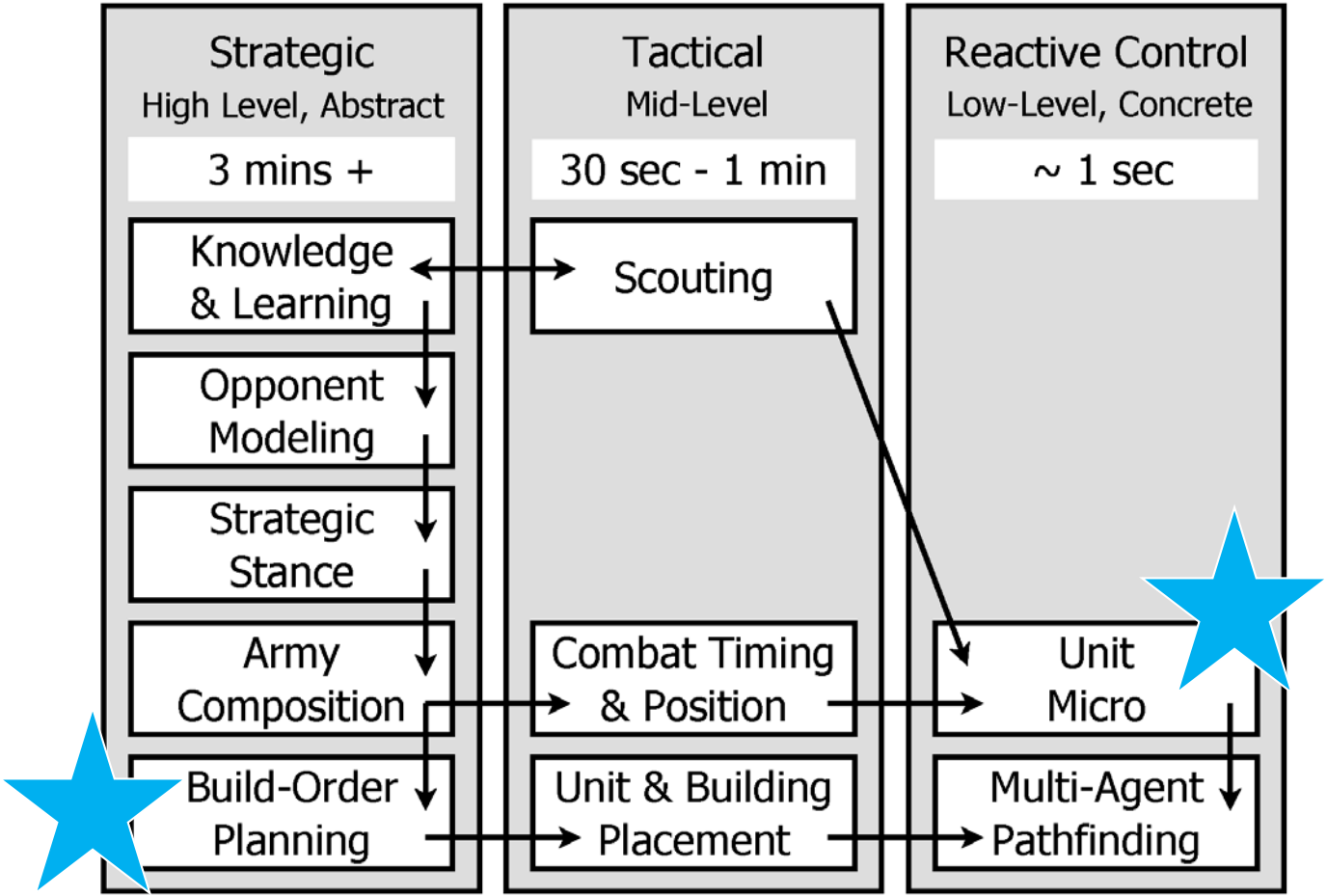
eSports

Human vs. Machine



StarCraft (Huge)

