

Advanced Artificial Intelligence

“Minimax”

- You want to win
 - maximizing the outcome for you
- Your opponent wants you to loose
 - Minimizing the outcome for you
- You think one move at a time
 - Your move (maximize), opponents move (minimize), your move (maximize), opponents move (minimize), your move (maximize), ...
- What you do, needs to depend on what you believe will be your opponents reaction

IBM's Deep Blue Used Minimax



Deep Blue, at the Computer History Museum

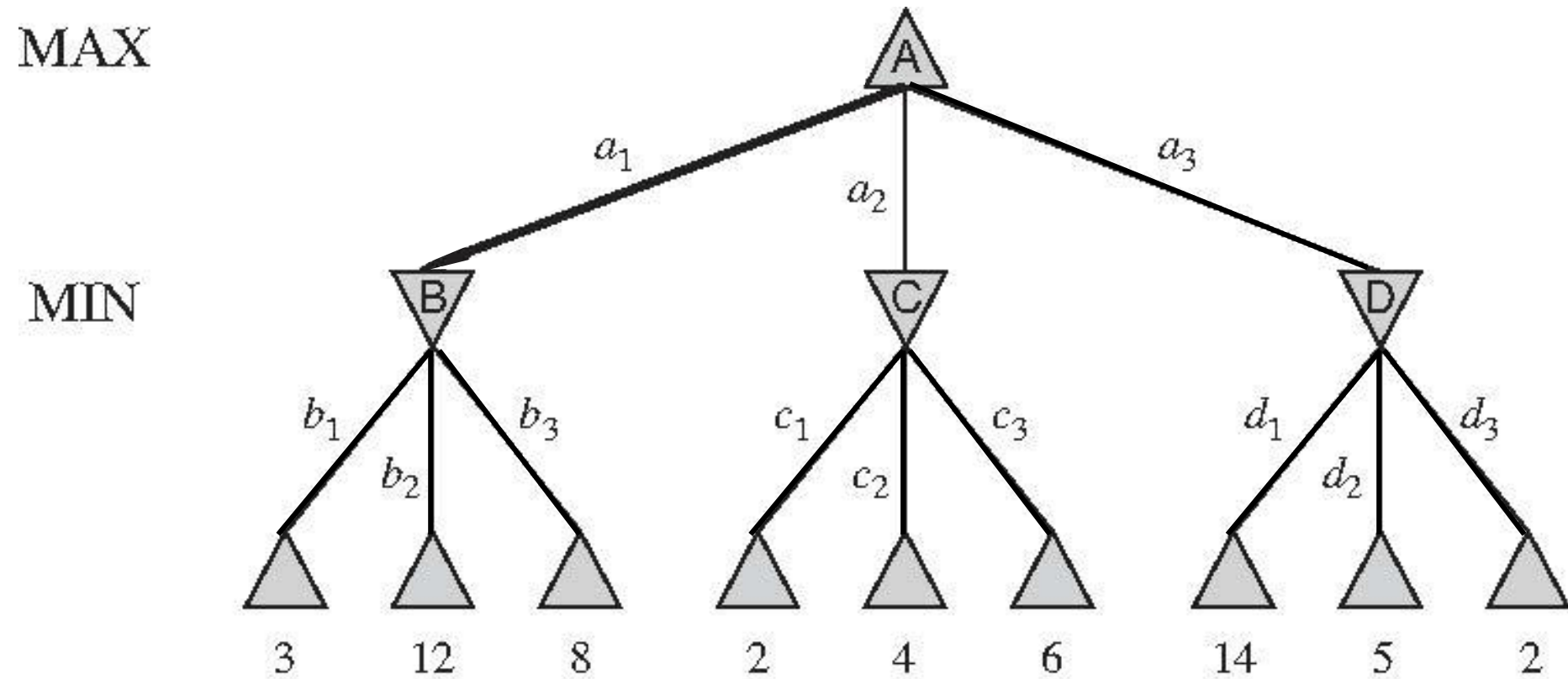


Deep Blue beats Garry Kasparow 1997

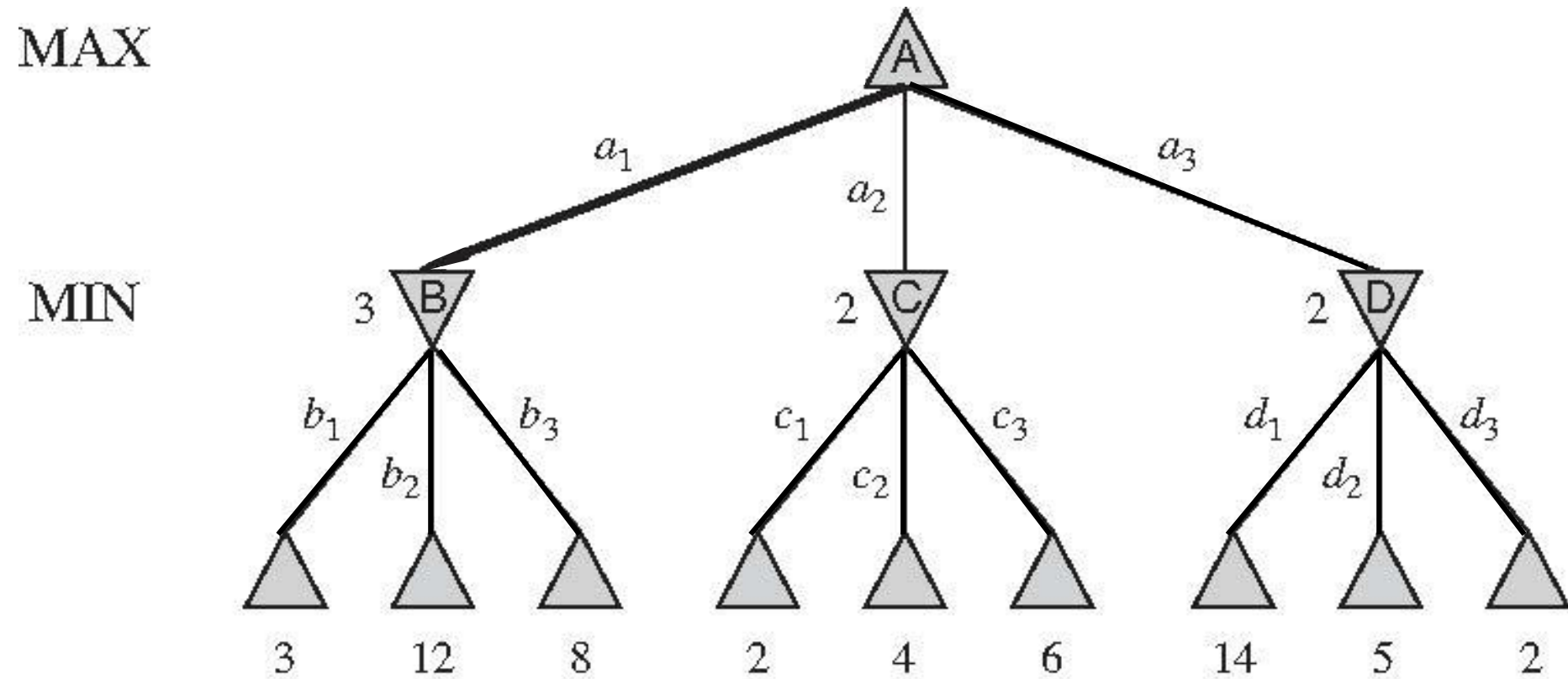
From the perspective of the game/algorithm

- The first player is always Max
 - Max wants the result to be +1
- The second player is always Min
 - Min wants the result to be -1
(which means -1 for Max,
which automatically is +1 for Min)

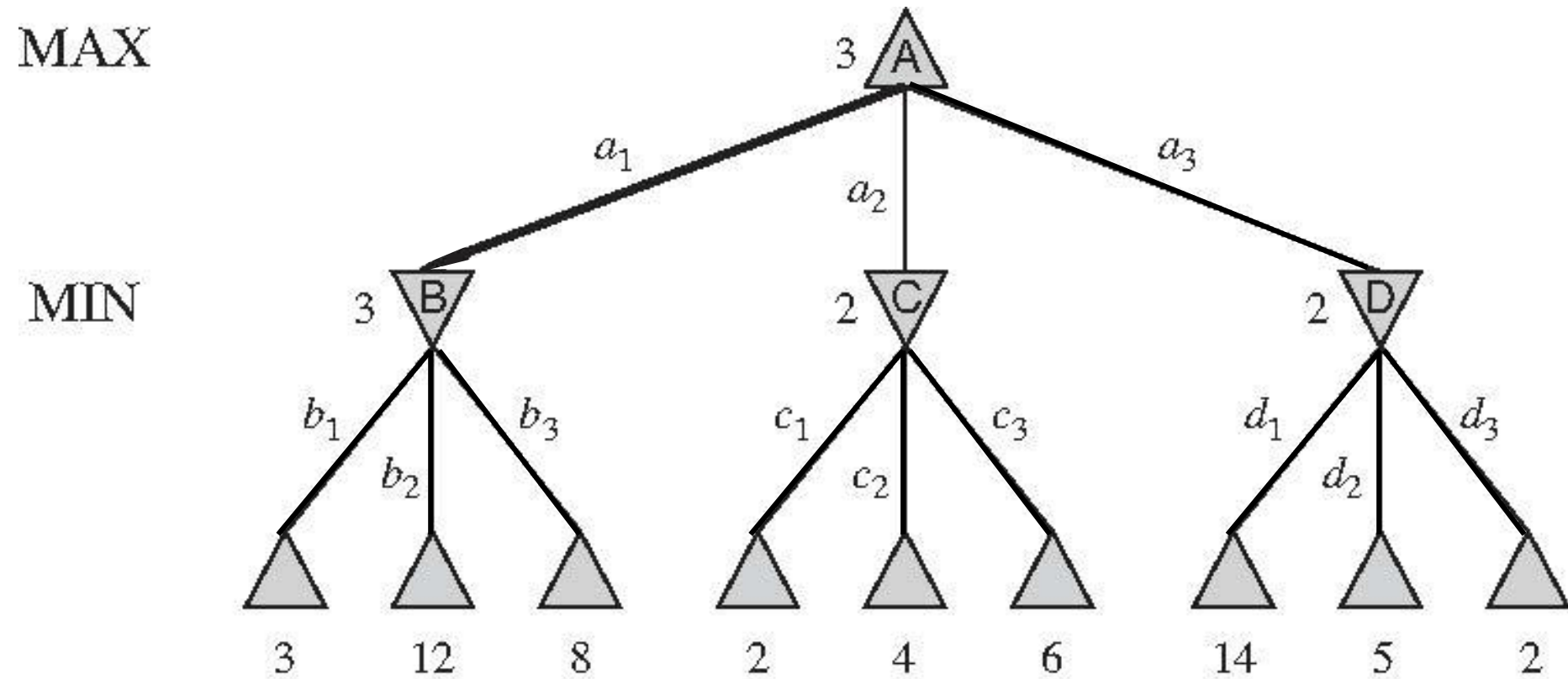
Minimax algorithm



Minimax algorithm



Minimax algorithm



Pruning

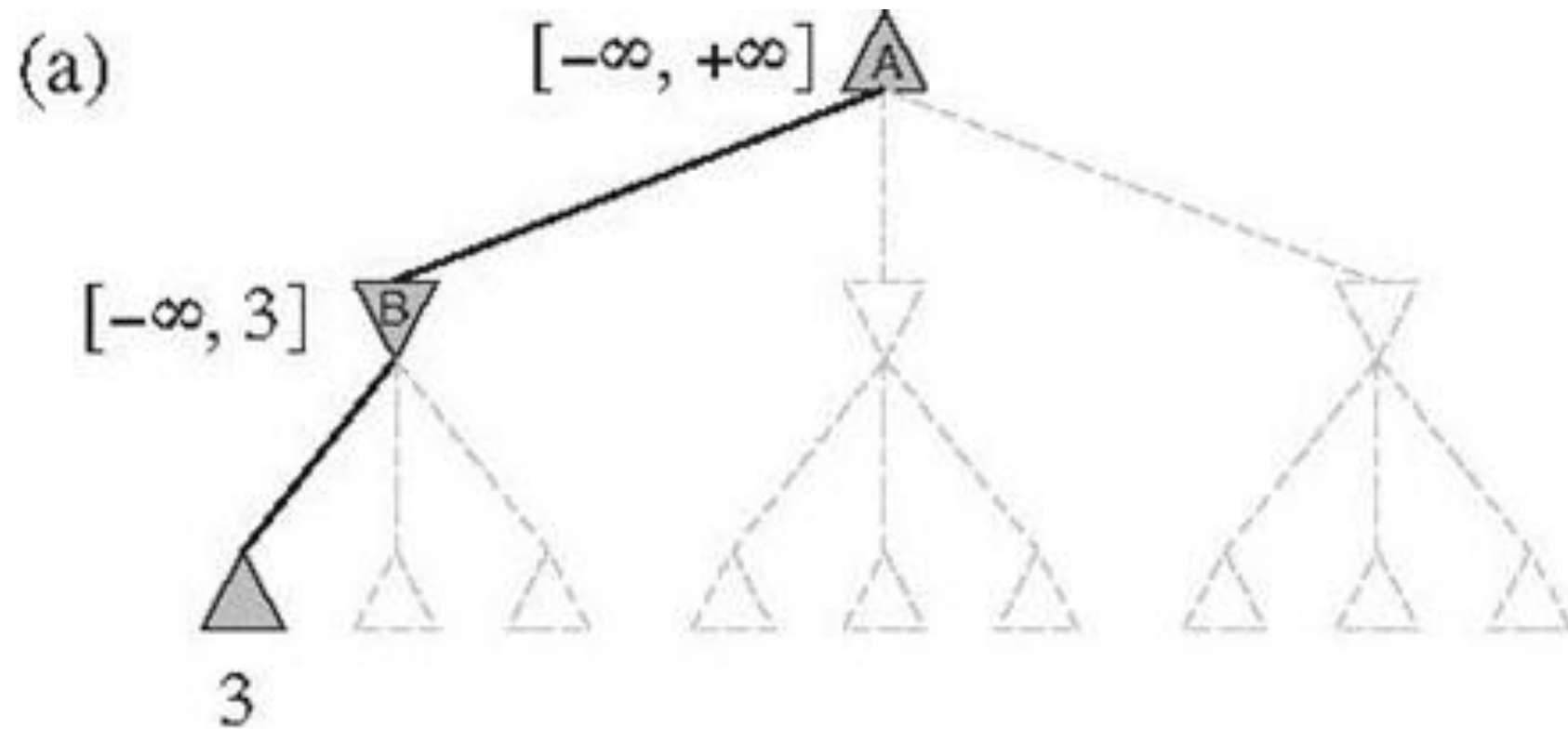
- Trying to minimize the search space
 - Find sections of the search space that can be ignored



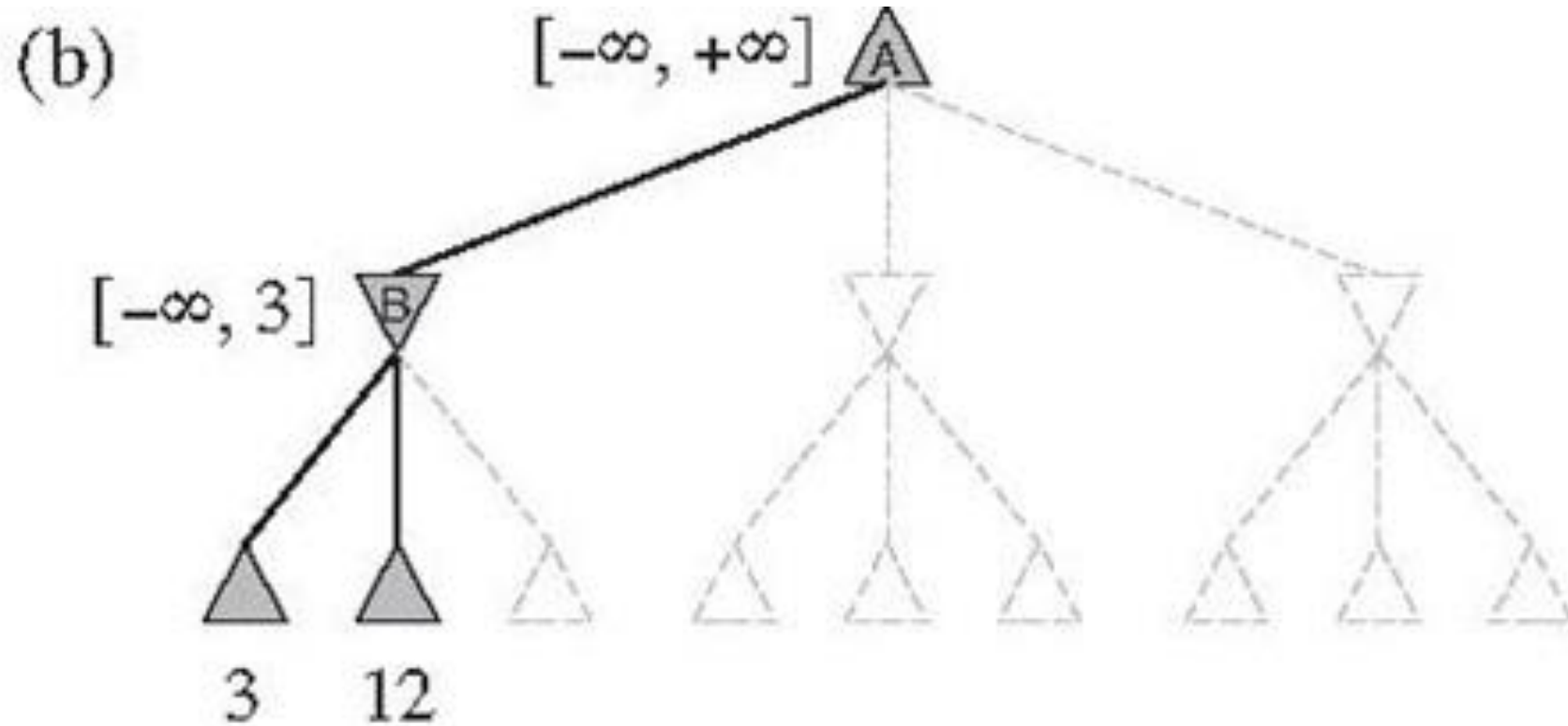
Alpha-Beta pruning

- Since we have a min-player and a max-player we have two values
- alpha: the best value for the max-player
 - For previous game: the highest value
- beta: the best value for the min-player
 - For previous game: the lowest value

