UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3547 Intelligent Agents & Reinforcement Learning

Module 2: Search

Course Plan

Module Titles
Module 1 – Introduction to Intelligent Agents
Module 2 – **Current Focus: Search**
Module 3 – Logical Inference
Module 4 – Planning and Knowledge Representation
Module 5 – Probabilistic Reasoning
Module 6 – Intro to Reinforcement Learning and Finite Markov Decision Processes
Module 7 – Dynamic Programming and Monte Carlo Methods
Module 8 – Temporal Difference Learning
Module 9 – Function Approximation for RL
Module 10 – Deep Reinforcement Learning and Policy Gradient Methods
Module 11 – Introduction to Advanced DRL
Module 12 – Presentations (no content)

Learning Outcomes for this Module

- Enable you to select the appropriate search technique for a problem from a wide variety of methods
- Enable you to compare the feasibility of using alternative search algorithms
- Prepare you to use constraint solvers to tackle more difficult problems
- Enable you to recognize business opportunities to use these tools

Topics for this Module

- **2.1** Problem Solving as State-Space Searching
- **2.2** Uninformed Search
- **2.3** Heuristic Search
- **2.4** Adversarial Search
- **2.5** Genetic Algorithms
- **2.6** Constraint Satisfaction Problems
- **2.7** Go Code
- **2.8** Resources and Wrap-up

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 1

Problem Solving as State-Space Searching

Problem Solving as State-Space Searching

- State-space problems:
  - A set of states
  - A start state (usually the environment's current state)
  - A set of possible actions for each state
  - An action function that maps a state and action to a new state
  - A goal function that can determine if a state is a goal state
  - Optionally a solution quality criterion
- State-space searching assumes:
  - The agent has perfect knowledge of the state space and can observe what state it is currently in
  - The agent has a set of actions that have known deterministic effects
  - The agent can determine whether a state satisfies a goal