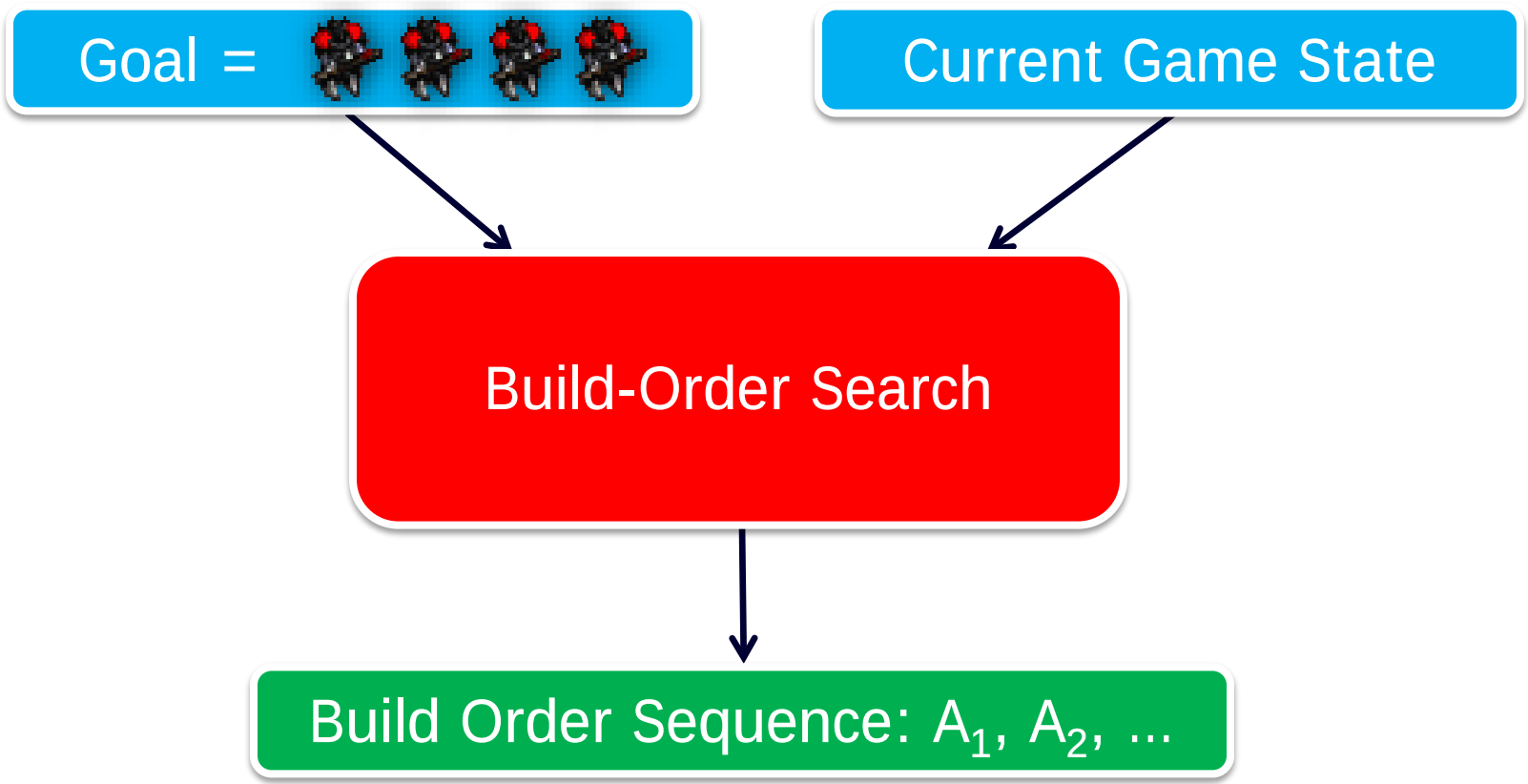

Strategic
High Level, Abstract
3 mins +
Tactical
Mid-Level
30 sec - 1 min
Reactive Control
Low-Level, Concrete
~ 1 sec
Knowledge & Learning
Scouting
Opponent Modeling
Strategic Stance
Army Composition
Combat Timing & Position
Unit Micro
Build-Order Planning
Unit & Building Placement
Multi-Agent Pathfinding


Strategic
High Level, Abstract
3 mins +
Knowledge & Learning
Opponent Modeling
Strategic Stance
Army Composition
Build-Order Planning
Tactical
Mid-Level
30 sec - 1 min
Scouting
Combat Timing & Position
Unit & Building Placement
Reactive Control
Low-Level, Concrete
~ 1 sec
Unit Micro
Multi-Agent Pathfinding

Build-Order Planning and Optimization

What is a Build-Order?

- Sequence of economic actions
- List of buildings / units to build in order
- Players memorize ‘opening books’


Goal =
Current Game State
Build-Order Search
Build Order Sequence: A_1, A_2, ...

Algorithm - DFBB

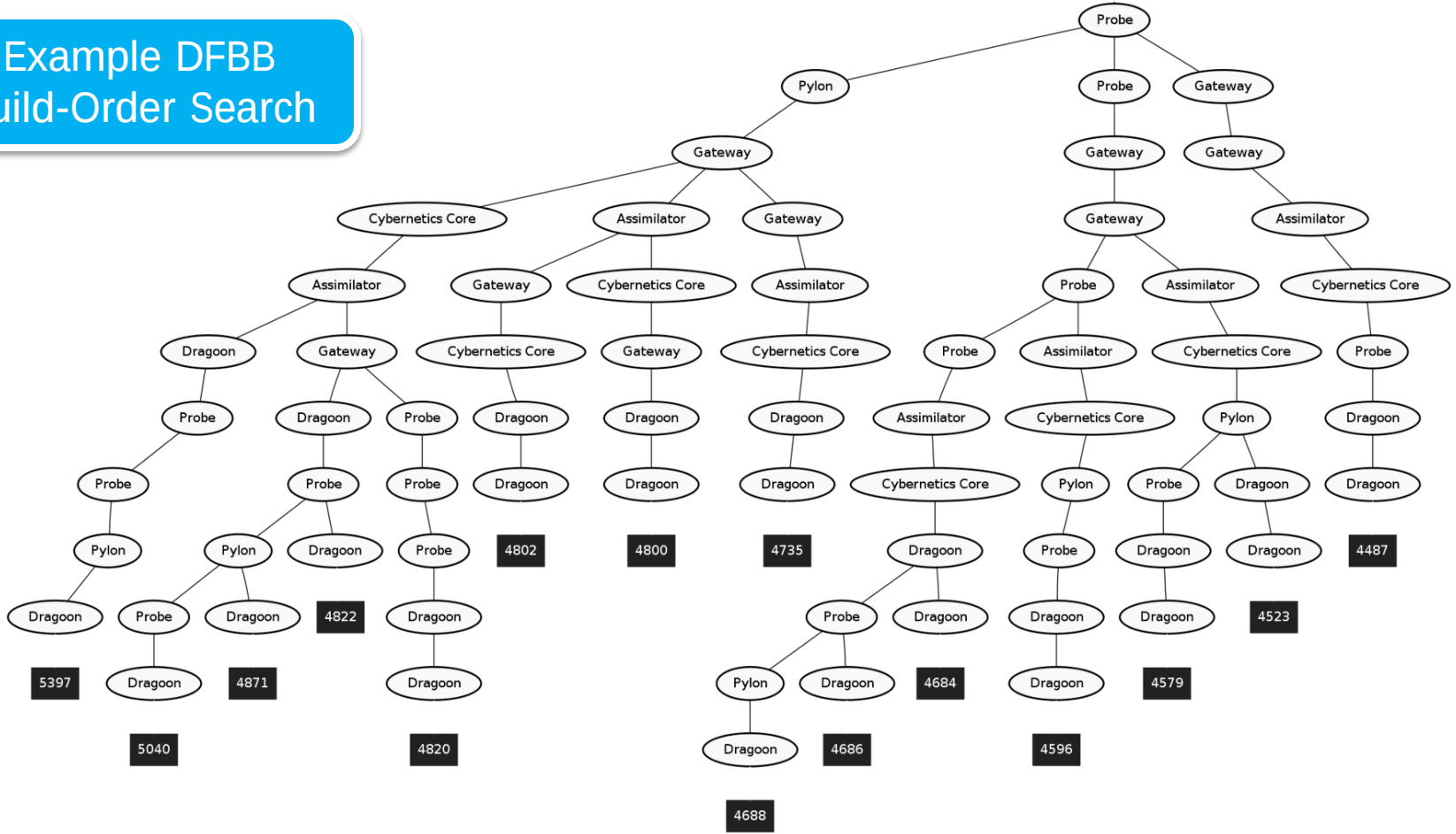
- Depth-First Branch and Bound
- Optimize: Build-Order Makespan