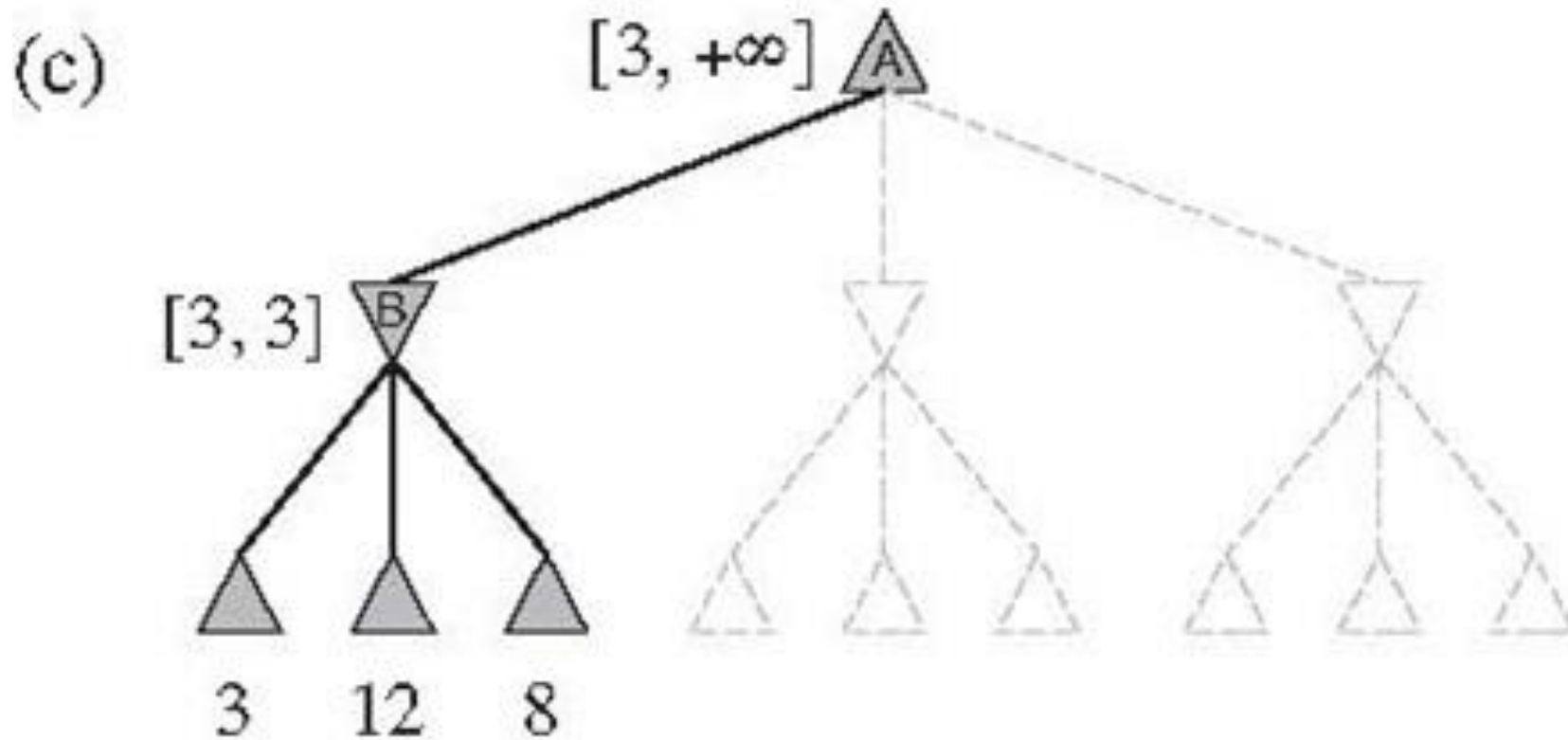
Alpha-Beta pruning



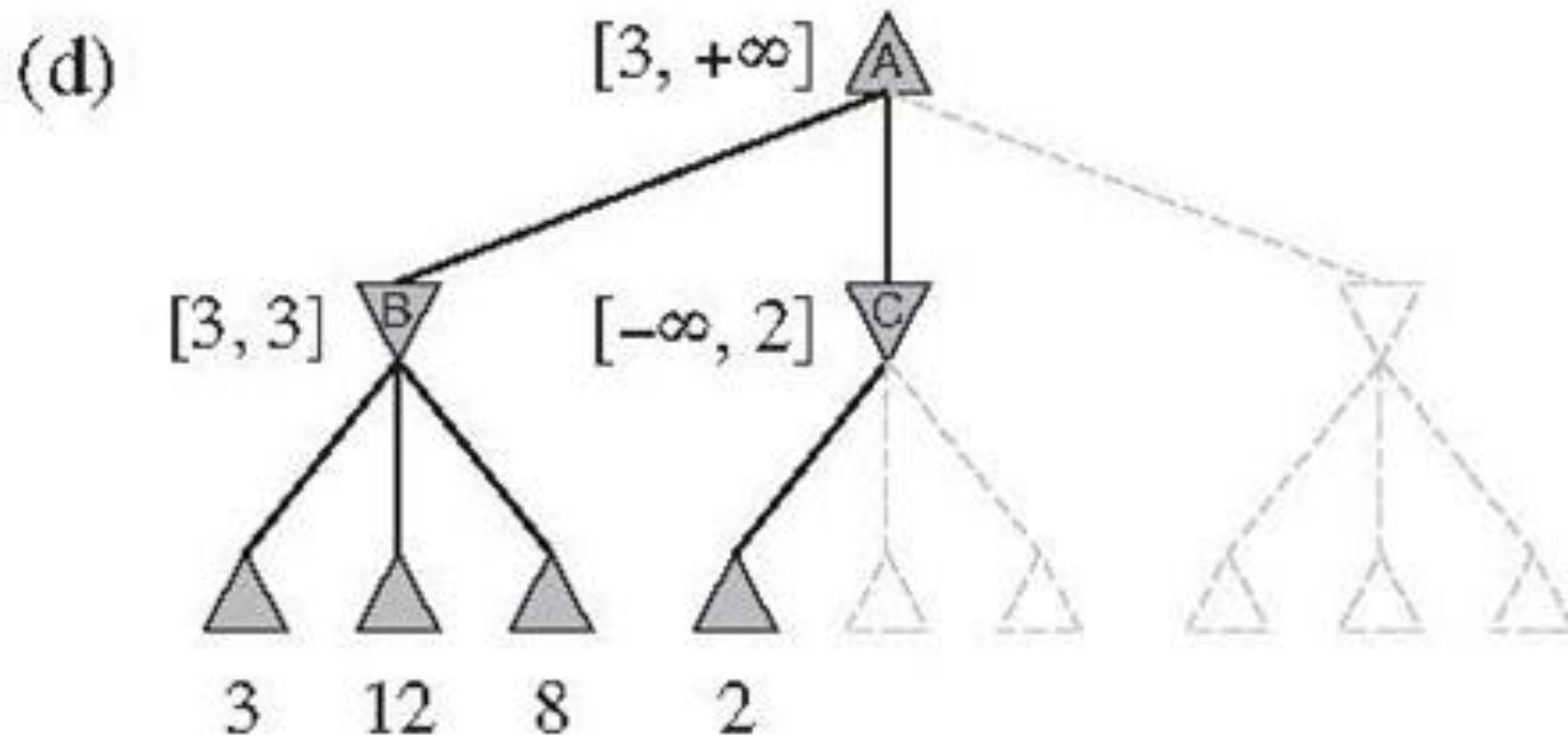
Alpha-Beta pruning



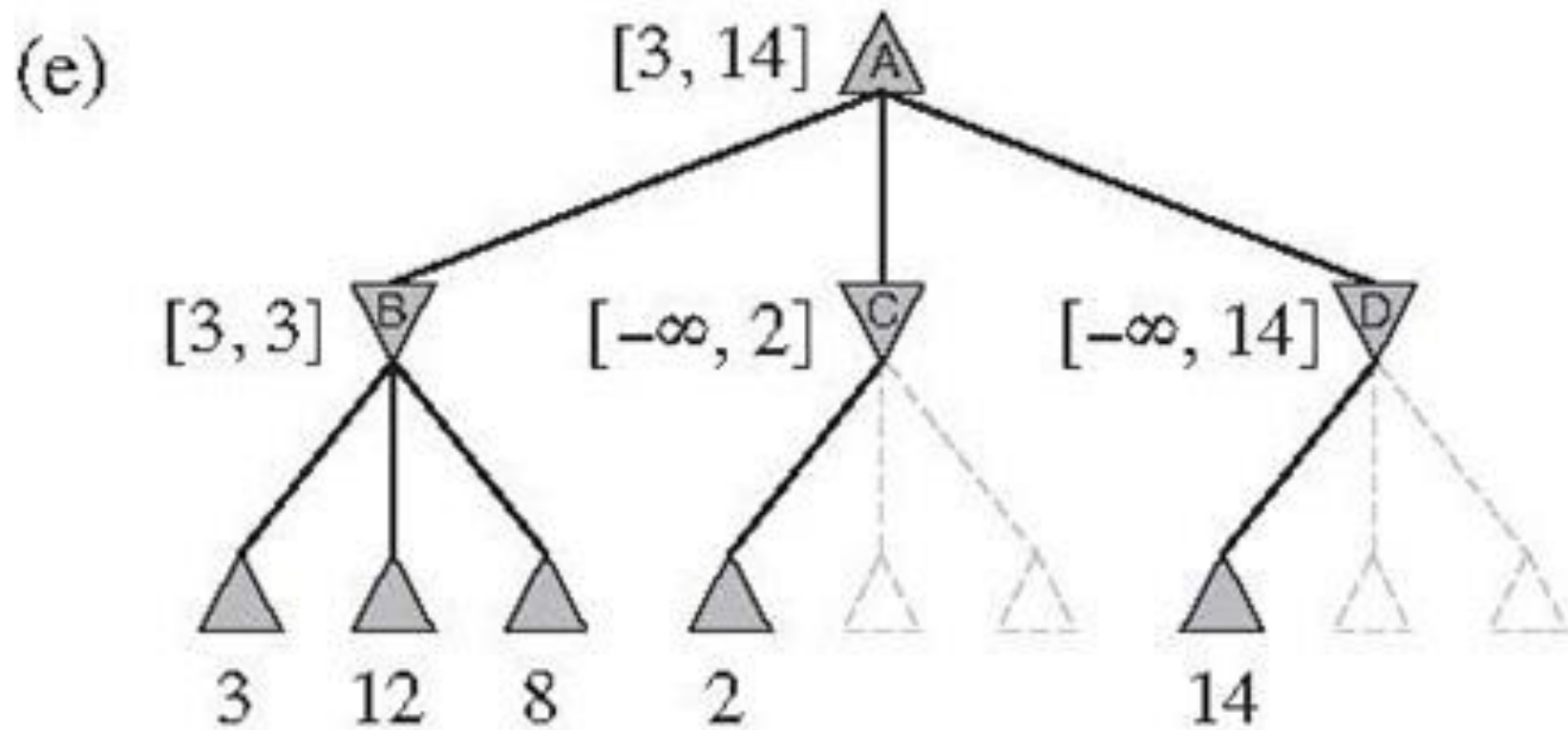
Alpha-Beta pruning



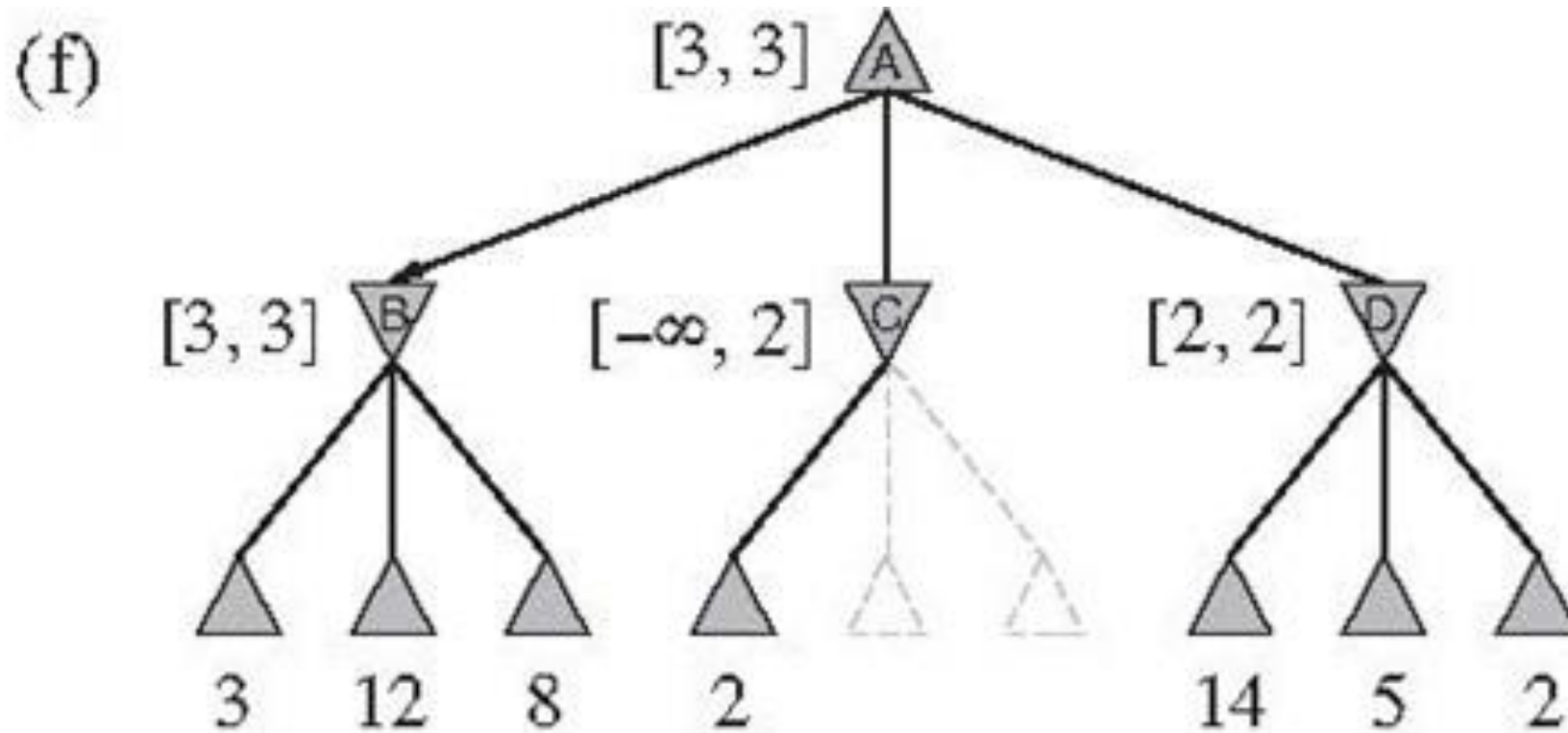
Alpha-Beta pruning



Alpha-Beta pruning



Alpha-Beta pruning



Monte Carlo Tree Search

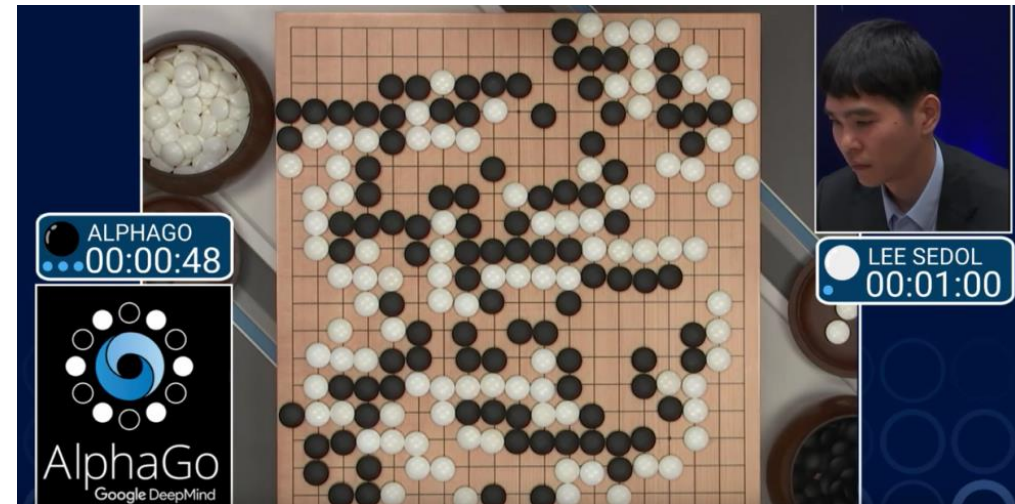
Heuristic Search Algorithm including simulation often used in games

Monte Carlo Tree Search (MCTS)

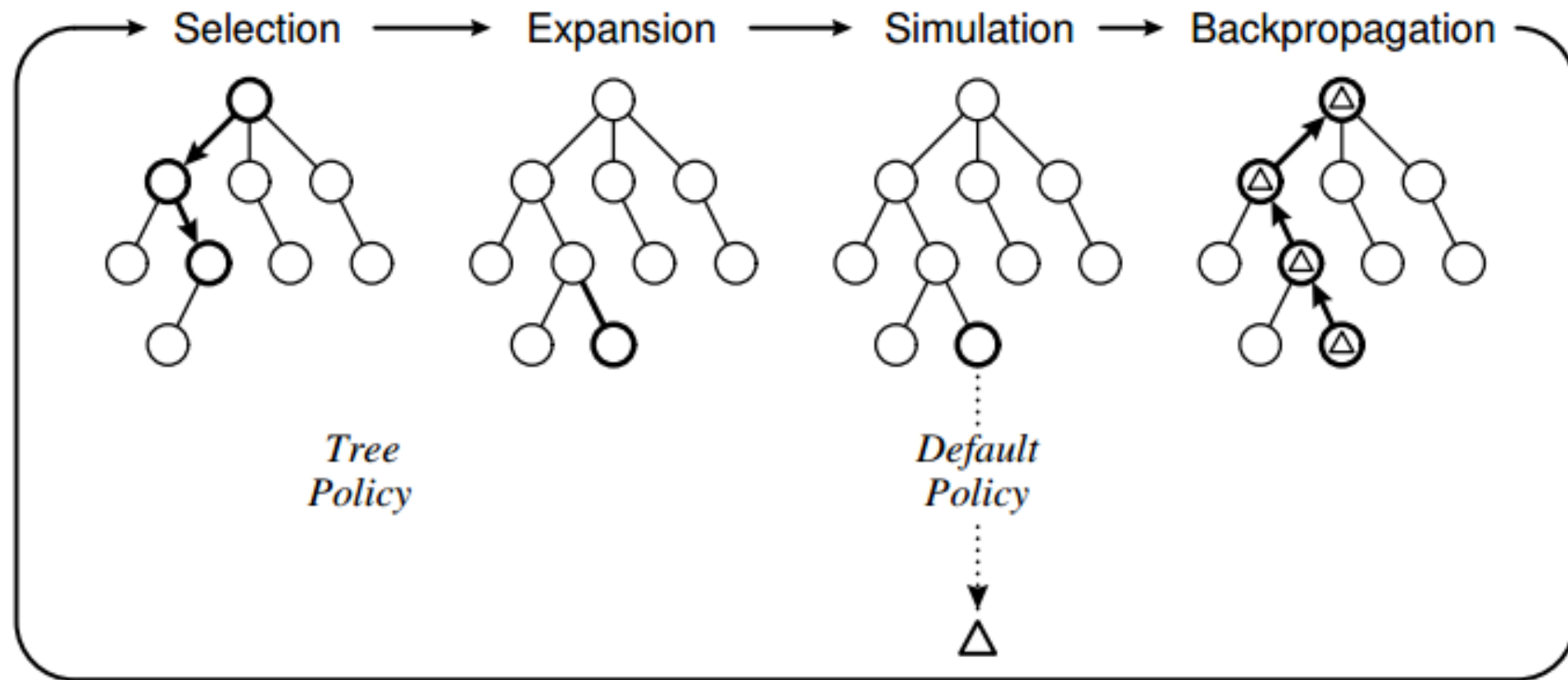
When:

- The search tree would be very big?
Time for each move is limited (so you have no time to build the complete search tree)
 - E.g. in the game Go
 - The branching factor is so high that the algorithm (e.g. Minimax) could only look maybe one step ahead
- It is difficult to tell what is a good position and what is not
 - The algorithm can not have a policy how to determine what move to make

Google Deep minds Alpha Go used a combination of Neural networks, Reinforcement learning and Monte Carlo Tree Search