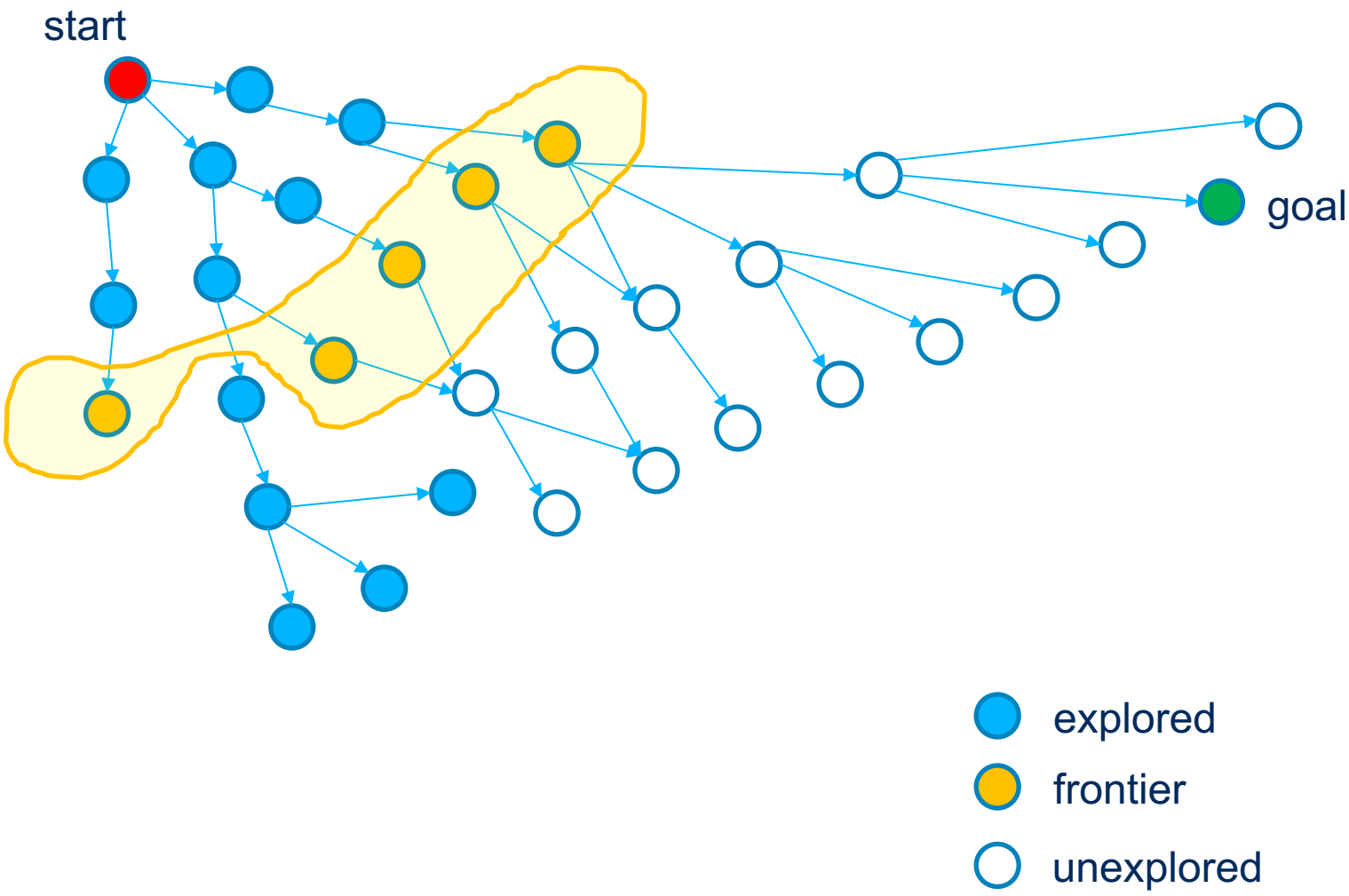
Graphs

- Graphs are a natural structure for many kinds of search problems
- They consist of a set of nodes and arcs between nodes
- Typically in state-space problems states are represented as nodes and actions as arcs
- Graphs can be
  - Directed or undirected
  - Cyclic or acyclic
  - Trees
  - With or without arc costs
- Key metrics
  - Forward branching factor
  - Backward branching factor

Module 2 – Section 2

Uninformed Search Strategies

The Search Frontier



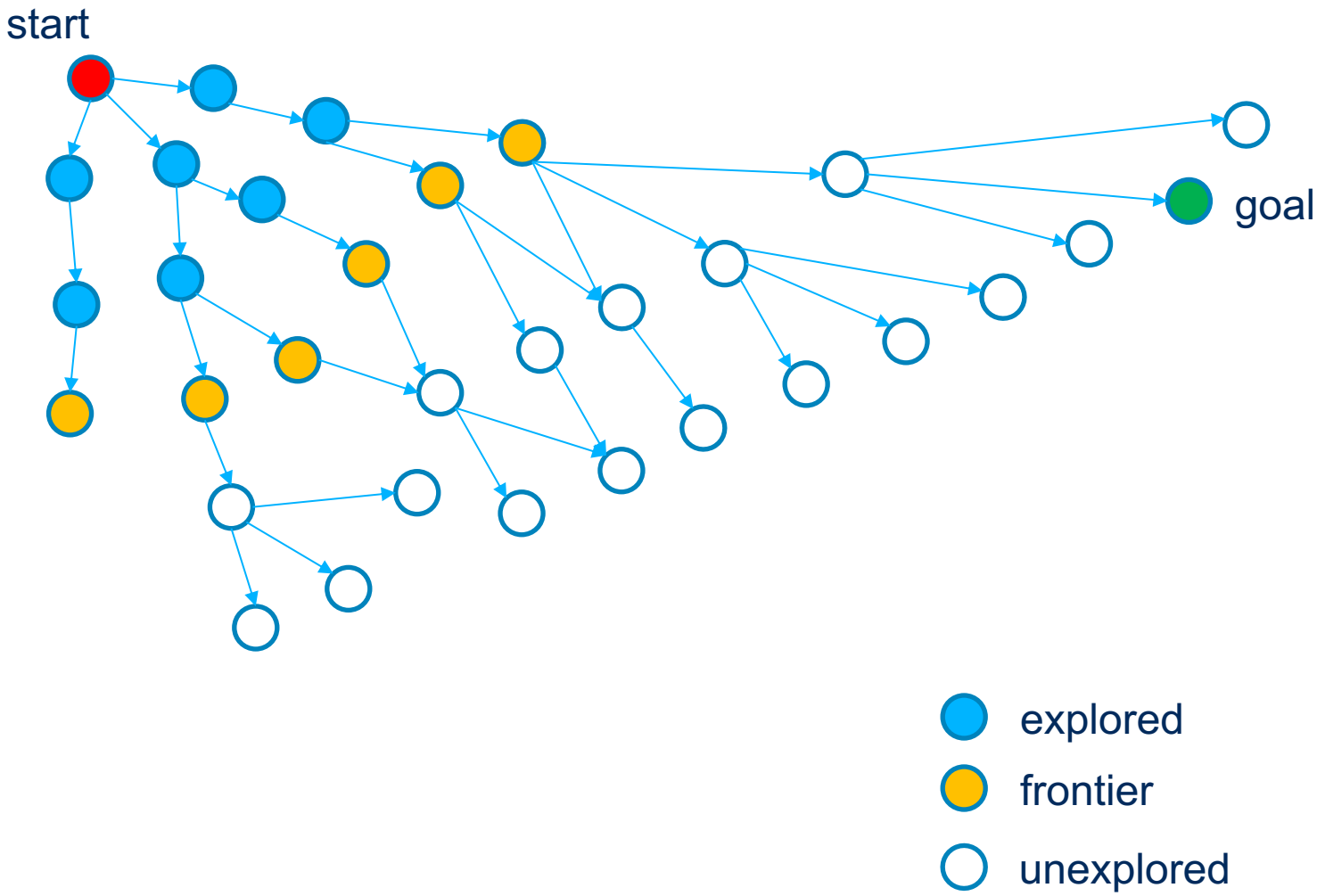
Generic Search Approach

```
while the frontier is not empty {
        choose a path from the frontier and remove it
        if the node at the end of the path is a goal node {
                return the path as a solution
        } else {
                extend the path to include its neighbours
        }
}
return NO SOLUTION
```