Algorithm - DFBB

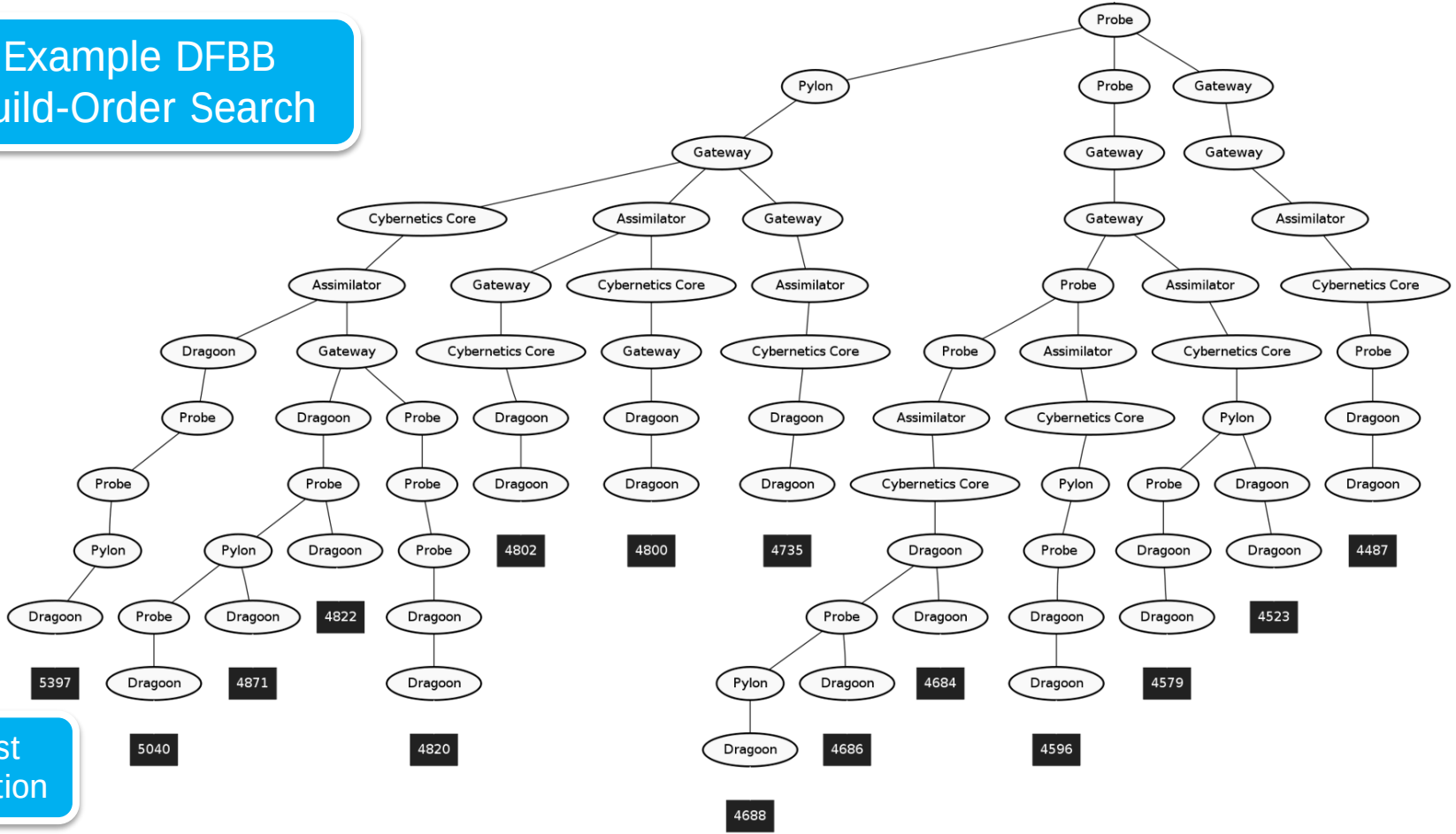
- Depth-First Branch and Bound
- Optimize: Build-Order Makespan

- Low Memory Usage
- Any-Time
- Save / Load Search
- Upper + Lower Bound Heuristics