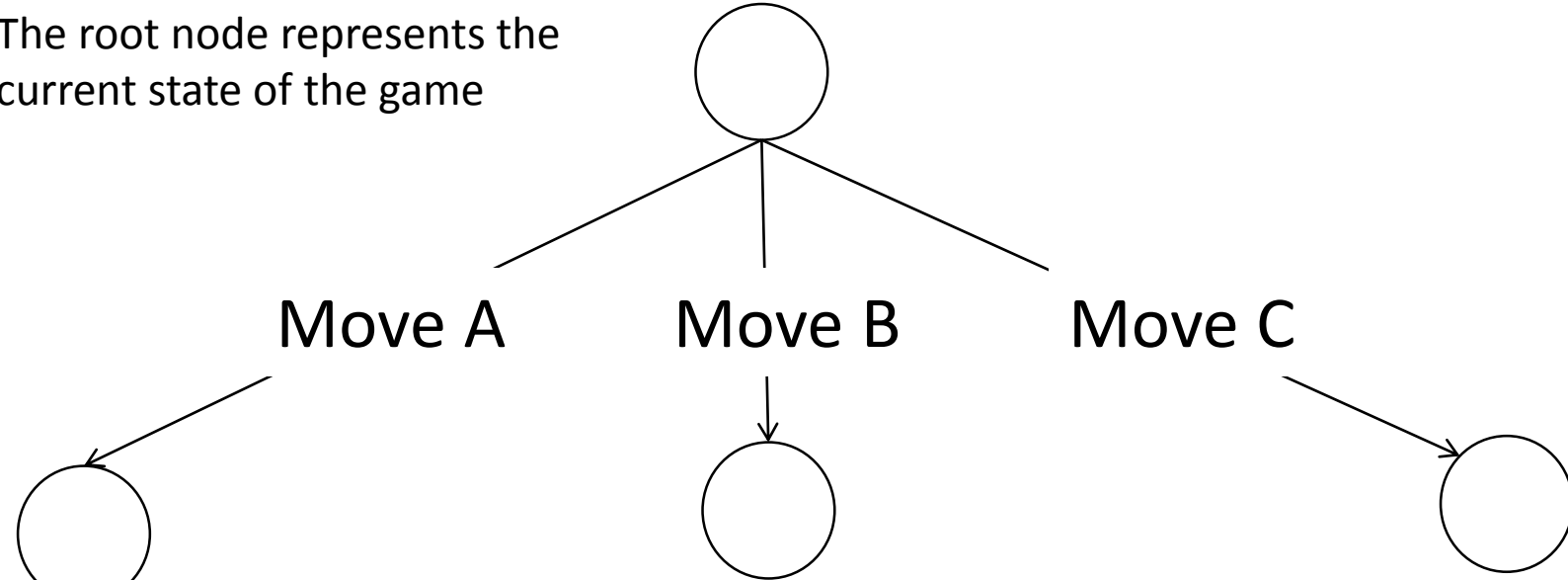


Monte Carlo Tree search



Monte Carlo Tree Search SELECTION

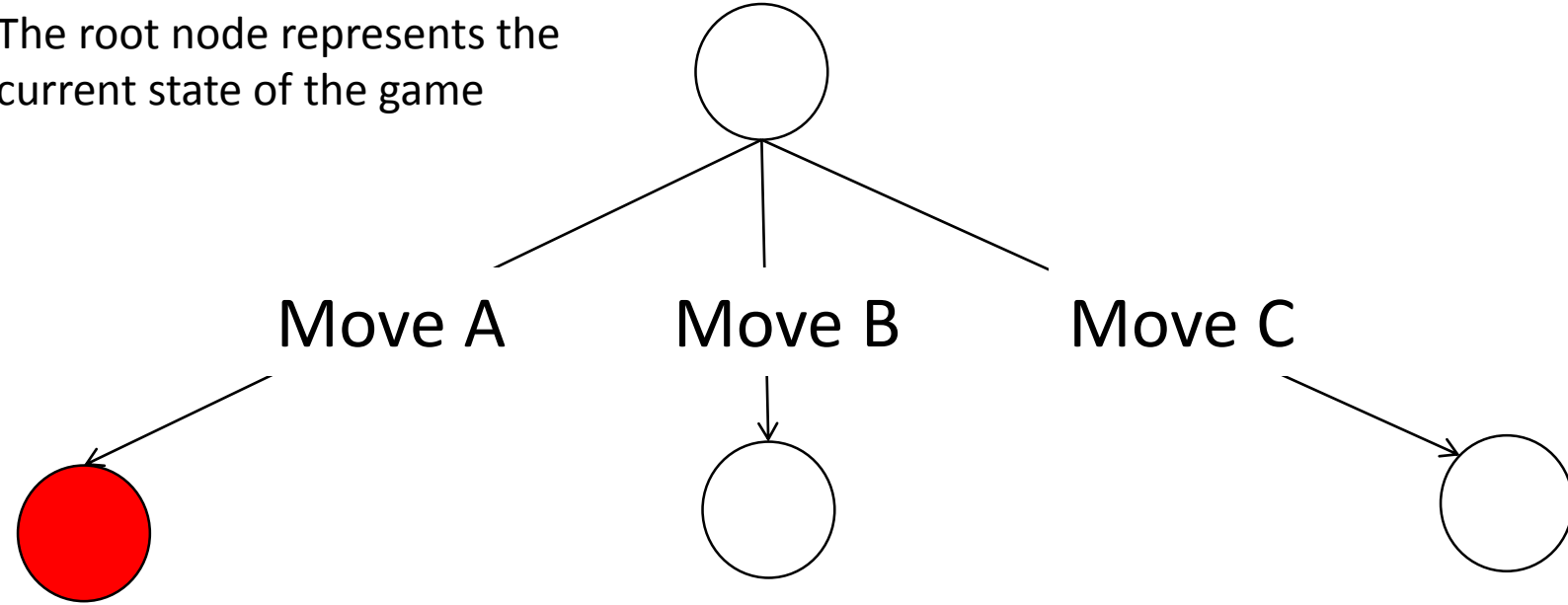
The root node represents the
current state of the game



Possible moves,
none of them has
been further explored yet
They are all leaf nodes at the moment

Monte Carlo Tree Search SELECTION

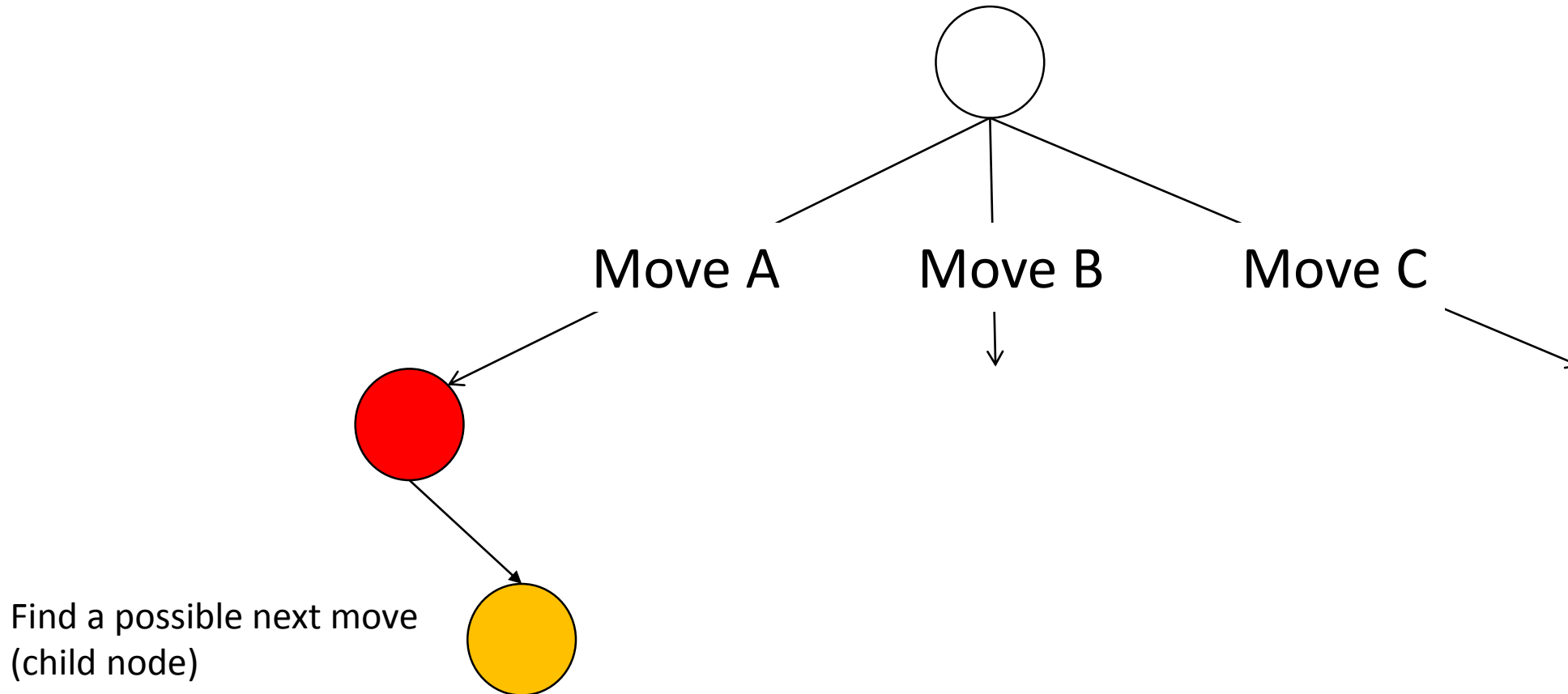
The root node represents the current state of the game



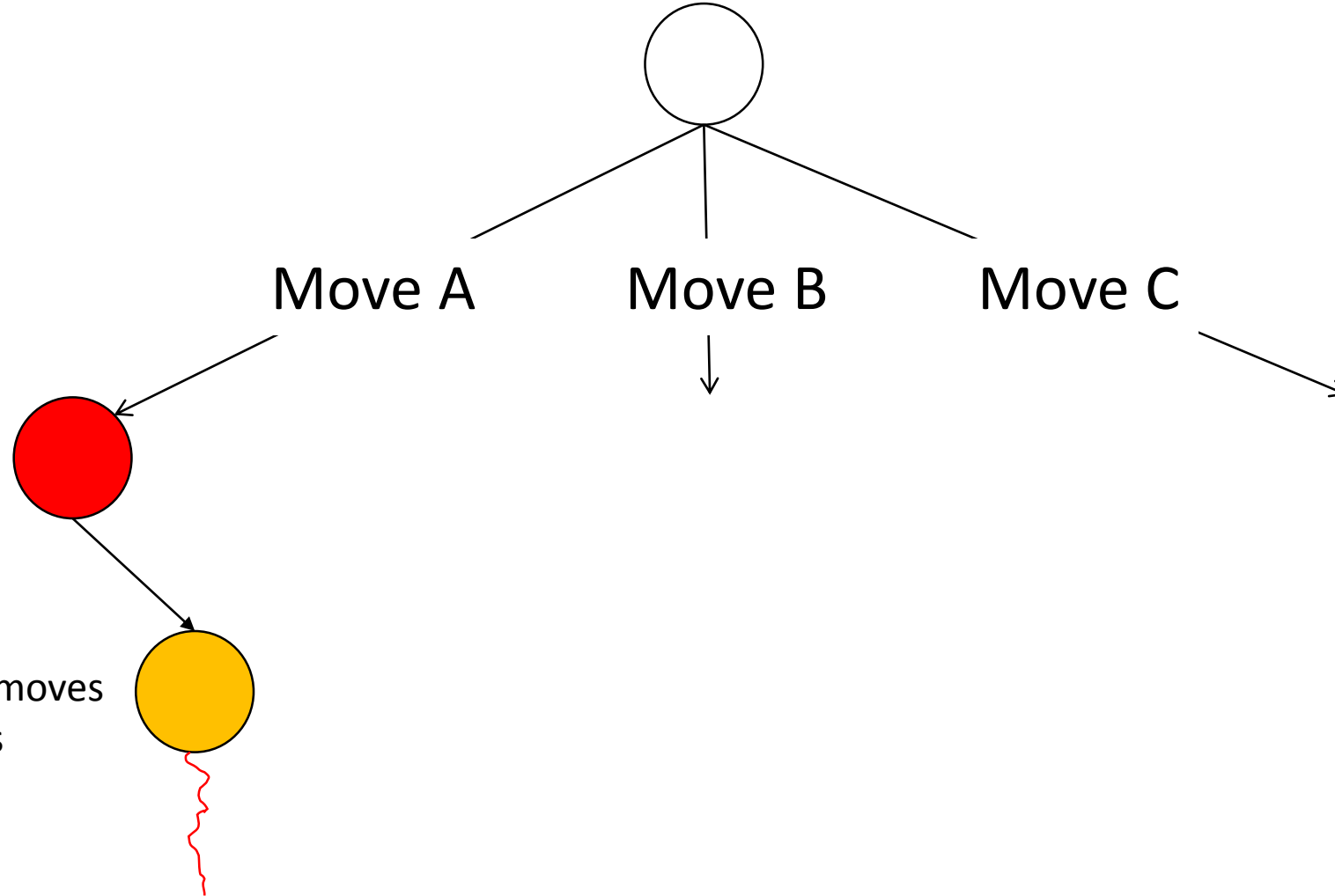
Possible moves,
none of them has
been further explored

Choose one of
them for simulation

Monte Carlo Tree Search **Expansion**

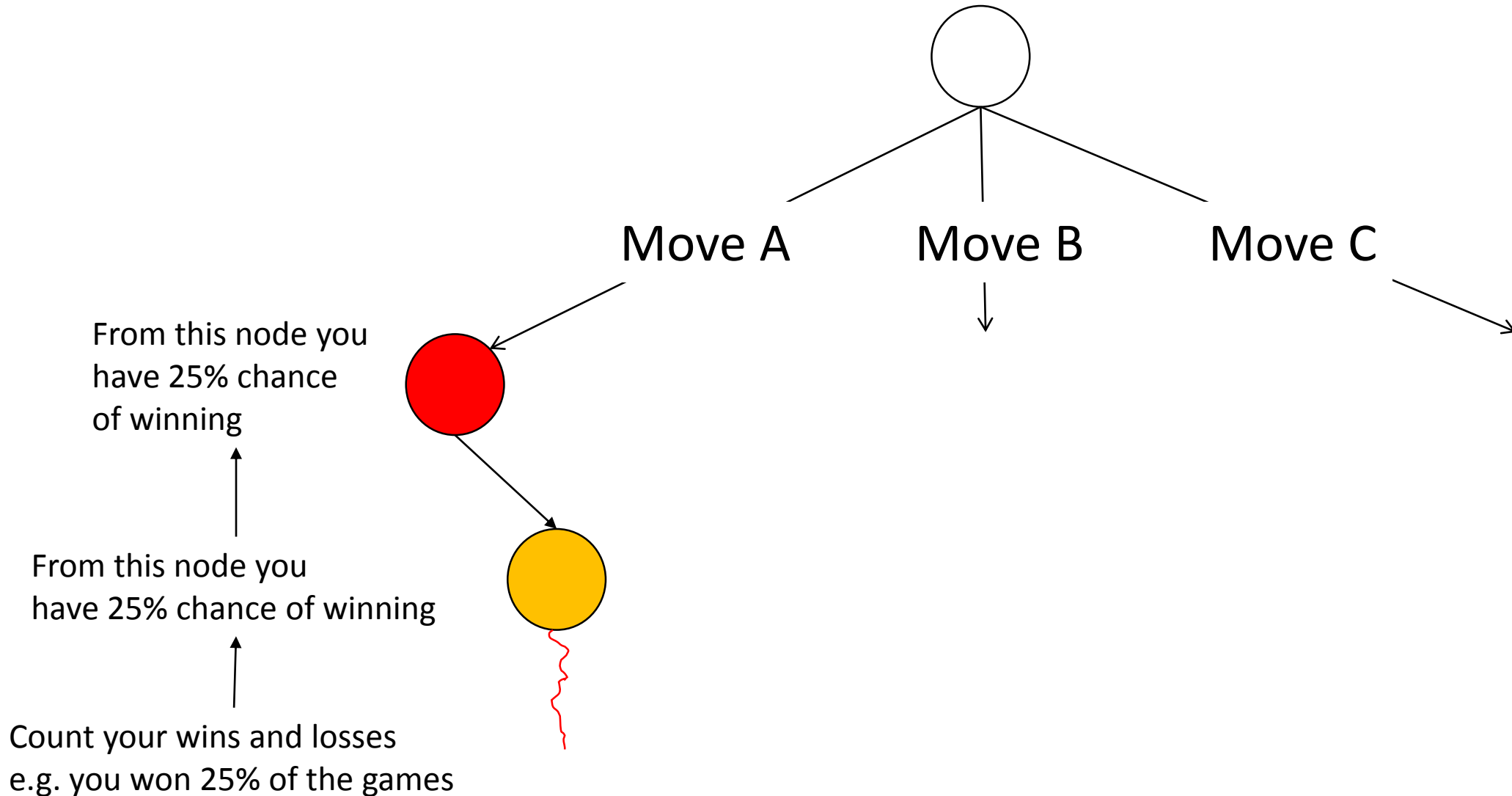


Monte Carlo Tree Search **Simulation**

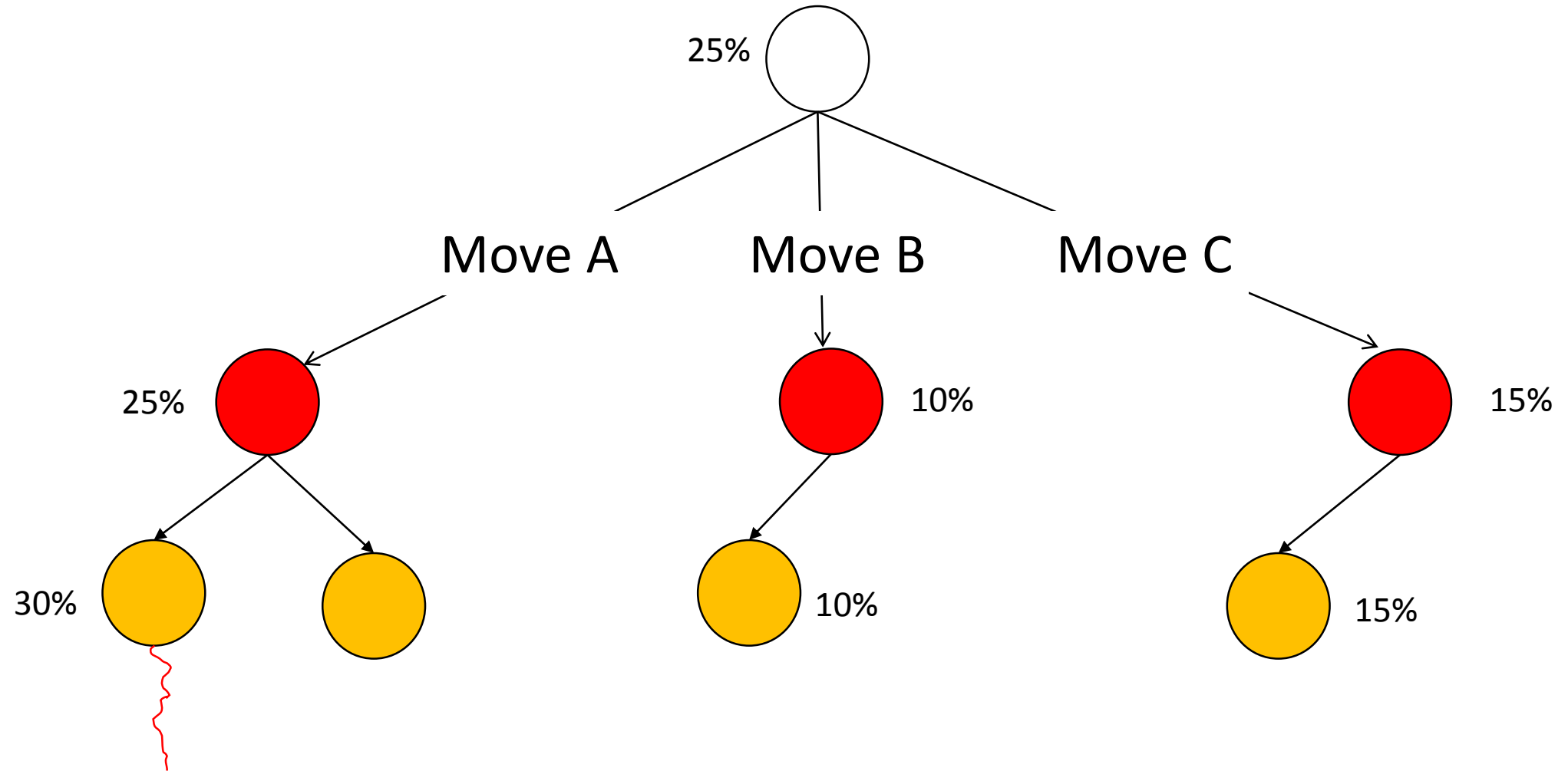


Then just play from there
e.g. by selecting random moves
One time or several times