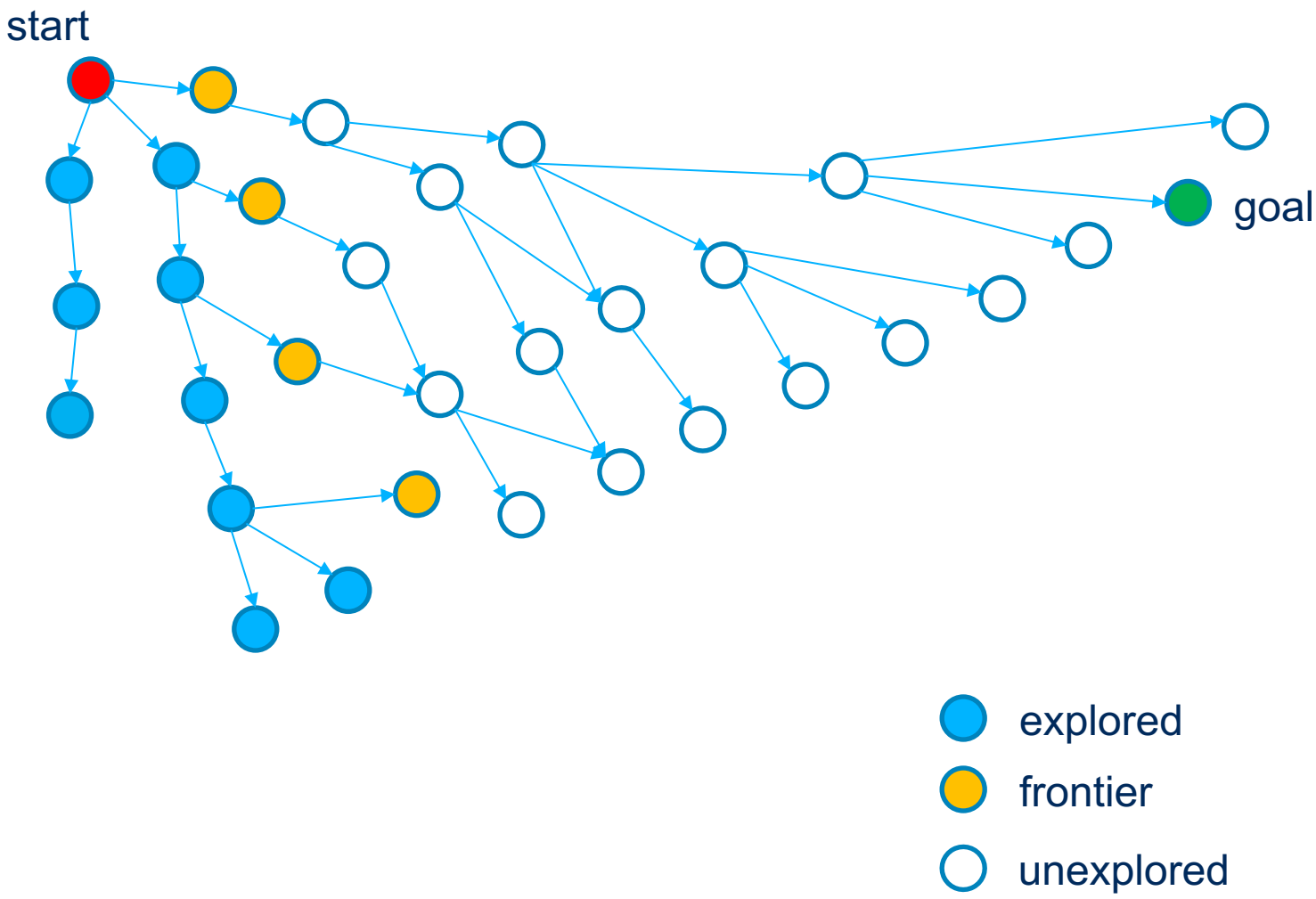
Uninformed Search Strategies

- **Uninformed search**: Techniques to use when we don't have any information in advance about where the solution might be until we find it
- Most popular techniques
  - Breadth-first
  - Depth-first
  - Iterative deepening
  - Lowest-cost-first