Example DFBB Build-Order Search

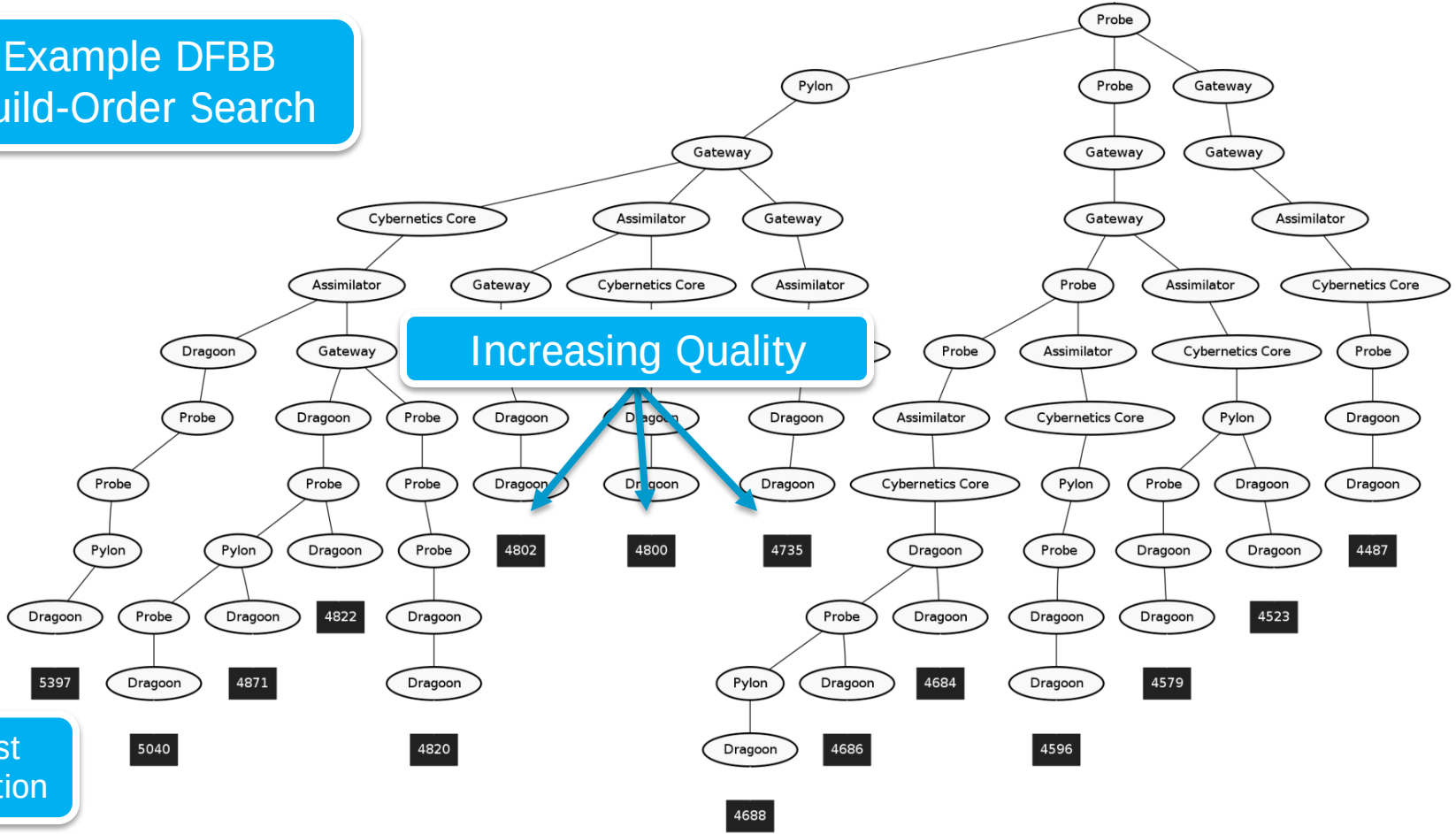


Example DFBB Build-Order Search

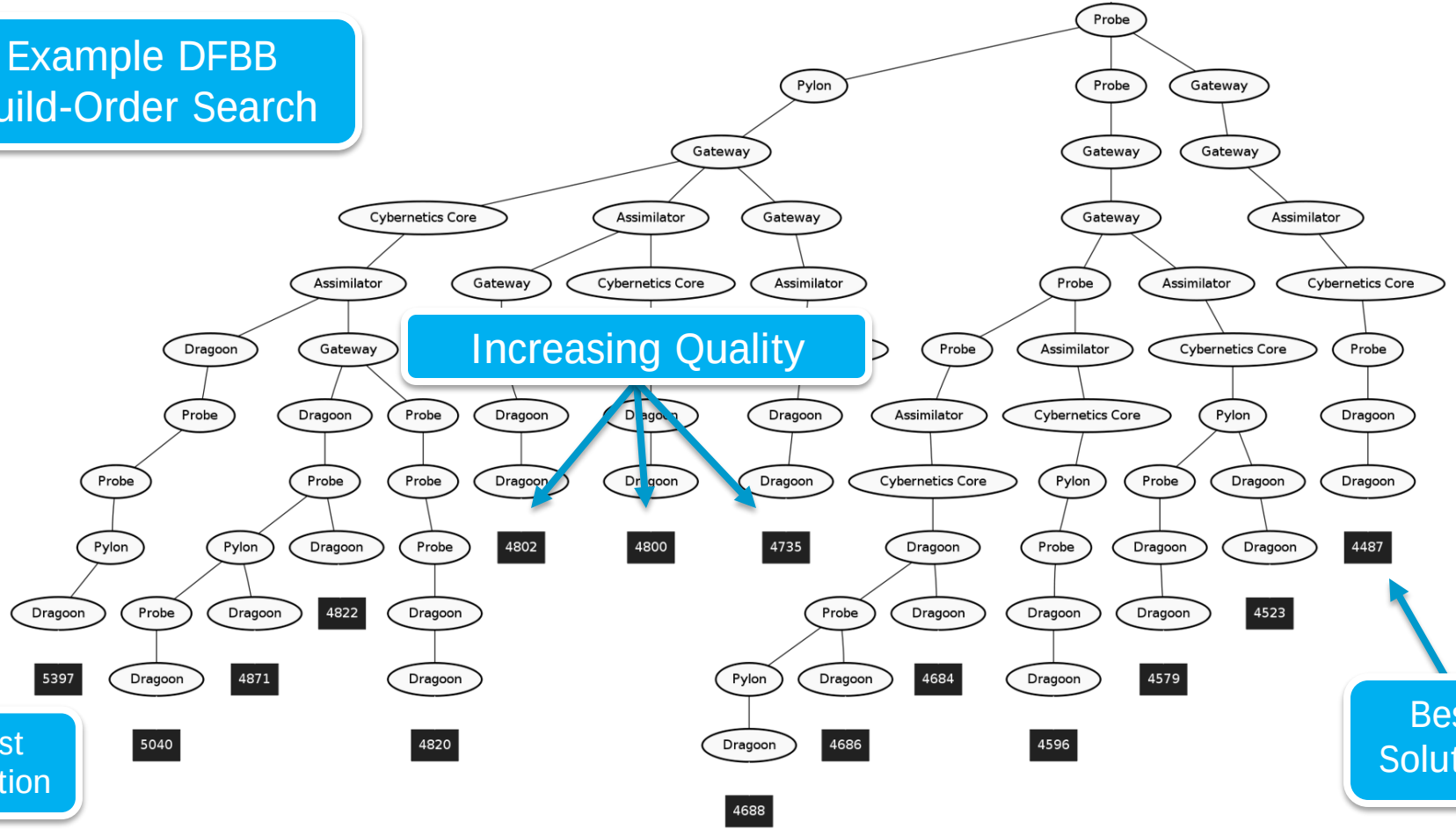


First Solution

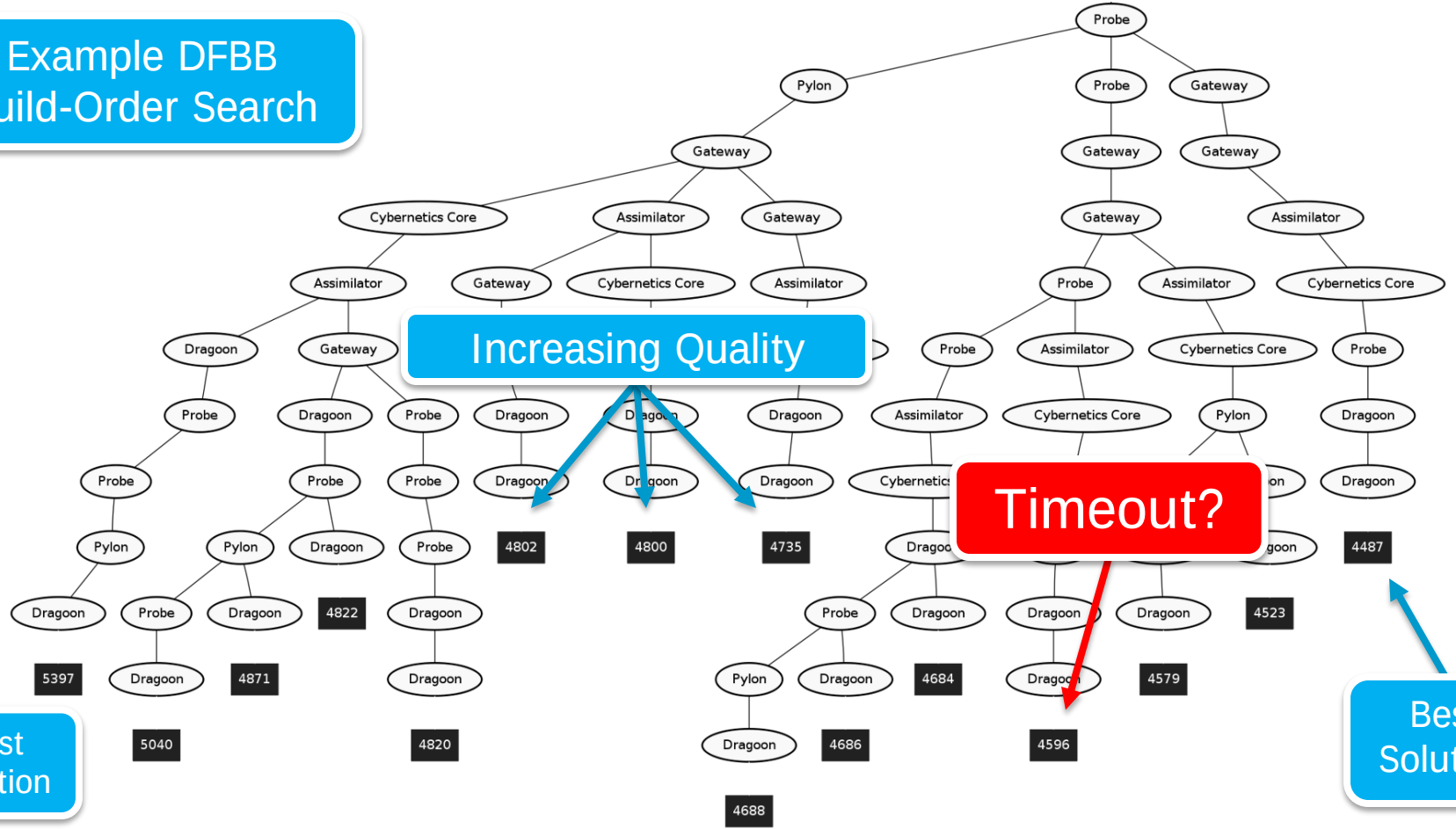
Example DFBB Build-Order Search



Example DFBB Build-Order Search



Example DFBB Build-Order Search



Pro Human Results

- Extract Pro Build-Orders
- Re-plan with BOSS
- Compare to original

- Results:
  - 100% of build orders solved
  - Faster than humans
  - Solve Time: 4% of makespan