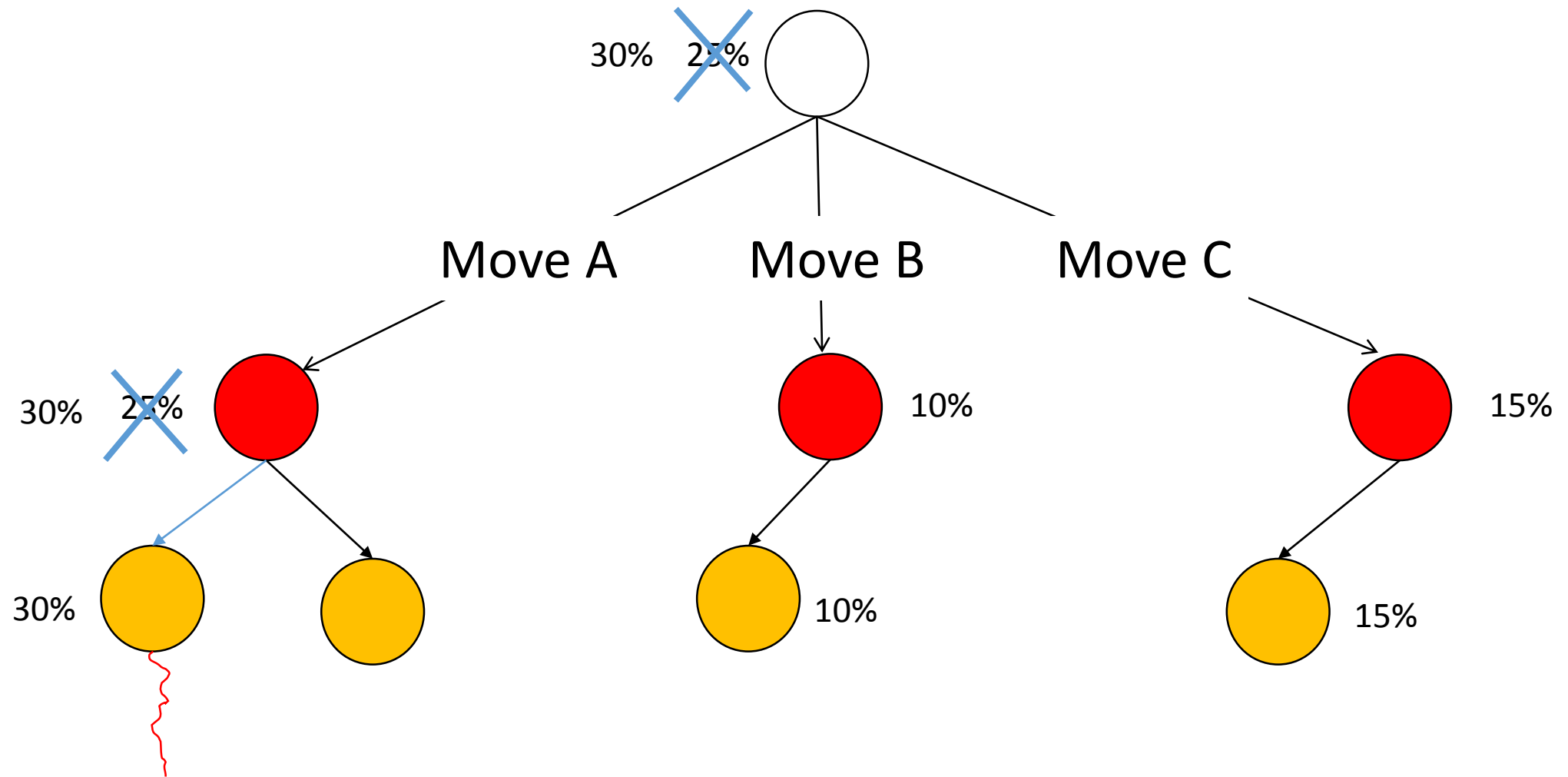
Monte Carlo Tree Search Back propagation



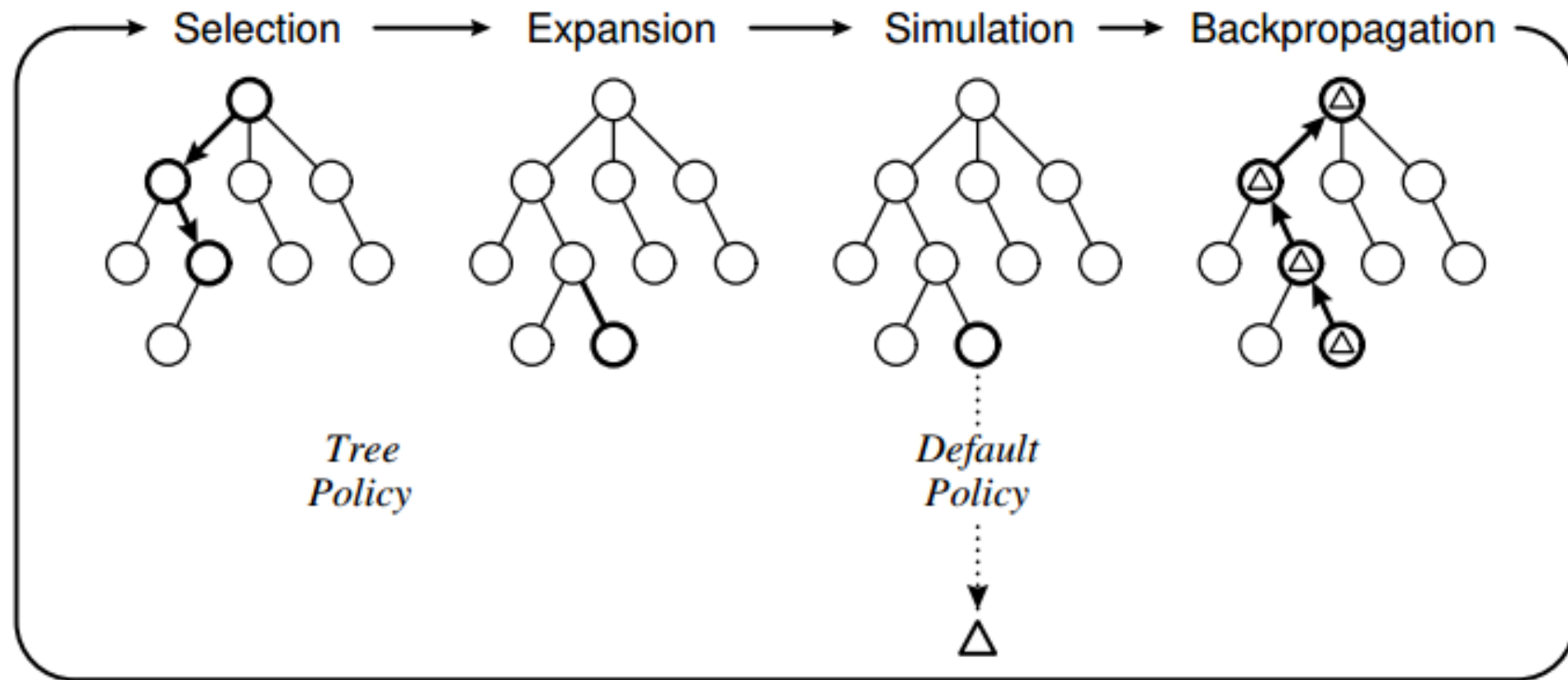
Monte Carlo Tree search Back propagation



Monte Carlo Tree Search Back propagation



Monte Carlo Tree Search



Optimization / Local Search

Looking for improvement from the current state

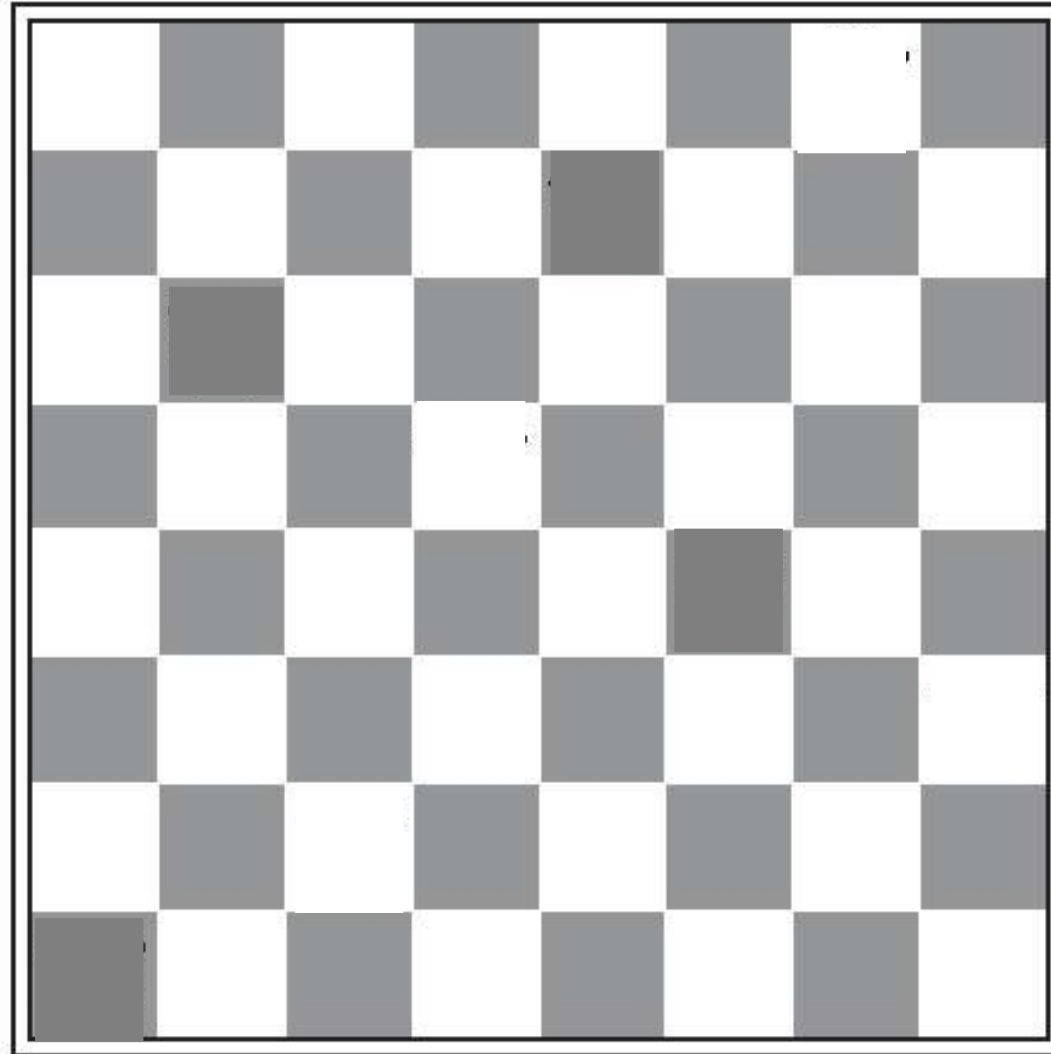
Local Search

- Often used in optimization problems
- It looks “locally” if the solution that it already has can be improved
- Useful when only the goal state counts and not the path to the goal state
- Local search does not use a search tree
- Often you do not have a goal? function, you can only see if the new solution is better or worse than the old one

The eight queens problem

Best Score
= 0

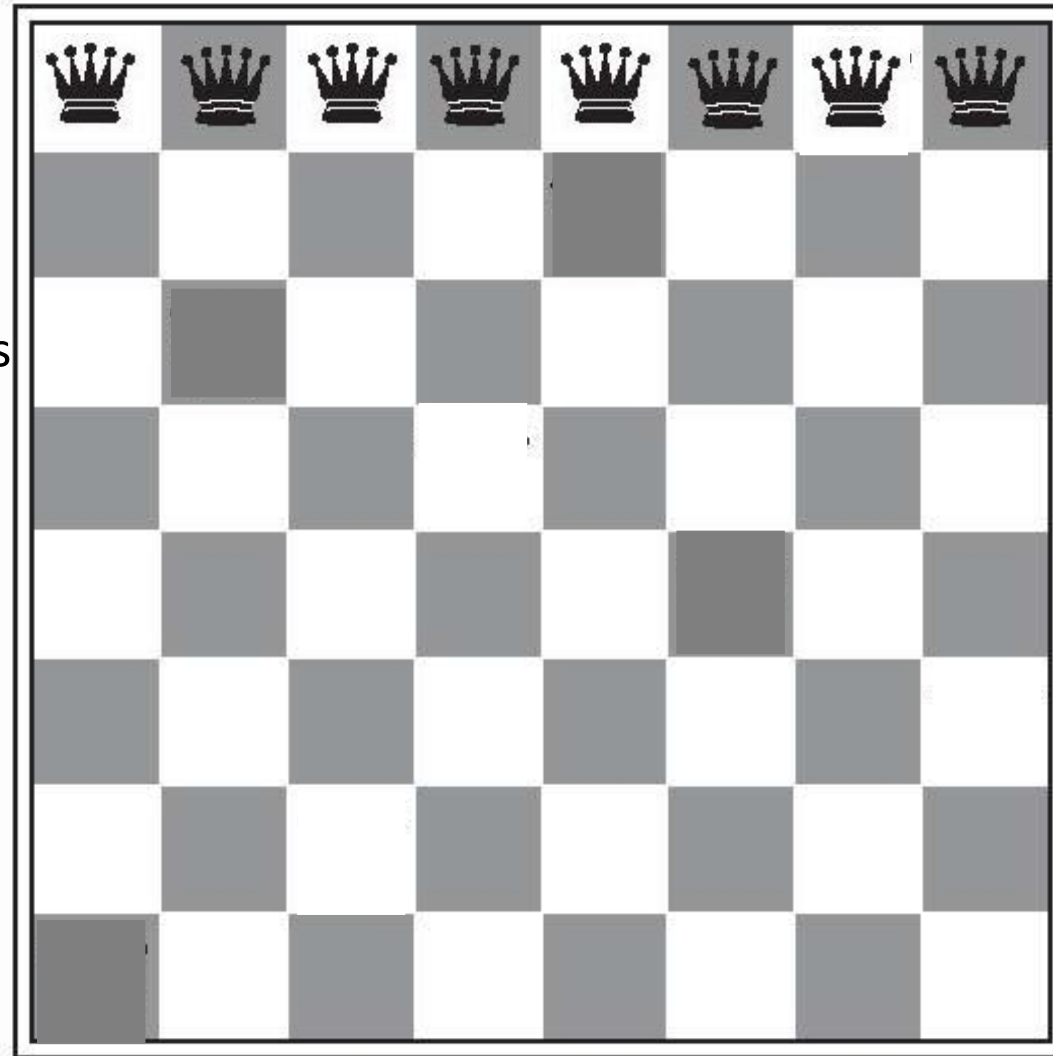
(no pair of queens
is attacking
each other)



The eight queens problem

Score
= 28









(all pairs of queens
are attacking
each other)



The eight queens problem

Score = 17

(1 for each pair of
attacking queens)

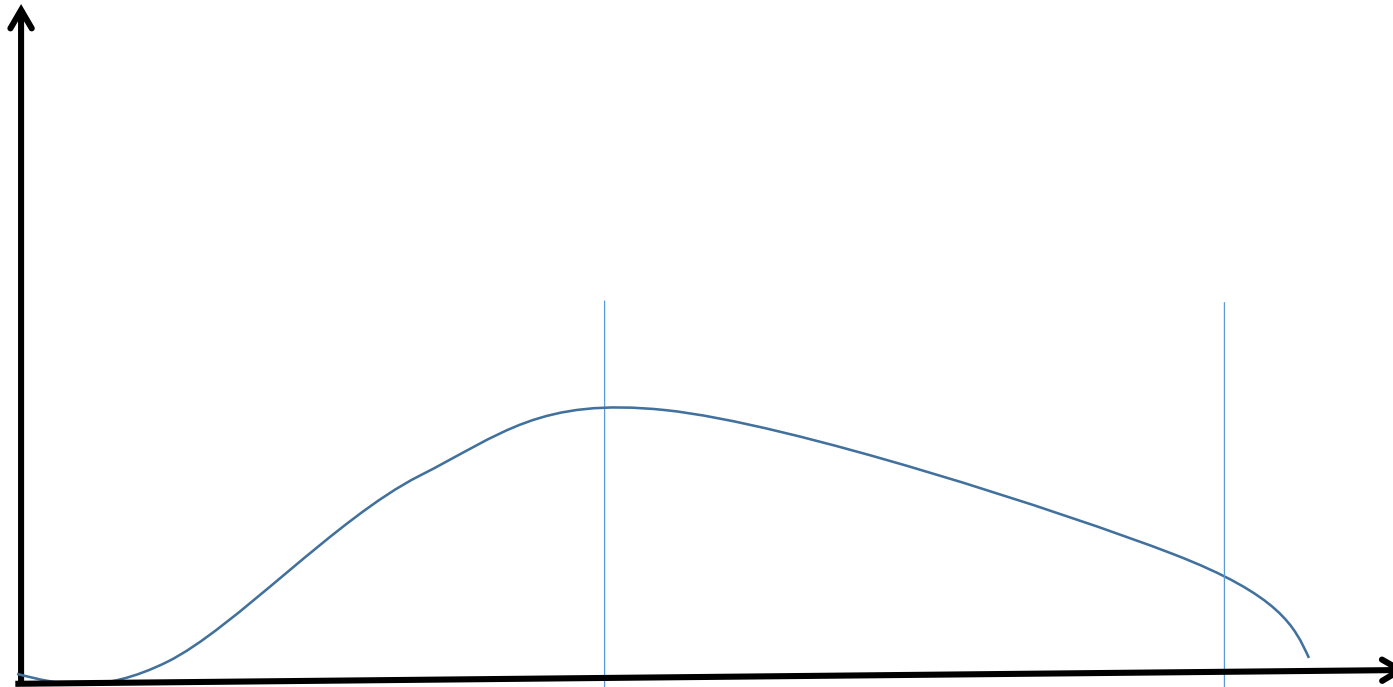
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

Hill climbing



Hill climbing

well being
(objective function)