Breadth-First Search

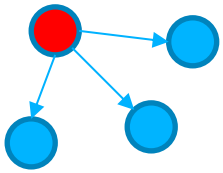


Depth-First Search

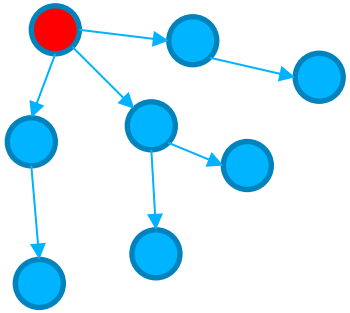


Iterative Deepening

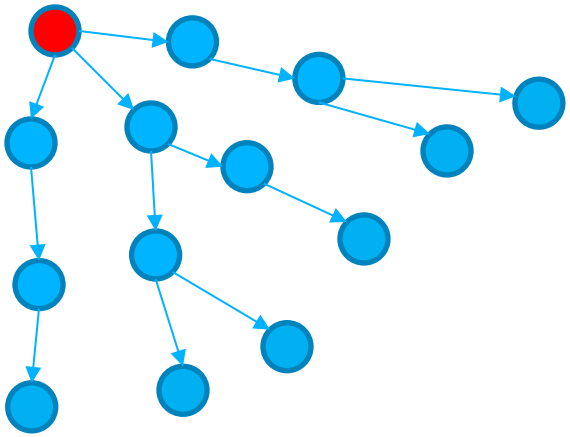
start



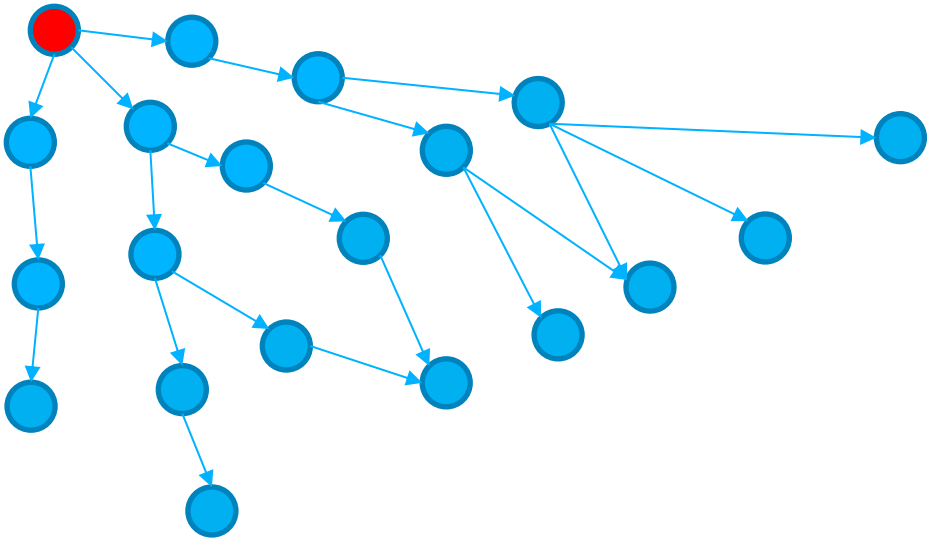
1



2



3



4

Lowest-Cost-First

- Consider a problem where the arcs have attached costs and we want to find the lowest-cost path
- Similar to breadth-first but choose the lowest-cost path on the frontier to expand next
- None of the previous search techniques would find it first
- A lowest-cost-first search would
- Guaranteed to find lowest-cost path if the costs are all greater than some positive constant and the branching factor is finite
- Examples: shortest route between two cities, minimum cost manufacturing process

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 3

Heuristic Search

Heuristics

- **Heuristics** generally:
  - ”Rules of thumb”
  - Practical procedure that may not be optimal
- Heuristics for our purposes:
  - **Heuristic Function**: A function (often denoted h(n) that maps a node n to a non-negative real number that estimates the cost of the least-cost path from *n* to a goal node
  - **Admissible Heuristic**: a heuristic function that never overestimates the actual cost e.g. Euclidean distance is an admissible heuristic for the road travel distance between locations
- Heuristics are useful because:
  - They may guide and accelerate the search process
  - An admissible heuristic will allow us to determine, once we’ve found a path to the goal, whether that path is optimal and we can stop

A*

- A* is optimally efficient if we have an *admissible heuristic*
- Estimates the total cost path: $f(p) = cost(p) + h(p)$ where $cost(p)$ is the cost from the start node to a node on the frontier and $h(p)$ is the estimated cost from the frontier node to the goal node
- To implement A*, always select the frontier node with smallest $f(p)$ for expansion
- If $h(p)$ is an admissible heuristic, the A* algorithm is guaranteed to find the optimal solution (assuming the branching factor is finite and all costs are greater than some number > 0)

Strategies for Improving on A* (if possible)

- Iterative Deepening (IDA*)
- Use graphs rather than trees
- Employ smart selection of the order of expansion of branches
- Remove symmetries
- Use solution (pattern) databases
- Take advantage of problem-specific features