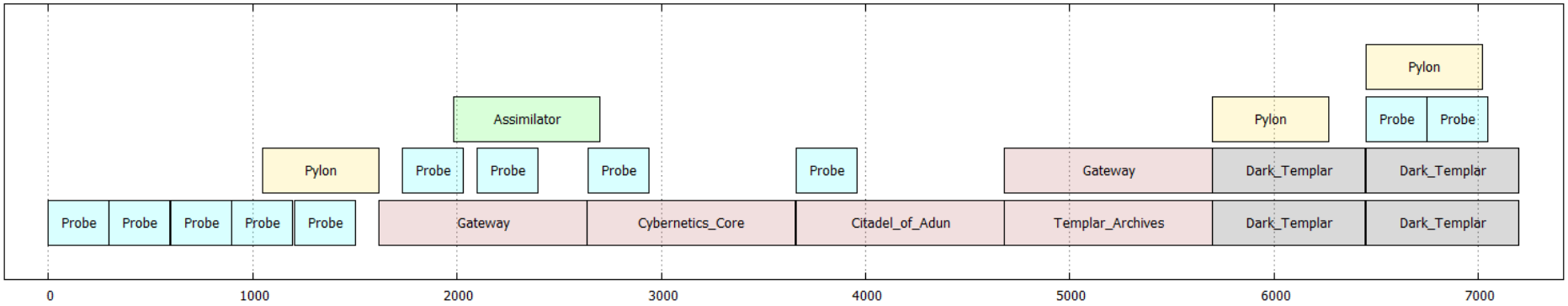
Build-Order Heuristics

- Resource Gathering Bound
- Landmark Heuristic

- Macro Actions
  - Repeat actions common in pro build orders

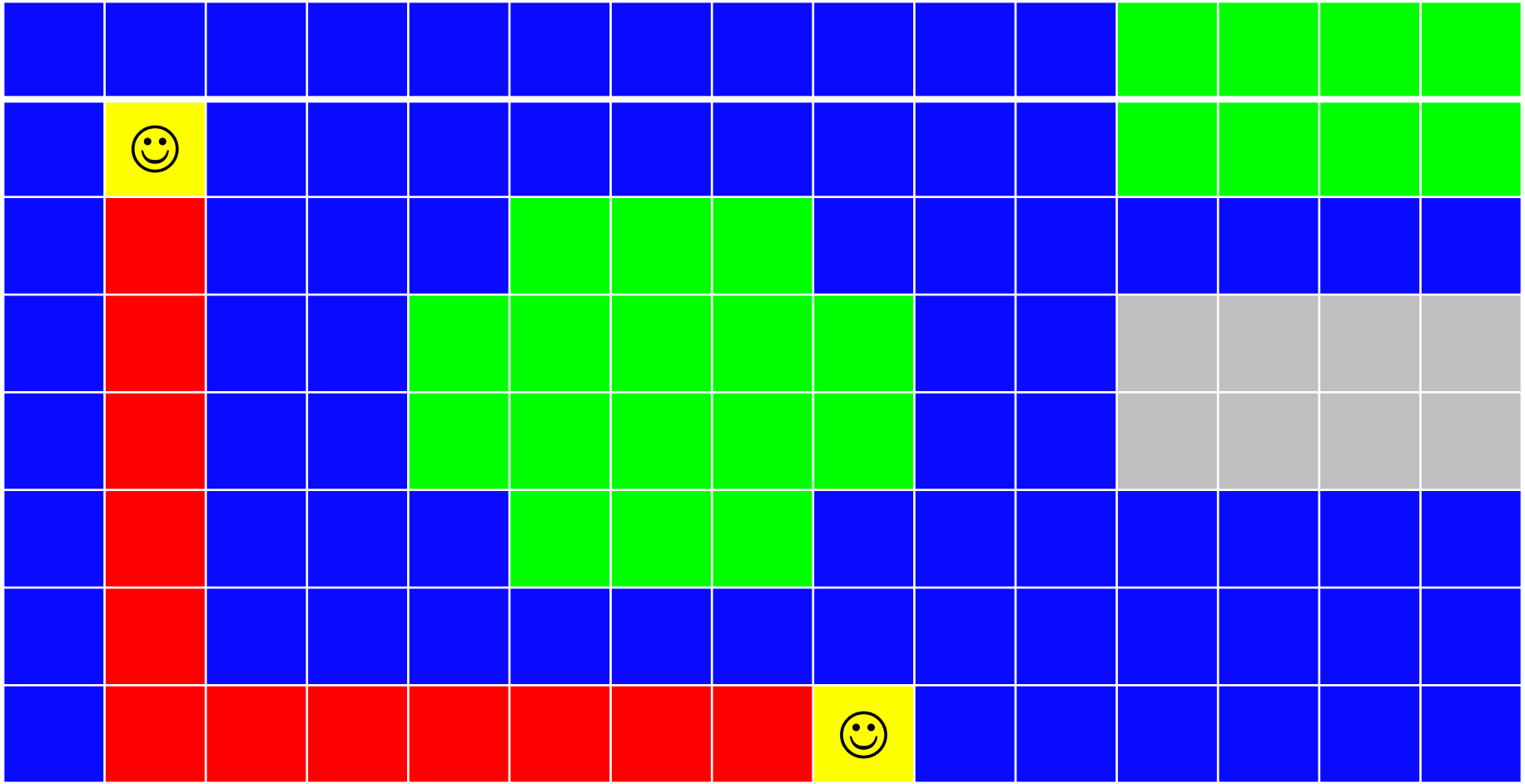
Landmark Heuristic

- How long do we need to build the sequential prerequisites of an action



Macro Actions

- Some problems have common sequences of actions that appear in optimal paths
- These commonly repeated action sequences are called “macro actions”
- Adding macro actions to the search can sometimes help us “shortcut” to better nodes further down the optimal path