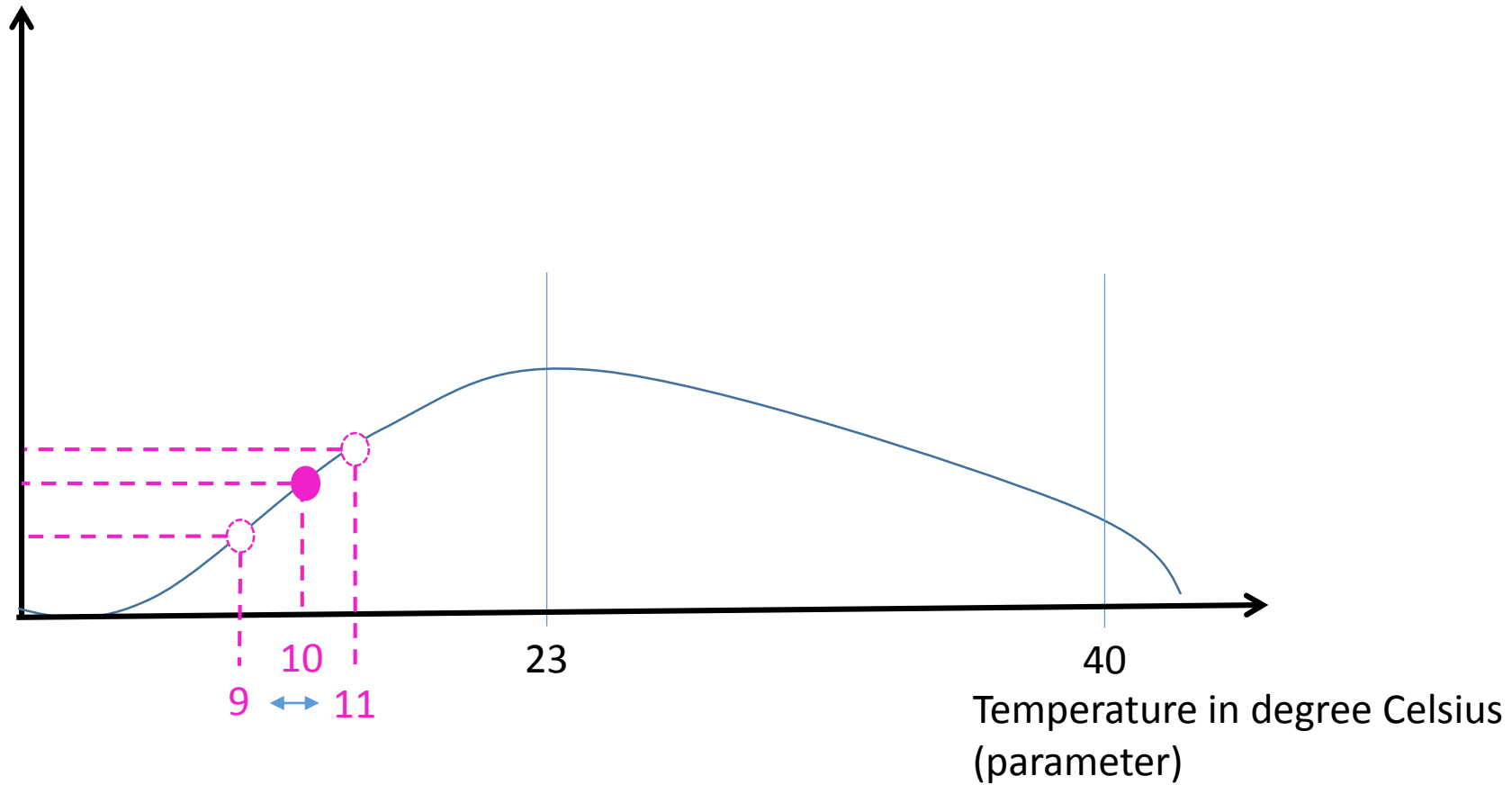
23

40

Temperature in degree Celsius
(parameter)

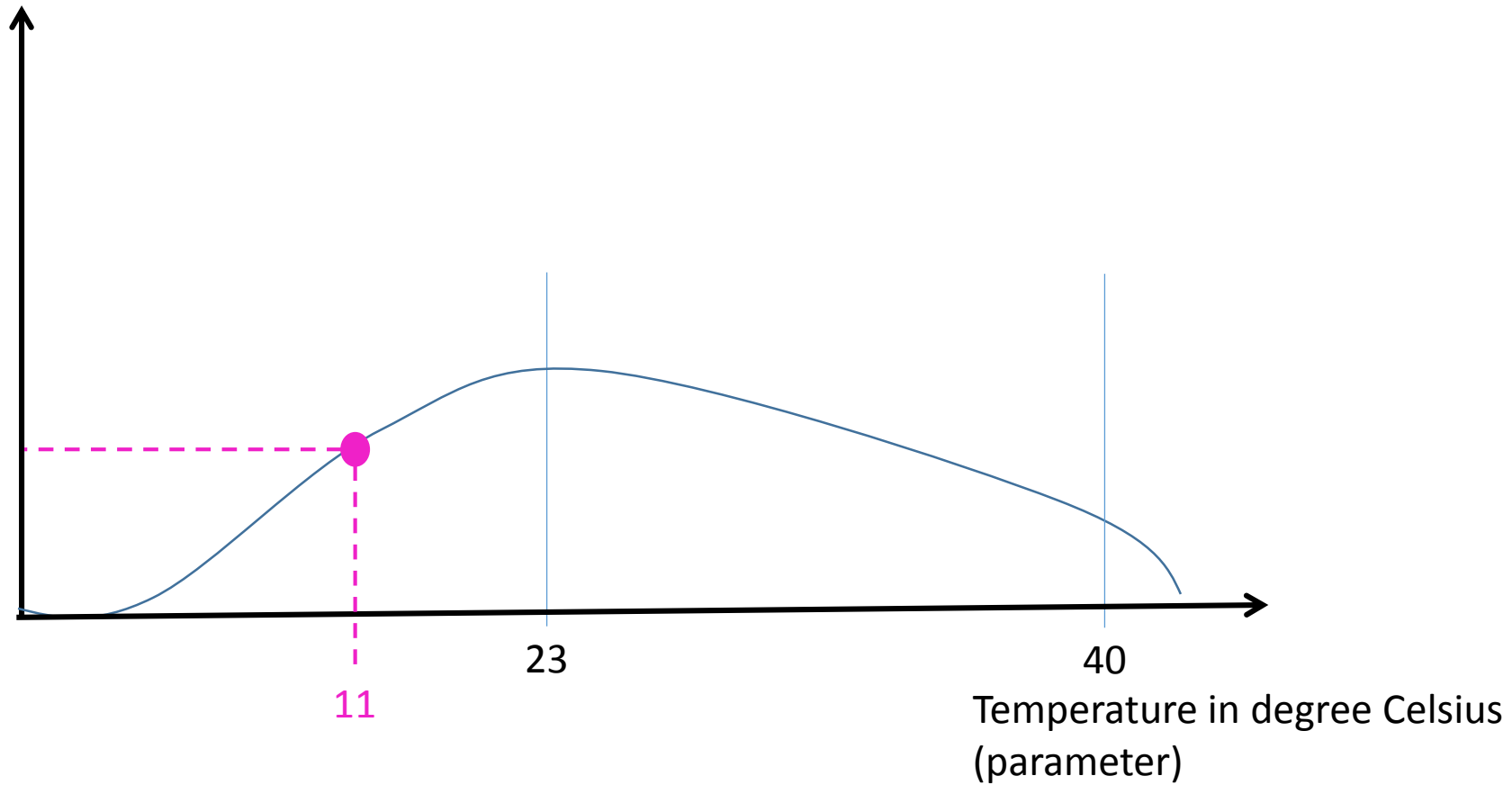
Hill climbing

well being
(objective function)



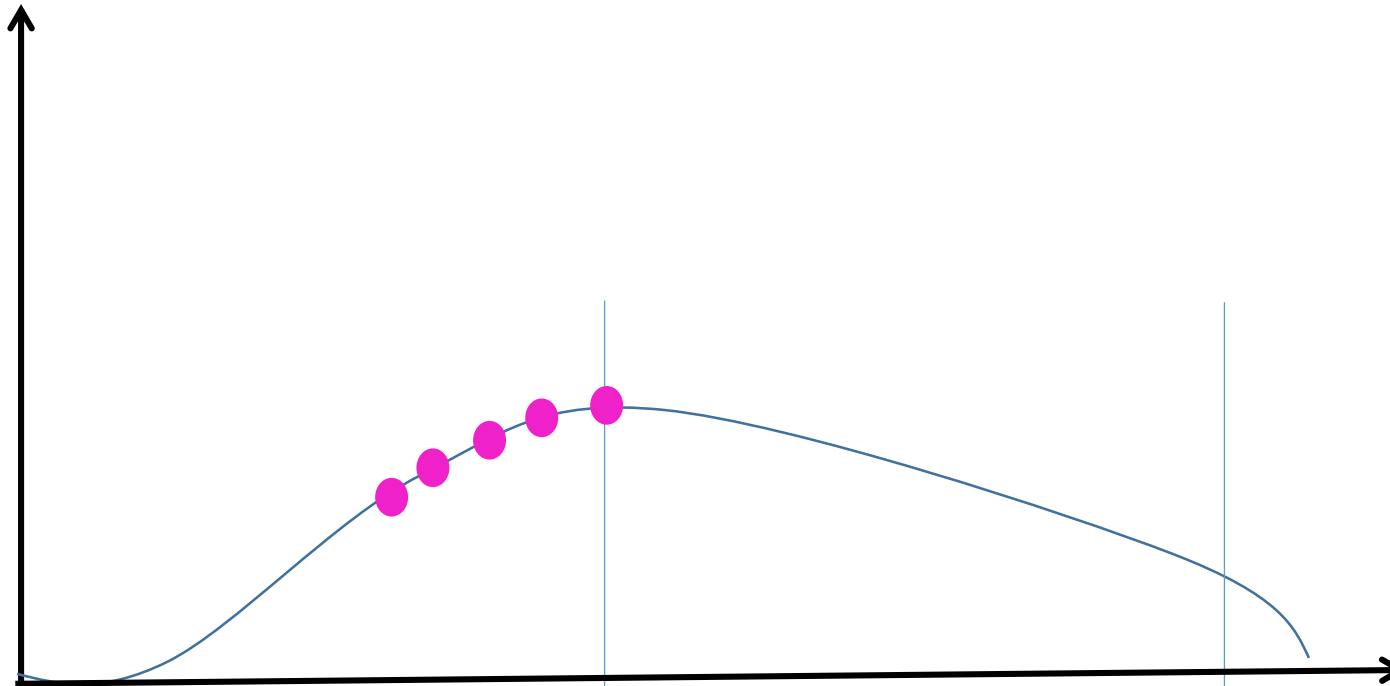
Hill climbing

well being
(objective function)



Hill climbing

well being
(objective function)



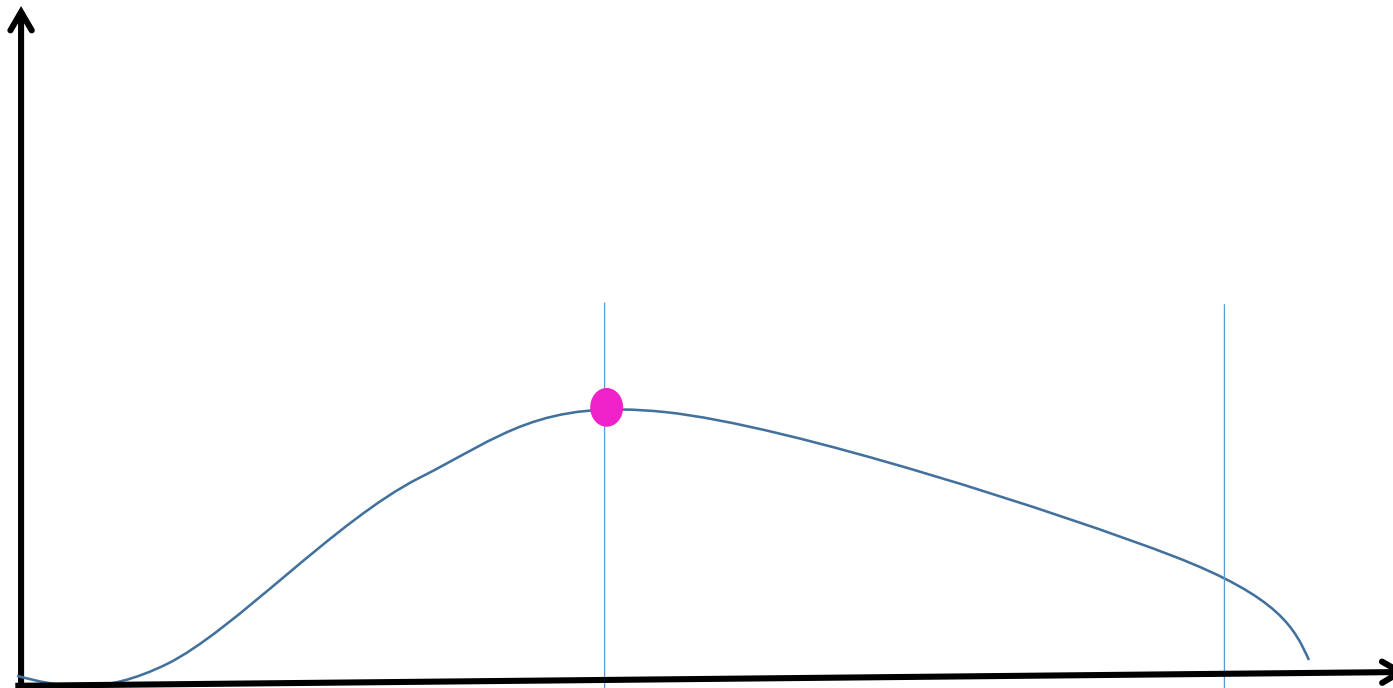
23

40

Temperature in degree Celsius
(parameter)

Hill climbing

well being
(objective function)

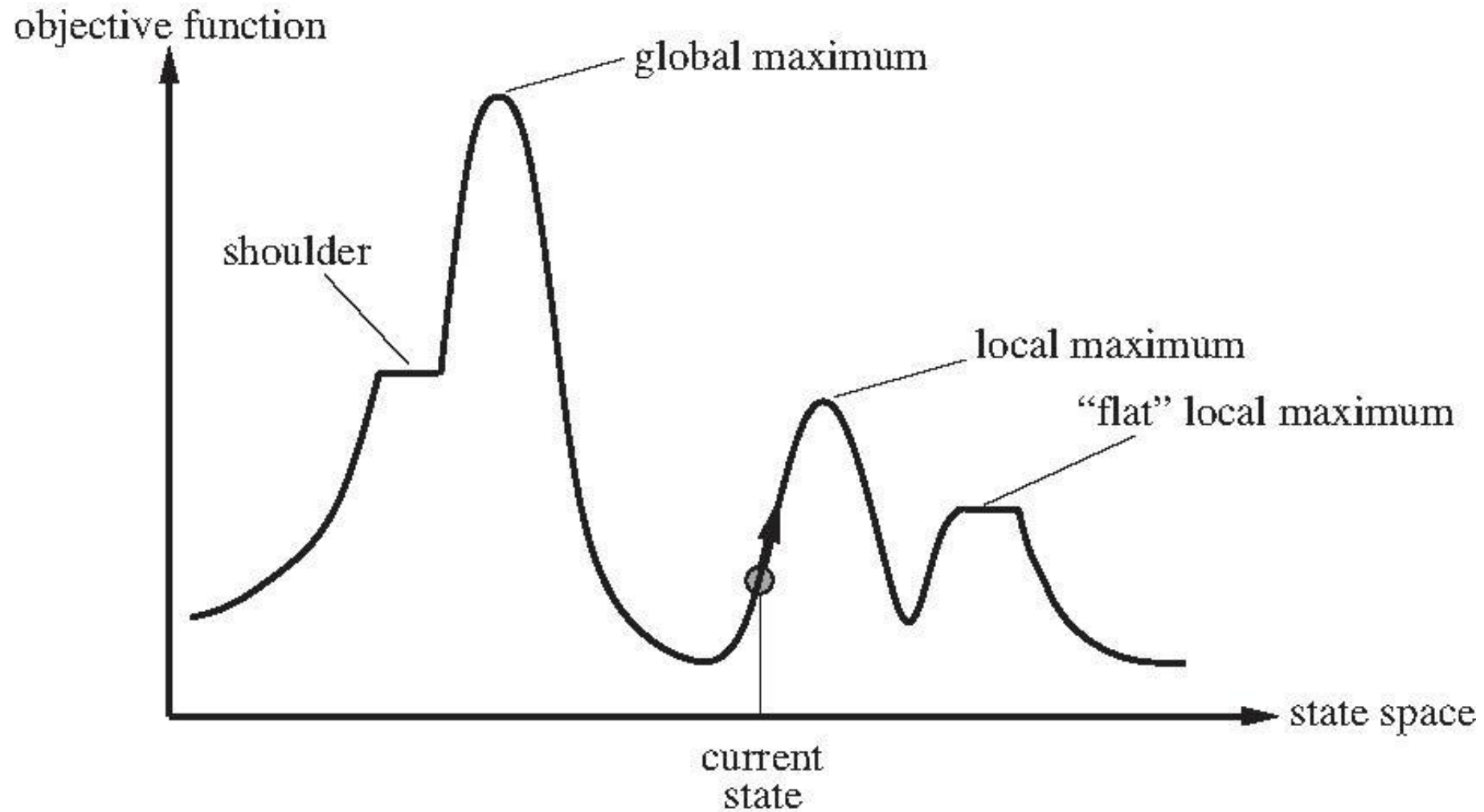


23

40

Temperature in degree Celsius
(parameter)