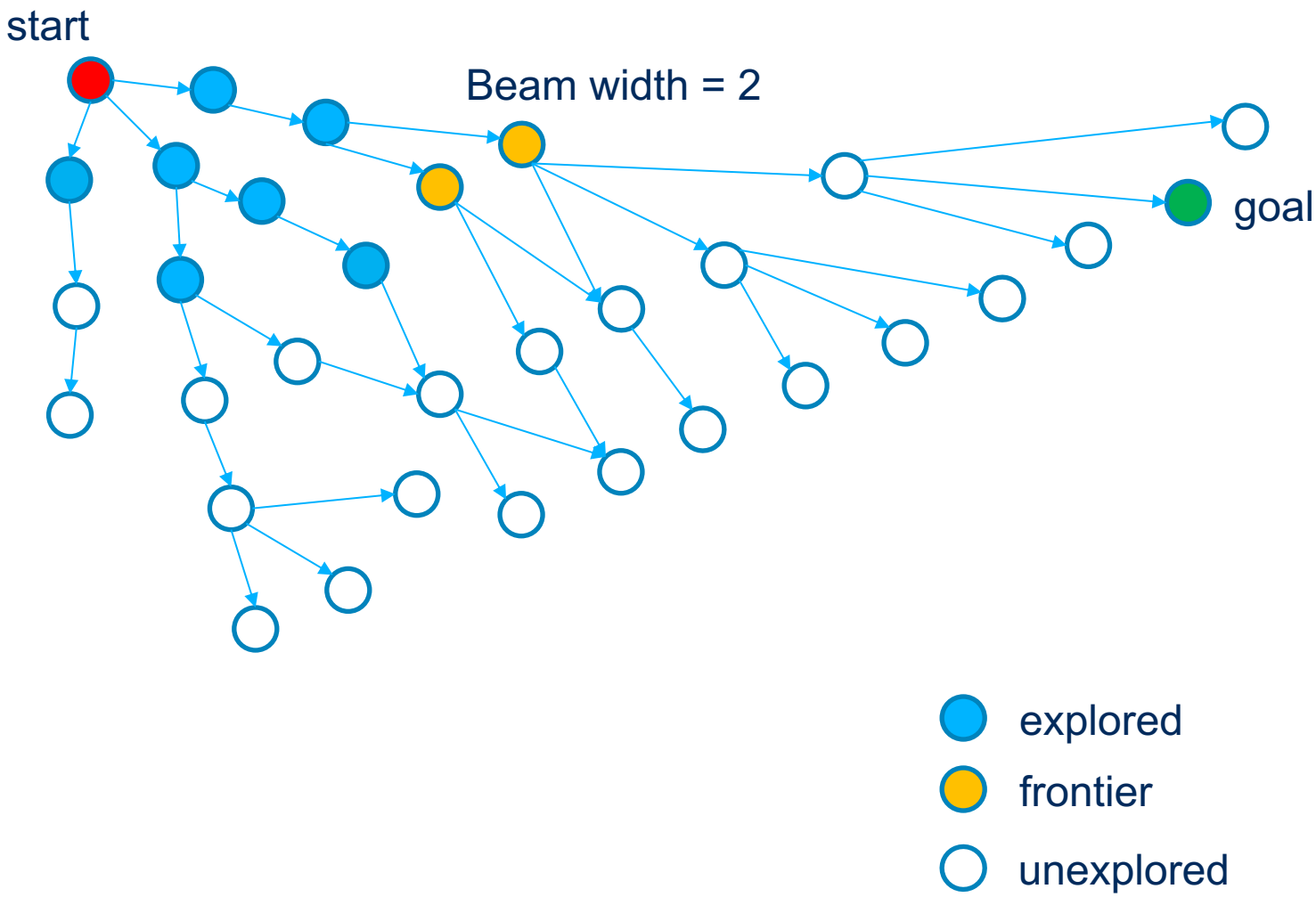
Simulated Annealing

- Useful when an approximate global optimum is required in a fixed amount of time
- Idea based on the metallurgical concept of annealing
- The concept is to randomly explore the solution space using large jumps at first and slowly reduce the size of the jumps and focus on promising areas of the space
- An adaptation of Monte Carlo methods (the Metropolis algorithm)

Monte Carlo Tree Search

- Expand the search tree using random sampling
- Useful for game-playing (e.g. Go)
- The strategy is to expand the tree to the end of the game to see who wins, then adjust the probabilities of selection of nodes so that the more promising nodes are selected more often
- Repeat many times

Beam Search



Pruning

- Graph cycle pruning: Remove cycles
- Graph multi-path pruning: Remove duplicate paths

Search Direction

- Forward
- Backward
- Meet-in-the-Middle (Bidirectional)
- Island-Driven Search: Find the nearest “island” and take a known path to other islands, then branch from there e.g. routing from neighbourhood to avenue to highway then reverse at the other end