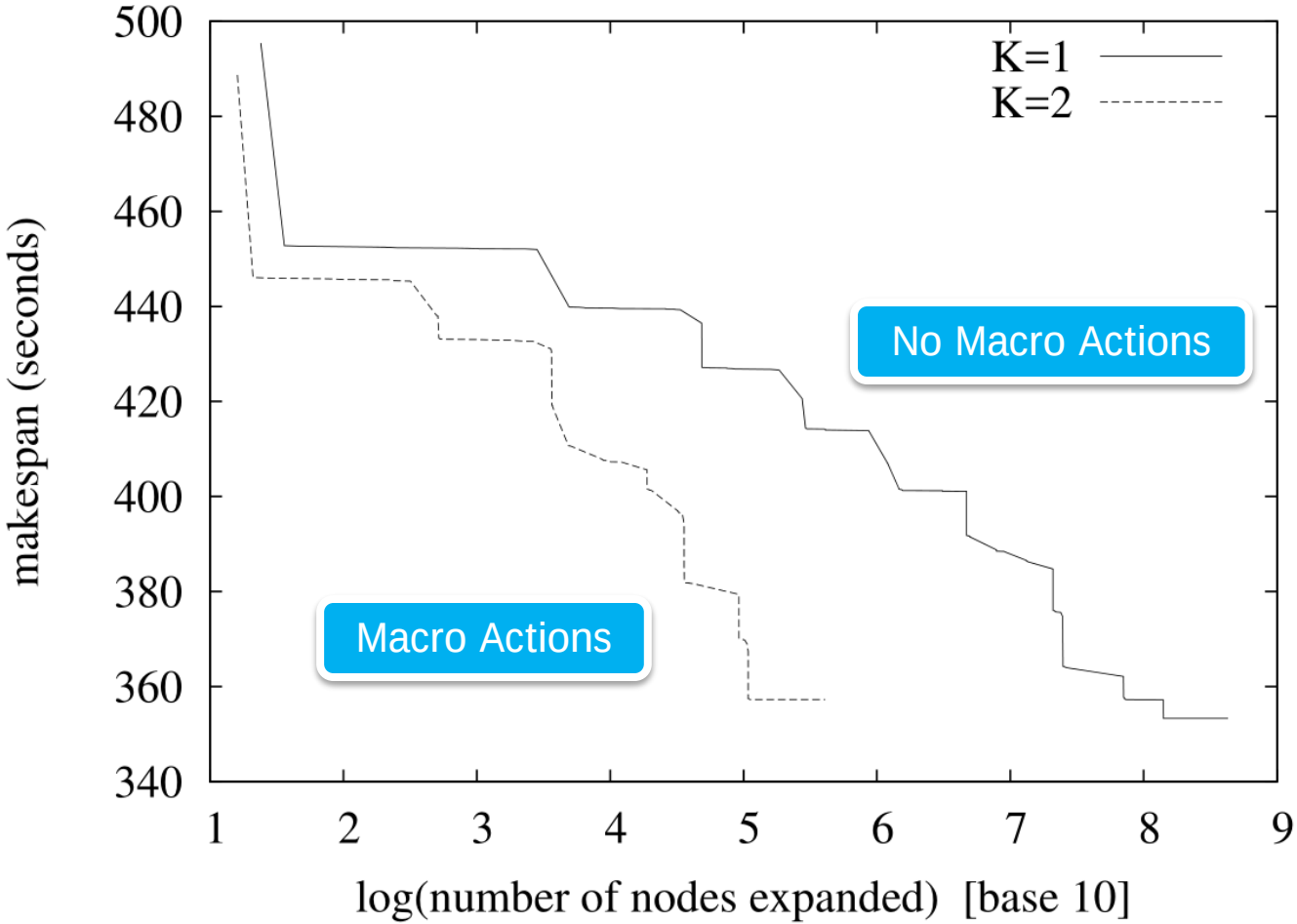


Macro Actions

- Macro actions are built of atomic actions
- They have cost equal to the sum of the actions contained within them
- Example: > > ^
  - Built From: Right Right Up
  - Cost: 100 + 100 + 141 = 341