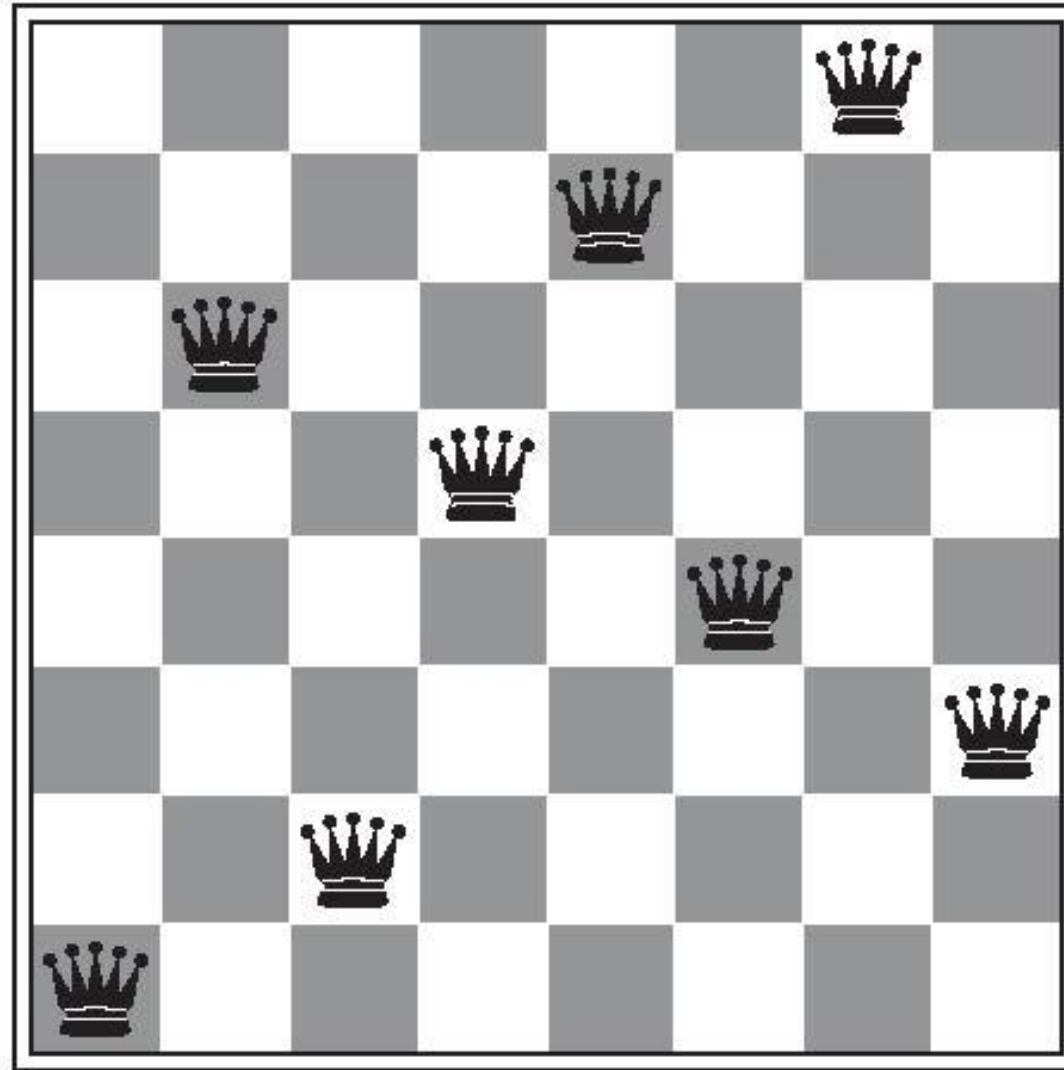
Problems of local search



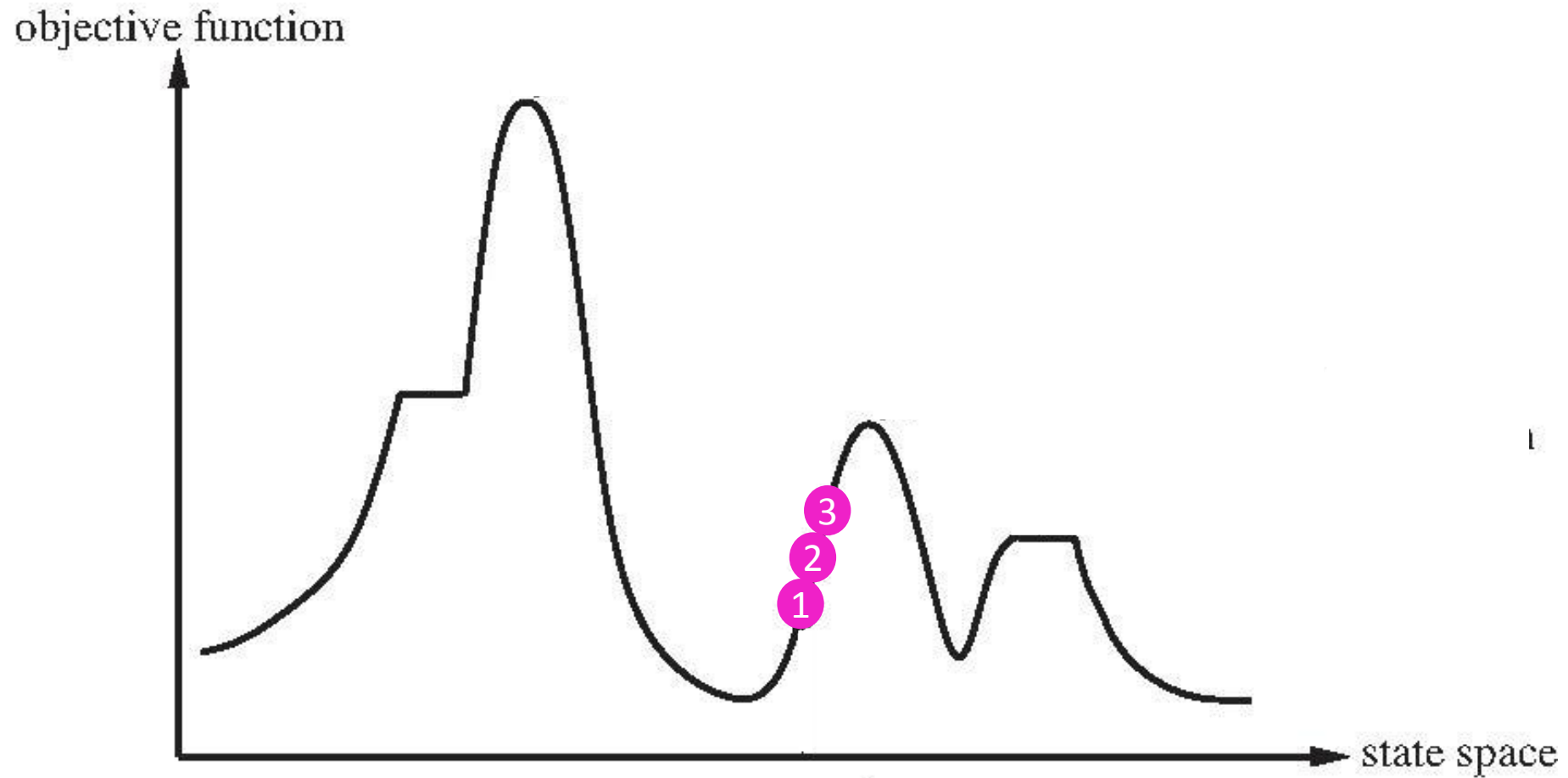
Local Minimum

Score = 1

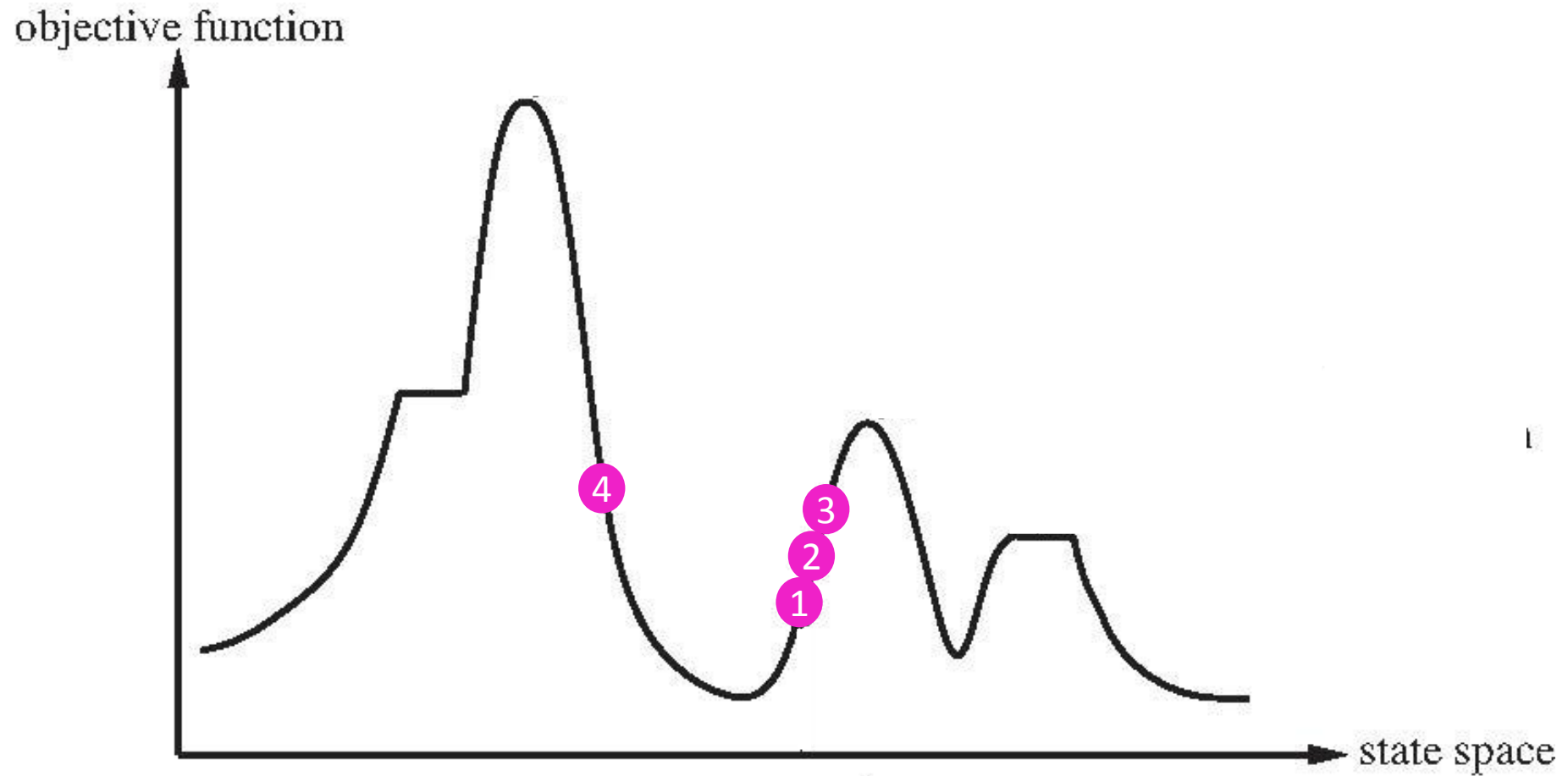
(you cant get
a lower score
than 1 in just
one move)



Simulated Annealing

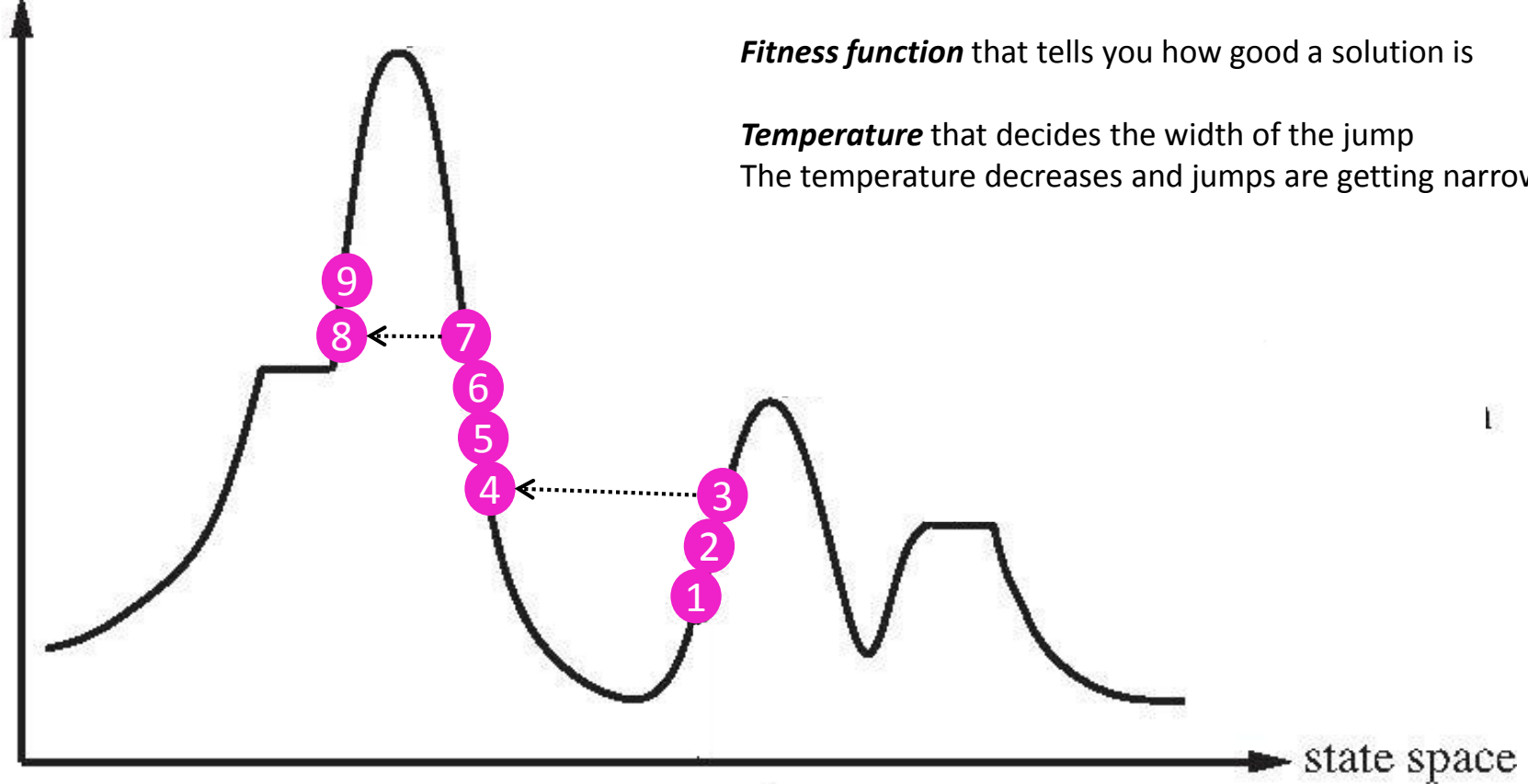


Simulated Annealing

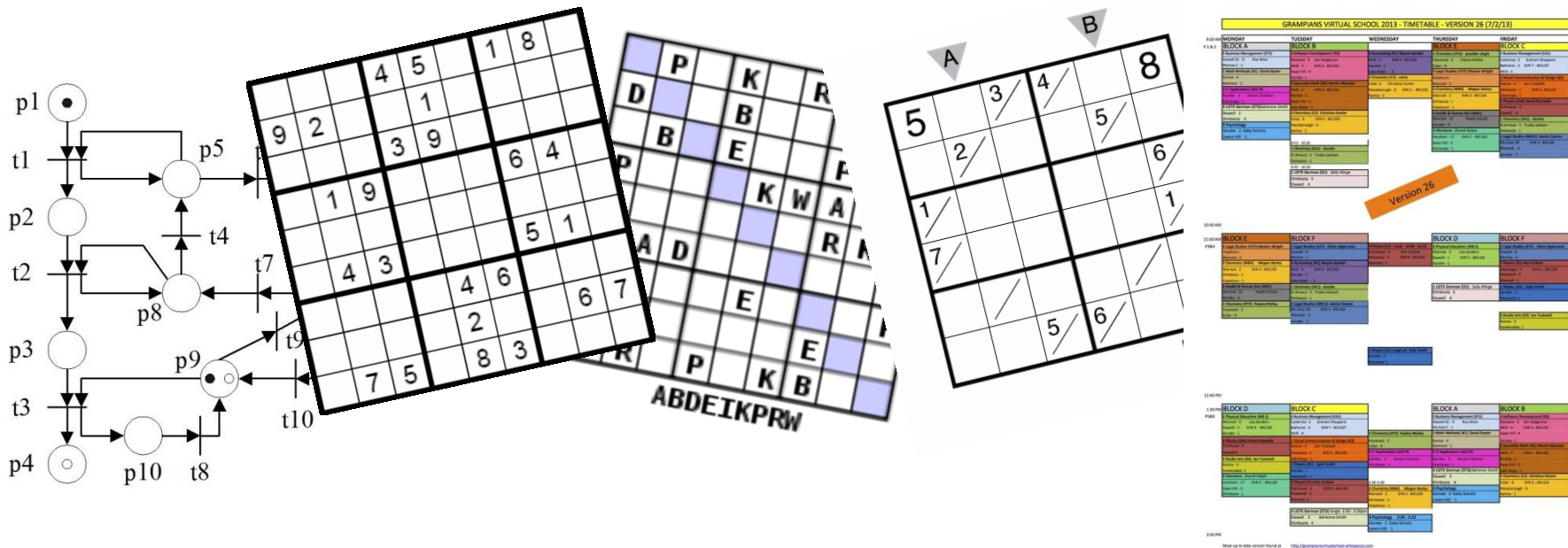


Simulated Annealing

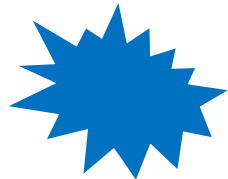
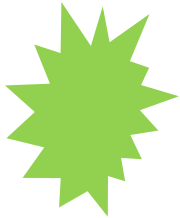
objective function



Constraint Satisfaction Problems



Example: Map coloring



Constraint satisfaction problem

X is a set of variables $\{X_1, \dots, X_n\}$

$\{WA, NT, Q, NSW, V, SA, T\}$

D is a set of domains $\{D_1, \dots, D_n\}$ one for each variable

$D = \{\text{red}, \text{green}, \text{blue}\}$

C is a set of constraints that specify allowable combinations of values.

$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

D_i consists of a set of allowable values $\{v_1, \dots, v_k\}$ for variable X_i

$D_i = \{\text{red}, \text{green}, \text{blue}\}$

C_i consists of a pair $\langle \text{scope}, \text{rel} \rangle$
 $\text{scope} = \text{tuple of variable, rel} = \text{their possible values}$

For example: $\text{scope} = (WA, NT)$

$\text{rel} = \{(\text{red}, \text{blue}), (\text{red}, \text{green}), (\text{blue}, \text{red}), (\text{blue}, \text{green}), (\text{green}, \text{red}), (\text{green}, \text{blue})\}$

Constraint Satisfaction Problem

Example: Map coloring

Variables (States of Australia)

Western Australia (WA)

Northern Territory (NT)

Queensland (Q)

New South Wales (NSW)

Victoria (V)

South Australia (SA)

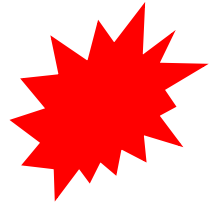
Tasmania (T)



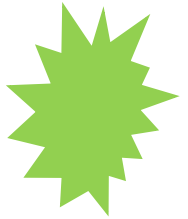
Constraint Satisfaction Problem

Example: Map coloring

Each variable (state of Australia) can either be red, green, or blue.