Dynamic Programming

- Useful when:
  - The goal nodes are explicit
  - A lowest-cost path is needed
  - The graph is fairly small and finite
  - The goal doesn't change very often
  - The solution can be reused many times
- Can be used to construct a heuristic function that can be used for A* by solving a simplified problem then using a **pattern database** to solve the bigger problem
- Much more about Dynamic Programming later in the course

Pattern Databases

- Useful for single-agent search
- The concept is to create a database of all possible feasible subgoals
- The search can then use this information to make better choices of next steps based on its progress toward these subgoals
- The subgoals are represented as patterns that can be pattern-matched
- Can encode symmetries

Neural Search

- Ideally we'd like an estimate of the quality of a node, even if we can't find a path to the goal
- We'd like a heuristic **evaluation function** that tells us whether a path is leading us toward a better or worse state (relative to achieving our goal)
- There are many ways of constructing such a function
- For example, for chess:
  - Counting pieces won/lost weighted by their value
  - Assigning values to configurations of the pieces on the board e.g. pawn structure, mobility in the centre, how exposed the king is, etc.
- We know neural nets make terrific trainable function approximators so can use them to recognize and evaluate patterns (on the board, in percepts, in data, etc.)

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 4

Adversarial Search

Minimax

- Most games that AI theorists have focused on are:
  - Deterministic
  - Turn-taking
  - Two-player
  - Zero-sum
  - Perfect information
- The Minimax algorithm is a good choice for these kinds of situations
- The player about to move (you, “Max”) wants to maximize their advantage
- The other player (“Min”) will respond in a way that maximizes theirs

Minimax (Cont'd)

- The objective is to choose moves that maximize Max's advantage and minimize Min's opportunity to cause damage
- Max has to assume Min will play optimally, which means leaving aside moves that have big gains if it gives Min even a single opportunity for a response that would be more valuable than Max's
- Ideally we can rollout to the end of the game and precisely evaluate each move but that isn't possible for other than simple games such as tic-tac-toe

Alpha-Beta Pruning

- Consider a node in the minimax tree you can move to
- If you already have a better choice you've found higher in the tree you can prune the subtree you're considering
- Maintain two values, α and β
- α is the minimum score the maximizing player is assured of
- β is the maximum score the minimizing player is assured of
- Whenever $\beta \leq \alpha$ don't consider descendants any further

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 5

Genetic Algorithms

Genetic Algorithms

- Based on the idea of biomimicry of sexual reproduction
- We create a set of candidate solutions to an optimization problem (usually random to begin)
- These solutions are often expressed as algorithms in a simplified programming language
- Algorithms compete with each other to allow their fitness to be measured
- A number of the top-scoring solutions is taken and used to attempt to construct even better solutions by combining their strengths

Biological Equivalents

- Crossover: Split pairs of solutions at some point in their program and swap the pieces
- Mutation: Make a small number of random changes
- Genotype-phenotype mapping: If the solution is encoded in some way (e.g. as a bit string that needs to be converted to a Python program), perform the conversion
- Fitness measurement: Evaluate the resulting solutions

- (Repeat until a good solution is found)

Example: Wumpus World Agent Programs

1

1 If *stench* goto 1
2 Shoot arrow
3 Shoot arrow
4 Turn left
5 Grab gold
6 Climb out
7 Turn right
8 Go forward
9 If *glitter* goto 3
etc.

2

1 If *breeze* goto 4
2 Turn left
3 Shoot arrow
4 Climb out
5 Grab gold
6 Turn right
7 Turn right
8 If *bump* goto 78
9 Goto 45
etc.

3

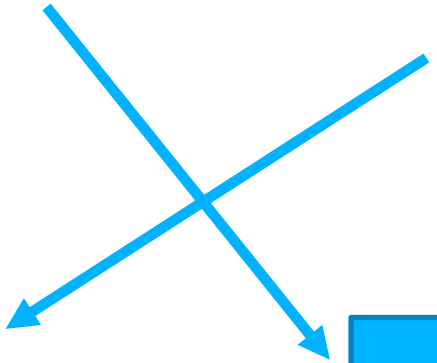
1 If *stench* goto 4
2 Go forward
3 If *bump* goto 22
4 Shoot arrow
5 If *glitter* goto 8
6 Turn right
7 Turn left
8 Grab gold
9 If *breeze* goto 91
etc.