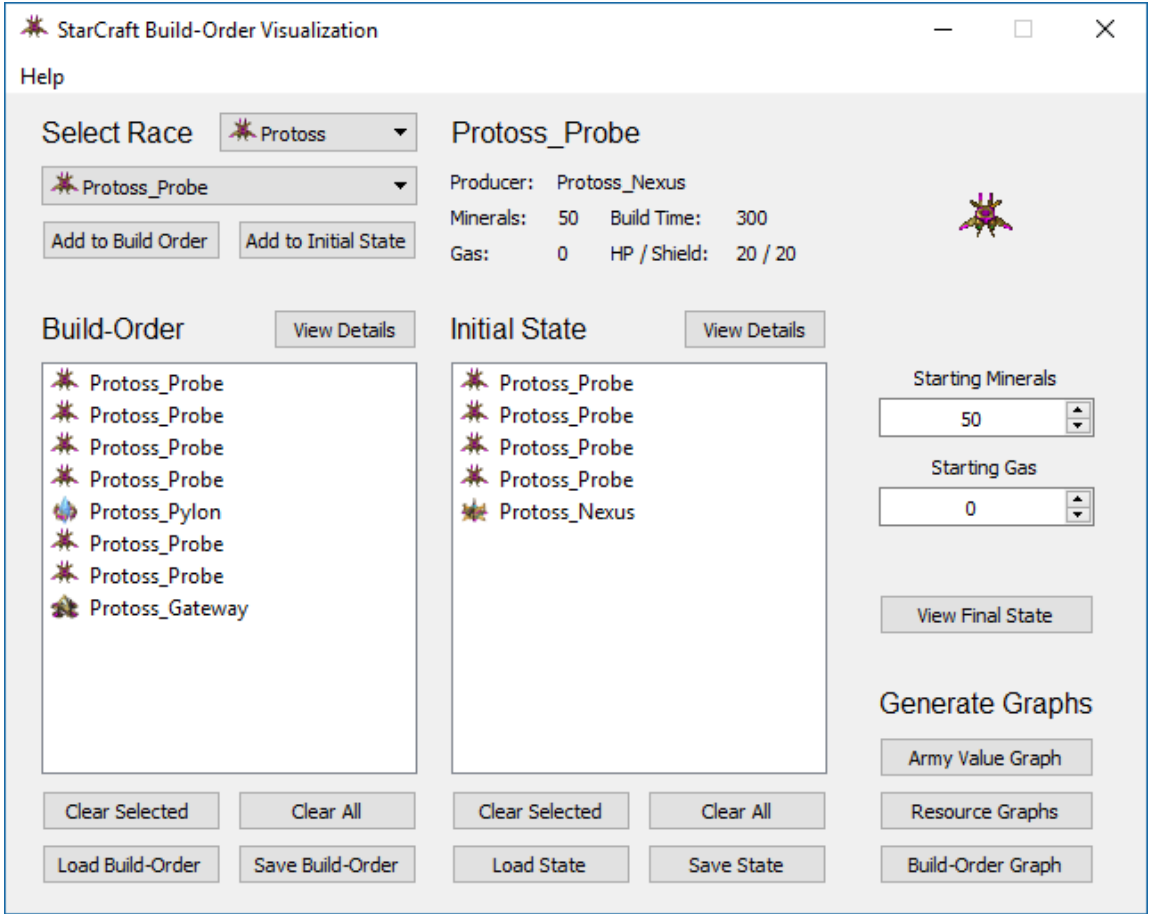
Macro Actions

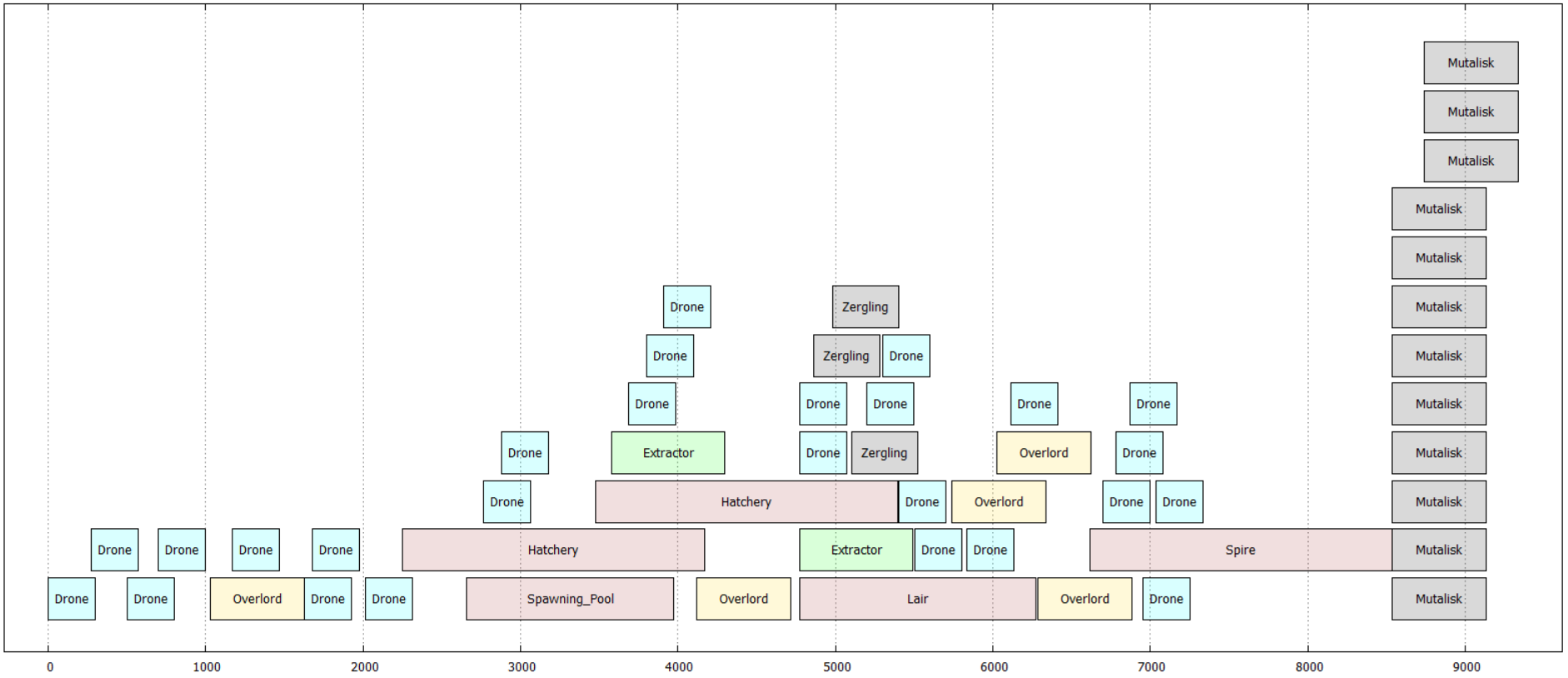
- Trade-off: branching factor vs. depth
- We must keep all atomic actions to ensure that the optimal solution is still found
- Macro actions search deeper into the tree with a single transition, but increase the branching factor of a given node
- They may speed up the search, but a poor choice of macro action can also slow down the search


500
480
460
440
420
400
380
360
340
makespan (seconds)
1
2
3
4
5
6
7
8
9
log(number of nodes expanded) [base 10]
K=1
K=2
No Macro Actions
Macro Actions

Build-Order Planning Impact

- Implemented into UAlbertaBot
  - Used in other bots in past few years
- All build-orders planned in real-time
  - First real-time planning solution
- Ran in real time with no time-outs
- UAlbertaBot placed 2nd when implemented


StarCraft Build-Order Visualization
Help
Select Race
Protoss
Protoss_Probe
Add to Build Order
Add to Initial State
Protoss_Probe
Producer: Protoss_Nexus
Minerals: 50
Build Time: 300
Gas: 0
HP / Shield: 20 / 20
Build-Order
View Details
Protoss_Probe
Protoss_Probe
Protoss_Probe
Protoss_Probe
Protoss_Pylon
Protoss_Probe
Protoss_Probe
Protoss_Gateway
Initial State
View Details
Protoss_Probe
Protoss_Probe
Protoss_Probe
Protoss_Probe
Protoss_Nexus
Starting Minerals
50
Starting Gas
0
View Final State
Generate Graphs
Army Value Graph
Resource Graphs
Build-Order Graph
Clear Selected
Clear All
Load Build-Order
Save Build-Order
Clear Selected
Clear All
Load State
Save State


Mutalisk
Mutalisk
Mutalisk
Mutalisk
Mutalisk
Mutalisk
Mutalisk
Mutalisk
Mutalisk
Drone
Zergling
Drone
Zergling
Drone
Drone
Drone
Drone
Drone
Drone
Drone
Extractor
Drone
Zergling
Overlord
Drone
Drone
Drone
Hatchery
Drone
Overlord
Drone
Drone
Drone
Drone
Drone
Drone
Drone
Hatchery
Extractor
Drone
Drone
Spire
Drone
Drone
Overlord
Drone
Drone
Spawning_Pool
Overlord
Lair
Overlord
Drone
Mutalisk
0
1000
2000
3000
4000
5000
6000
7000
8000
9000