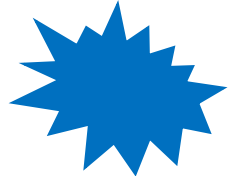
Domain for variables:
 $D_i = \{\text{red, green, blue}\}$



$D_{WA} = \{\text{red, green, blue}\}$

$D_{NT} = \{\text{red, green, blue}\}$

$D_Q = \{\text{red, green, blue}\}$



$D_{NSW} = \{\text{red, green, blue}\}$

$D_V = \{\text{red, green, blue}\}$

$D_{SA} = \{\text{red, green, blue}\}$

$D_T = \{\text{red, green, blue}\}$



Constraint Satisfaction Problem

Example: Map coloring

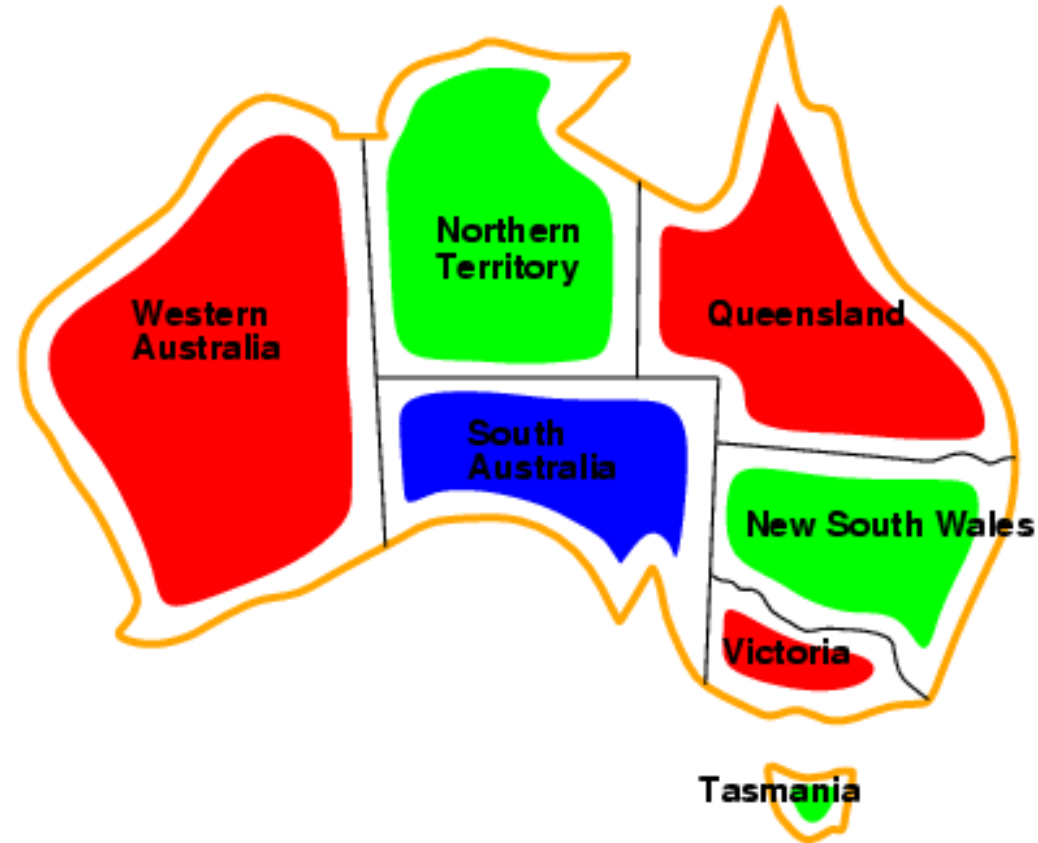
Neighboring regions must have distinct colors

$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$



Constraint Satisfaction Problem

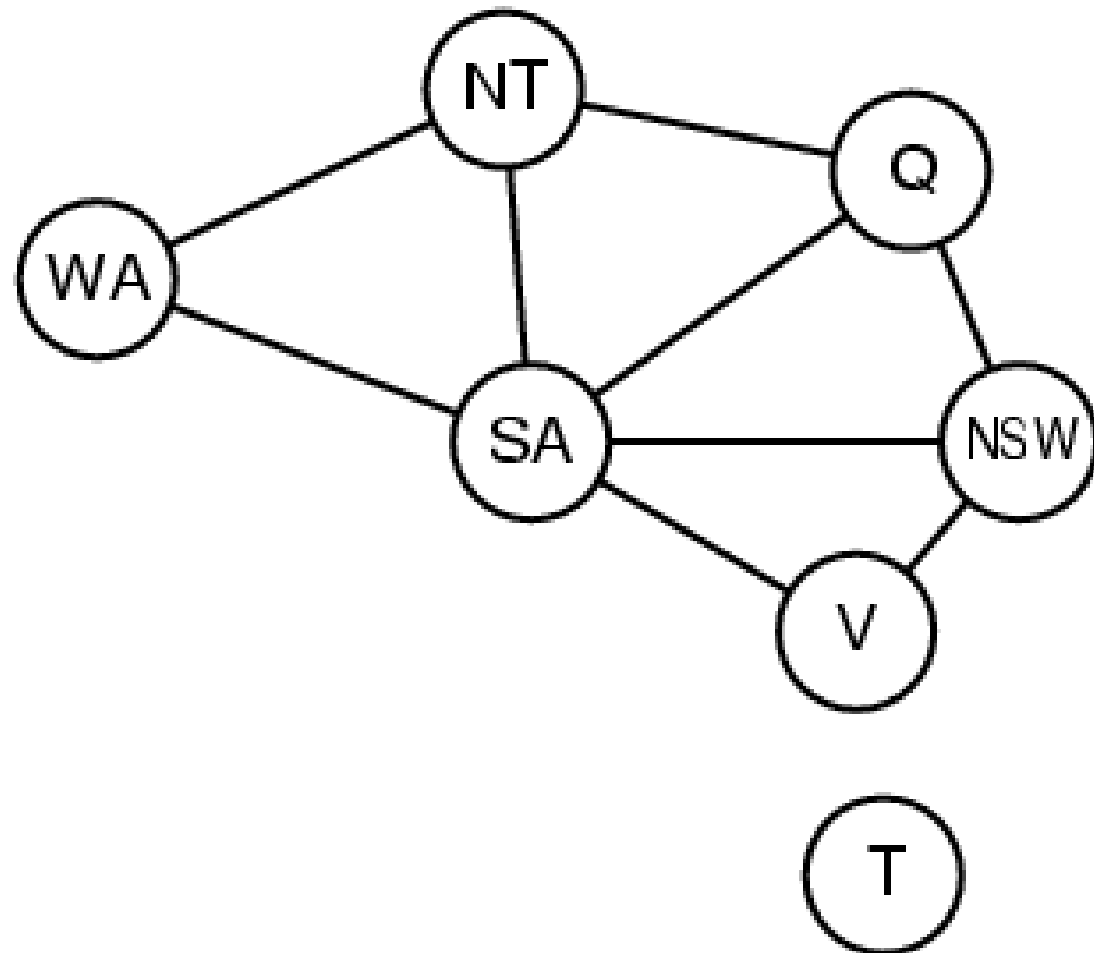
Example: Map coloring



{WA = red, NT = green, Q = red, NSW = green,
V = red, SA = blue, T = green}

Constraint Satisfaction Problem

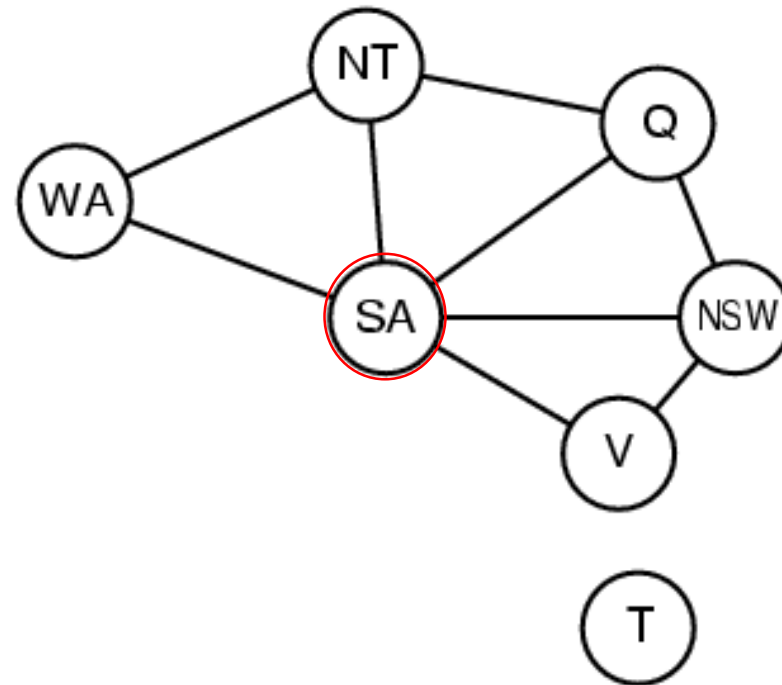
Example: Map coloring = graph coloring



Constraint Satisfaction Problem

Example: Map coloring

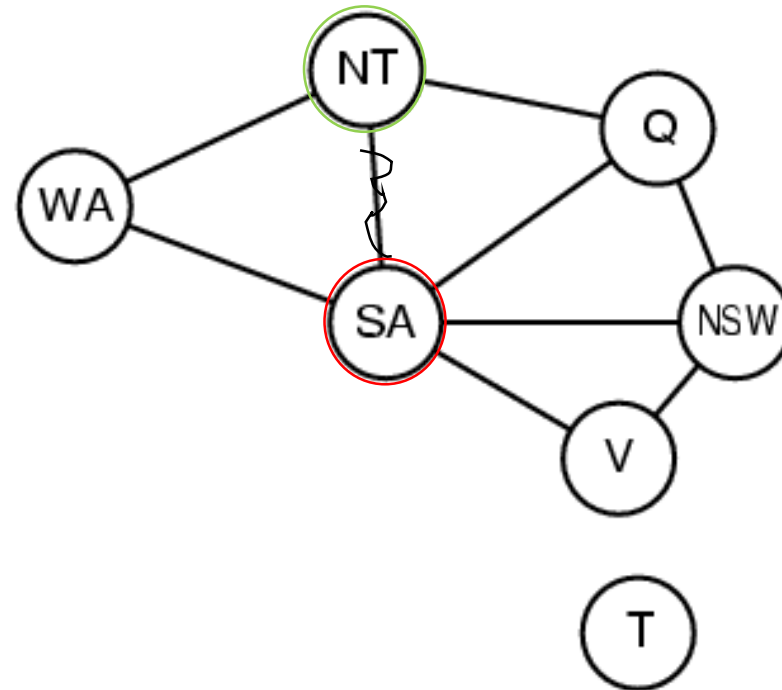
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

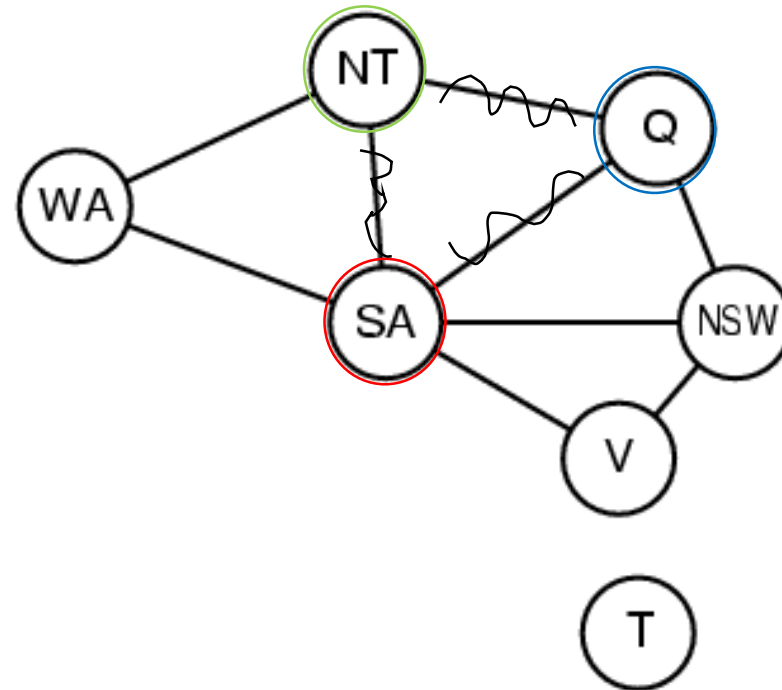
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

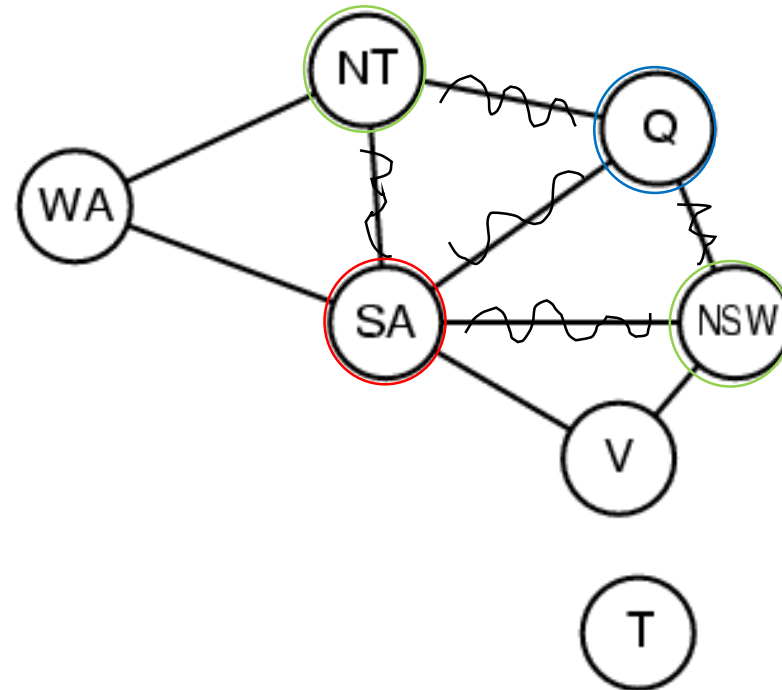
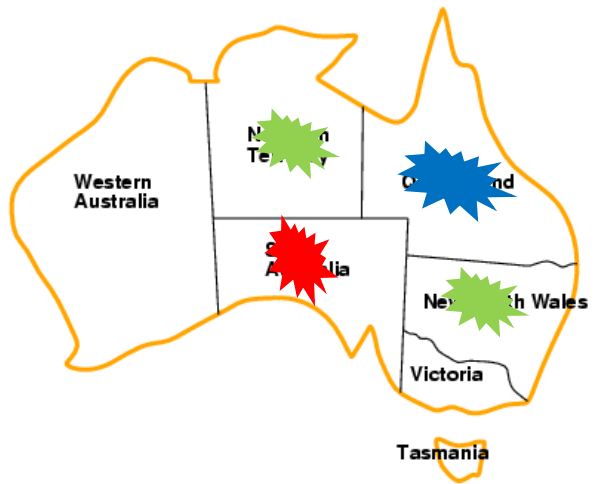
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

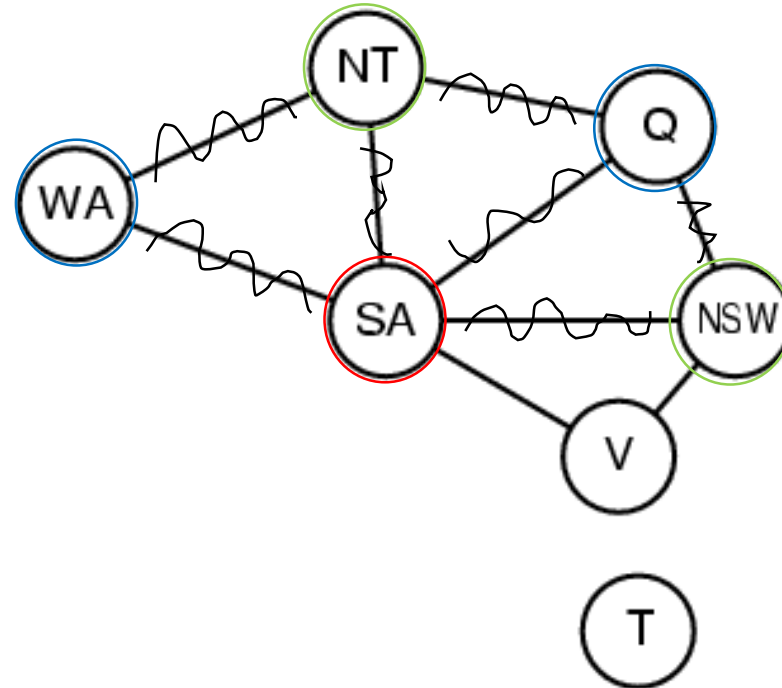
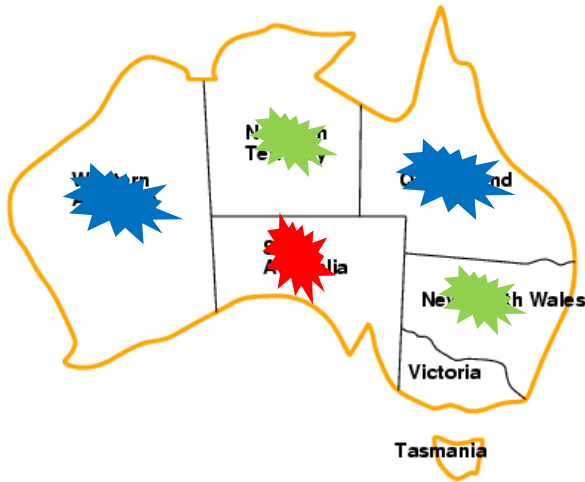
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

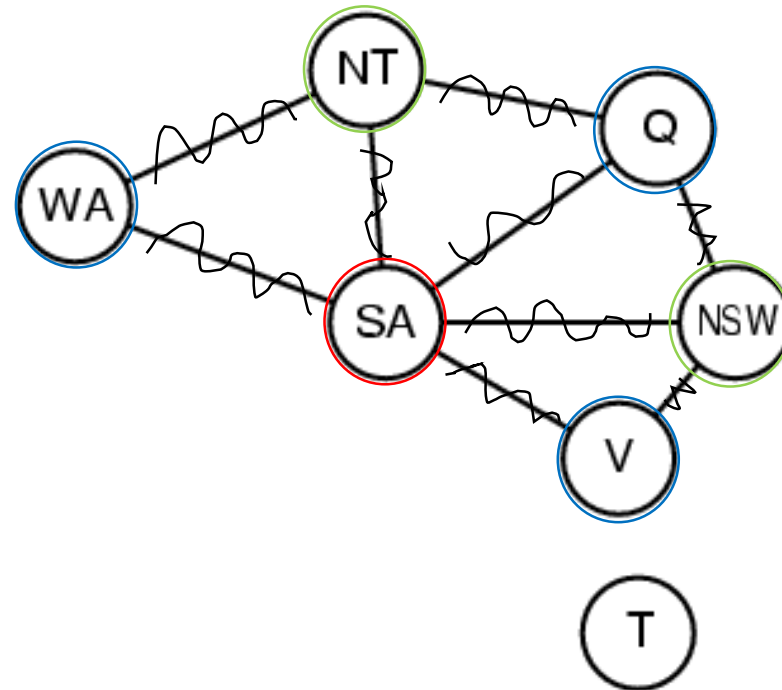
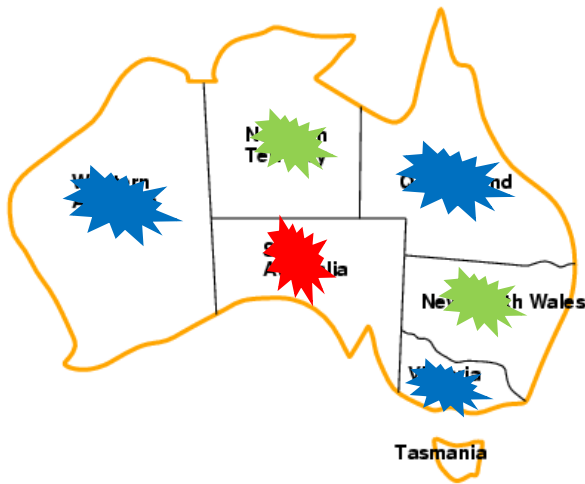
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

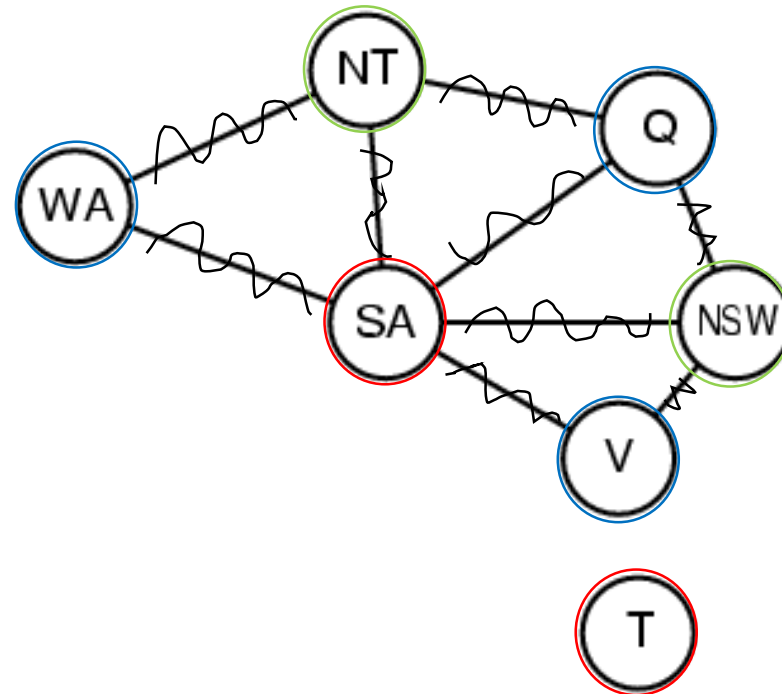
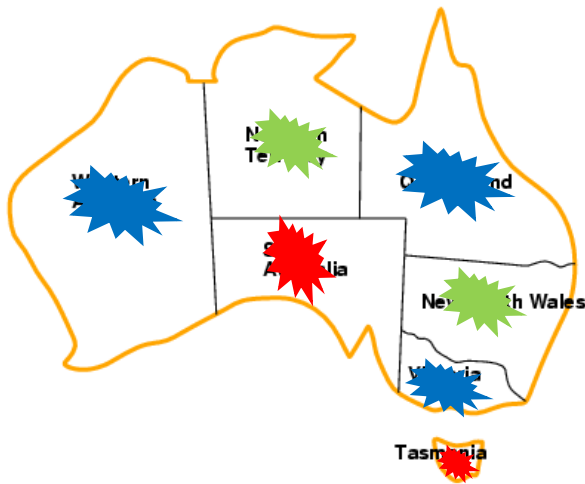
- Always start with the one with the most constraints



Constraint Satisfaction Problem

Example: Map coloring

- Always start with the one with the most constraints



Training Problems

Questions that you should be able to answer

Agents that you should be able to build

Environments that you should be able to define

Sliding Puzzle

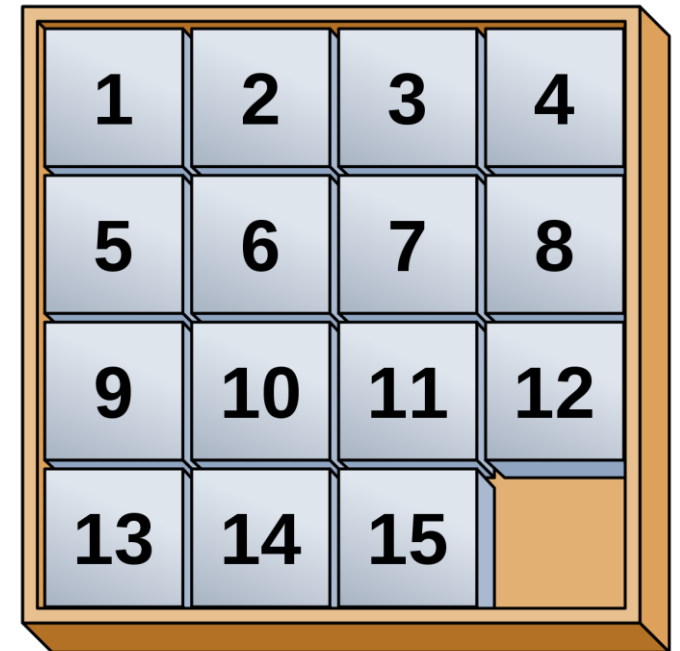
Initial state

2	3
1	

Goal state

1	2
3	

1. Build the search tree
2. What algorithm would be best and why?
3. What depth has the tree?
4. What branching factor has the tree?
5. Do you need to model any costs (why/why not?)
6. Are infinite loops possible?
7. If yes in 6 how can you avoid infinite loops?
8. Does your solution scale to very big instances of the sliding puzzle?