4

…
91 Record *breeze* here
92 Turn left
93 Turn left
94 Go forward
95 If record of *breeze* here goto 93
96 If *stench* goto 1
97 Turn left
98 Grab gold
99 If *breeze* goto 34
100 Goto 42

1 If *stench* goto 4
2 Go forward
3 If *bump* goto 22
4 Shoot arrow
5 If *glitter* goto 8
6 Turn right
7 Grab gold
8 Grab gold
9 If *breeze* goto 34
…

91 Record *breeze* here
92 Turn left
93 Turn left
94 Go forward
95 If record of *breeze* here goto 93
96 If *stench* goto 1
97 Turn left
98 Grab gold
99 If *breeze* goto 34
100 Goto 42



Beginning of 4 followed by end of 3

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 6

Constraint Satisfaction Problems

Constraint Satisfaction Problems

- Some search problems can be solved more efficiently by using a *factored* representation for each state
- We will replace the set of states with:
  - A set of variables
  - A set of domains for each variable
  - A set of constraints on which combinations of values the variables are allowed to take

CSP Applications

- Microsoft Project
- Natural resource extraction and marketing
- Robotics decision-making
- Economic resource allocation
- Process control

Definitions

- A **constraint satisfaction problem** has three components:
  - X is a set of variables $\{X_1, X_2, \ldots, X_n\}$
  - D is a set of domains $\{D_1, D_2, \ldots, D_n\}$, one for each variable
  - C is a set of constraints that specify allowable combinations of values
- Each D_i consists of a set of allowable values $\{v_1, v_2, \ldots, v_n\}$ for variable X_i
- Each C_i consists of a pair $\langle scope, rel \rangle$, where $scope$ is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on

Definitions (Cont'd)

- **Assignment**: An assignment of values to some or all of the variables
- **Partial assignment:** An assignment to only some of the variables
- **Complete assignment/Total Assignment/Possible World**: An assignment to all of the variables
- **Consistent assignment**: An assignment that does not violate any of the constraints
- **CSP Solution/Model**: A consistent, complete assignment

CSP Domains and Constraints

- The domains can be:
  - Finite and discrete: We can enumerate the states
  - Infinite and discrete: Can't enumerate; need a constraint language
  - Continuous: Linear and non-linear programming
- The constraints can be:
  - Unary
  - Binary
  - n-ary
  - Global
- Constraints can also be:
  - Absolute
  - Preference

Some Useful Tasks Given a CSP

- Determine whether or not there is a solution
- Find a solution
- Count the number of solutions
- Identify all the solutions
- Find the best solution subject to some quality measure
- Determine whether some statement holds true across all solutions

Strategies for Solving a CSP

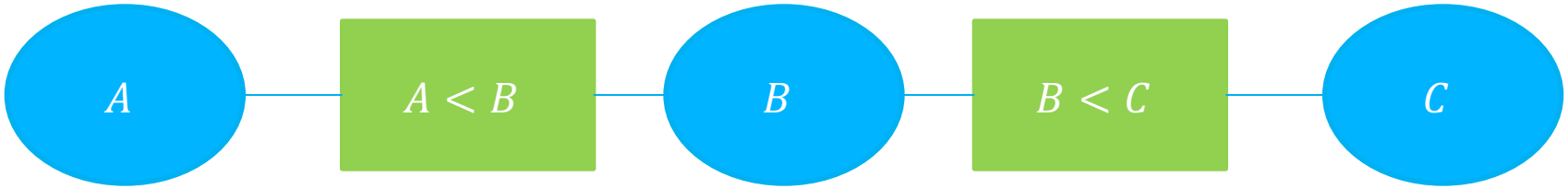
- Generate-and-Test: Exhaustively try every possible combination of variables
- Graph Search: Build a graph and apply a simplification algorithm followed by search
  - Consistency Algorithm(s)
  - Domain Splitting (Case Analysis)
  - Variable Elimination

Graph Search

- Nodes are assignments of values to some subset of the variables
- Neighbour nodes are created by selecting a variable that is not assigned in this node and creating a new node for each value the variable can take that does not violate any constraint $\cup$ the values in this node
- The start node is an empty node (no assignments)
- A goal node is one that assigns a value to every variable
- We can now do a depth-first search for a solution

Constraint Networks

- We can be more efficient yet by creating a **constraint network**:
  - Create a node for each variable
  - Create a node for each constraint
  - Associate with each variable a set of possible values, initially the members of the domain of the variable
  - For each constraint c and each variable X in the scope of the constraint add an arc $\langle X, c \rangle$