The wolf-goat-cabbage problem

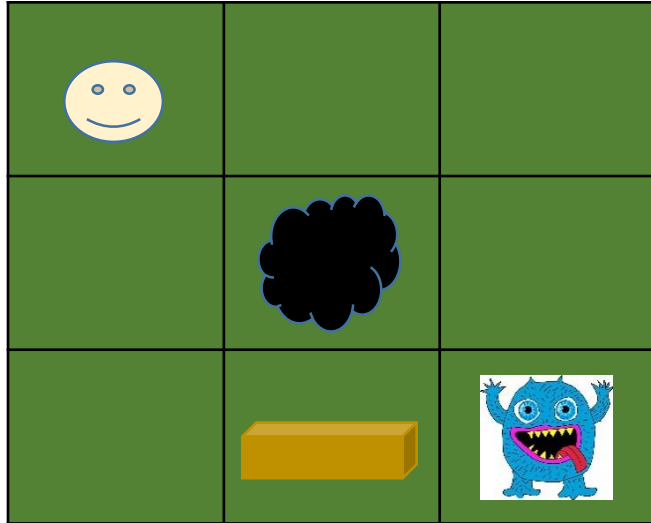
You want to cross a river with your boat, a cabbage, a goat, and a wolf. You need to get everything on the other side.

1. But there are some difficulties
2. There is only space in the boat for two items
3. you are the only one that can row the boat
4. If you leave the goat with the cabbage alone the goat will eat the cabbage
5. If you leave the goat with the wolf alone the wolf will eat the goat.

The wolf-goat-cabbage problem

1. Find a representation for the four objects on either side of the river
2. Build a search tree, where the goal state is that everything including you are safely on the other side of the river
3. Can you prune your search tree?
4. What are the constraints?
5. How is the maximum depth of the tree?
6. On what depth lies the shallowest solution?
7. Are infinite loops possible?
8. If yes in 7, what can you do to avoid it?
9. In order to find the shallowest solution what search algorithm would you use?

The Wumpus world



Environmental Input (Senses)

1. In a field adjacent to the field with a pit, you feel a cold breeze
2. In a field adjacent to the field with the gold, you see a glitter
3. In a field adjacent to the field with the Wumpus you smell a stench

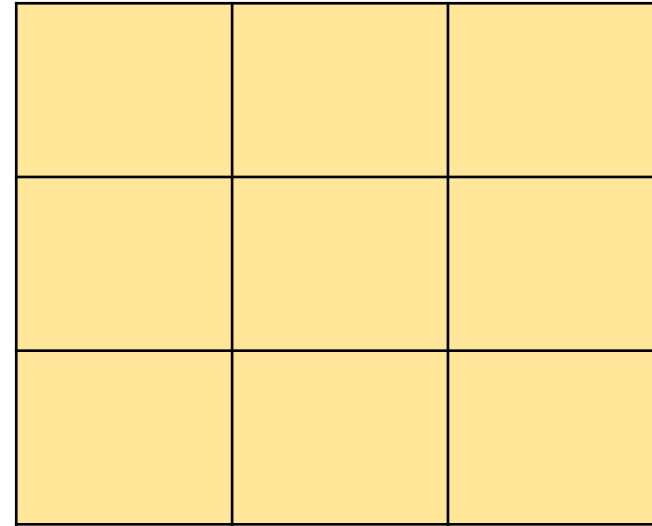
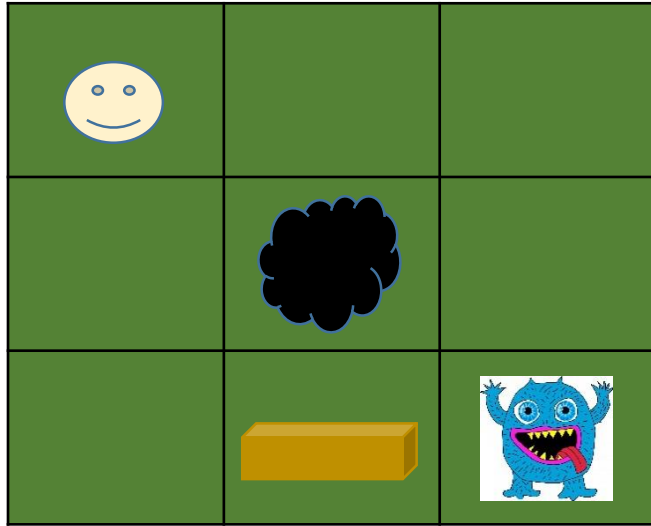
Actions possible:

You can only move vertically or horizontally one field at the time

How would you solve this problem?

The Wumpus world

The agent can make an internal representation based on what it has observed

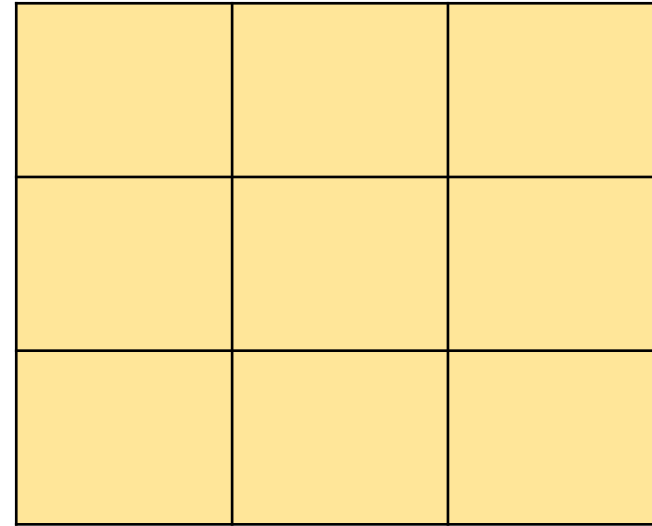
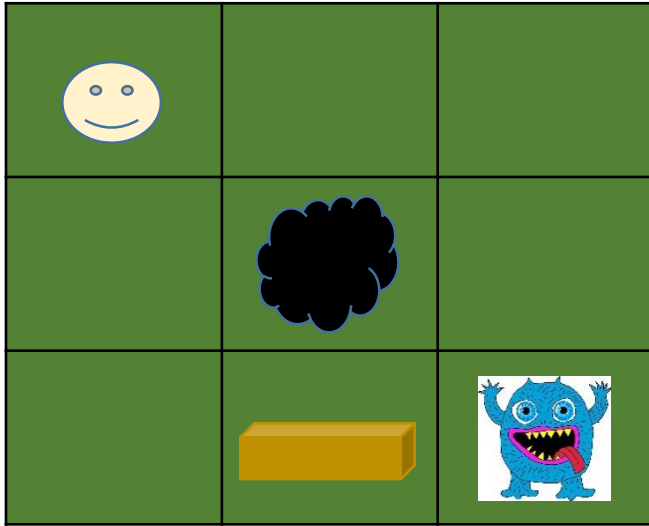


What does your agent know?

- The size of the world?
- The general layout of the world (e.g. 3x3 and which fields are adjacent to each other)
- How many Wumpus there are?
- How many pits there are?
- How many gold bars it may find?
- That it feels a breeze when adjacent to a pit?
- That it smells a stench when adjacent to the Wumpus?
- That it sees a glitter when adjacent to the gold?
- What are your agents reasoning capabilities?
 - When it smells a stench?
 - When it sees a glitter?
 - When it feels a breeze

The Wumpus world

The agent can make an internal representation based on what it has observed

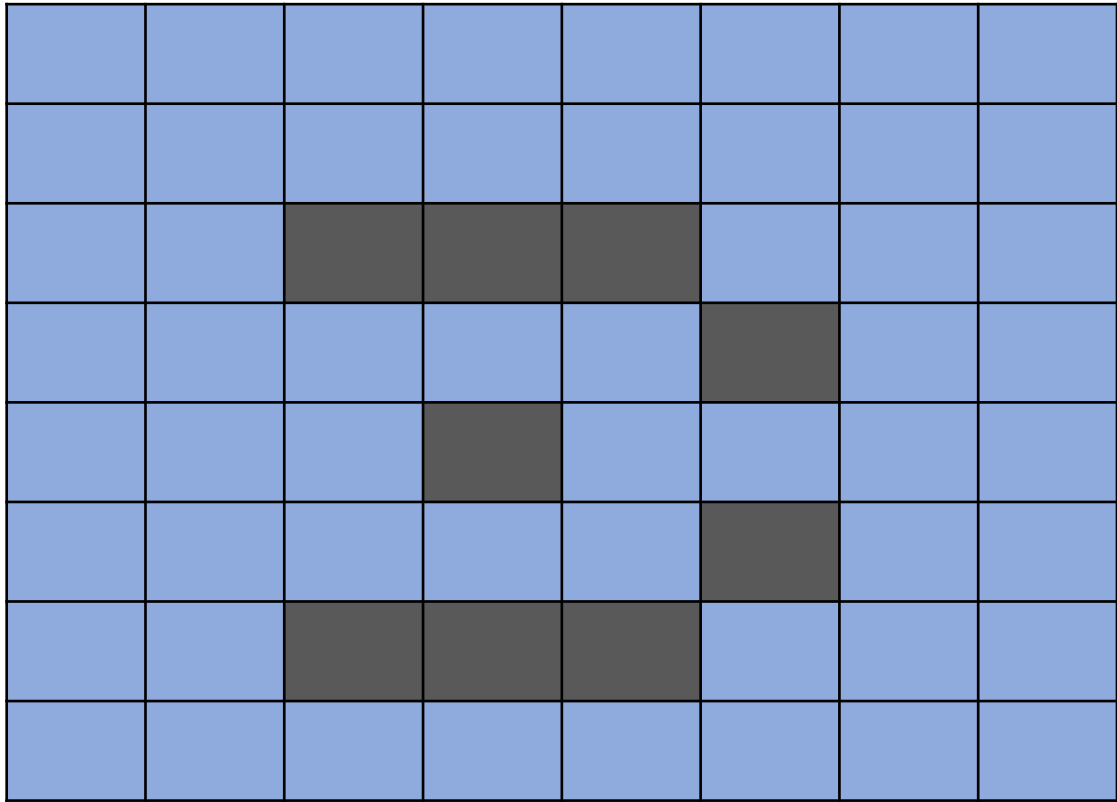


Can you combine your skills about search, agents and environments and constraint satisfaction to solve this problem?

How about for much bigger worlds?

How about an unknown number of pits and wumpi?

Autonomous vacuum cleaner



- Possible moves are (go straight, turn left, turn right)
- Should there be a first choice of action as long as nothing else is known that makes this action unreasonable? (e.g. go straight?)
- What search algorithm would that match?
- What does the vacuum cleaner know about the environment initially?
 - It knows that obstacles exist but not where they are
 - Walls are treated in the same way as obstacles
- The goal is to clean the whole area
- Can you implement this vacuum cleaner using your knowledge about agents, environments, constraints and search?

Machine learning

Machine Learning vs Data Mining

- Machine learning
 - The algorithms (that can be used for data mining)
 - Development, refinement, adaptation of ML algorithms
- Data mining
 - The application of a machine learning algorithms with a specific purpose
 - Data mining...is the process of discovering valuable and hidden knowledge in multidimensional data sets that you never knew existed
 - is a largely automated process
 - Due to machine learning algorithms
 - Encompasses knowledge discovery (description) and prediction

Mitchell's definition of learning (1997)

*“A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”*

Hand et al.'s definition of data mining (2001)

“... the analysis of ... observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.”

Machine Learning

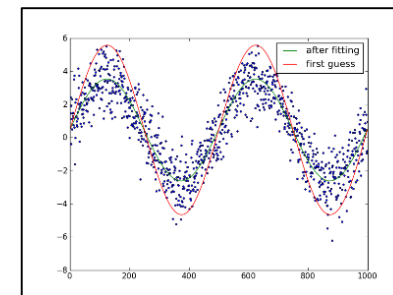
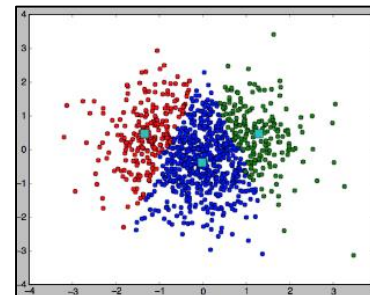
- In a conventional programmed an algorithm gives a step by step description of the what has to be done
 - All possible alternatives need to be considered (e.g. if-then-else, case)

- Machine Learning

- Algorithms that figures out the sequence of steps to take by itself



- Find patterns, trends in data based on statistics



Types of Learning

- **Supervised learning**

- Learning from examples
 - E.g. learn to distinguish between cats and dogs by looking at many cat and dog pictures that are labeled until you can identify cats and dogs on unseen pictures good enough

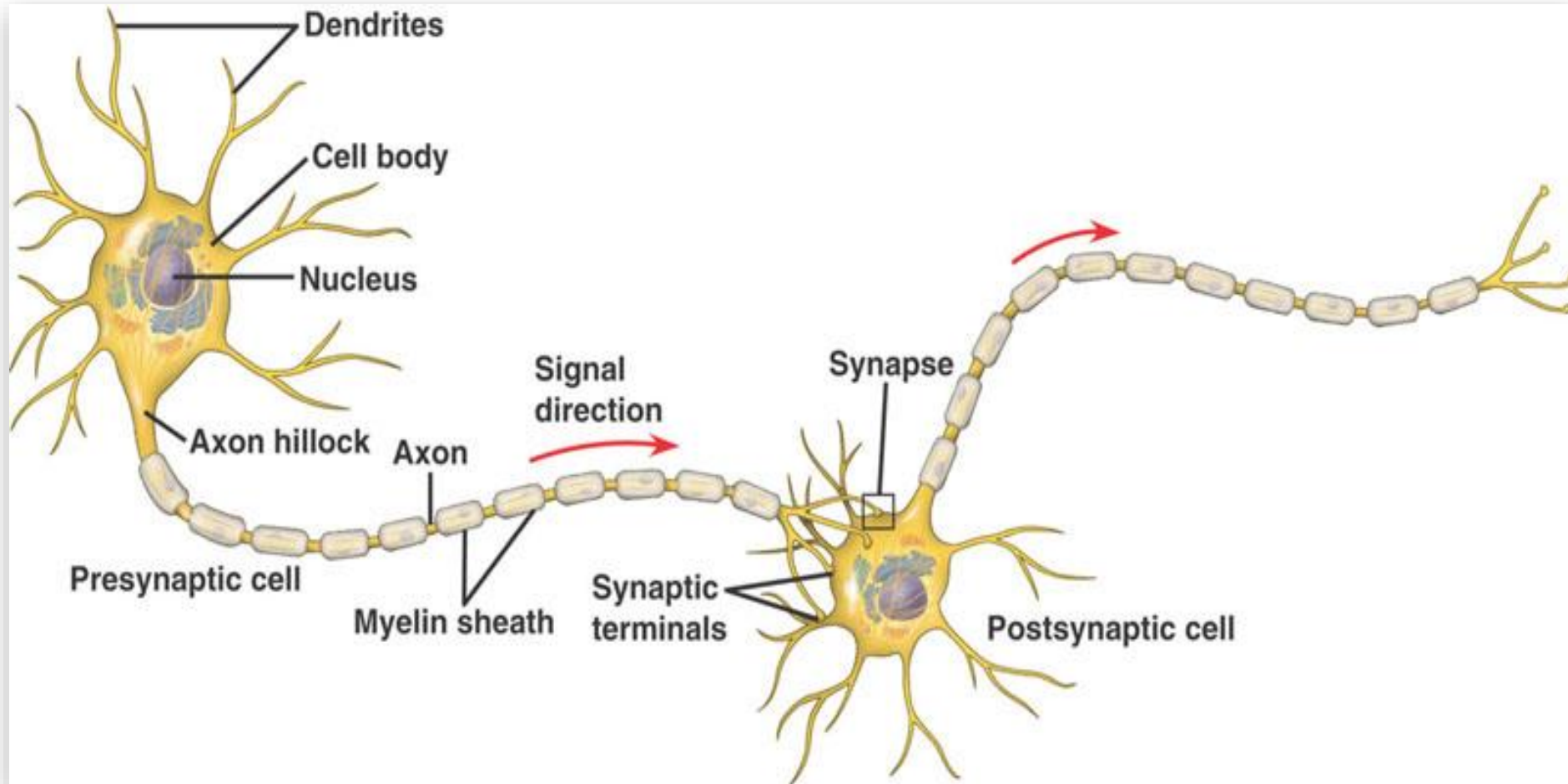
- **Unsupervised learning**

- Figuring out the differences themselves
 - E.g. knowing there are two groups to divide the pictures into, comparing the pictures and find the most important combination of variables that are different and hence divide the pictures into two groups.
 - You don't know what the agent learns.
 - It could be cats vs dogs. It could be sunny pictures vs rainy pictures regardless of the animal in the pictures.

- **Reinforcement learning**

- Figuring out the right action (sequence of actions) to do in order to achieve the goal of getting the highest reward
- The agent chooses the action(s) to do it by itself
- It needs an evaluation function to decide if the action(s) lead to something good or not so good and presenting it with the reward

Neural network



Perceptron (artificial neuron)