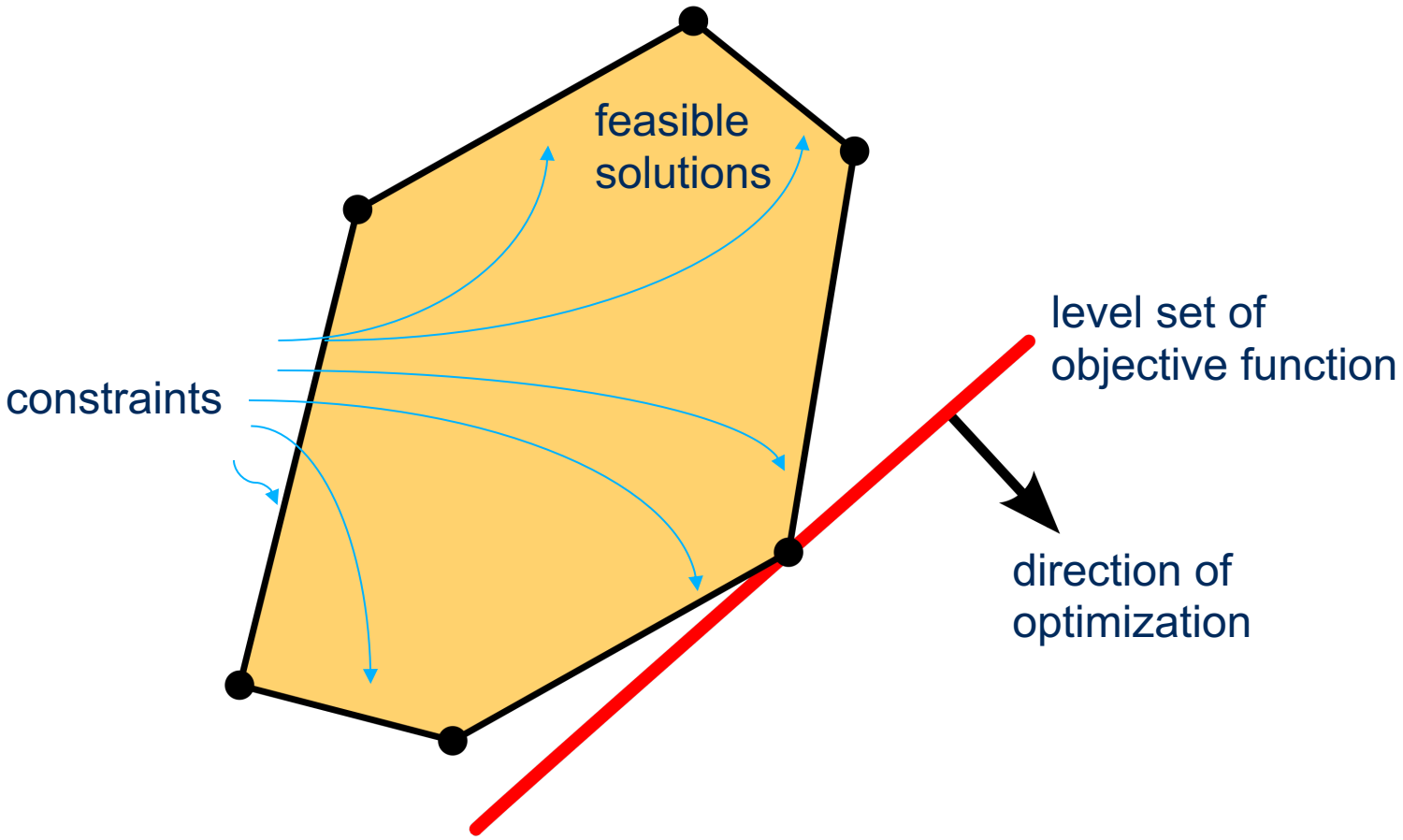
Consistency Algorithms

- Node consistency
- Arc consistency
- Path consistency
- Global constraints
- Local search

Domain Splitting (Case Analysis)

- Split the problem into a number of disjoint cases and solve each separately

Linear Programming



By Ylloh - Own work, CC0,
https://commons.wikimedia.org/w/index.php?curid=19413134

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 7

Go Code

Download & Play

- https://github.com/maxpumperla/deep_learning_and_the_game_of_go
- Change directory to the folder named code
- `python human_v_bot.py`
- Let's play! (Careful, the UI is unforgiving)
- https://github.com/maxpumperla/ScalphaGoZero

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 8

Resources and Wrap-up

Resources

- Russell & Norvig. Artificial Intelligence: A Modern Approach, 3rd Ed. Pearson. 2010. Chaps. 3-6.
- Poole & Mackworth. Artificial Intelligence, 2nd Ed. Cambridge University Press. 2017. Chaps. 3-4.
- Pumperla & Ferguson. Deep Learning and the Game of Go. Manning. 2019. Chap. 4.
- Teofili, Tommaso. Deep Learning for Search. Manning. 2019.
- Kramer, Oliver. Genetic Algorithm Essentials. Springer. 2017.
- Levy, Steven. Artificial Life: A Report from the Frontier Where Computers Meet Biology. Vintage. 1993.

Resources (Cont'd)

- https://en.wikipedia.org/wiki/Linear_programming
- Pattern Databases: https://webdocs.cs.ualberta.ca/~jonathan/publications/ai_publications/compi.pdf
- U. of Alberta Chinook checkers player (uses a pattern database): https://webdocs.cs.ualberta.ca/~chinook/play/
- Neo4j Graph Database (site often has free books about graph databases and/or algorithms): https://neo4j.com/

Summary

- Many problems can be described as searches on a tree or graph
- These searches are usually computationally exponential in the depth of the search so heuristics are essential for other than toy problems
- We can often reduce the size of the search space by factoring the states

Next Week

- We will factor the states further into logical variables representing real-world concepts
- We'll use first order logic to allow our AI to infer new knowledge from what it already knows
- We'll learn a little Prolog

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies