Artificial Intelligence
Sessions 3-4-5

Nicolas Durand

ENAC

Game Theory

- A study of strategic decision making

Game Theory

- A study of strategic decision making
- Used in logic, economics, political science, psychology...

Game Theory

- A study of strategic decision making
- Used in logic, economics, political science, psychology...
- First addressed zero-sum games

Game Theory

- A study of strategic decision making
- Used in logic, economics, political science, psychology...
- First addressed zero-sum games
- John Von Neuman proof of existence mixed strategy equilbria in 2 person zero-sum games.

Game Theory

- A study of strategic decision making
- Used in logic, economics, political science, psychology...
- First addressed zero-sum games
- John Von Neuman proof of existence mixed strategy equilbria in 2 person zero-sum games.
- Minimax theorem (1928)

Minimax Theorem (Neuman 1928)

- Two-person, zero-sum game with finitely many strategies

Minimax Theorem (Neuman 1928)

- Two-person, zero-sum game with finitely many strategies
- There exists a value G and a mixed strategy such that

Minimax Theorem (Neuman 1928)

- Two-person, zero-sum game with finitely many strategies
- There exists a value G and a mixed strategy such that
- Given player 2's strategy, the best payoff for player 1 is G

Minimax Theorem (Neuman 1928)

- Two-person, zero-sum game with finitely many strategies
- There exists a value G and a mixed strategy such that
- Given player 2's strategy, the best payoff for player 1 is G
- Given player 1's strategy, the best payoff for player 2 is $-G$

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

- Minimax choice for A = A2

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

- Minimax choice for A = A2
- Minimax choice for B = B2

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

- Minimax choice for A = A2
- Minimax choice for B = B2
- Not stable, $A \rightarrow A2$, $B \rightarrow B1$, $A \rightarrow A1$, $B \rightarrow B2$ ...

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

- Minimax choice for A = A2
- Minimax choice for B = B2
- Not stable, $A \rightarrow A2$, $B \rightarrow B1$, $A \rightarrow A1$, $B \rightarrow B2$ ...
- Dominated choices: $A3 < A1$, $A3 < A2$, $B3 < B2$

Minimax Theorem: Example

Payoff Matrix for A	B chooses B1	B chooses B2	B chooses B3
A chooses A1	(+3,-3)	(-2,+2)	(+2,-2)
A chooses A2	(-1,+1)	(0,0)	(+4,-4)
A chooses A3	(-4,+4)	(-3,+3)	(+1,-1)

- Minimax choice for A = A2
- Minimax choice for B = B2
- Not stable, $A \rightarrow A2$, $B \rightarrow B1$, $A \rightarrow A1$, $B \rightarrow B2$ ...
- Dominated choices: $A3 < A1$, $A3 < A2$, $B3 < B2$
- Which mixed strategy optimizes A's payoff?

Minimax Theorem: Example

- A3 and B3 are not chosen

Minimax Theorem: Example

- A3 and B3 are not chosen
- A chooses A1 with prob a and A2 with prob (1-a)

Minimax Theorem: Example

- A3 and B3 are not chosen
- A chooses A1 with prob a and A2 with prob (1-a)
- B chooses B1 with prob b and B2 with prob (1-b)

Minimax Theorem: Example

- A3 and B3 are not chosen
- A chooses A1 with prob a and A2 with prob (1-a)
- B chooses B1 with prob b and B2 with prob (1-b)
- Expected payoff is $6ab - 2a - b$

Minimax Theorem: Example

- A3 and B3 are not chosen
- A chooses A1 with prob a and A2 with prob (1-a)
- B chooses B1 with prob b and B2 with prob (1-b)
- Expected payoff is $6ab - 2a - b$
- Optimal for $a = \frac{1}{6}$ and $b = \frac{1}{3}$

Minimax Theorem: Example

- A3 and B3 are not chosen
- A chooses A1 with prob a and A2 with prob (1-a)
- B chooses B1 with prob b and B2 with prob (1-b)
- Expected payoff is $6ab - 2a - b$
- Optimal for $a = \frac{1}{6}$ and $b = \frac{1}{3}$
- Expected payoff for $G_A = \frac{-1}{3}$, $G_B = \frac{1}{3}$

Nash Equilibrium

- non-cooperative game involving two or more players

Nash Equilibrium

- non-cooperative game involving two or more players
- each player knows the equilibrium strategies of other players

Nash Equilibrium

- non-cooperative game involving two or more players
- each player knows the equilibrium strategies of other players
- and has nothing to gain by changing only its own strategy unilaterally

Nash Equilibrium

- non-cooperative game involving two or more players
- each player knows the equilibrium strategies of other players
- and has nothing to gain by changing only its own strategy unilaterally
- payoffs constitute a Nash equilibrium

Nash Equilibrium: Formal definition

- (S, f) game with n players

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles
- $f = (f_1(x), \ldots, f_n(x))$: payoff function for $x \in S$

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles
- $f = (f_1(x), \ldots, f_n(x))$: payoff function for $x \in S$
- x_i be a strategy profile of player i

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles
- $f = (f_1(x), \ldots, f_n(x))$: payoff function for $x \in S$
- x_i be a strategy profile of player i
- x_{-i} be a strategy profile of all players except for player i

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles
- $f = (f_1(x), \ldots, f_n(x))$: payoff function for $x \in S$
- x_i be a strategy profile of player i
- x_{-i} be a strategy profile of all players except for player i
- if i chooses strategy x_i then player i obtains payoff $f_i(x)$

Nash Equilibrium: Formal definition

- (S, f) game with n players
- S_i is the strategy set for player i,
- $S = S_1 \times S_2 \times \cdots \times S_n$ set of strategy profiles
- $f = (f_1(x), \ldots, f_n(x))$: payoff function for $x \in S$
- x_i be a strategy profile of player i
- x_{-i} be a strategy profile of all players except for player i
- if i chooses strategy x_i then player i obtains payoff $f_i(x)$
- $x^* \in S$ is a Nash equilibrium (NE) if
  $\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)$

Nash's Existence Theorem

- Mixed strategies

Nash's Existence Theorem

- Mixed strategies
- Every game with a finite number of players in which each player can choose from finitely many pure strategies has at least one Nash equilibrium.

Nash equilibria in a payoff matrix

- First number maximizes the column
- Second number maximizes the row

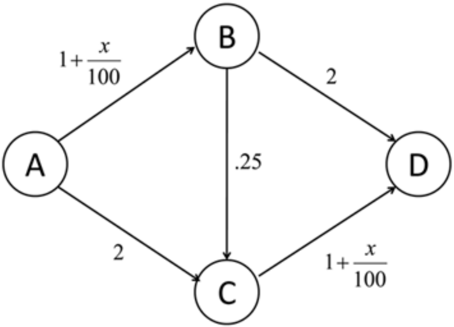
	Option A	Option B	Option C
Option A	(0,0)	**(20,30)**	(5,10)
Option B	**(30,25)**	(0,0)	(5,15)
Option C	(10,5)	(15,5)	**(10,10)**

Prisoner's dilemma

Payoff Matrix	B cooperates	B defects
A cooperates	(-1,-1)	(-10,0)
A defects	(0,-10)	(-5,-5)

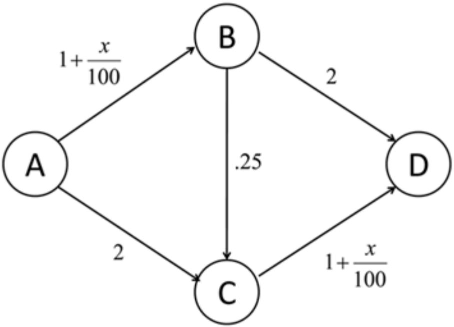
- What is the Nash equilibrium?

Traffic Network



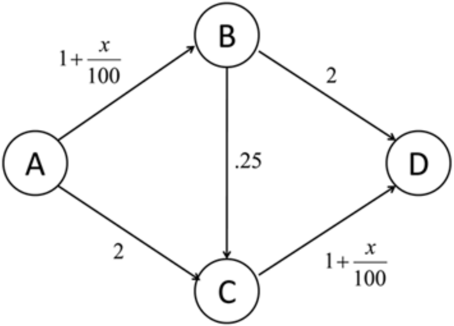
- 100 cars want to go from A to D

Traffic Network



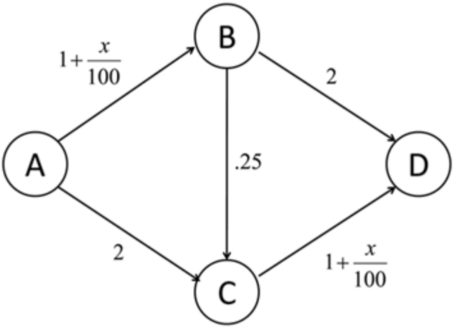
- 100 cars want to go from A to D
- Minimize their time

Traffic Network



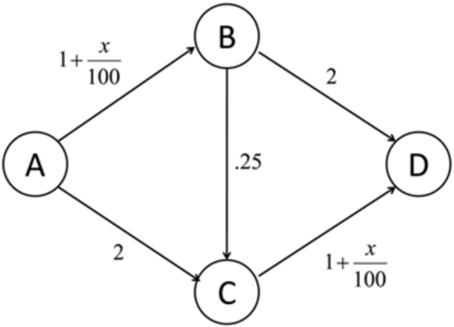
- 100 cars want to go from A to D
- Minimize their time
- Each car can choose either ABD ACD or ABCD

Traffic Network



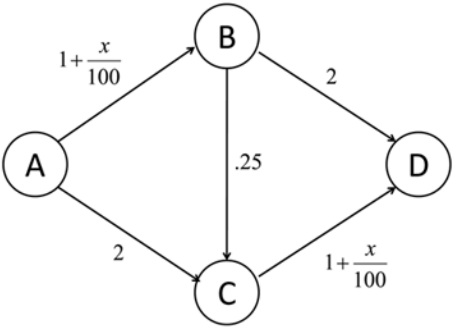
- 100 cars want to go from A to D
- Minimize their time
- Each car can choose either ABD ACD or ABCD
- What is the Nash Equilibrium?

Traffic Network



- 100 cars want to go from A to D
- Minimize their time
- Each car can choose either ABD ACD or ABCD
- What is the Nash Equilibrium?
- What is the optimum?

Traffic Network



- 100 cars want to go from A to D
- Minimize their time
- Each car can choose either ABD ACD or ABCD
- What is the Nash Equilibrium?
- What is the optimum?
- Remove BC: Braess Paradox

Combinatorial Game Theory

- Sequential games with perfect information

Combinatorial Game Theory

- Sequential games with perfect information
- No incomplet information (games of chance, for ex poker)

Combinatorial Game Theory

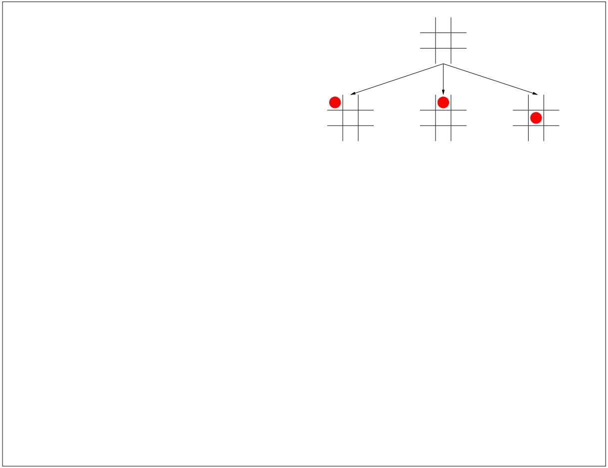
- Sequential games with perfect information
- No incomplet information (games of chance, for ex poker)
- Moves are represented as a game tree

Combinatorial Game Theory

- Sequential games with perfect information
- No incomplet information (games of chance, for ex poker)
- Moves are represented as a game tree
- Some games are solved (for ex tic tac toe, checkers)

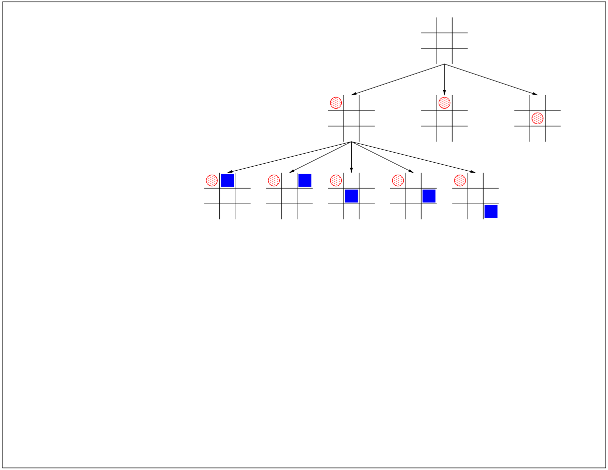
Example on tic-tac-toe

- Alternative choices of two players
- Can be represented in a decision tree



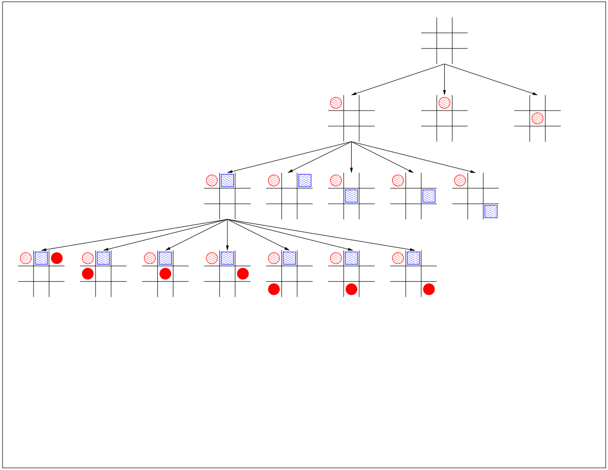
Example on tic-tac-toe

- Alternative choices of two players
- Can be represented in a decision tree



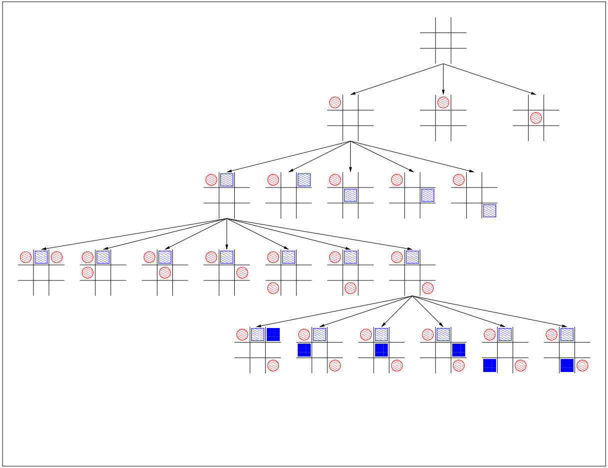
Example on tic-tac-toe

- Alternative choices of two players
- Can be represented in a decision tree



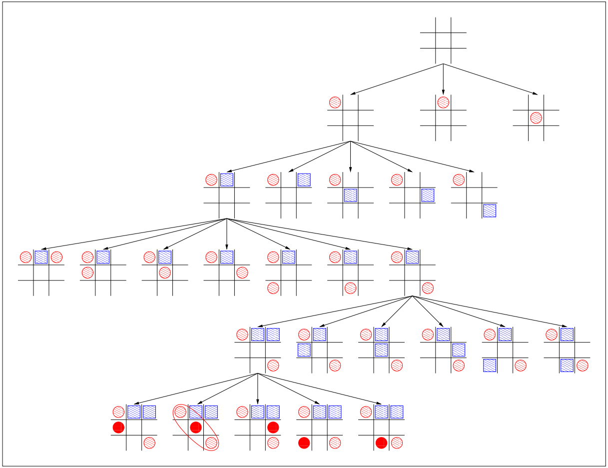
Example on tic-tac-toe

- Alternative choices of two players
- Can be represented in a decision tree

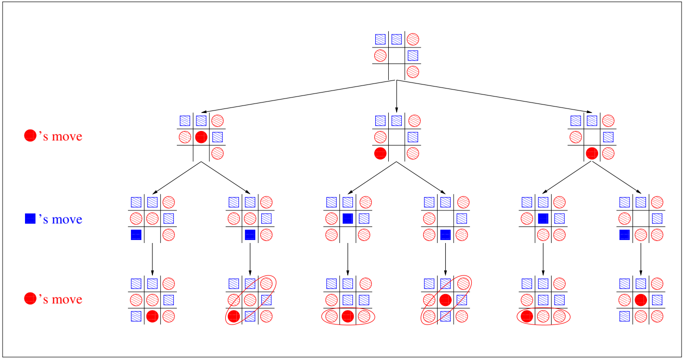


Example on tic-tac-toe

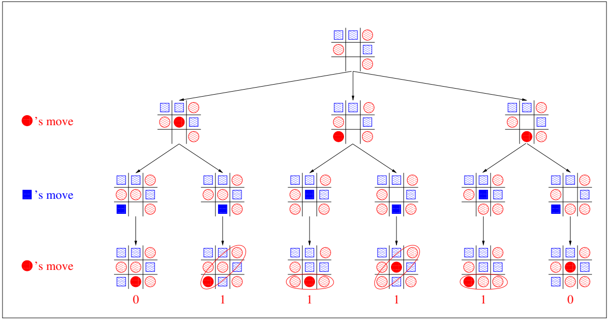
- Alternative choices of two players
- Can be represented in a decision tree



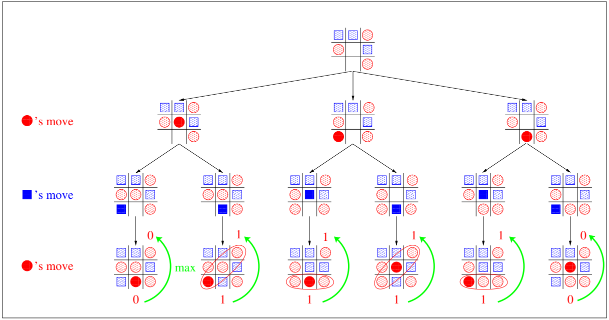
What is the best next choice for the red?



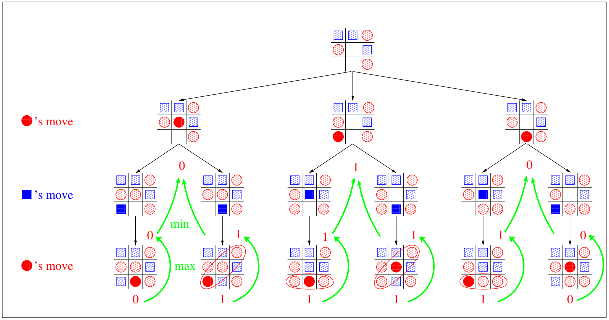
Minimax algorithm on tic-tac-toe



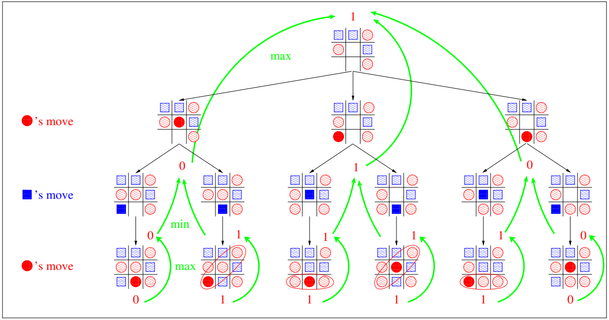
Minimax algorithm on tic-tac-toe



Minimax algorithm on tic-tac-toe



Minimax algorithm on tic-tac-toe



Naive approach

- Exhaustive expand tree generally too big

Naive approach

- Exhaustive expand tree generally too big
- Chess: typical board has around 35 moves

Naive approach

- Exhaustive expand tree generally too big
- Chess: typical board has around 35 moves
- for 40 move game: 35^{40} leaves in the tree

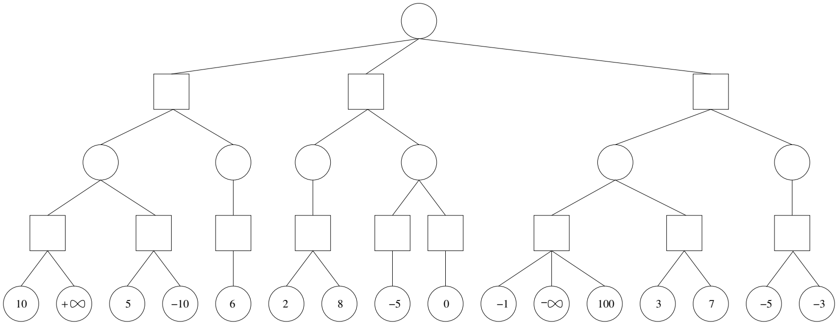
Naive approach

- Exhaustive expand tree generally too big
- Chess: typical board has around 35 moves
- for 40 move game: 35^{40} leaves in the tree
- Choose next move on a path that leads to win

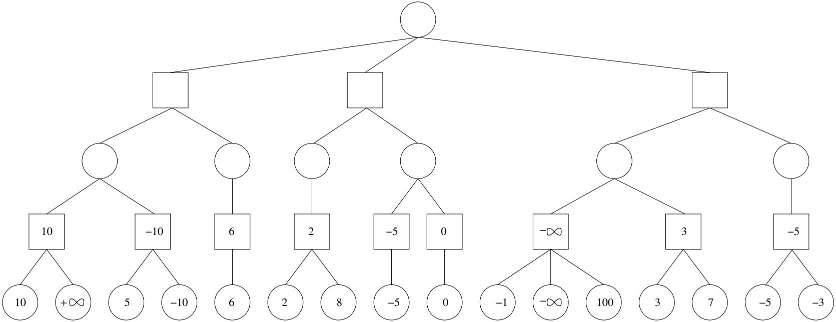
Naive approach

- Exhaustive expand tree generally too big
- Chess: typical board has around 35 moves
- for 40 move game: 35^{40} leaves in the tree
- Choose next move on a path that leads to win
- Your opponent is most likely to choose worst case for you

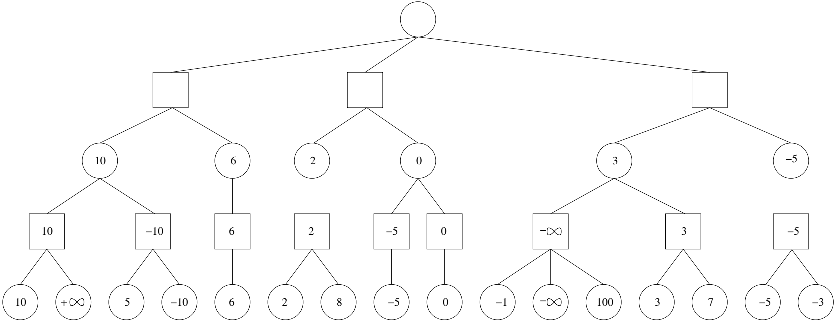
Minimax algorithm



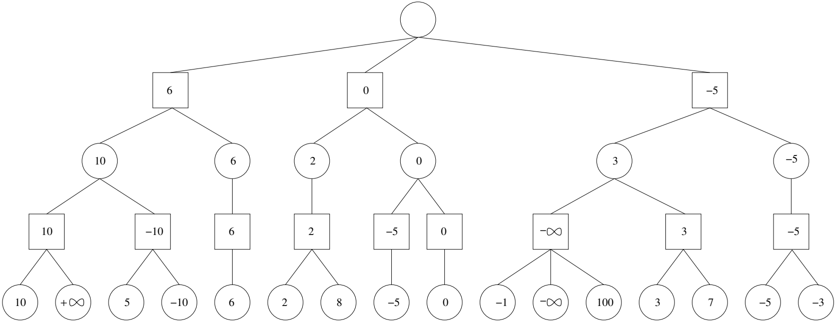
Minimax Algorithm



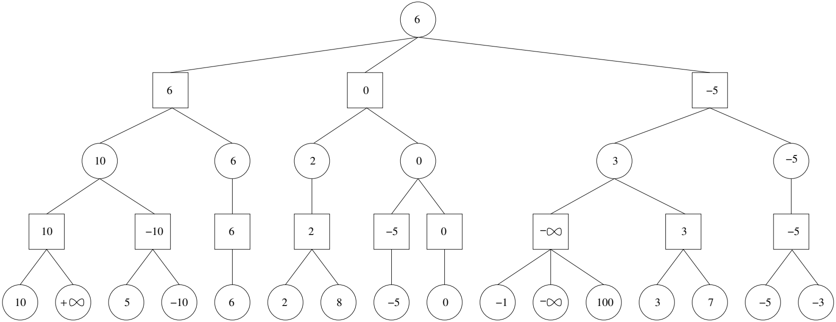
Minimax Algorithm



Minimax Algorithm



Minimax Algorithm



Tree Structure

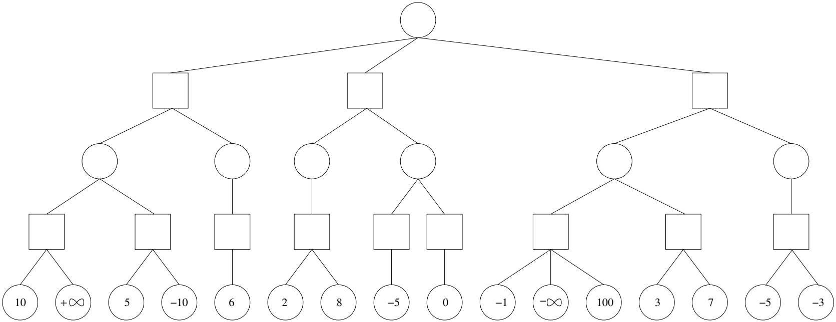
```
type 'a tree =
    Leaf of 'a
  | Nodemin of 'a tree list
  | Nodemax of 'a tree list
```

Minimax Algorithm

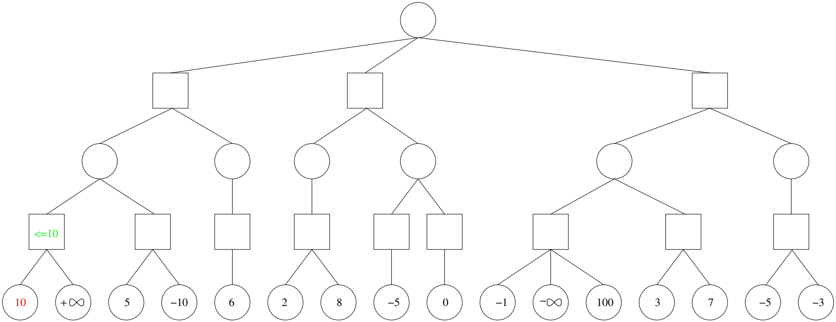
Minimax(t)

match t **with**
Leaf x**:**
    return x
Nodemin l_{min}**:**
    m:=$+\infty$
    for all $t_{min} \in l_{min}$ **do**
        m:=min m Minimax(t_{min})
    end for
    return m
Nodemax l_{max}**:**
    m:=$-\infty$
    for all $t_{max} \in l_{max}$ **do**
        m:=max m Minimax(t_{max})
    end for
    return m
end match

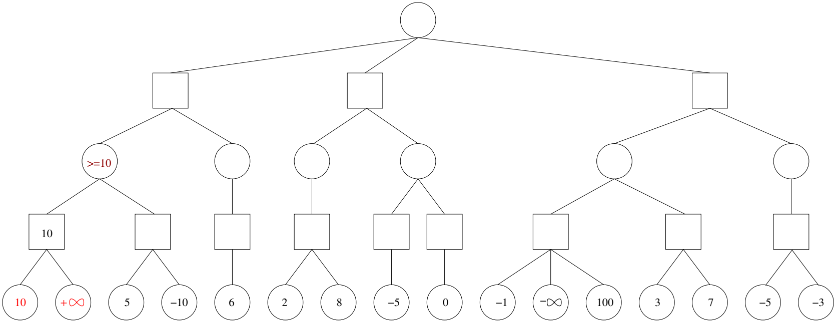
Alpha-Beta Pruning: general idea



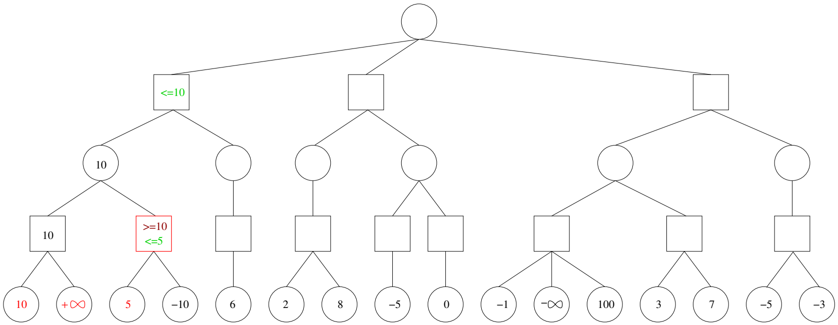
Alpha-Beta Pruning: general idea



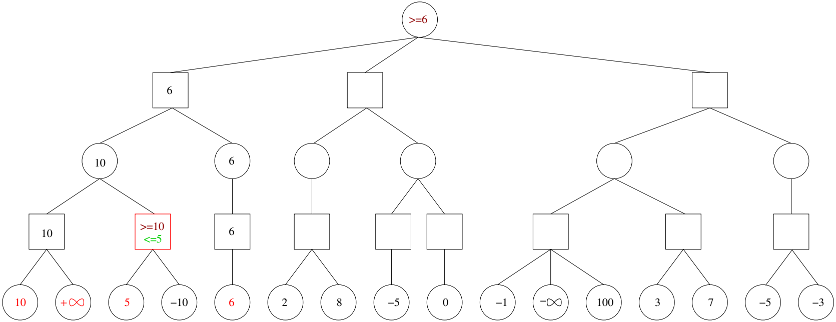
Alpha-Beta Pruning: general idea



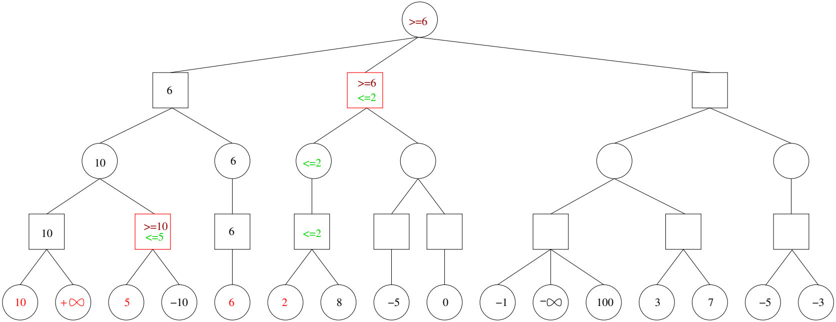
Alpha-Beta Pruning: general idea



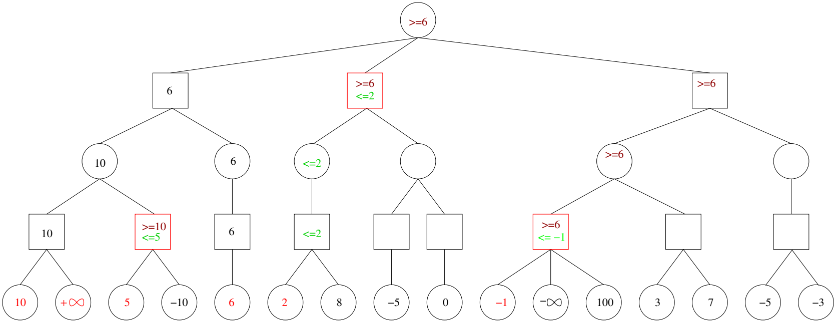
Alpha-Beta Pruning: general idea



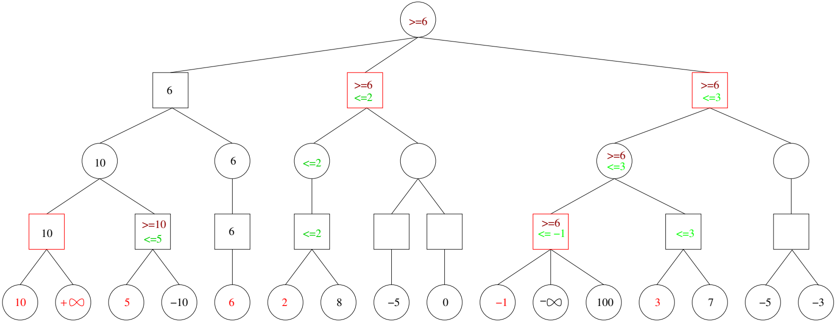
Alpha-Beta Pruning: general idea



Alpha-Beta Pruning: general idea



Alpha-Beta Pruning: general idea



AlphaBeta Pruning: $alphabeta(t, -\infty, \infty)$

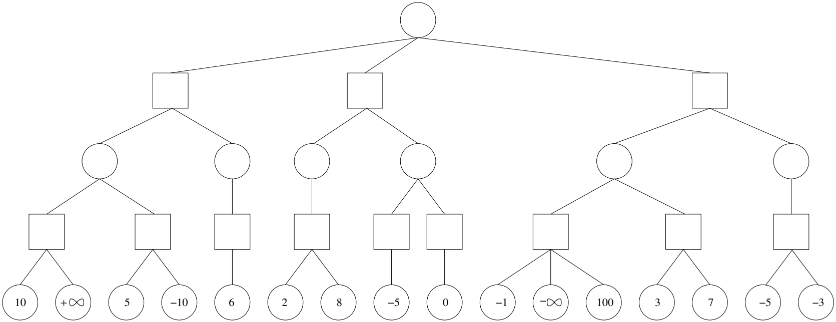
alphabeta(t, a, b)

```
match t with
Leaf s:
    return s
Nodemin l_min:
    s:=b
    for all t_min ∈ l_min do
        s:=min s alphabeta(t_min, a, s)
        if s ≤ a then
            return s
        end if
    end for
Nodemax l_max:
    s:=a
    for all t_max ∈ l_max do
        s:=max s alphabeta(t_max, s, b)
        if s ≥ b then
            return s
        end if
    end for
end match
```

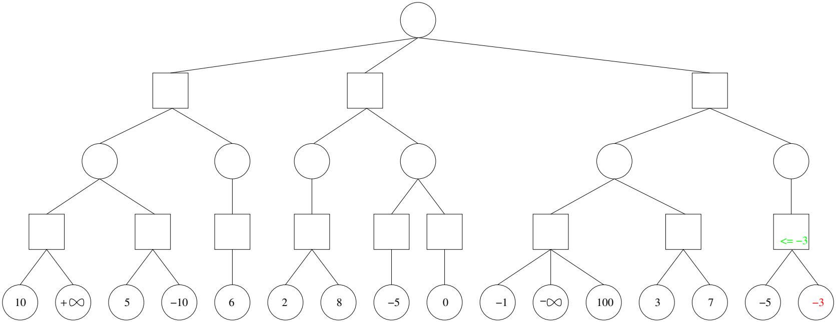
Detailed Example

$Leaf$: **10** $(alpha = -inf, beta = +inf)$
$Node_{min}$: 10 $(alpha = -inf, beta = 10)$
$Leaf$: **+inf** $(alpha = -inf, beta = 10)$
$Node_{min}$: +inf $(alpha = -inf, beta = 10)$
$Node_{max}$: 10 $(alpha = 10, beta = +inf)$
$Leaf$: **5** $(alpha = 10, beta = +inf)$
$Node_{min}$: 5 $(alpha = 10, beta = 5)$ **beta $\leq$ alpha**
$Node_{max}$: 5 $(alpha = 10, beta = +inf)$
$Node_{min}$: 10 $(alpha = -inf, beta = 10)$
$Leaf$: **6** $(alpha = -inf, beta = 10)$
$Node_{min}$: 6 $(alpha = -inf, beta = 6)$
$Node_{max}$: 6 $(alpha = 6, beta = 10)$
$Node_{min}$: 6 $(alpha = -inf, beta = 6)$
$Node_{max}$: 6 $(alpha = 6, beta = +inf)$
$Leaf$: **2** $(alpha = 6, beta = +inf)$
$Node_{min}$: 2 $(alpha = 6, beta = 2)$ **beta $\leq$ alpha**
$Node_{max}$: 2 $(alpha = 6, beta = +inf)$
$Node_{min}$: 6 $(alpha = 6, beta = 6)$ **beta $\leq$ alpha**
$Node_{max}$: 6 $(alpha = 6, beta = +inf)$
$Leaf$: **-1** $(alpha = 6, beta = +inf)$
$Node_{min}$: -1 $(alpha = 6, beta = -1)$ **beta $\leq$ alpha**
$Node_{max}$: -1 $(alpha = 6, beta = +inf)$
$Leaf$: **3** $(alpha = 6, beta = +inf)$
$Node_{min}$: 3 $(alpha = 6, beta = 3)$ **beta $\leq$ alpha**
$Node_{max}$: 3 $(alpha = 6, beta = +inf)$
$Node_{min}$: 6 $(alpha = 6, beta = 6)$ **beta $\leq$ alpha**
$Node_{max}$: 6 $(alpha = 6, beta = +inf)$

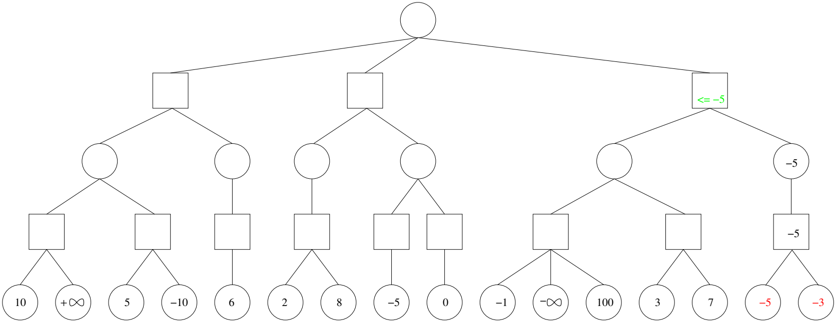
Alpha-Beta Pruning: right to left



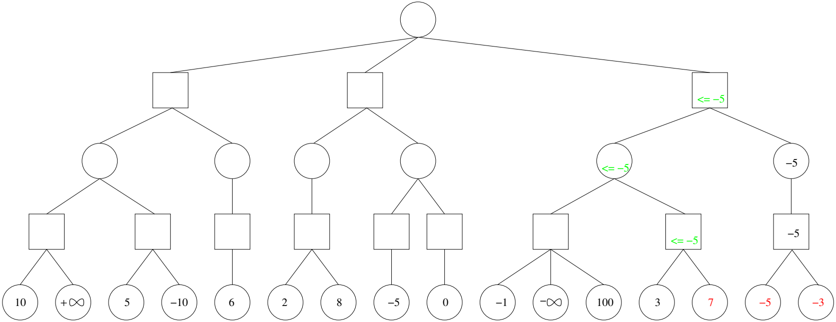
Alpha-Beta Pruning: right to left



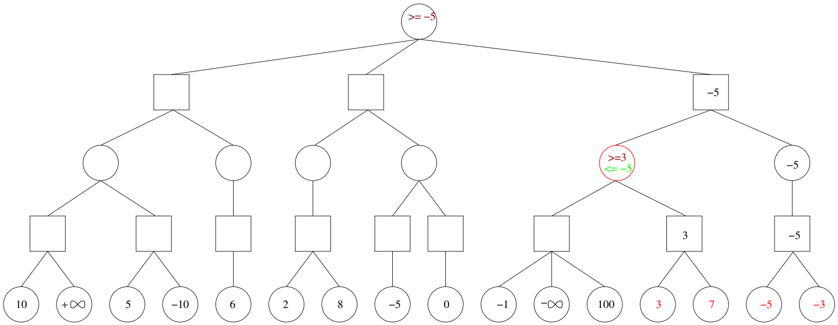
Alpha-Beta Pruning: right to left



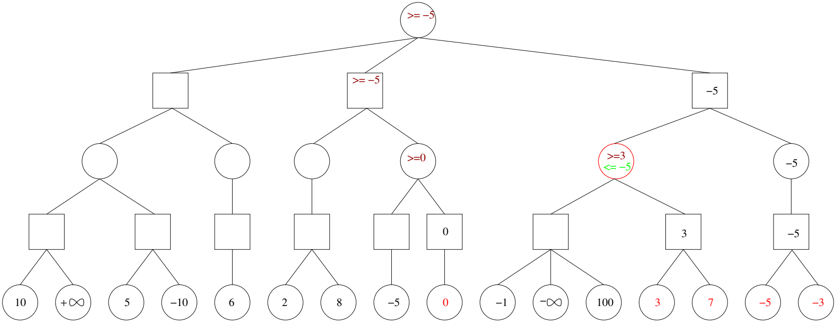
Alpha-Beta Pruning: right to left



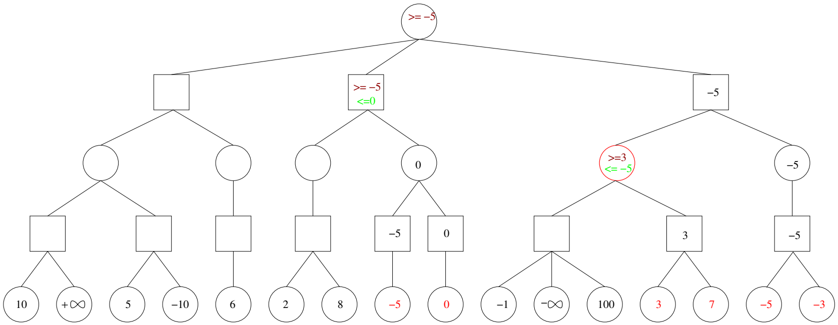
Alpha-Beta Pruning: right to left



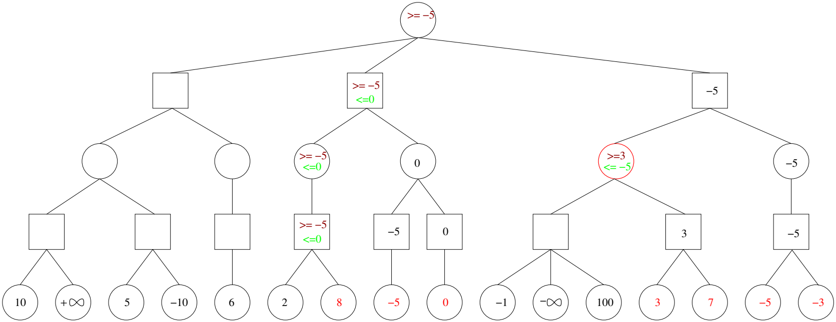
Alpha-Beta Pruning: right to left



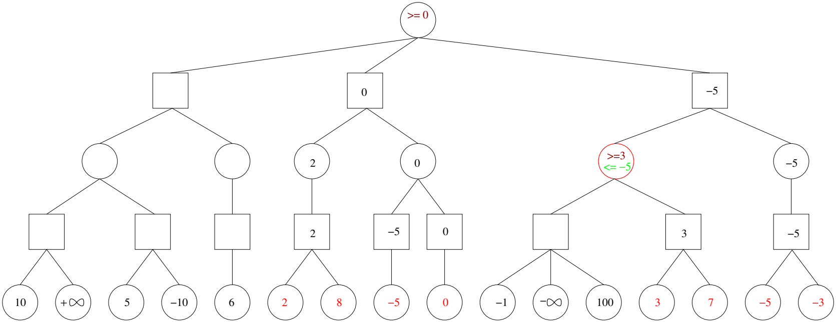
Alpha-Beta Pruning: right to left



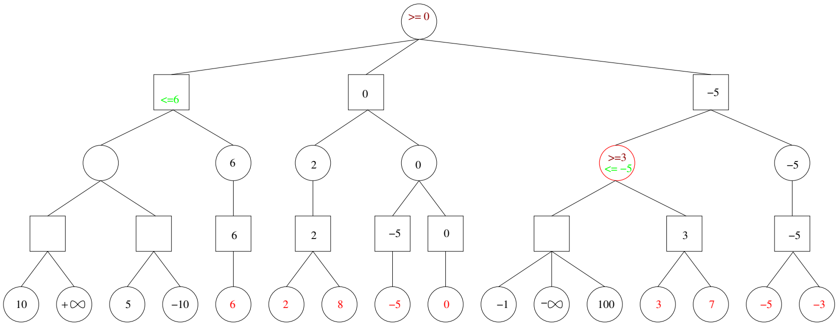
Alpha-Beta Pruning: right to left



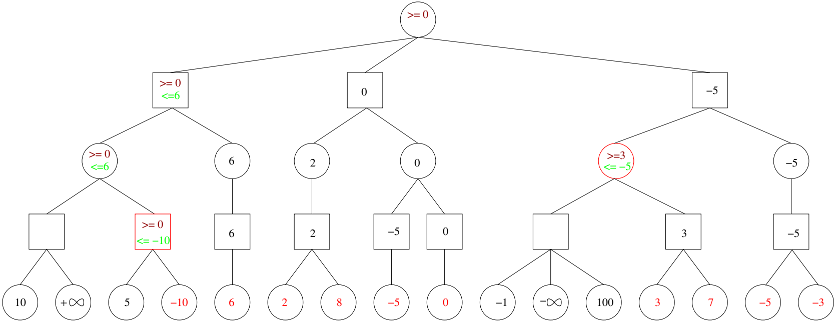
Alpha-Beta Pruning: right to left



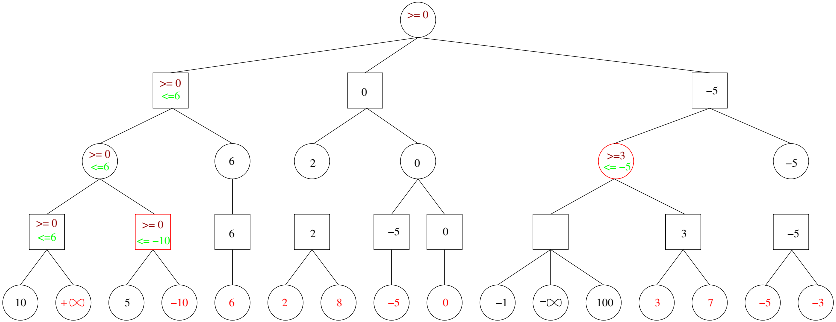
Alpha-Beta Pruning: right to left



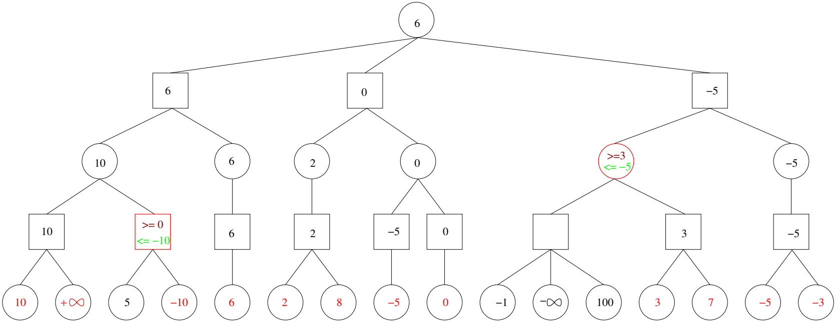
Alpha-Beta Pruning: right to left



Alpha-Beta Pruning: right to left



Alpha-Beta Pruning: right to left



Negamax: $negamax(t, -\infty, \infty)$

negamax(t, a, b)

```
match t with
Leaf s:
    return s
Node l:
    s:=a
    for all t ∈ l do
        s:=max s -negamax(t, −b, −s)
        if s ≥ b then
            return s
        end if
    end for
end match
```

Tic Tac Toe example

- Minimax algorithm: $n = 3 \Rightarrow 46080$ leaves explored

Tic Tac Toe example

- Minimax algorithm: $n = 3 \Rightarrow 46080$ leaves explored
- Alpha-Beta pruning: $n = 3 \Rightarrow 725$ leaves explored, $\frac{1}{63}$ times less.

Tic Tac Toe example

- Minimax algorithm: $n = 3 \Rightarrow 46080$ leaves explored
- Alpha-Beta pruning: $n = 3 \Rightarrow 725$ leaves explored, $\frac{1}{63}$ times less.
- Alpha-Beta pruning: $n = 4 \Rightarrow 18082185$ leaves explored.

Tic Tac Toe example

- Minimax algorithm: $n = 3 \Rightarrow 46080$ leaves explored
- Alpha-Beta pruning: $n = 3 \Rightarrow 725$ leaves explored, $\frac{1}{63}$ times less.
- Alpha-Beta pruning: $n = 4 \Rightarrow 18082185$ leaves explored.
- Minimax algorithm: $n = 4 \Rightarrow$???? leaves explored

Coding issues

- Alpha-Beta works better if the search tree is correctly ordered.

Coding issues

- Alpha-Beta works better if the search tree is correctly ordered.
- Iterative deepening can be used.

Coding issues

- Alpha-Beta works better if the search tree is correctly ordered.
- Iterative deepening can be used.
- Avoid to create the tree and then expore it.

Coding issues

- Alpha-Beta works better if the search tree is correctly ordered.
- Iterative deepening can be used.
- Avoid to create the tree and then expore it.
- Dynamically create the tree in order to save memory.

Coding issues

- Alpha-Beta works better if the search tree is correctly ordered.
- Iterative deepening can be used.
- Avoid to create the tree and then expore it.
- Dynamically create the tree in order to save memory.
- Transposition tables to avoid repeated evaluations.

Iterative Deepening

- Start with 1-ply Depth.

Iterative Deepening

- Start with 1-ply Depth.
- Continue with k-ply Depth (k++) until we run out of time.

Iterative Deepening

- Start with 1-ply Depth.
- Continue with k-ply Depth (k++) until we run out of time.
- Due to exponential nature of game tree search the D-1 iterations are only a fraction of D-ply search.

Iterative Deepening

- Start with 1-ply Depth.
- Continue with k-ply Depth (k++) until we run out of time.
- Due to exponential nature of game tree search the D-1 iterations are only a fraction of D-ply search.
- Use k-1 iteration to order the new iteration.

Iterative Deepening

- Start with 1-ply Depth.
- Continue with k-ply Depth (k++) until we run out of time.
- Due to exponential nature of game tree search the D-1 iterations are only a fraction of D-ply search.
- Use k-1 iteration to order the new iteration.
- Improved move order generally catches up time lost with previous deepening searches

Transposition tables

- Interior nodes of game trees are not always distinct.

Transposition tables

- Interior nodes of game trees are not always distinct.
- Same positions may be re-visited multiple times.

Transposition tables

- Interior nodes of game trees are not always distinct.
- Same positions may be re-visited multiple times.
- Record the information of each sub-tree in a transposition table

Transposition tables

- Interior nodes of game trees are not always distinct.
- Same positions may be re-visited multiple times.
- Record the information of each sub-tree in a transposition table
- The information saved includes the score, best move, search depth, whether the value is an upper bound, lower bound or exact value.

Transposition tables

- Interior nodes of game trees are not always distinct.
- Same positions may be re-visited multiple times.
- Record the information of each sub-tree in a transposition table
- The information saved includes the score, best move, search depth, whether the value is an upper bound, lower bound or exact value.
- If the previous search is at least the desired depth then use it.

Transposition tables

- Interior nodes of game trees are not always distinct.
- Same positions may be re-visited multiple times.
- Record the information of each sub-tree in a transposition table
- The information saved includes the score, best move, search depth, whether the value is an upper bound, lower bound or exact value.
- If the previous search is at least the desired depth then use it.
- If the previous search is less than the desired depth then try first the previous search.

NegaScout Algorithm

- In alpha-beta, the narrower the search window, the higher chance to get a cutoff.

NegaScout Algorithm

- In alpha-beta, the narrower the search window, the higher chance to get a cutoff.
- Idea: use alpha=beta-1

NegaScout Algorithm

- In alpha-beta, the narrower the search window, the higher chance to get a cutoff.
- Idea: use alpha=beta-1
- Start after the first move.

NegaScout Algorithm

- In alpha-beta, the narrower the search window, the higher chance to get a cutoff.
- Idea: use alpha=beta-1
- Start after the first move.
- If the subtree search results to a cutoff, you may save time.

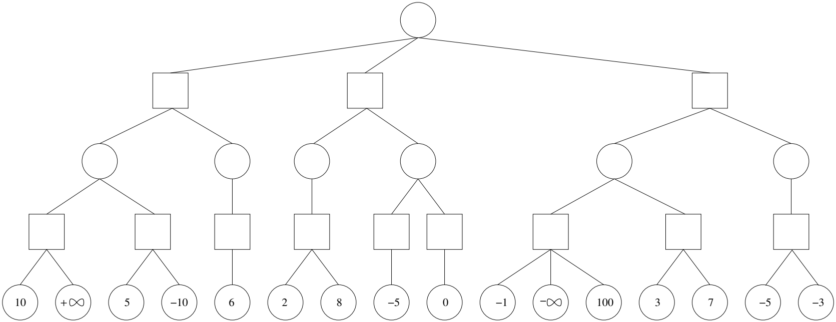
NegaScout Algorithm

- In alpha-beta, the narrower the search window, the higher chance to get a cutoff.
- Idea: use alpha=beta-1
- Start after the first move.
- If the subtree search results to a cutoff, you may save time.
- Sometimes you need to explore the subtree because it leads to a better move.

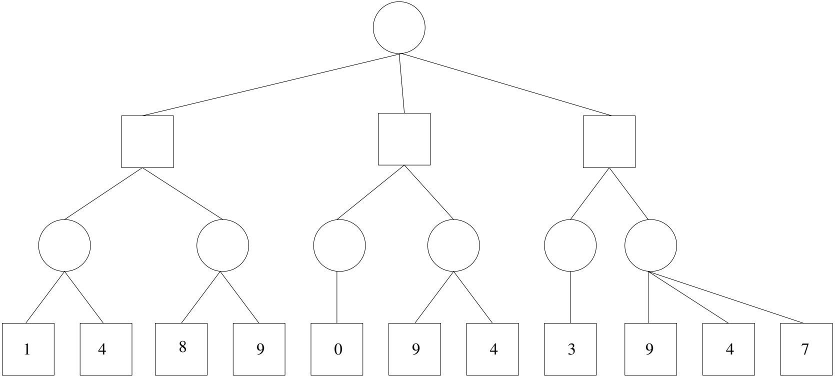
The competition Game

- Players choose a number from 1 to 10
- Everybody wins the smallest number
- If one gives a larger number, he/she gives 1 point to people who chose the smallest number and looses 2 points.

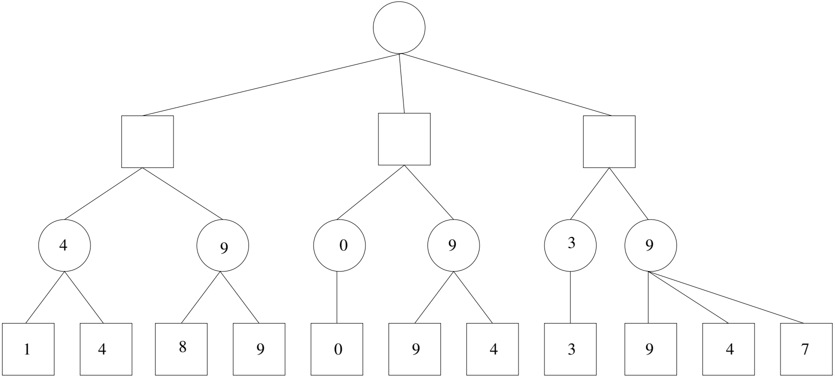
Exercice: alpha-beta pruning



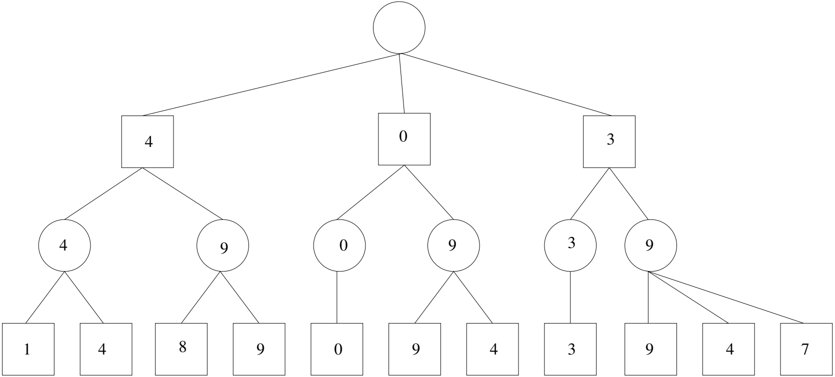
Exercice: alpha-beta pruning



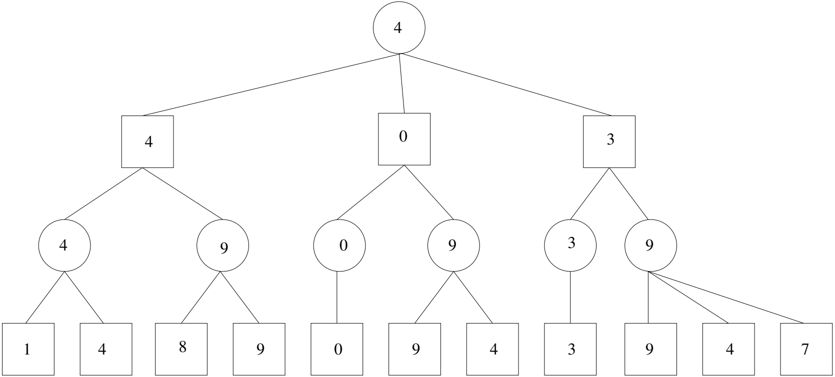
Exercice: alpha-beta pruning



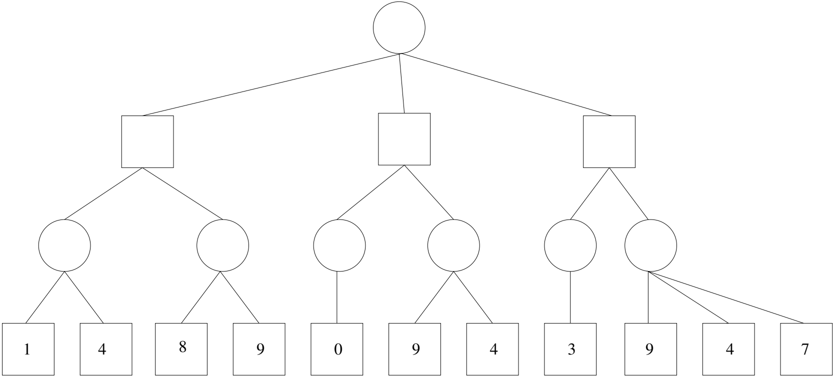
Exercice: alpha-beta pruning



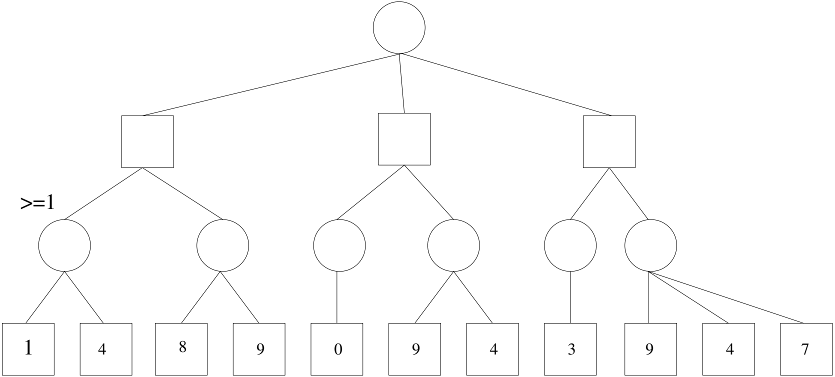
Exercice: alpha-beta pruning



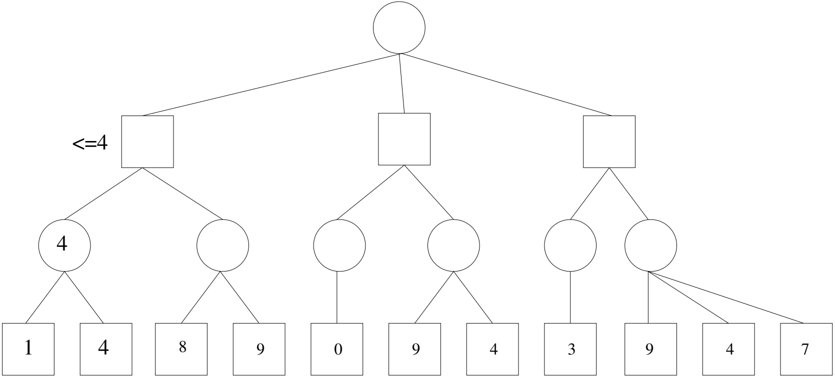
Exercice: alpha-beta pruning



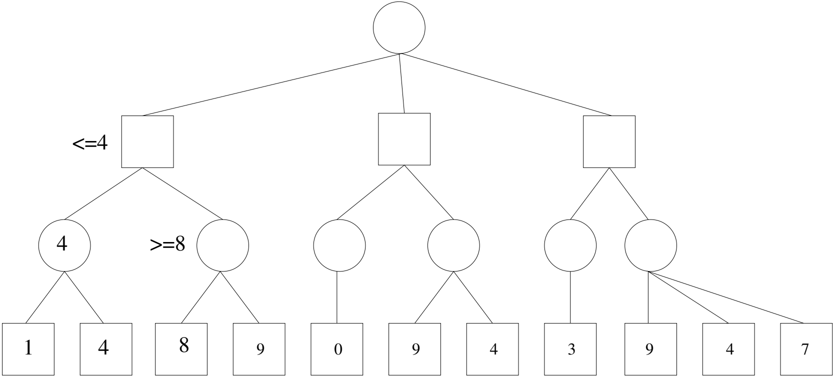
Exercice: alpha-beta pruning



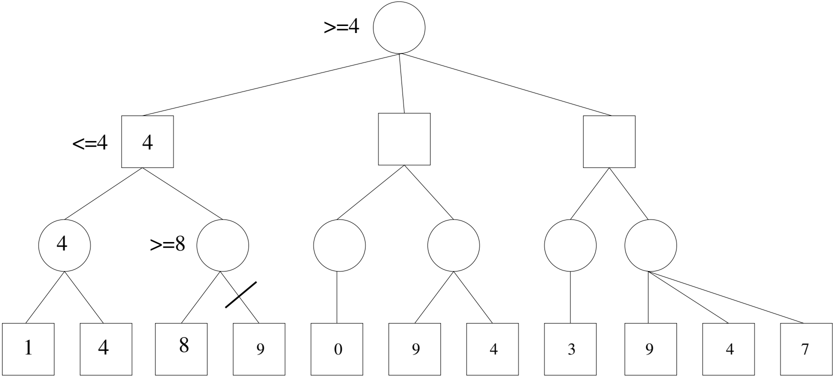
Exercice: alpha-beta pruning



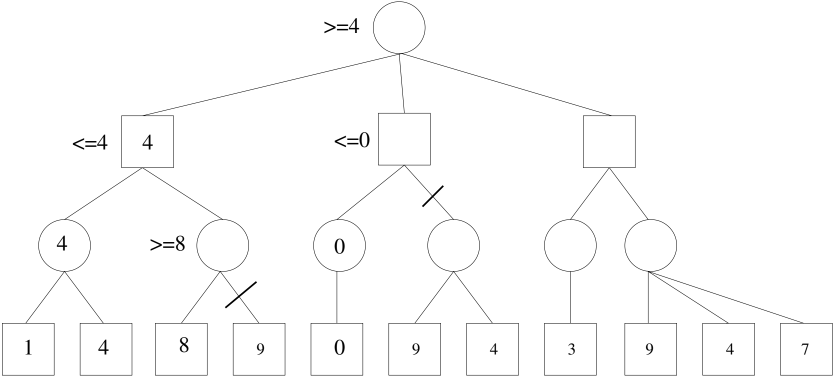
Exercice: alpha-beta pruning



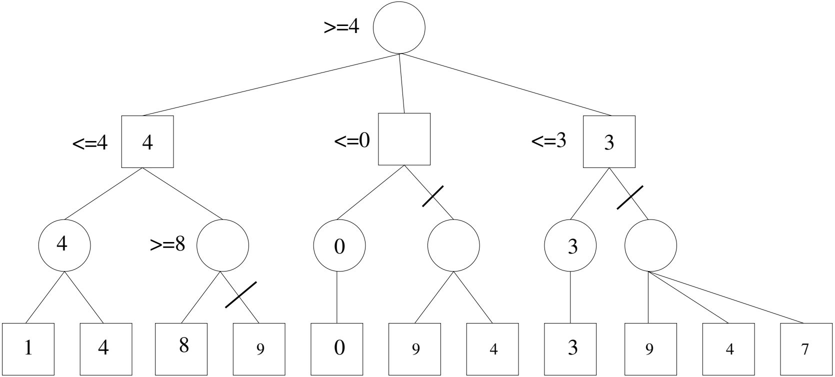
Exercice: alpha-beta pruning



Exercice: alpha-beta pruning



Exercice: alpha-beta pruning



Weak Methods in AI

- Little knowledge of the environment
- Based on Tree or Graph search
- Can be adapted to many domains
- Generally less efficient than "ad hoc" methods

Missionaries and Cannibals problem

- Three missionaries and three cannibals must cross a river using a boat.
- The boat can carry at most two people.
- Missionaries cannot be outnumbered by cannibals.
- The boat cannot cross the river by itself.
- Initial state (missionaries:3,canibals:3,boat:left): (3,3,l)
- Final state (0,0,r)

States, State Variables, State Spaces

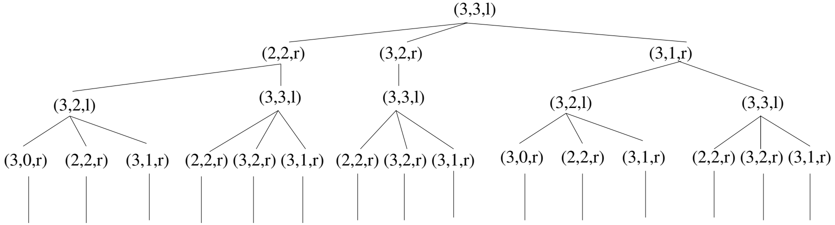
- **State Variables**: Every problem can be modeled by sets of objects that represent the State Variables of the problem.
- **Problem States**: A Problem State is the set of values given to each variable at a fixed time.
- **Space States**: The Space State of a problem is the set of possible states of the problem.

Production System

- **Production System**: Set of Rules used to find the reachable States (starting from a specific State).
- **Deductive Solution**: Start from the Initial State to end up with the Final State (Forward Chaining).
- **Inductive Solution**: Start from the Final State to end up with the Initial State (Backward Chaining).

Trees

- **Trees**: Each State is linked to its predecessor without testing the occurence of same States.

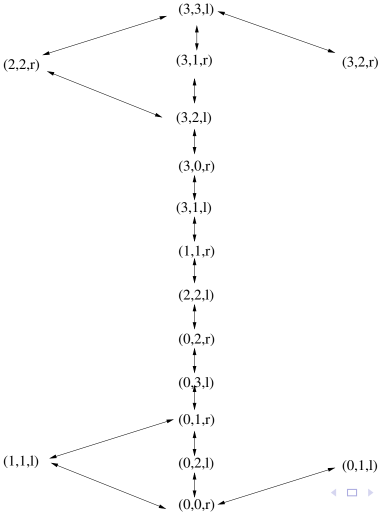


Trees

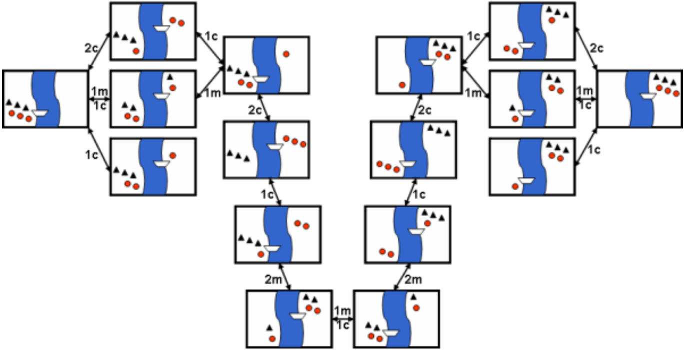
- **Advantage**: Fast (occurence test are time consuming)
- **Drawbacks**:
  - Memory consuming
  - How to avoid endless loops
- Very often used in Game Theory (generally combined with occurence test techniques such as transposition tables).

Graphs

- **Graphs**: When a new state is generated, we check if it has already been generated.



MC problem: graph representation



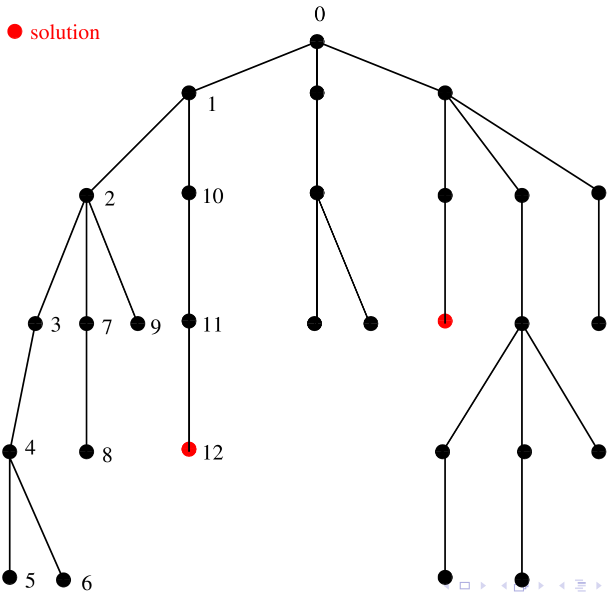
British Museum Algorithm

- Generate only one node using a production rule.
- Iterate the process starting with this new node until a solution is found.
- if a dead-end is reached, restart from the tree root.
- Monkey and Typing Machine Algorithm
- 7000 billions of billions of years to type the title “Notre Dame de Paris” if the monkey types a letter per second.

Depth-first search and Backtracking

- Smarter implementation of the previous algorithm
- After each failure, we come back to the node prior the dead end was observed, and choose a production rule not already used.

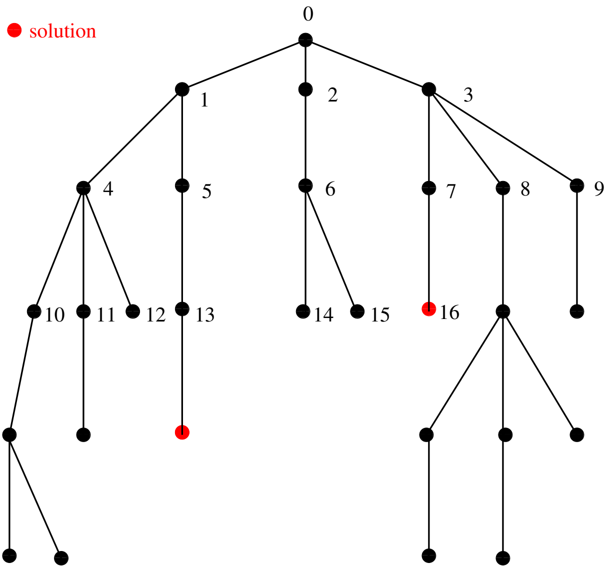
Depht-first search and Backtracking



Depth-first search and Backtracking

- **Advantage**: easy to implemant.
- **Drawbacks**:
  - more effective on graphs than on trees
  - does not take advantage of the problem structure
- Classic Algorithm used by PROLOG language. (iterative depth is an alternative used in Game Theory).

Breadth-first search



Breadth-first search

- **Advantages**:
  - will find the solution, even with infinite cycles.
  - the less deep solution is found
- **Drawbacks**:
  - very high memory cost
  - does not take advantage of the problem structure

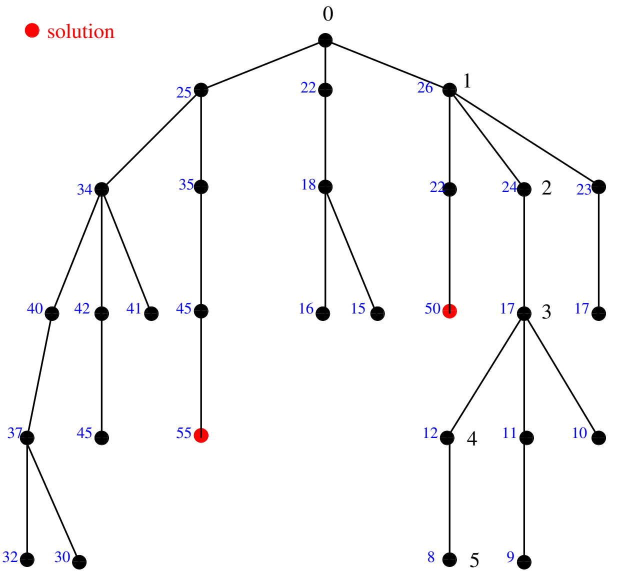
Heuristic Notion

- A **Heuristic** aims at directing the tree search, in order to reduce the problem time resolution
- It uses mecanisms depending on specific knowledge of the problem
- It is often combined to methods that do not guarantee the resolution completeness in order to improve efficiency.

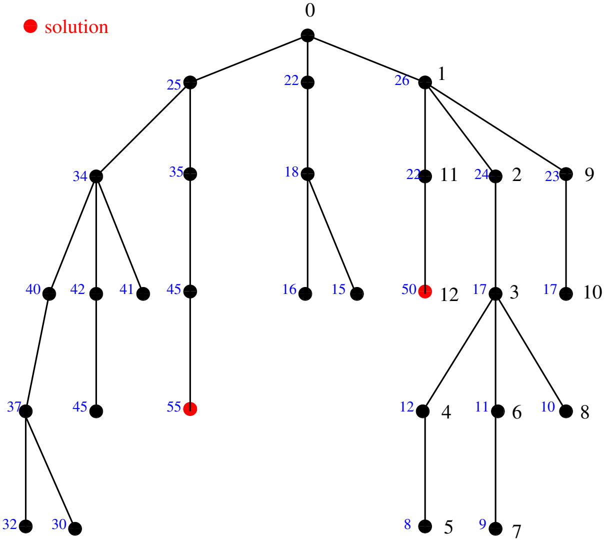
Hill climbing

- Hill climbing is a depth first method for which the node generation rule uses a Heuristic.
- Example for the Traveling Salesman Problem (TSP): choose the closest city not already visited.
- **Advantage**: very fast algorithm
- **Drawback**: does not often find the optimal solution

Hill Climbing



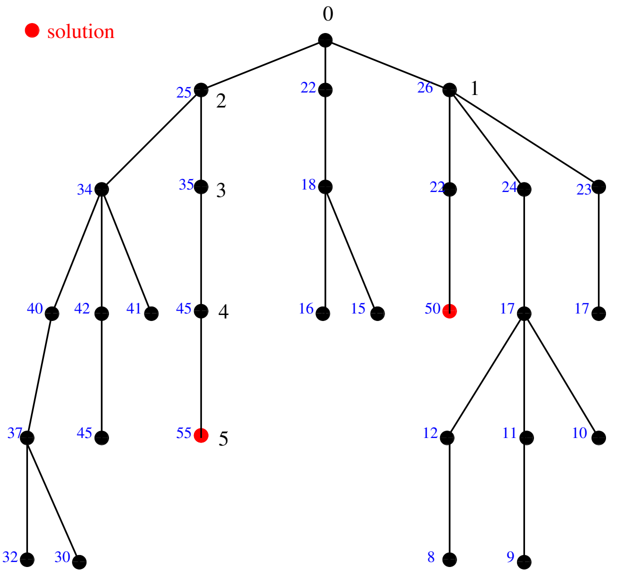
Hill Climbing and Backtracking



Best first search

- Compromise between Depth-first and Breadth-first search.
- At each step the best node (not visited) is developped. All the visited nodes are included to search the next best node

Best search first



A^* Algorithm

- Best-first algorithm
- For additive cost problems
- Classical algorithm in robotics

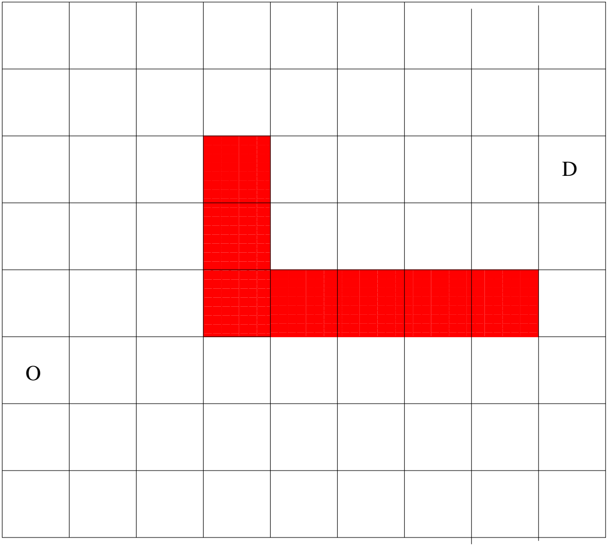
A^* Algorithm

- u_0: initital state
- T: set of terminal states
- $P = p_1, p_2, ..., p_n$: set of production rules
- $h(u)$: heuristic function that estimates the cost from the current state u to the final state.
- $k(u, v)$: cost function between state u and v.
- D: a list of states that have been developped
- G: a list of states that have been generated
- $Father$: a table that gives the preceding state of each state generated

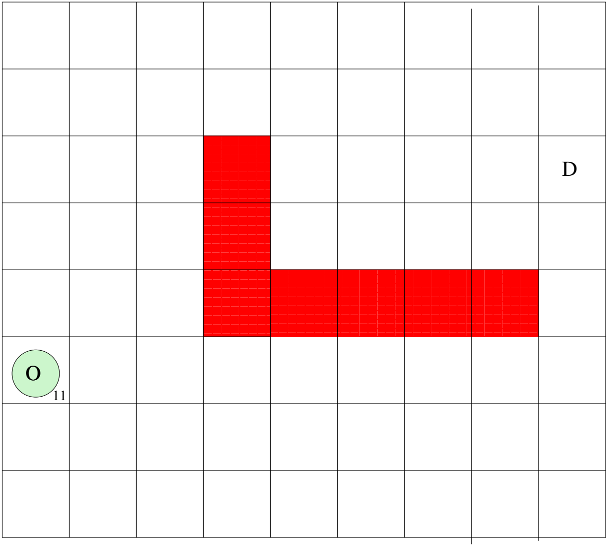
A^* Algorithm

$G = u_0$; $g(u_0) = 0$
while $G \neq \emptyset$ **do**
    $u \leftarrow \mathit{first}(G)$
    $G \leftarrow G - u$
    $D \leftarrow D + u$
    if $u \in T$ **then**
        path from u_0 to final state u
    end if
    for $i = 1$ to n **do**
        $v \leftarrow p_i(u)$
        if $v \notin D + G$ or $g(v) > g(u) + k(u, v)$ **then**
            $g(v) \leftarrow g(u) + k(u, v)$
            $f(v) \leftarrow g(v) + h(v)$
            $\mathit{father}(v) \leftarrow u$
            $G \leftarrow \mathit{insert}_{f(v)}(G, v)$
        end if
    end for
end while

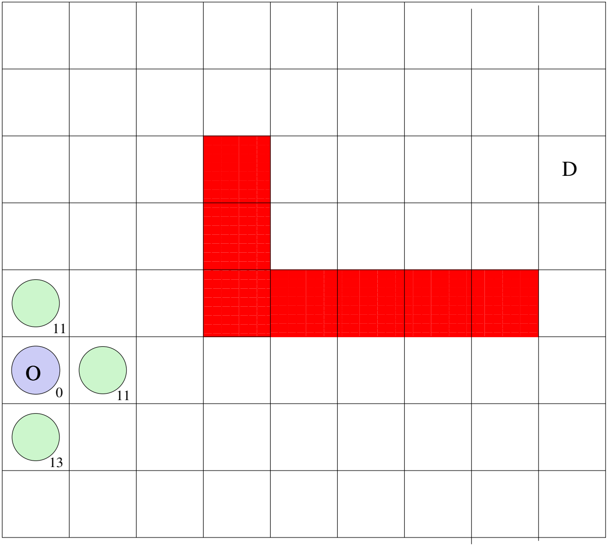
Example



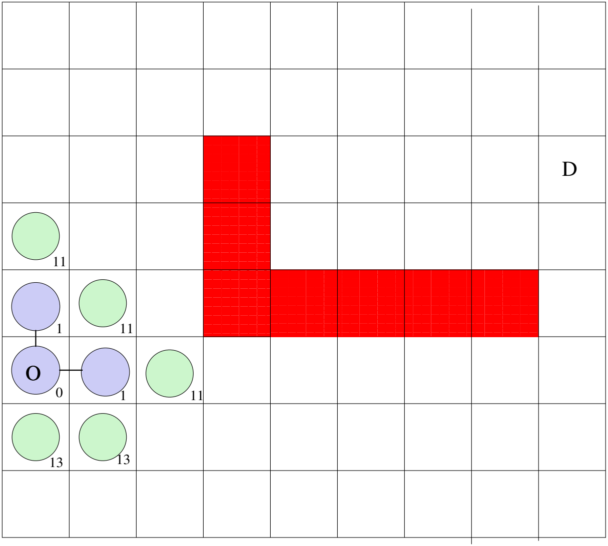
Example



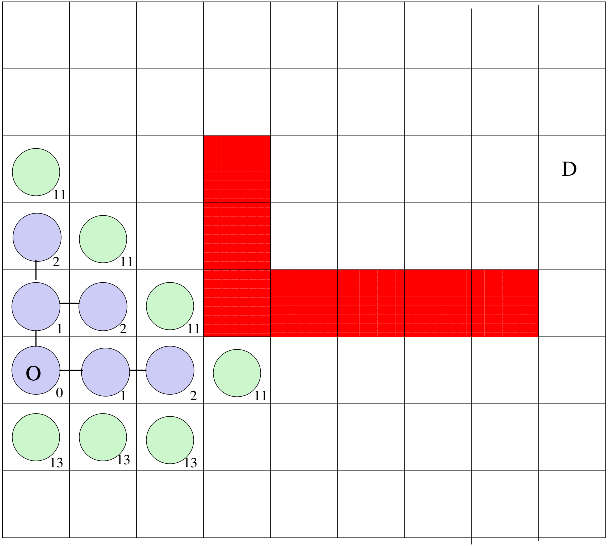
Example



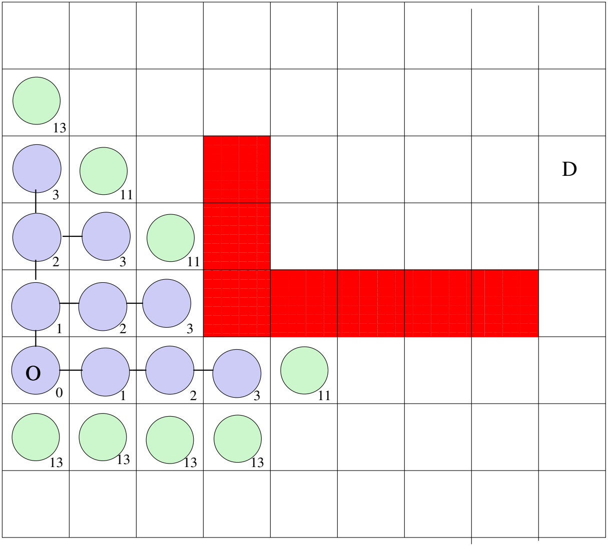
Example



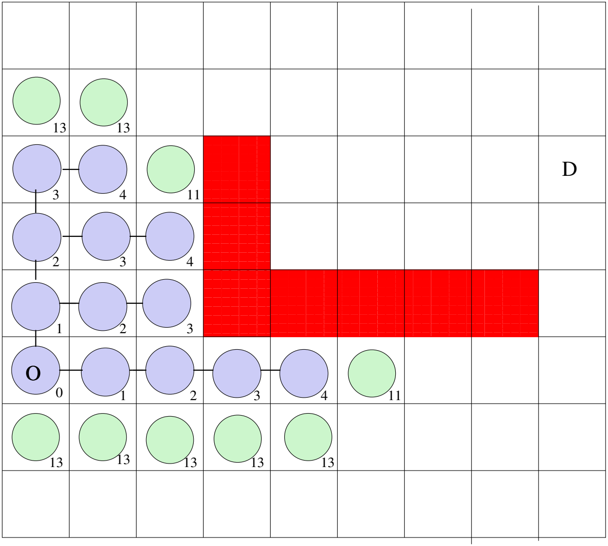
Example



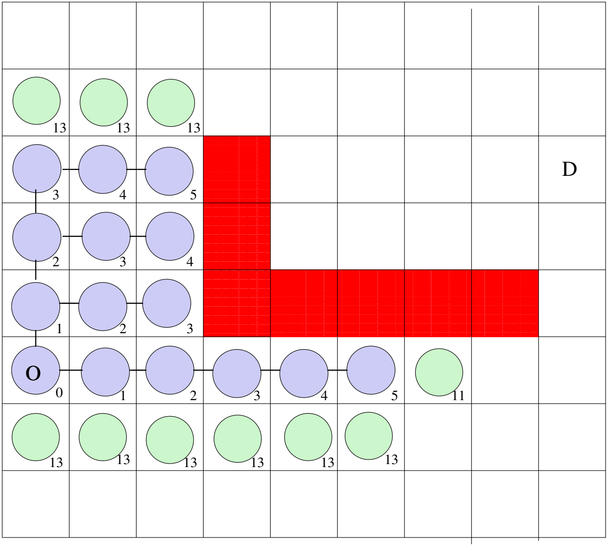
Example



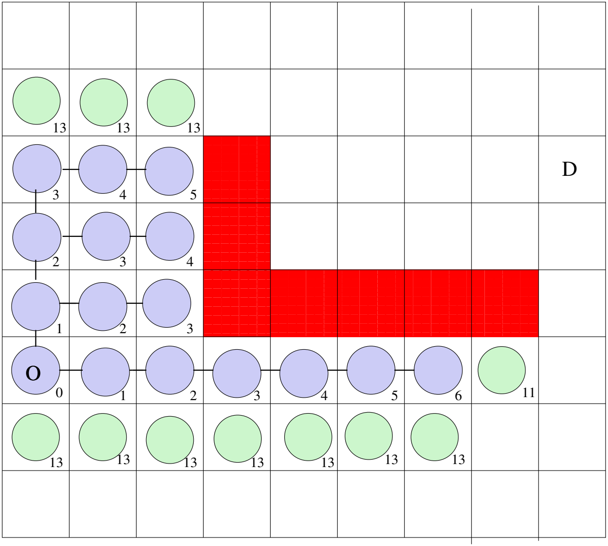
Example



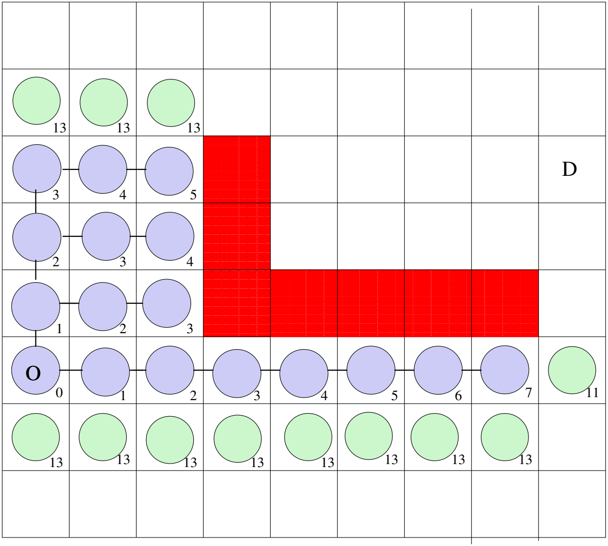
Example



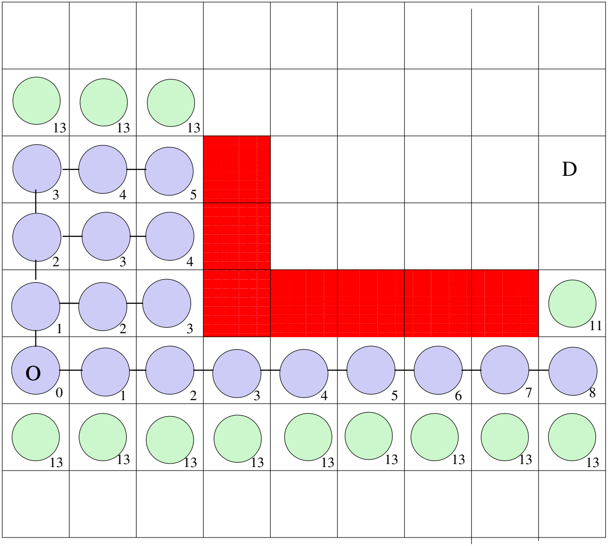
Example



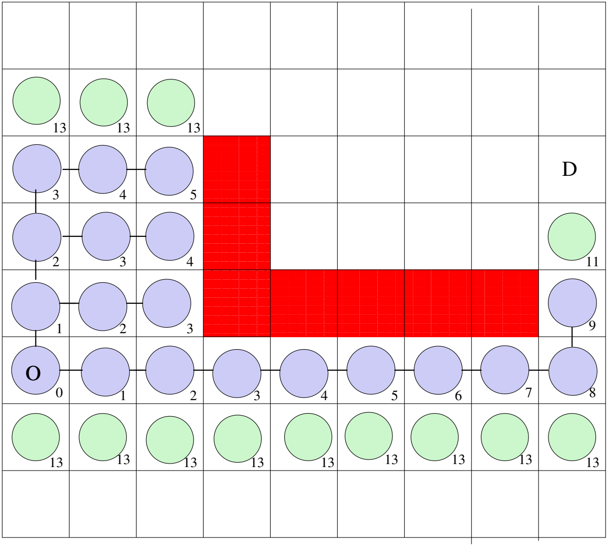
Example



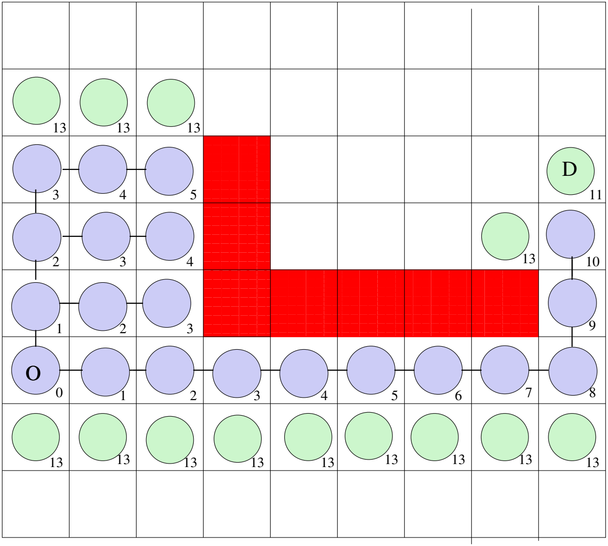
Example



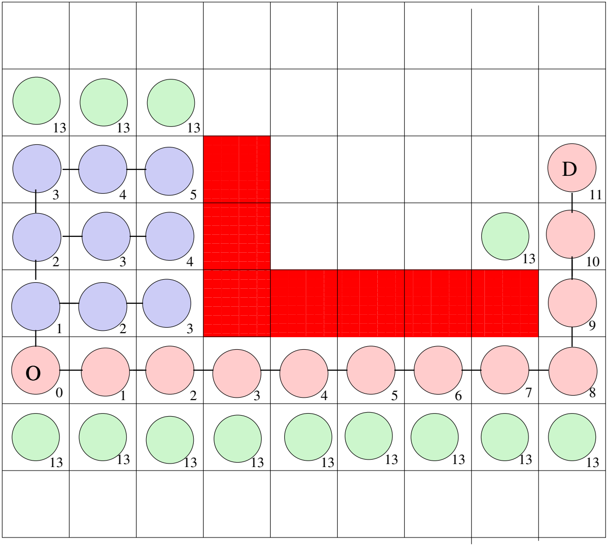
Example



Example


13
13
13
3
4
5
D
11
2
3
4
13
10
1
2
3
9
O
0
1
2
3
4
5
6
7
8
13
13
13
13
13
13
13
13
13

Example


13 13 13
D
3 4 5 11
2 3 4 13 10
1 2 3 9
O
0 1 2 3 4 5 6 7 8
13 13 13 13 13 13 13 13 13

A^* Algorithm

- If $k(u, v) = 0$: we do not care about the distance to the solution.
- If $k(u, v) = 1$: we want to minimize the number of arcs to the solution.
- $h(u)$ estimates the distance from u to the destination.
- $h(u)$ estimates the minimum distance among all the possible paths from u to the destination.
- Let us define $h^*(u)$ as this minimum distance.

H heuristic

- **Perfect heuristic**:
  For all (u, v), $h(u) = h(v) \Rightarrow h^*(u) = h^*(v)$
- **Nearly perfect heuristic**:
  For all (u, v), $h(u) < h(v) \Rightarrow h^*(u) < h^*(v)$
- **Consistant heuristic**:
  If u generates v then $h(u) - h(v) \leq k(u, v)$
- **Admissible heuristic**:
  For all u, $h(u) \leq h^*(u)$

Robot Motion Planning

- Find the shortest path in a labyrinth
- $k(u, v)$: effective distance between u and v.
- $h(u)$: distance as the crow flies to destination.

H heuristic

- If h is perfect, the algorithm converges directly to the optimum solution
- if h is admissible the optimum is always found
- if h is consistant, then h is admissible and for every developped state u, g(u) is the minimum distance leading to u.

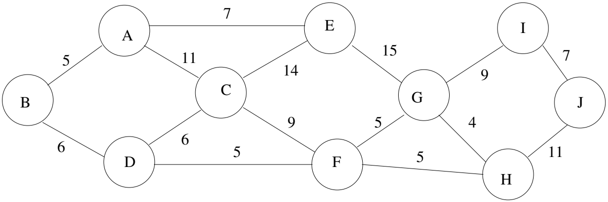
Complexity as a function of the number of nodes

- Worse case: 2^N
- If h is admissible: N^2
- If h is consistant: N
- Even a linear complexity can be very high. For the TSP the number of states is $N = !n$

Complexity as a function of K and M

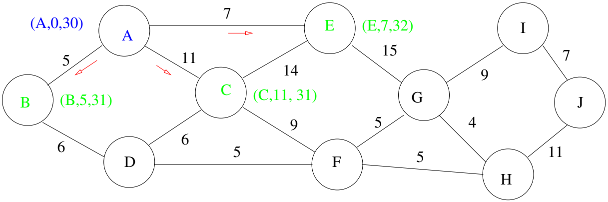
- Let us define
  - K number of sons for a node.
  - M minimal number if edges between the root and final state.
- If h is perfect: M
- If h is consistant: k^M
- If h is admissible:
  - $(1-r)\,h^* \leq h \Rightarrow K^{aM}$
  - $h^* - (h^*)^{\frac{1}{2}} \leq h \Rightarrow M^{\frac{1}{2}} K^{M^{\frac{1}{2}}}$
  - $h^* - log(h^*) \leq h \Rightarrow M^{log(K)}$
  - $h^* - r \leq h \Rightarrow MK^r$

Exercice: find the shortest path from A to J



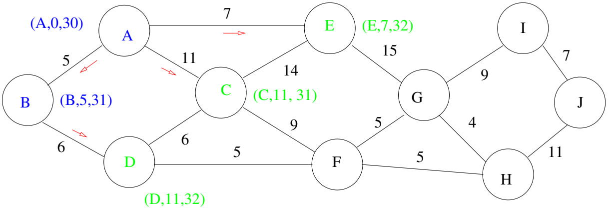
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



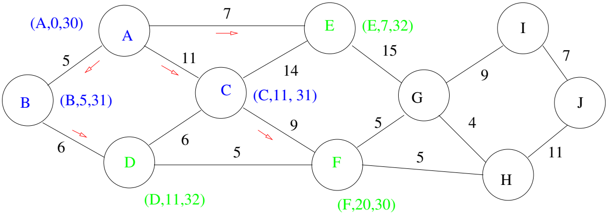
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



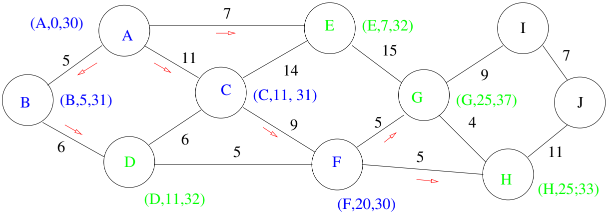
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



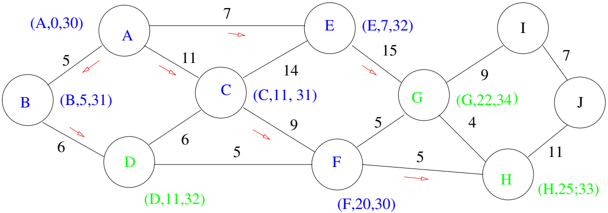
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



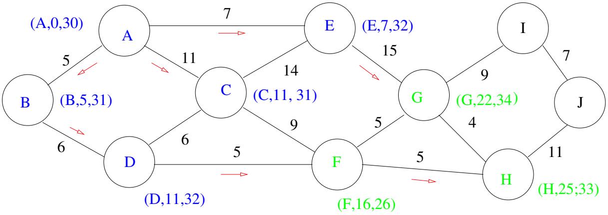
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



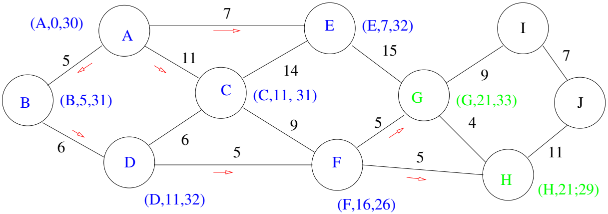
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



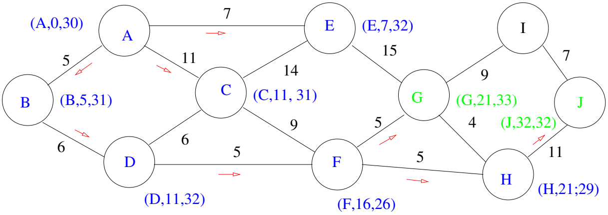
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



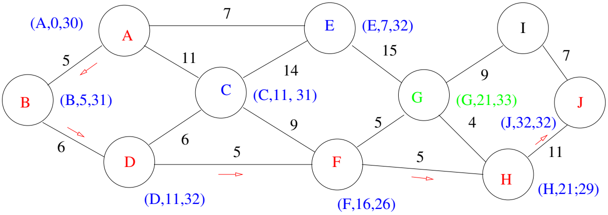
A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Exercice: find the shortest path from A to J



A	B	C	D	E	F	G	H	I	J
30	26	20	21	25	10	12	8	5	0

Dijkstra's algorithm

- Edsger Dijkstra in 1956: graph search algorithm.

Dijkstra's algorithm

- Edsger Dijkstra in 1956: graph search algorithm.
- Shortest path in a graph with non-negative edge path cost.

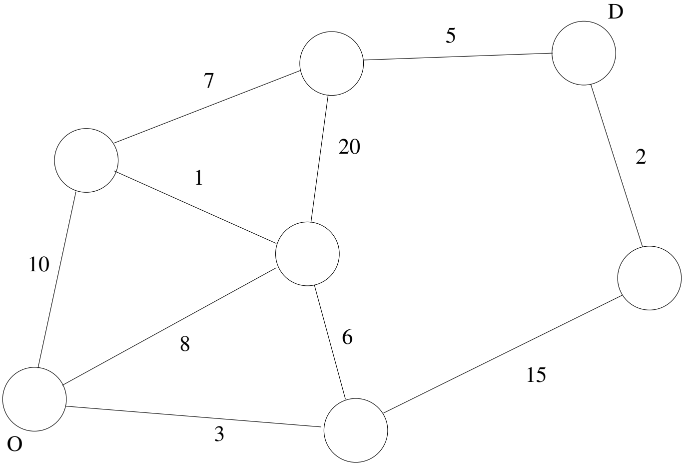
Dijkstra's algorithm

- Edsger Dijkstra in 1956: graph search algorithm.
- Shortest path in a graph with non-negative edge path cost.
- Complexity in $O(V^2)$ where V is the number of vertices.

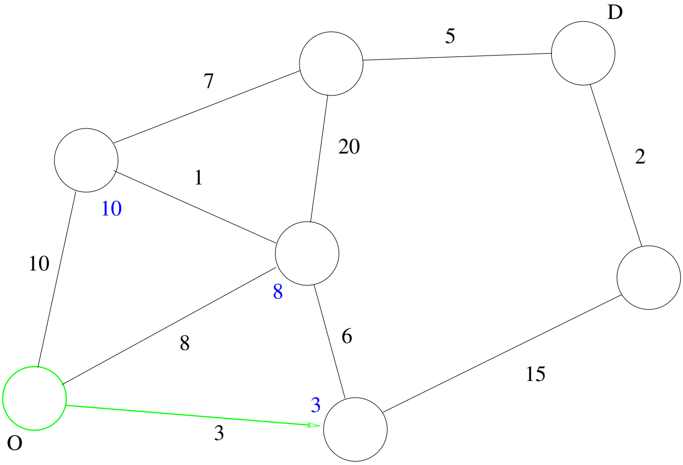
Dijkstra's algorithm

- Edsger Dijkstra in 1956: graph search algorithm.
- Shortest path in a graph with non-negative edge path cost.
- Complexity in $O(V^2)$ where V is the number of vertices.
- Can be reduced to $O(E + Vlog(V))$ where E is the number of edges. with min-priority queues

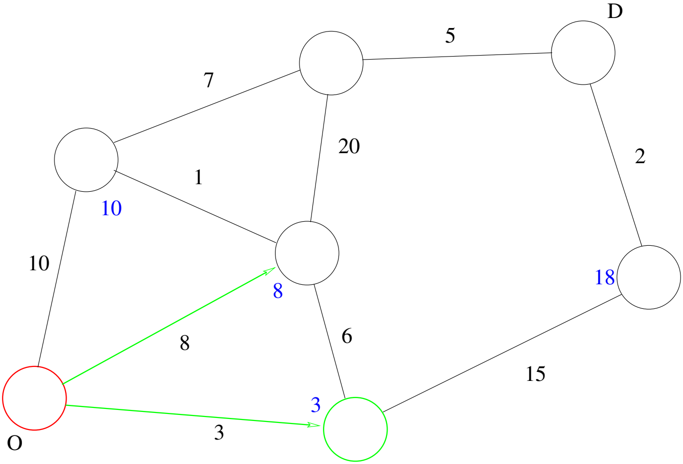
Dijkstra's algorithm



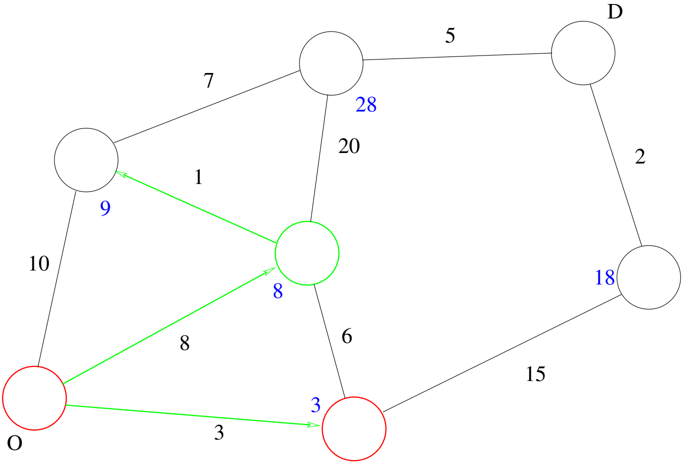
Dijkstra's algorithm



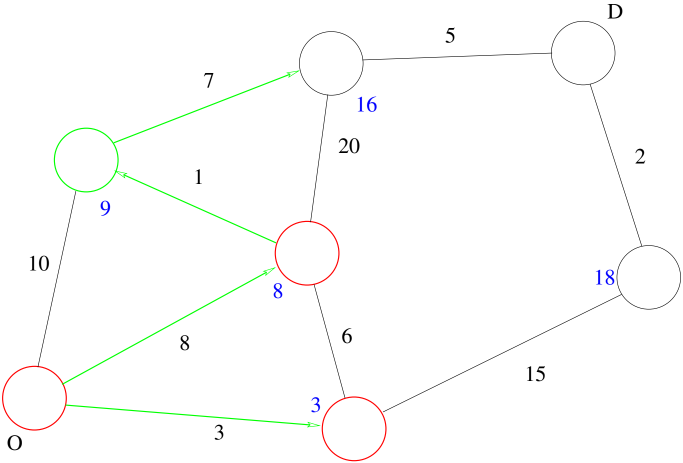
Dijkstra's algorithm



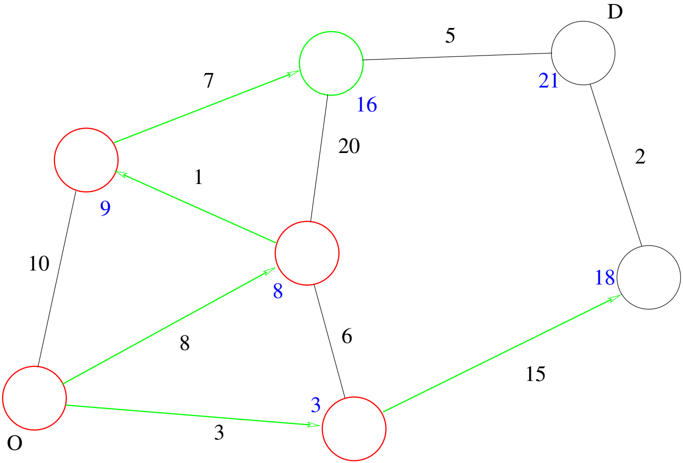
Dijkstra's algorithm



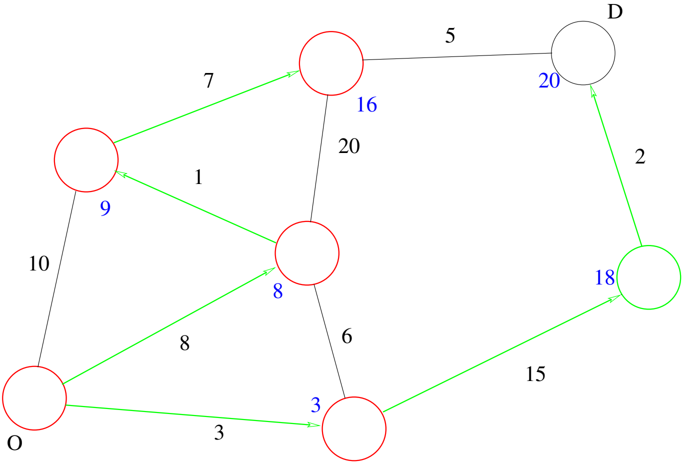
Dijkstra's algorithm



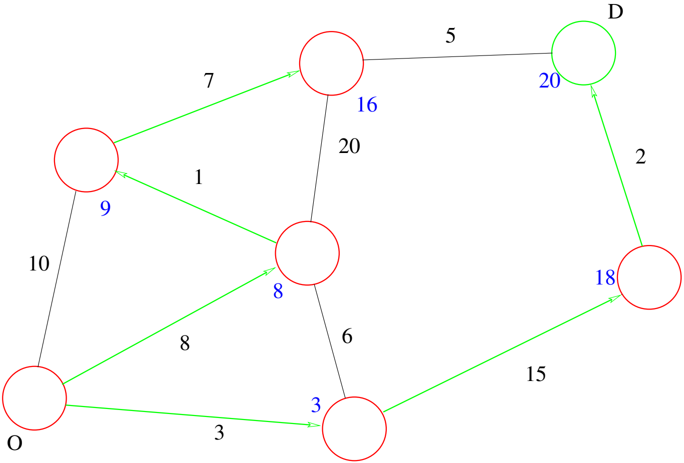
Dijkstra's algorithm



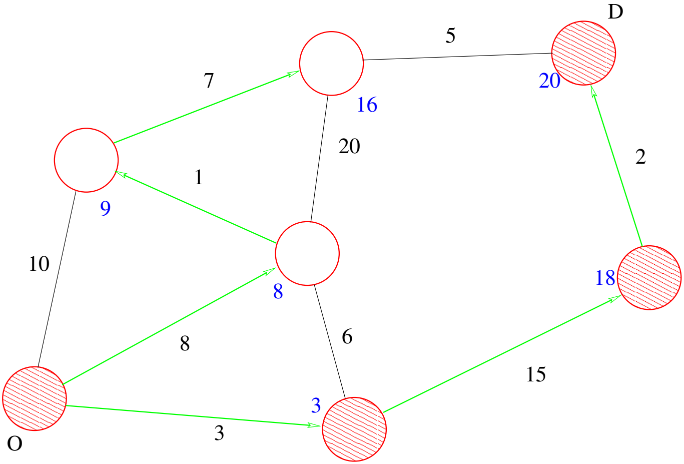
Dijkstra's algorithm



Dijkstra's algorithm



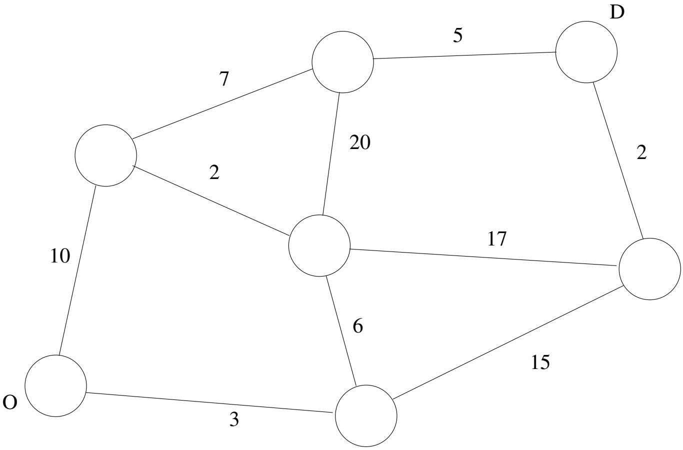
Dijkstra's algorithm



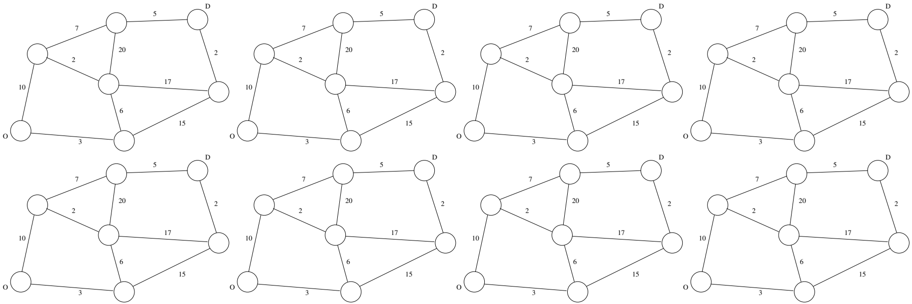
Dijkstra Algorithm(Graph,Source,Target)

```
for all nodes v ∈ Graph do
    dist(v):= ∞, previous(v):=undefined
end for
dist(source):=0
Q:= the set of all nodes in Graph
while Q ≠ ∅ do
    u:= node in Q with smallest distance
    if u=Target or dist(u)=∞ then
        END
    end if
    Q:=Q-u
    for each neighbor v of u do
        alt:=dist(u)+distbetween(u,v)
        if alt<dist(v) then
            dist(v):=alt,previous(v):=u
            Reorder v in Q
        end if
    end for
end while
```

Dijkstra's algorithm: exercise

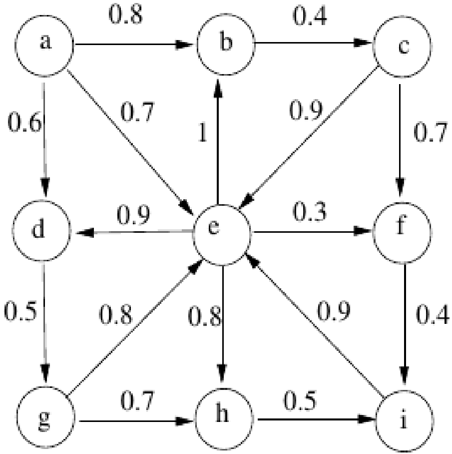


Dijkstra's algorithm: exercise



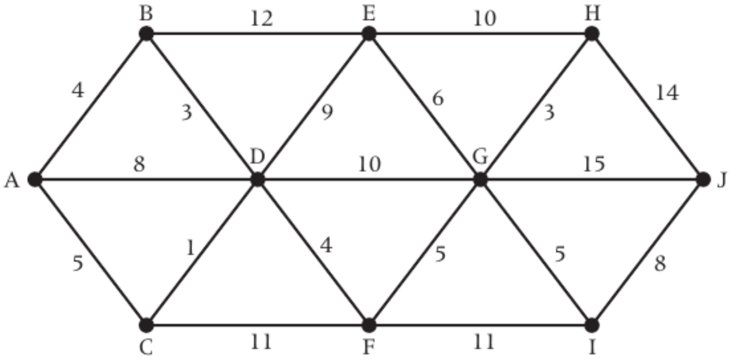
Dijkstra's algorithm: exercise

Let's consider a communication network with a reliability expressed in the following network. Which is the most reliable path from *a* to *i* ?



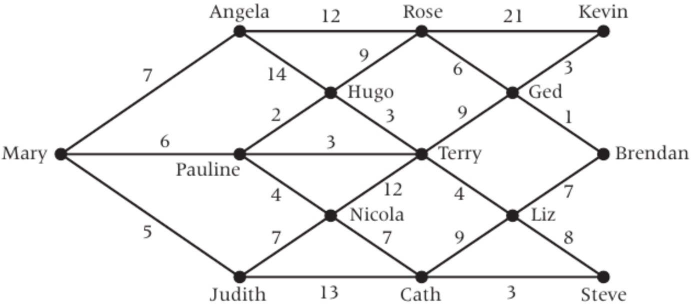
Dijkstra's algorithm: exercise

Find the shortest distance from A to J on the network below.



Dijkstra's algorithm: exercise

Every day, Mary thinks of a rumour to spread on her way to school.



- Find the time taken for the rumour to reach each person.
- List the route through which Brendan first hears the rumour.
- One day Pauline is not at school. What is the extra time needed by Brendan to hear about the rumour.

Motivation

- Prove that a set of logic propositions is satifiable.
- Scheduling problems.
- Optimization problems.
- Type of constraints:
    - depend on the domains ($\mathbb{R}$, $\mathbb{N}$, time intervals, binary variables, qualitative variables)
    - Real variable constraints (ex: linear constraints).
    - Finite Domain constraints. (The case we are interested in!)

Definitions

- A *CSP* $P = (X, D, C)$ is defined by
  - a sequence $X = (x_1, x_2, ..., x_n)$ of n variables
  - a sequence $D = (d_1, d_2, ..., d_n)$ of n finite domains for the variables of X.
  - a sequence $C = (c_1, c_2, ..., c_m)$ of m constraints. Each constraint c_i is defined by a couple (v_i, r_i)
    - v_i is a sequence of variables $(x_{i_1}, ..., x_{i_{n_i}}) \subset X$
    - r_i is a relation defined by a subset of $(d_{i_1} \times ... \times d_{i_{n_i}})$ of the domains related to variables of v_i. It represents the allowed values for these variables.

Binary CSP

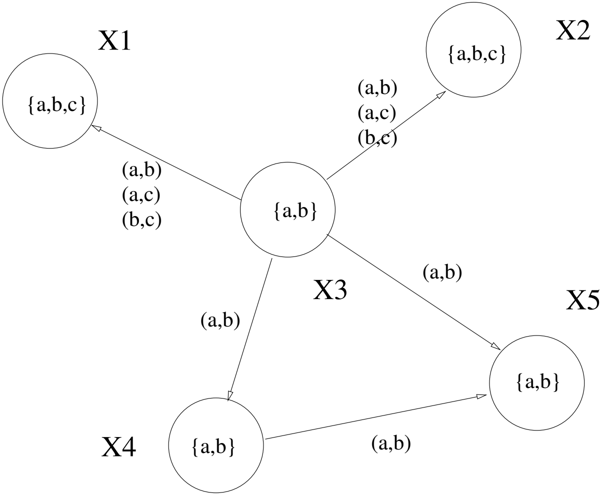
- A *CSP* is binary if the number of variables involved in every constraint is 2.
- Binary *CSP*'s can be represented by graphs
- Example in Air Traffic Control
  - Each aircraft can choose one trajectory out of a set of trajectories
  - Each aircraft i trajectory i_k might be in conflict with aircraft j trajectory j_l.
  - Find a conflict free combination of trajectories.

Example

Variables	Domains
x_1	a, b, c
x_2	a, b, c
x_3	a, b
x_4	a, b
x_5	a, b

Constraints	Variables	Relations
$c_1 = (v_1, r_1)$	$v_1 = (x_3, x_1)$	$r_1 = (a, b), (a, c), (b, c)$
$c_2 = (v_2, r_2)$	$v_2 = (x_3, x_2)$	$r_2 = (a, b), (a, c), (b, c)$
$c_3 = (v_3, r_3)$	$v_3 = (x_3, x_4)$	(a, b)
$c_4 = (v_4, r_4)$	$v_4 = (x_3, x_5)$	(a, b)
$c_5 = (v_5, r_5)$	$v_5 = (x_4, x_5)$	(a, b)

Example



Definitions

- For a CSP $P = (X, D, C)$, an **instantiation** $\mathscr{A}$ of $Y = \{x_{y_1}, ..., x_{y_{|Y|}}\} \subset X$ is an application that associates to each variable $x_{y_i} \in Y$ a value $\mathscr{A}(x_{y_i}) \in d_{y_i}$.
- An instantiation can be **complete** or **partial**.
- For a CSP $P = (X, D, C)$, an **instantiation** $\mathscr{A}$ of Y **satisfies** the constraint $c_i = (v_i, r_i)$ of C ($\mathscr{A} \models c_i$) iff $v_i \subset Y$ and $\mathscr{A}(v_i) \in r_i$.
- In the previous example, instantiation $\mathscr{A} = \{x_1 \to b, x_2 \to a, x_3 \to a\}$ satisfies c_1, violates c_2 and has no impact on the other constraints.

Definitions

- The **satisfiability rate** of a constraint c_i is the cardinal of r_i devided by the product of the cardinals of the domains of the variables of v_i.
- For example, the satisfiability rate of c_1 and c_2 is $\frac{1}{2}$ whereas the satisfiability rate of c_3, c_4 and c_5 is $\frac{1}{4}$.
- An instantiation $\mathscr{A}$ of variables of $Y \subset X$is **consistent** iff: $\forall c_i = (v_i, r_i) \in C$ with $v_i \subset Y, \mathscr{A} \models c_i$

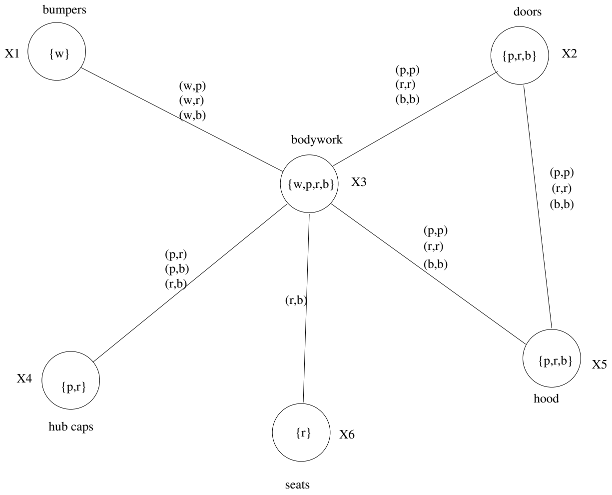
Definitions

- A **Solution** $\mathscr{S}$ of $P = (X, D, C)$ is a consistent instantiation of variables of X. We say: $\mathscr{S}$ satisfies P ($\mathscr{S} \models P$). S_P is the set of solutions of P
- A CSP $P = (X, D, C)$ is **consistent** iff $S_p \neq \emptyset$
- An instantiation $\mathscr{A}$ is **globally consistent** if $\exists \mathscr{S} \in S_P, \mathscr{A} \subset \mathscr{S}$
- Remark: if an instantiation is not globally consistent, it cannot be extended to a solution.
- Is the previous CSP consistent?

Example

- An automaker is building a new car. Different parts of the car come from all over Europe.
  - Lille provides doors and hoods painted in pink red or black
  - Hambourg provides the bodyworks painted in white, pink, red or black
  - Palerme provides the bumpers painted in white
  - Madrid provides the seats in red only
  - Athens provides the hub caps painted in pink and red
- The constraints are the following:
  - The bodywork and doors must be the same color
  - The doors and the hood must be the same color
  - The hood and the bodywork must be the same color
  - the hub caps, bumpers and seats color must be ligther than the bodywork

Example


bumpers
X1
{w}
doors
{p,r,b}
X2
(w,p)
(w,r)
(w,b)
(p,p)
(r,r)
(b,b)
bodywork
{w,p,r,b}
X3
(p,p)
(r,r)
(b,b)
(p,p)
(r,r)
(b,b)
(p,r)
(p,b)
(r,b)
(r,b)
X4
{p,r}
hub caps
{r}
X6
seats
{p,r,b}
X5
hood

Example

- This problem only has 2 solutions
  $\mathscr{S}_1 = \{x_1 \rightarrow w, x_2 \rightarrow b, x_3 \rightarrow b, x_4 \rightarrow p, x_5 \rightarrow b, x_6 \rightarrow r\}$
  $\mathscr{S}_2 = \{x_1 \rightarrow w, x_2 \rightarrow b, x_3 \rightarrow b, x_4 \rightarrow r, x_5 \rightarrow b, x_6 \rightarrow r\}$
- Give examples of globally consistent instantiations ?

Definitions

- Two CSPs P and P are **equivalent** ($P \equiv P'$) iff $S_p = S_{P'}$.
- A constraint c is **induced** by a CSP P if it satisfies all the solutions of P.
  $(\forall \mathscr{S} \in S_p, \mathscr{S} \models c)$.
- It can be added to a CSP without changing the solution set. For example, constraint $c : x_3 \in \{w, b\}$ is induced by the previous CSP.
- A CSP $P = (X, D, C)$ is **globally consistent** iff $\forall c_i = (v_i, r_i) \in C, \bigcup_{\mathscr{S} \in S_p} [\mathscr{S}(v_i)] = r_i$
- This means that any partial instantiation that satifies at least one constraint without violating any other is globally consistent and can be extended to a solution. (Very rare in real life problems, unfortunately).

Problem Definitions

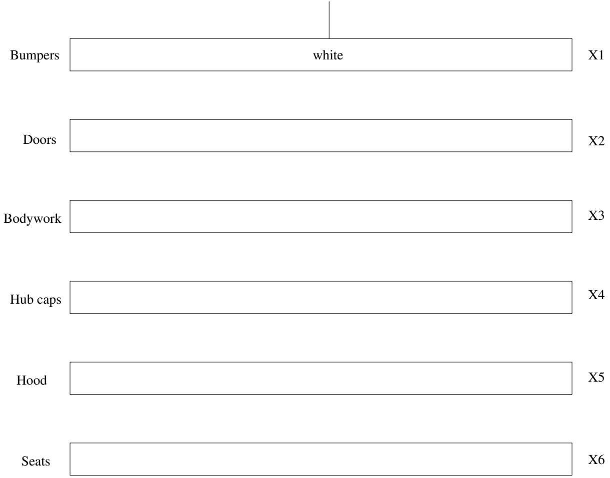
- Prove that a CSP is consistent
- Find every solution for a CSP
- Find a solution that maximizes one or several criterias.
- Estimate the number of solutions for a CSP
- Find the values of variables that are common to every solution

Standard resolution algorithm: Backtrack Algorithm

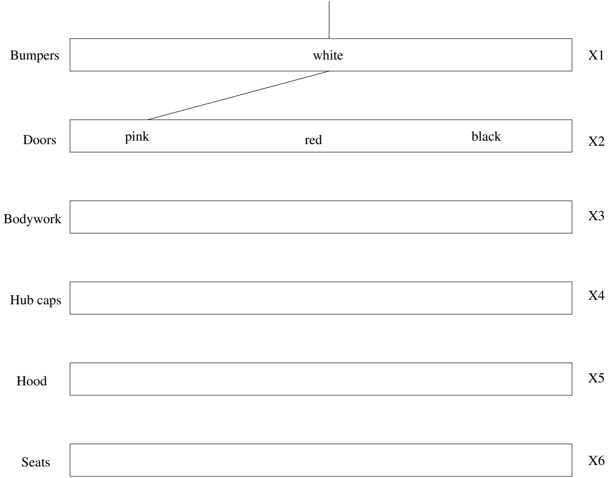
BT$(V, \mathscr{A})$

if $V = \emptyset$ **then**
    $\mathscr{A}$ is a solution
else
    choose $x_i \in V$
    for all $v \in d_i$ **do**
        if $\mathscr{A} \cup \{x_i \rightarrow v\}$ is consistent **then**
            BT$(V - \{x_i\}, \mathscr{A} \cup \{x_i \rightarrow v\})$
        end if
    end for
end if

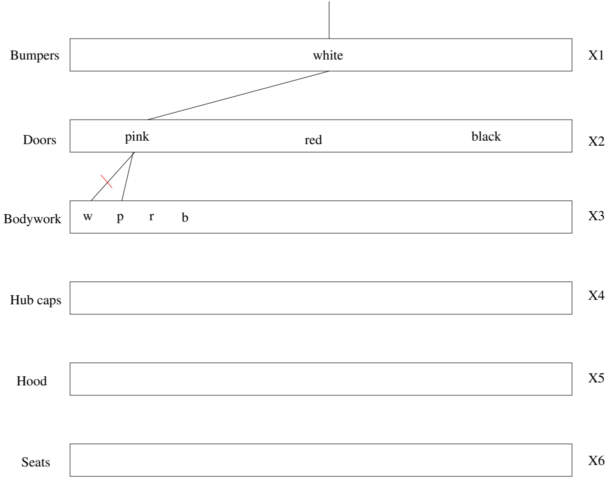
Backtrack algorithm example



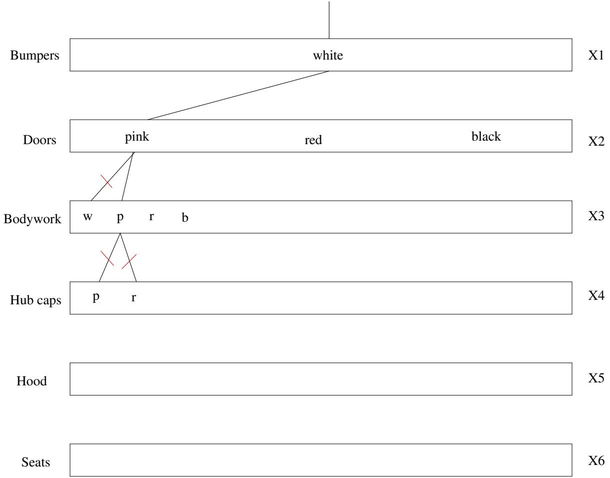
Backtrack algorithm example



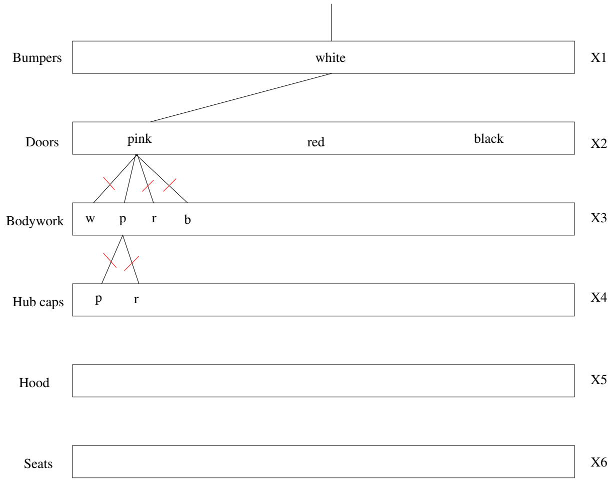
Backtrack algorithm example



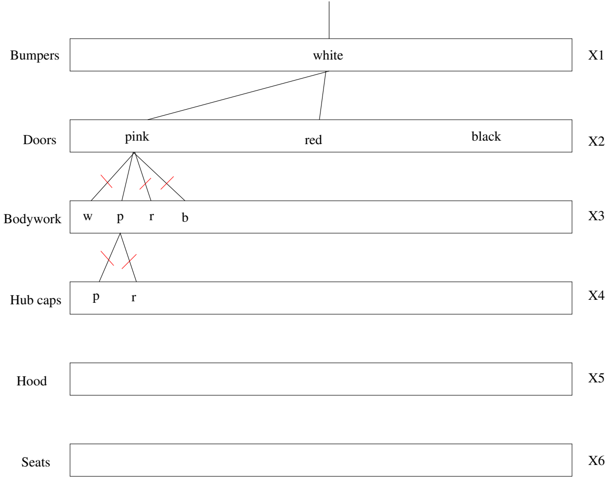
Backtrack algorithm example



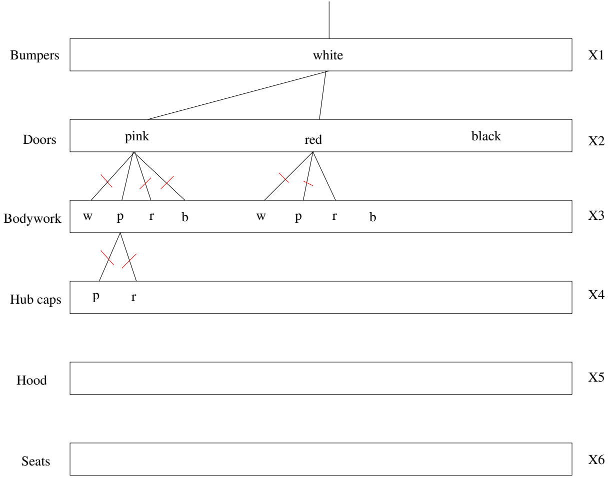
Backtrack algorithm example



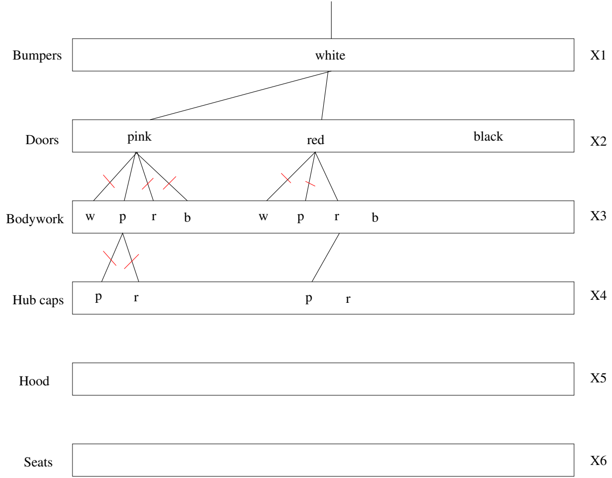
Backtrack algorithm example



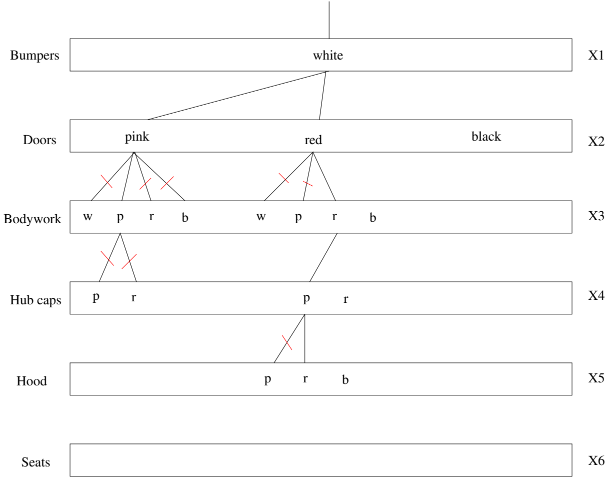
Backtrack algorithm example



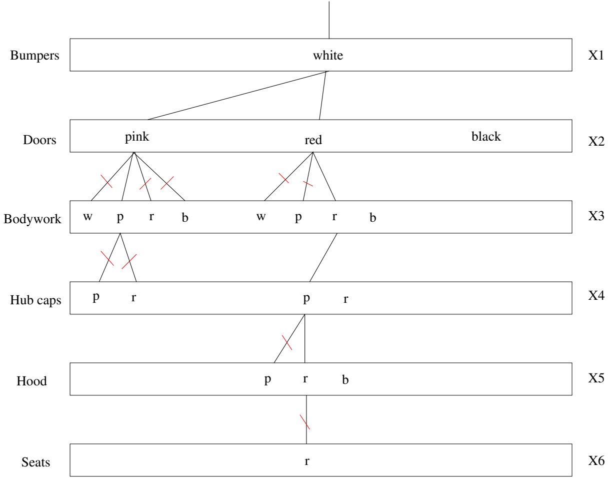
Backtrack algorithm example



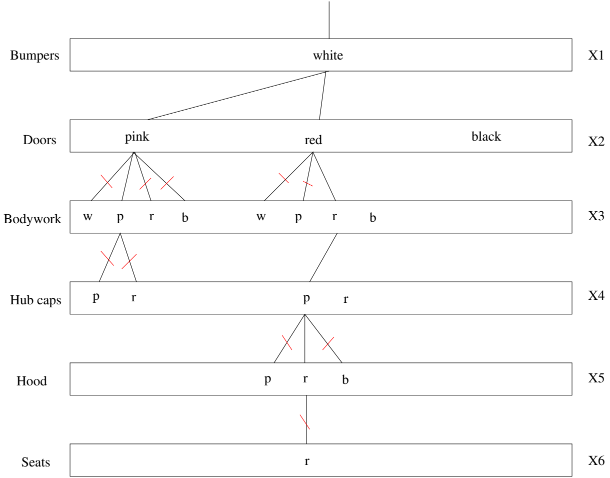
Backtrack algorithm example



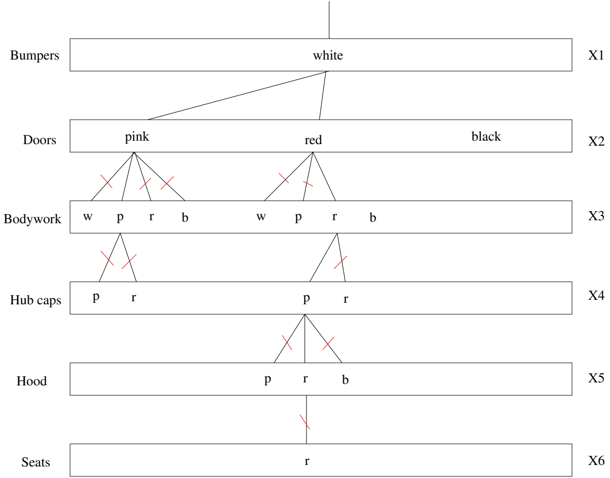
Backtrack algorithm example



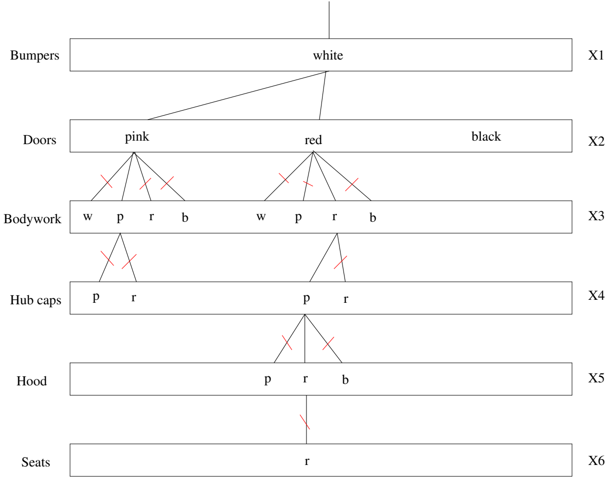
Backtrack algorithm example



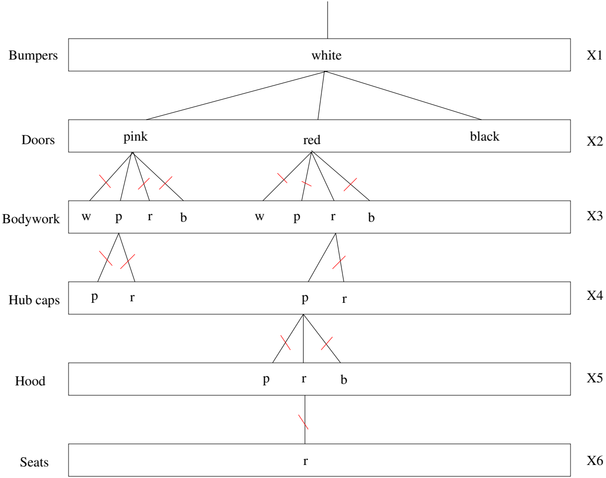
Backtrack algorithm example



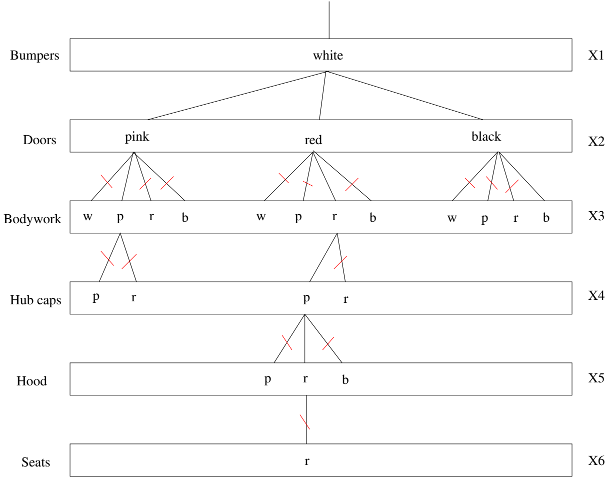
Backtrack algorithm example



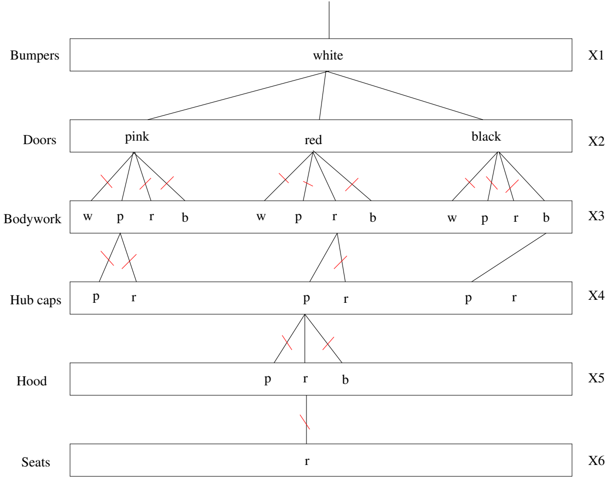
Backtrack algorithm example



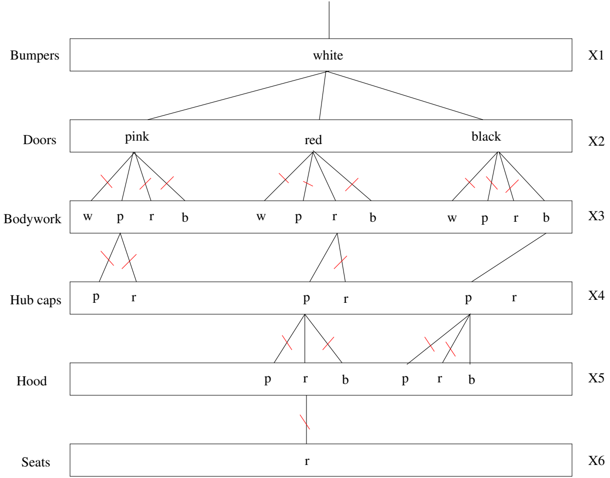
Backtrack algorithm example



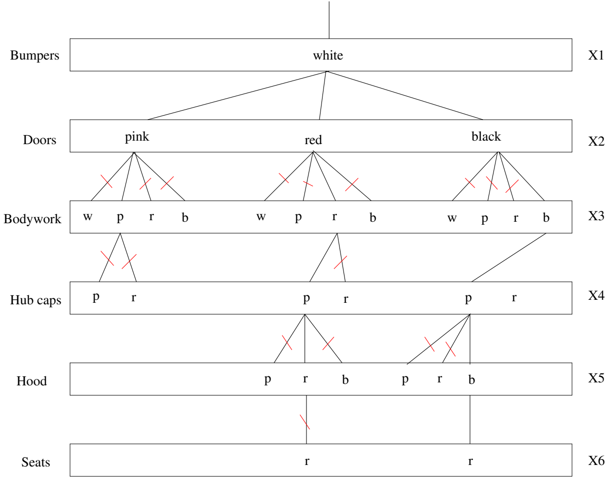
Backtrack algorithm example



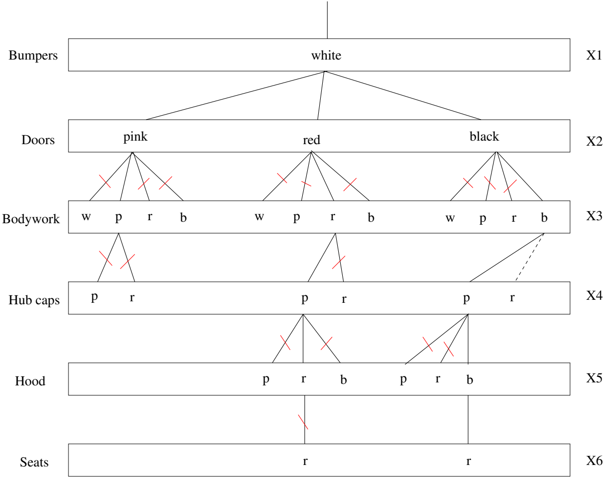
Backtrack algorithm example



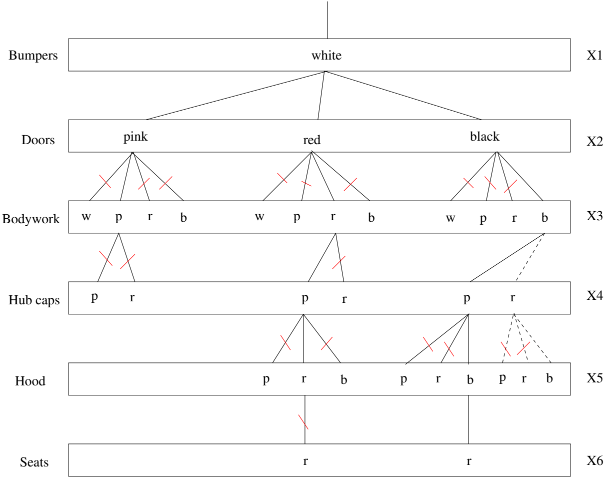
Backtrack algorithm example



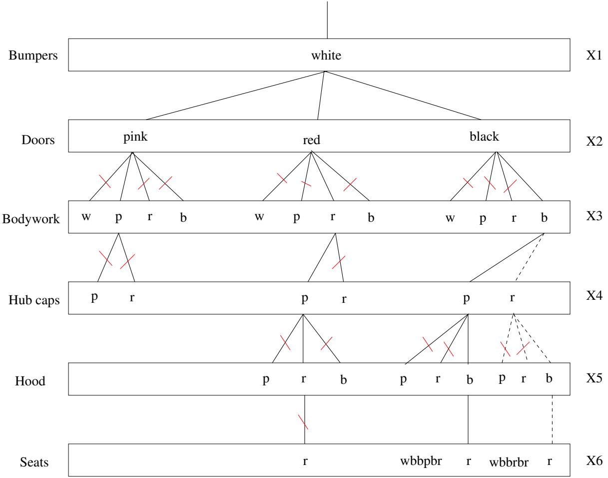
Backtrack algorithm example



Backtrack algorithm example



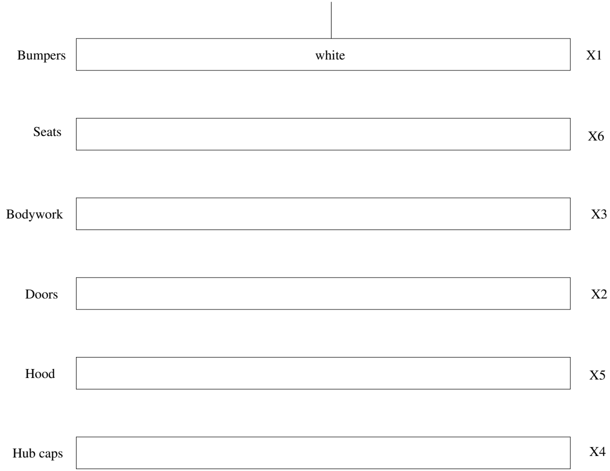
Backtrack algorithm example



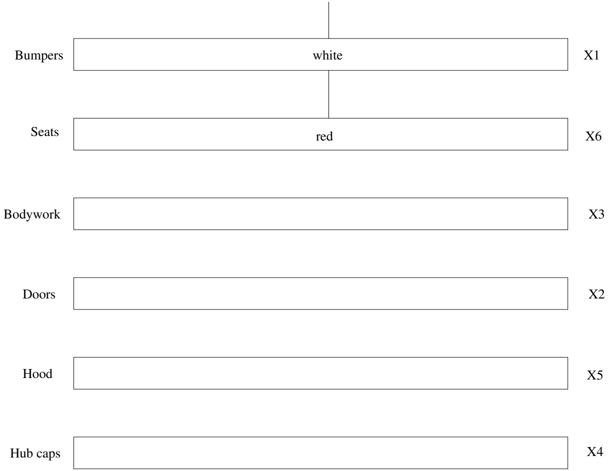
Remarks

- Required 29 nodes exploration and 30 constraints checks.
- Many drawbacks:
  - does not take advantage of non global consistency of partial instantiations
  - evaluates several times partial inconsistencies
- The backtrack algorithm can be improved.

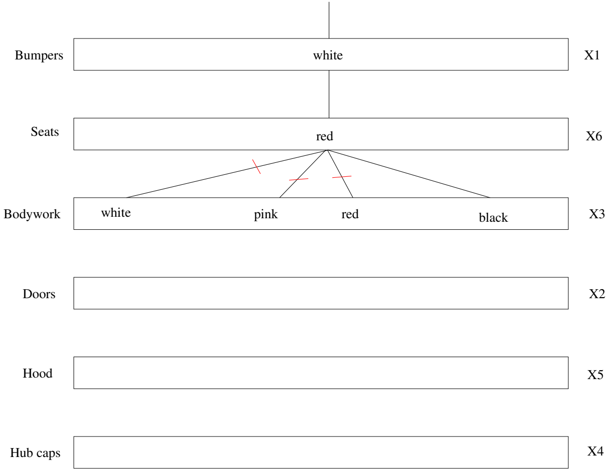
Backtrack algorithm example: another variable order



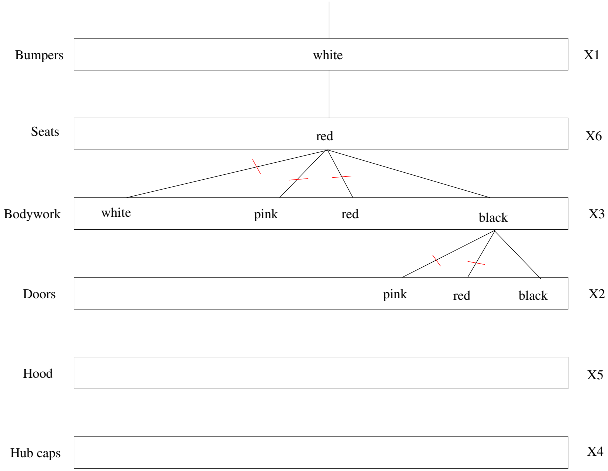
Backtrack algorithm example: another variable order



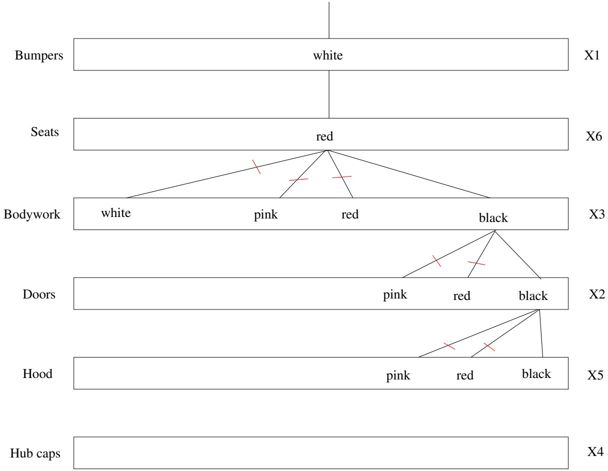
Backtrack algorithm example: another variable order



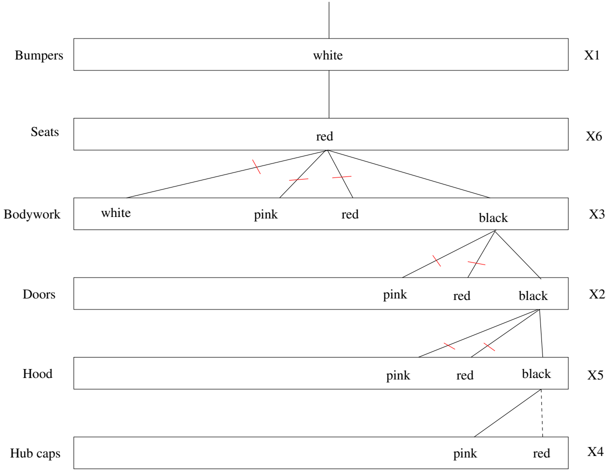
Backtrack algorithm example: another variable order



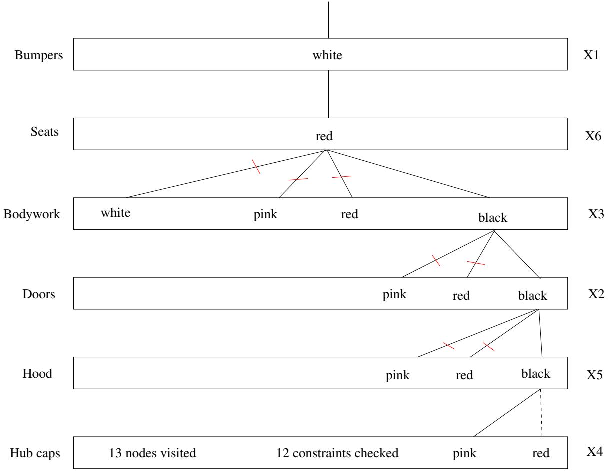
Backtrack algorithm example: another variable order



Backtrack algorithm example: another variable order



Backtrack algorithm example: another variable order



Definitions

- A CSP is **Node-consitent** if every values of every domain respect unary constraints.
- A CSP $P = (X, D, C)$ is **Arc-consitent** iff $\forall x_i \in X$, we have $d_i \neq \emptyset$ and $\forall v \in d_i, \forall c_j = (v_j, r_j) \in C, x_i \in v_j$, $\exists \mathscr{A} \in r_j$ such that $\mathscr{A}(x_i) = v$
- In other words, for every couple (x_i, x_j) of variables involved in a constraint, each value of the domain d_i is consistent with at least one value of the domain d_j.

Revise procedure

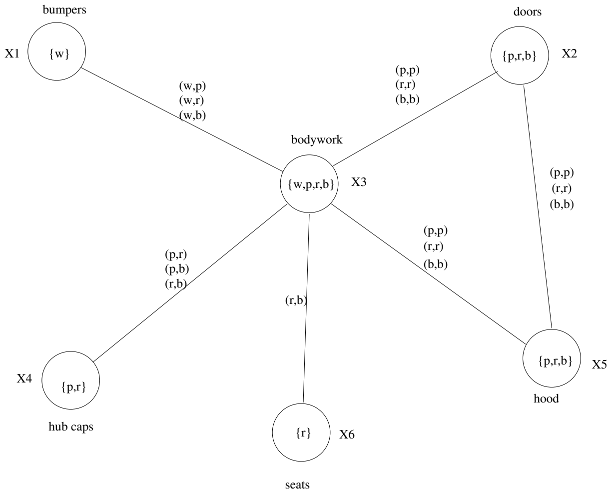
Revise(x_i, x_j)

modif $=$ false
for all $v \in d_i$ **do**
  if $\nexists v' \in d_j$ such that $\{x_i \rightarrow v, x_j \rightarrow v'\}$ is consistent **then**
    $d_i \leftarrow (d_i - \{v\})$
    modif $=$ true
  end if
end for
modif

Revise Procedure

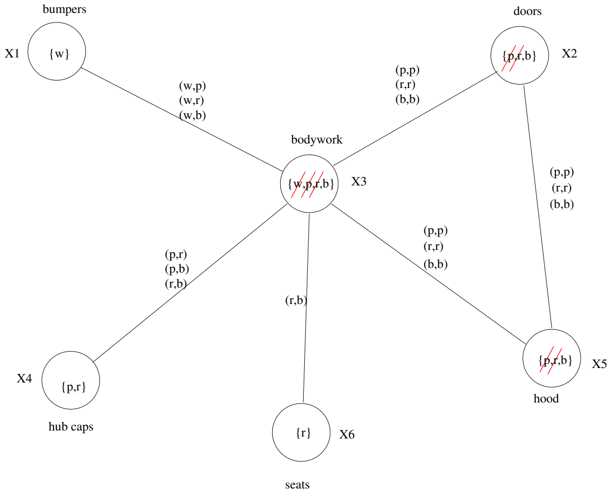
- Apply it on every variable couple
- May need to apply it several times until no domain can be reduced anymore
- $AC1$ repeats the procedure on every couple both ways as long as true values come out

Graph before Arc consitency

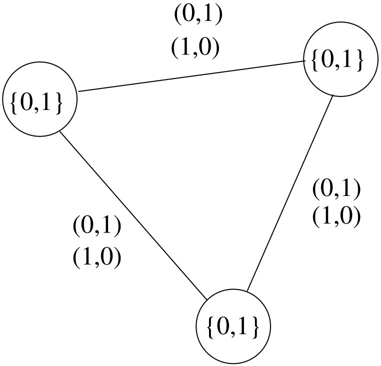


Graph after Arc consitency

The new CSP is globally consistent



Build an arc-consistent graph that is not globally consistent



AC3 Procedure

AC3

1: $L \leftarrow$ list of couples (x_i, x_j), $\exists$ a constraint between x_i and x_j
2: **while** $L \neq \emptyset$ **do**
3: Choose and eliminate a couple $(x_i, x_j) \in L$
4: **if** $revise(x_i, x_j) = true$ **then**
5: $L \leftarrow L \bigcup \{(x_k, x_i), \exists$ a constraint between x_i and $x_k\}$
6: **end if**
7: **end while**

Other procedures

- AC3 complexity is in $O(m.d^3)$ where m is the number of constraints and d the size of the domains.
- AC4 does not rely on revise and has a complexity in $O(m.d^2)$
- AC6 (same principles as AC4) but complexity reduced to $O(m.d)$

Path-Consistency

- A pair of variables is path-consistent with a third variable if each consistent evaluation of the pair can be extended to the other variable in such a way that all binary constraints are satisfied.
- x_i and x_j are path consistent with x_k if, for every pair of values (a_i, a_j) that satisfies the binary constraint between x_i and x_j, there exists a value a_k in the domain d_k such that (a_i, a_k) and (a_i, a_k) satisfy the constraint between x_i and x_k and between x_j and x_k respectively.

Intelligent backtrack

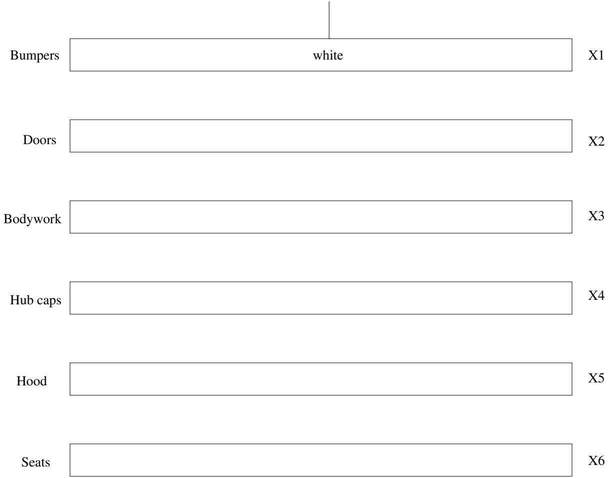
- Idea: Exploit constraint violations in order to build an instantiation belonging (but smaller) to the current instantiation that is not globally consistent. Each time it is encountered in the future it avoids further research in the tree.
- Example: **conflict based backjumping**
    - The idea is to calculate, each time a backtrack occurs, the values in $\mathscr{A}$ of *conflicting variables*.

Conflict Based Backjumping

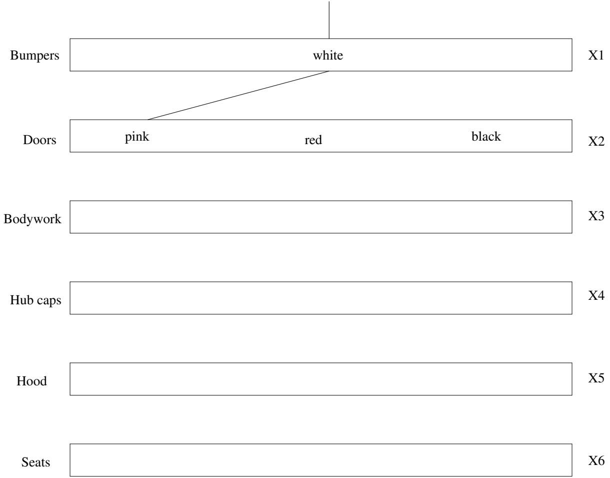
CBJ(V, $\mathscr{A}$)

```
if V = ∅ then
    𝒜 is a solution
else
    Choose x_i ∈ V
    conflict = ∅
    noBJ = true
    for all v ∈ d_i do
        while noBJ do
            localconflict = consistency (𝒜 ⋃{x_i → v})
            if localconflict = ∅ then
                sonconflict = CBJ (V − {x_i}, 𝒜 ⋃{x_i → v})
                if x_i ∈ sonconflict then
                    conflict ← conflict ⋃ sonconflict
                else
                    conflict ← sonconflict
                    noBJ = false
                end if
            else
                conflict ← conflict ⋃ localconflict
            end if
        end while
    end for
    conflict
end if
```

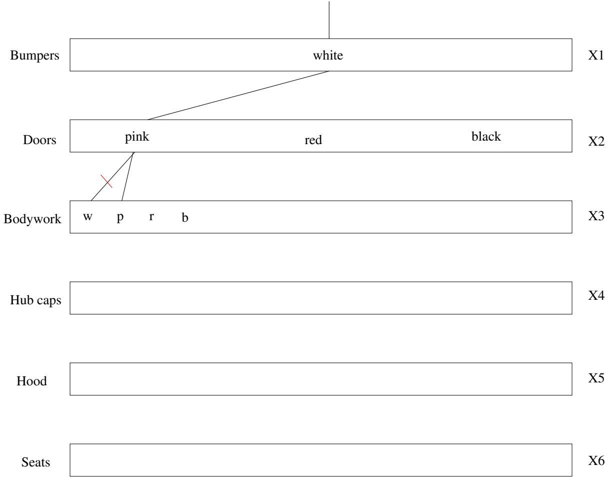
Backtrack algorithm example with CBJ procedure



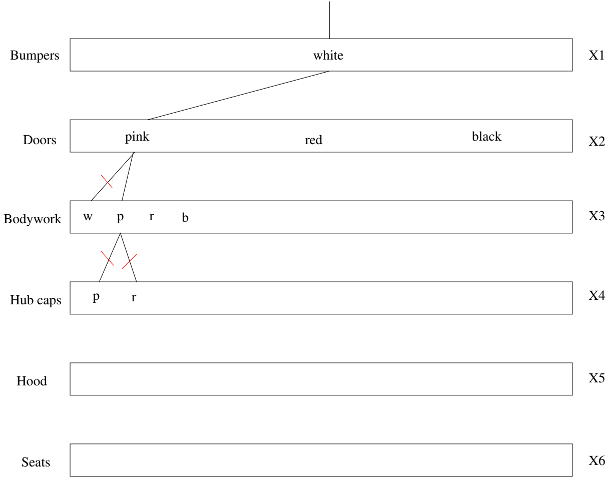
Backtrack algorithm example with CBJ procedure



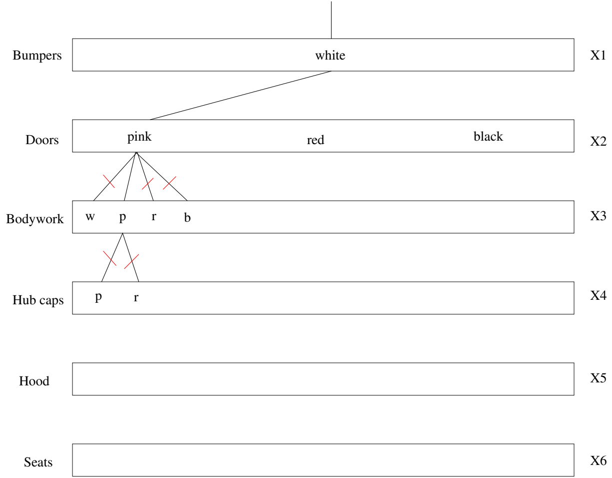
Backtrack algorithm example with CBJ procedure



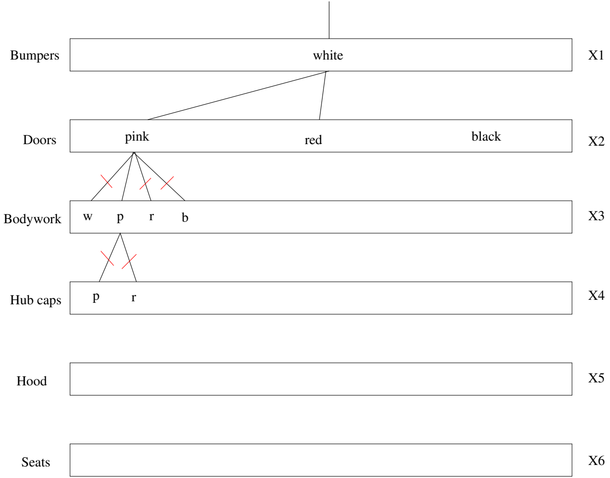
Backtrack algorithm example with CBJ procedure



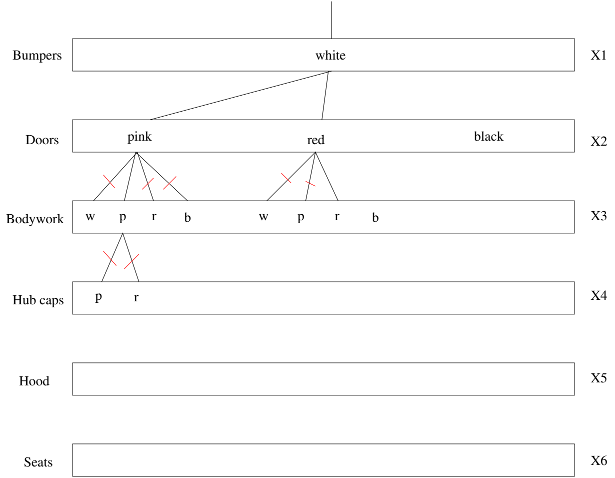
Backtrack algorithm example with CBJ procedure



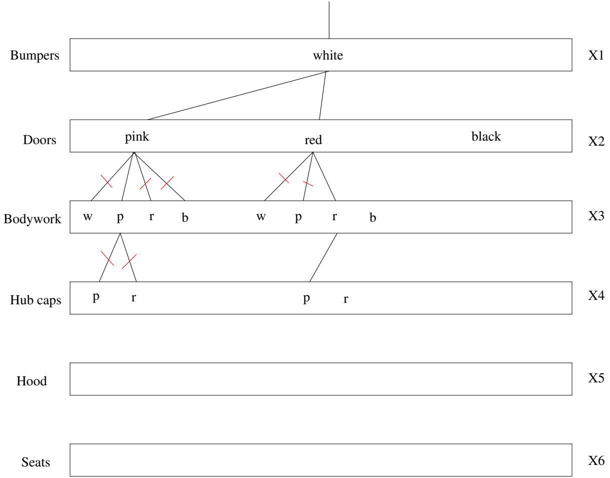
Backtrack algorithm example with CBJ procedure



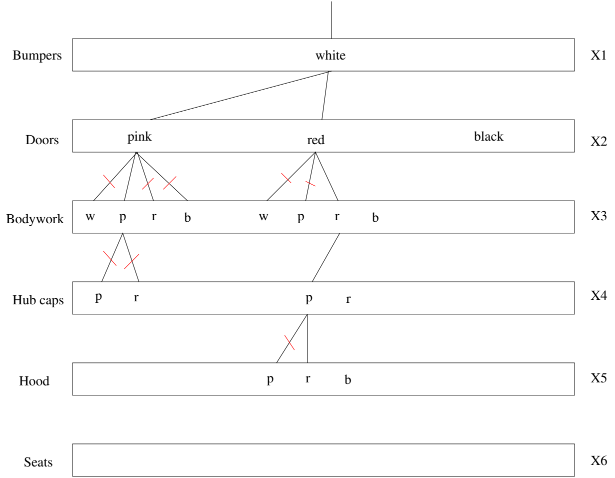
Backtrack algorithm example with CBJ procedure



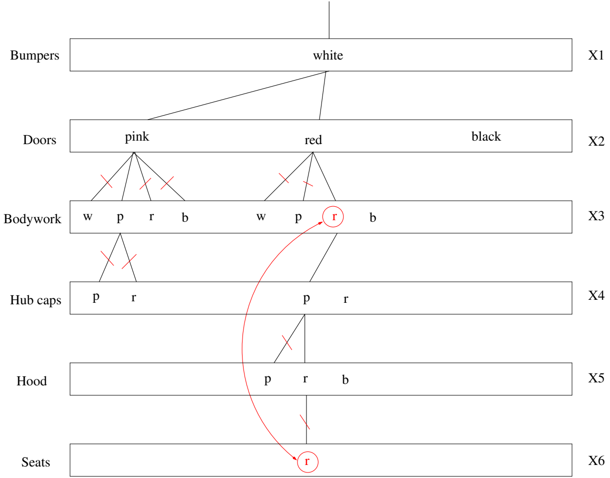
Backtrack algorithm example with CBJ procedure



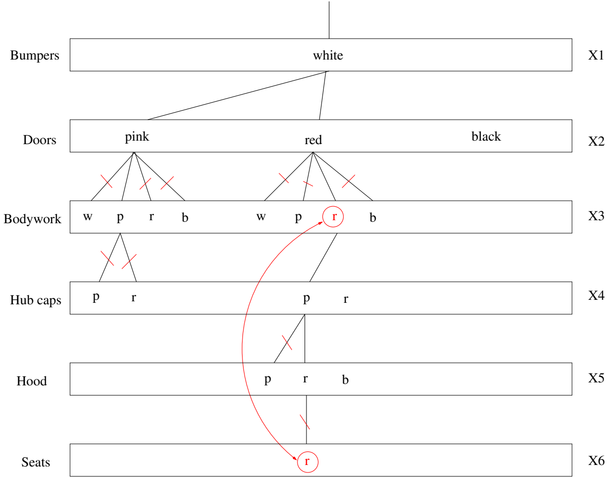
Backtrack algorithm example with CBJ procedure



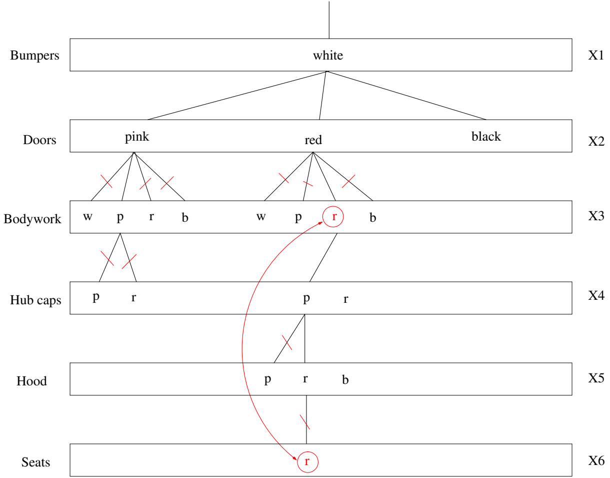
Backtrack algorithm example with CBJ procedure



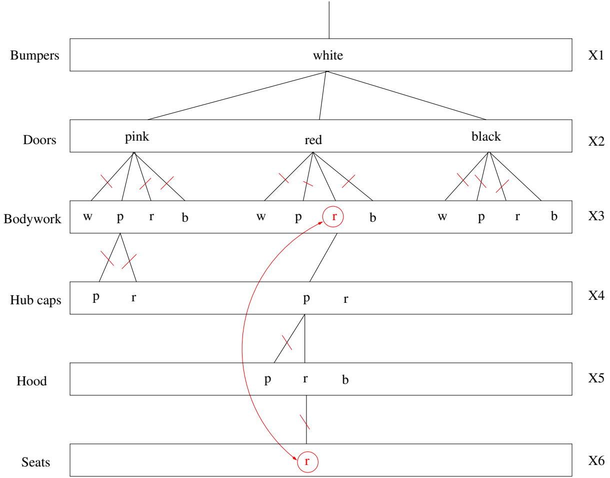
Backtrack algorithm example with CBJ procedure



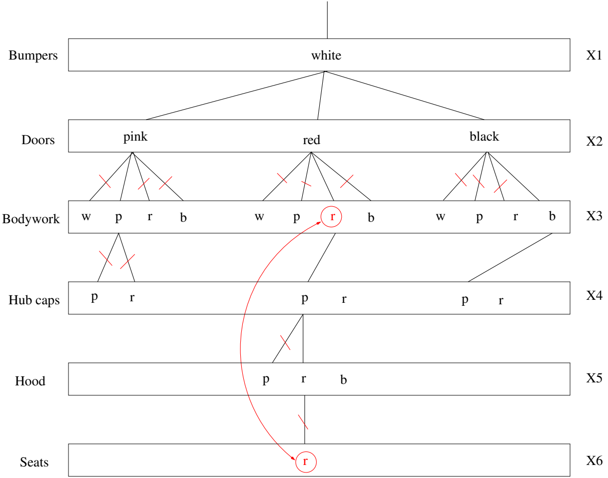
Backtrack algorithm example with CBJ procedure



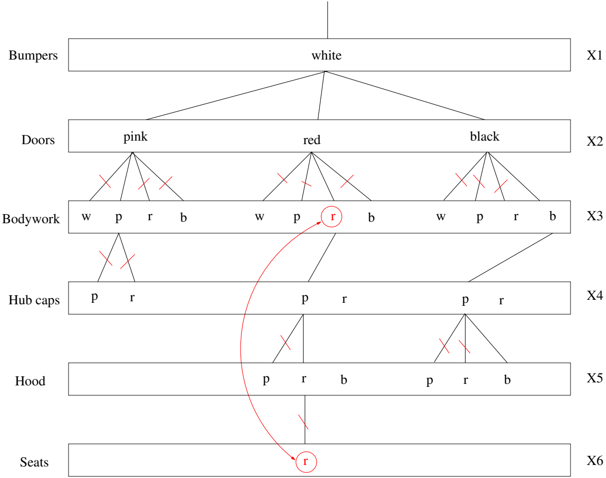
Backtrack algorithm example with CBJ procedure



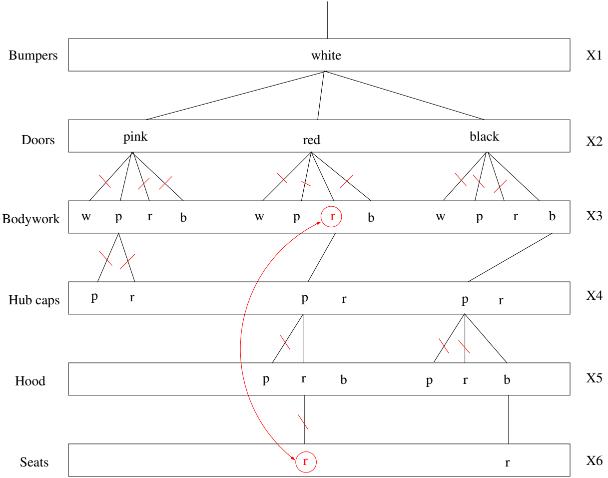
Backtrack algorithm example with CBJ procedure



Backtrack algorithm example with CBJ procedure



Backtrack algorithm example with CBJ procedure



Exercise

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

- every letter is a different number (from 0 to 9)
- the first letter of each word is represented by a number different from 0.
- Modelize the problem as a CSP.

Exercise

- Variables: S, E, N, D, M, O, R, Y

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $d_E = d_N = d_D = d_O = d_R = d_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:
    - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
    - $d_E = d_N = d_D = d_O = d_R = d_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $d_E = d_N = d_D = d_O = d_R = d_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - $1000(S + M) + 100(E + O) + 10(N + R) + D + E = 10000M + 1000O + 100N + 10E + Y$

Exercise

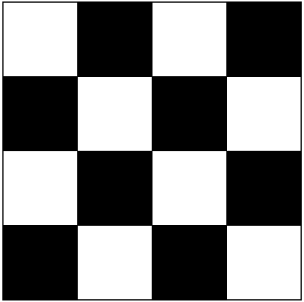
- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $d_E = d_N = d_D = d_O = d_R = d_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - $1000(S + M) + 100(E + O) + 10(N + R) + D + E = 10000M + 1000O + 100N + 10E + Y$
  - $allDifferent(S, E, N, D, M, O, R, Y)$

Exercise

- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - $d_S = d_M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $d_E = d_N = d_D = d_O = d_R = d_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - $1000(S + M) + 100(E + O) + 10(N + R) + D + E = 10000M + 1000O + 100N + 10E + Y$
  - $allDifferent(S, E, N, D, M, O, R, Y)$
- Solution: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2

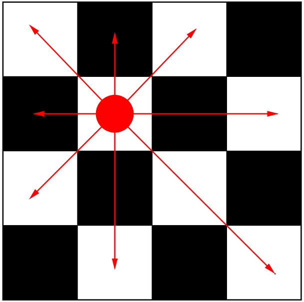
The queens problem

Place 4 chess queens on a 4×4 chess board so that no two queens attack each other.



The queens problem

Place 4 chess queens on a 4×4 chess board so that no two queens attack each other.



The queens problem: Modeling

- Variables: L_1, L_2, L_3, L_4 and C_1, C_2, C_3, C_4

The queens problem: Modeling

- Variables: L_1, L_2, L_3, L_4 and C_1, C_2, C_3, C_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$ and $C_1 \in \{1, 2, 3, 4\}$

The queens problem: Modeling

- Variables: L_1, L_2, L_3, L_4 and C_1, C_2, C_3, C_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$ and $C_1 \in \{1, 2, 3, 4\}$
- Constraints: lines, columns, and diagonals must be different.

The queens problem: Modeling

- Variables: L_1, L_2, L_3, L_4 and C_1, C_2, C_3, C_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$ and $C_1 \in \{1, 2, 3, 4\}$
- Constraints: lines, columns, and diagonals must be different.
  - $\forall i \neq j, L_i \neq L_j, C_i \neq C_j$

The queens problem: Modeling

- Variables: L_1, L_2, L_3, L_4 and C_1, C_2, C_3, C_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$ and $C_1 \in \{1, 2, 3, 4\}$
- Constraints: lines, columns, and diagonals must be different.
  - $\forall i \neq j, L_i \neq L_j, C_i \neq C_j$
  - $\forall i \neq j, L_i + C_i \neq L_j + C_j, L_i - C_i \neq L_j - C_j$

The queens problem: Another Modeling

- Variables: L_1, L_2, L_3, L_4

The queens problem: Another Modeling

- Variables: L_1, L_2, L_3, L_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$

The queens problem: Another Modeling

- Variables: L_1, L_2, L_3, L_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$
- Constraints:

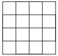
The queens problem: Another Modeling

- Variables: L_1, L_2, L_3, L_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$
- Constraints:
    - $\forall i \neq j, L_i \neq L_j$

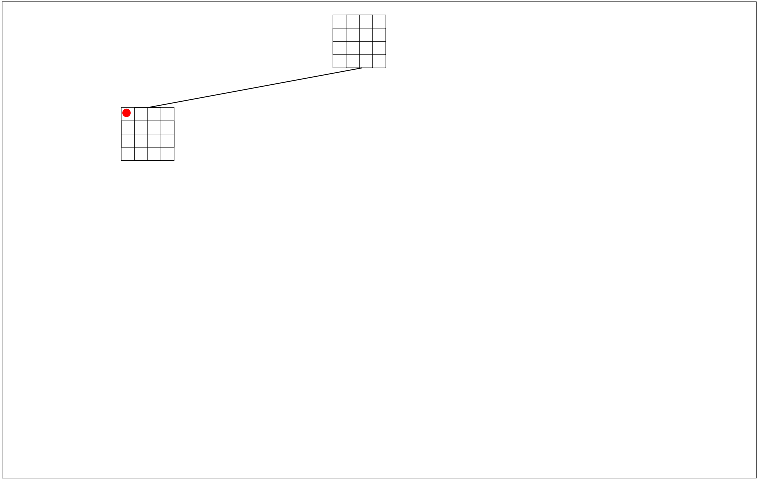
The queens problem: Another Modeling

- Variables: L_1, L_2, L_3, L_4
- Domains: $L_1 \in \{1, 2, 3, 4\}$
- Constraints:
  - $\forall i \neq j, L_i \neq L_j$
  - $\forall i \neq j, L_i + i \neq L_j + j, L_i - i \neq L_j - j$

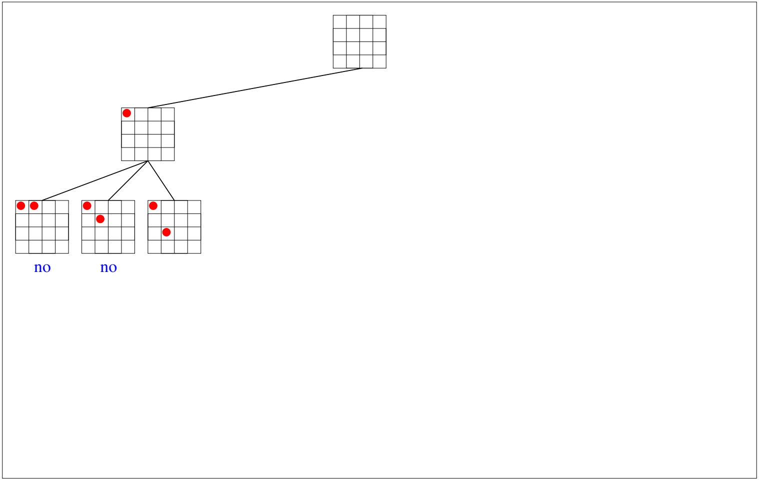
The queens problem: BackTrack Algorithm



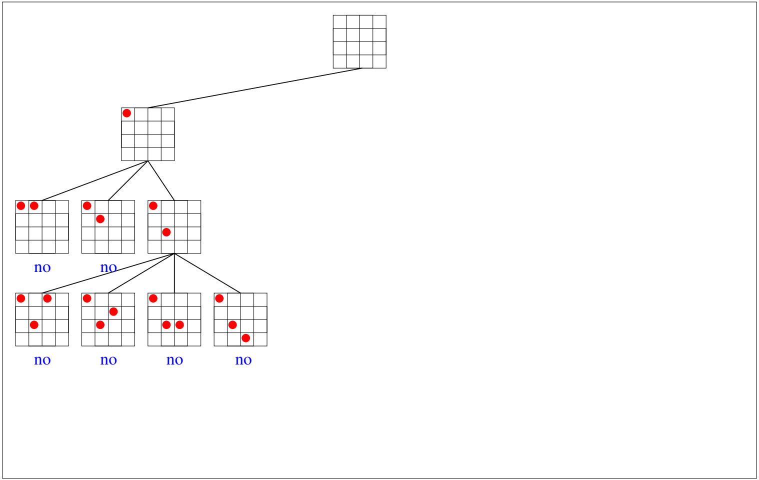
The queens problem: BackTrack Algorithm



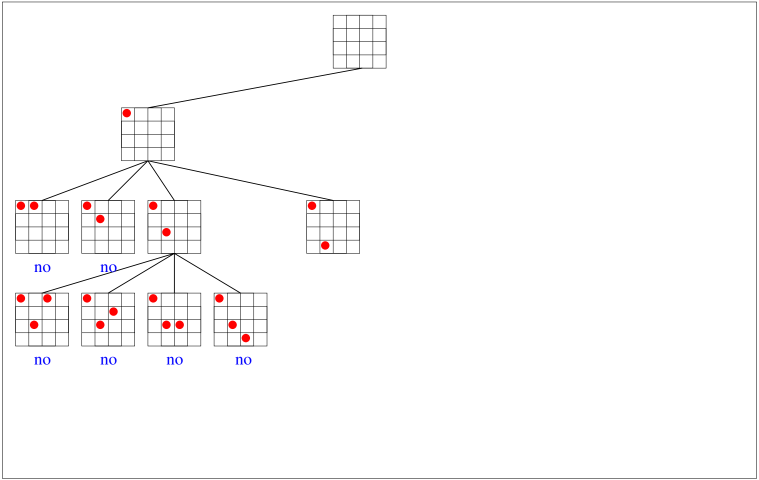
The queens problem: BackTrack Algorithm



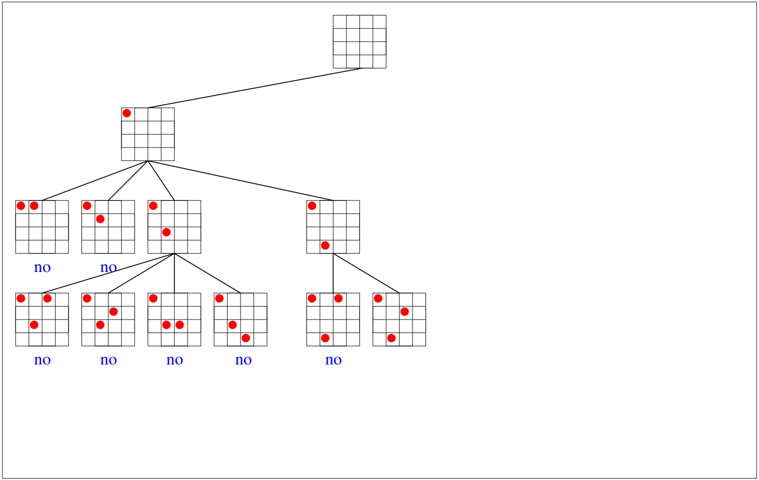
The queens problem: BackTrack Algorithm



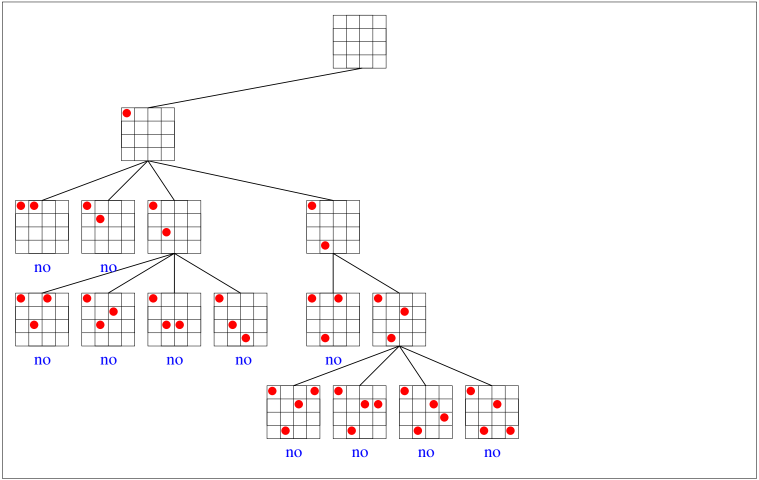
The queens problem: BackTrack Algorithm



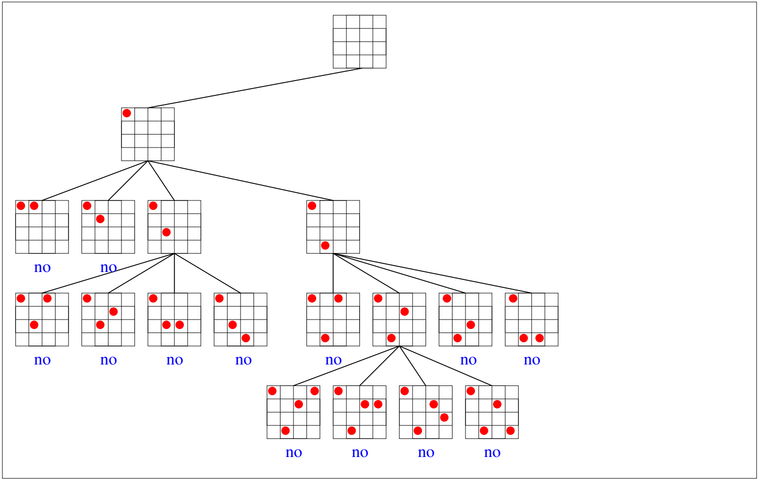
The queens problem: BackTrack Algorithm



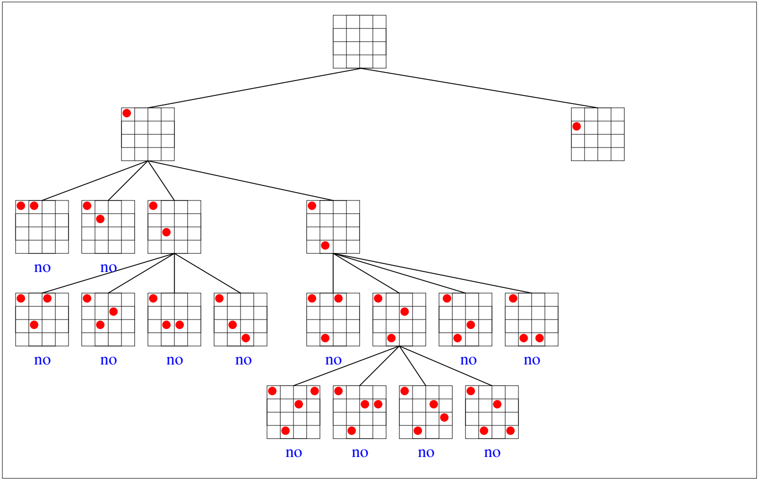
The queens problem: BackTrack Algorithm



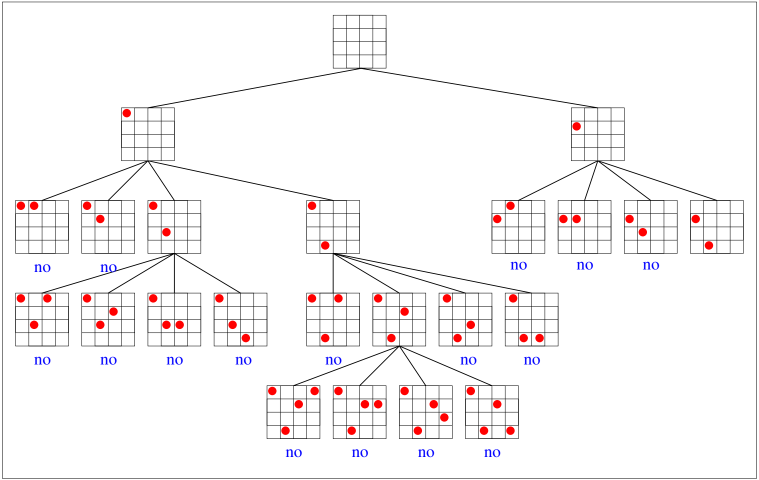
The queens problem: BackTrack Algorithm



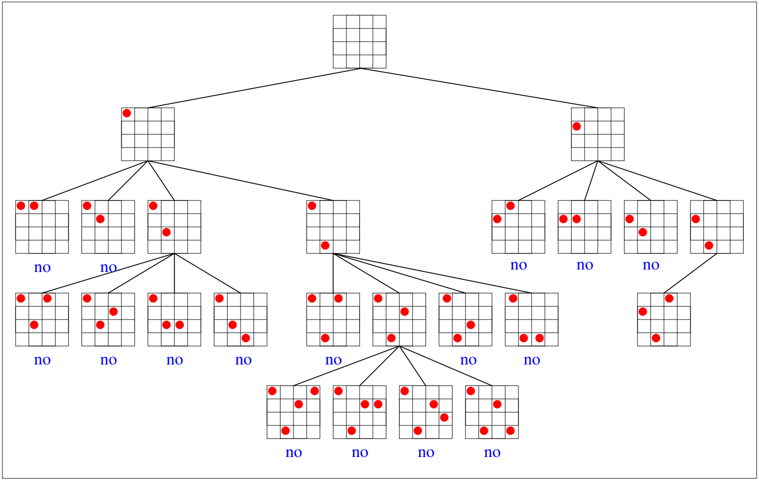
The queens problem: BackTrack Algorithm



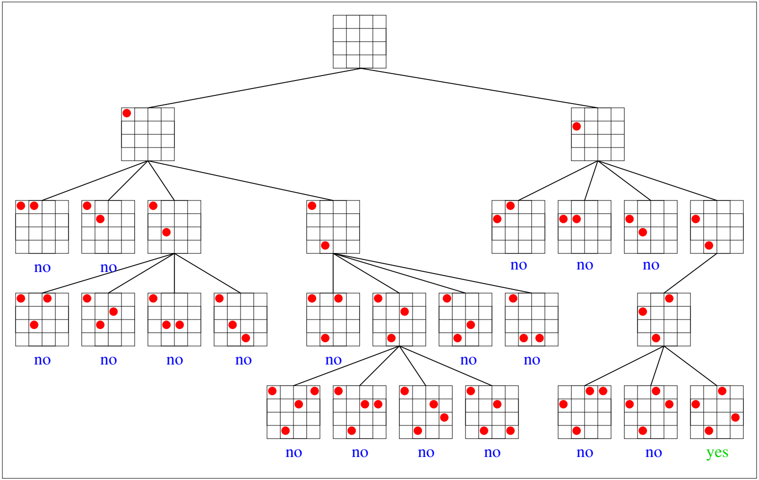
The queens problem: BackTrack Algorithm



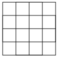
The queens problem: BackTrack Algorithm



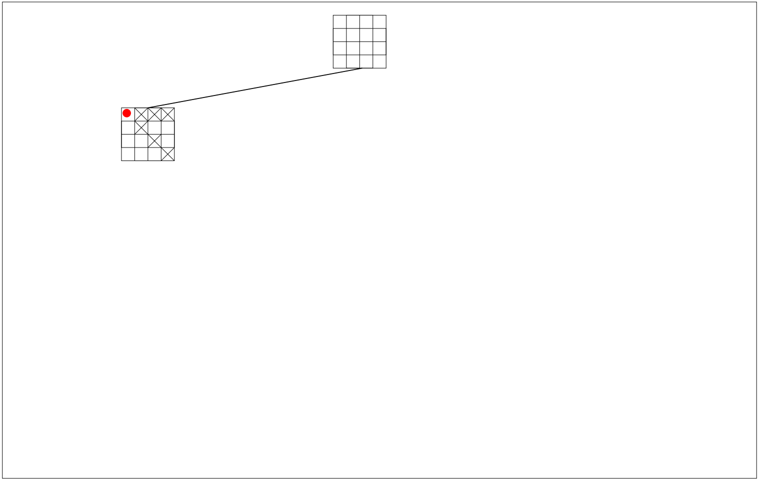
The queens problem: BackTrack Algorithm



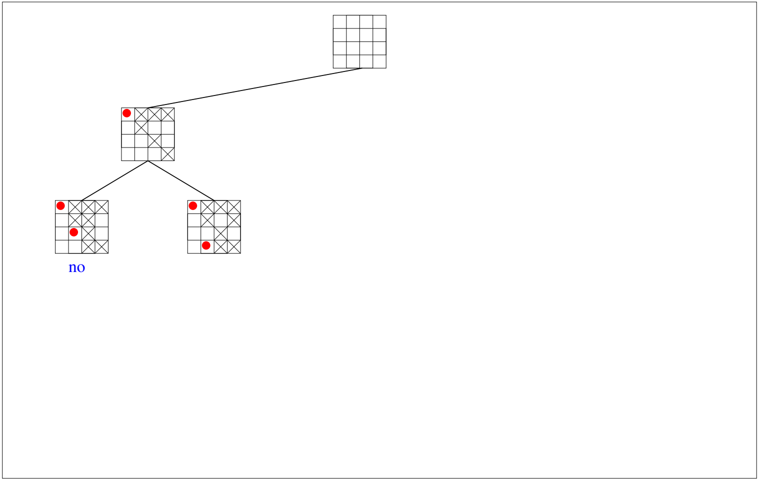
The queens problem: BT Algorithm with Node Consistency



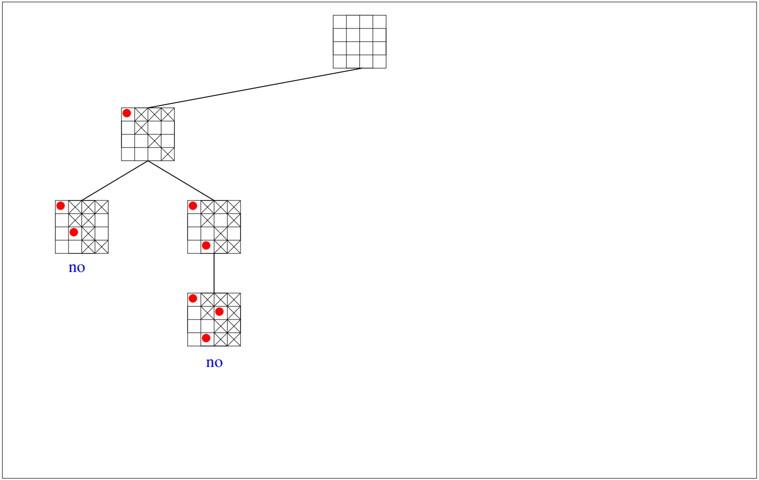
The queens problem: BT Algorithm with Node Consistency



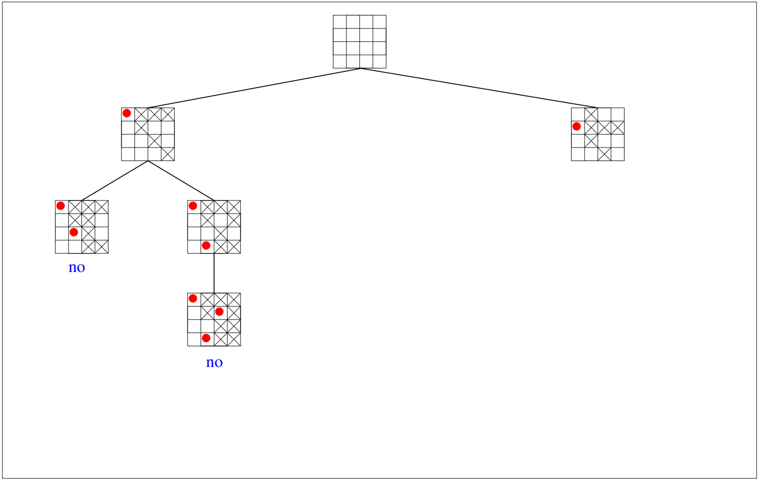
The queens problem: BT Algorithm with Node Consistency



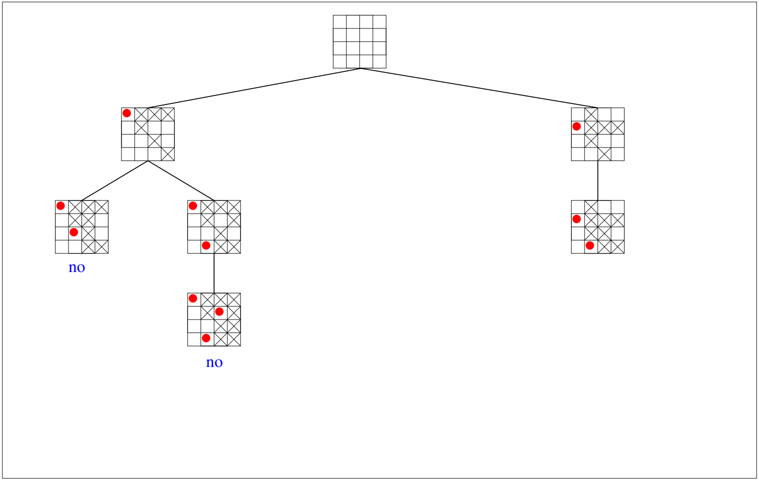
The queens problem: BT Algorithm with Node Consistency



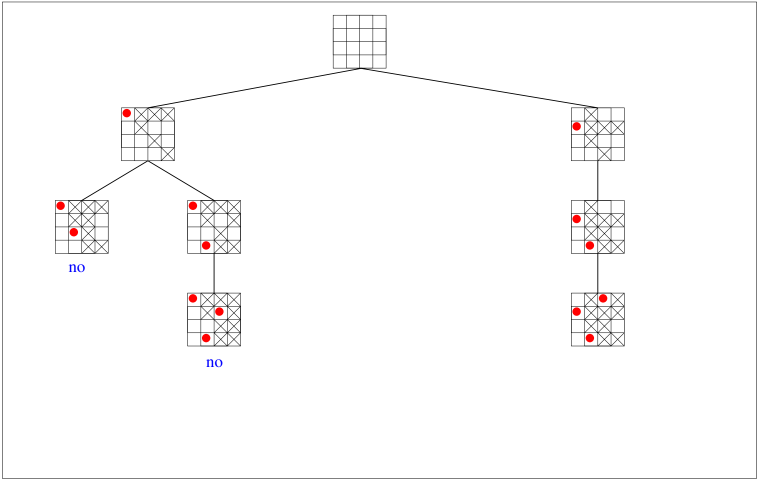
The queens problem: BT Algorithm with Node Consistency



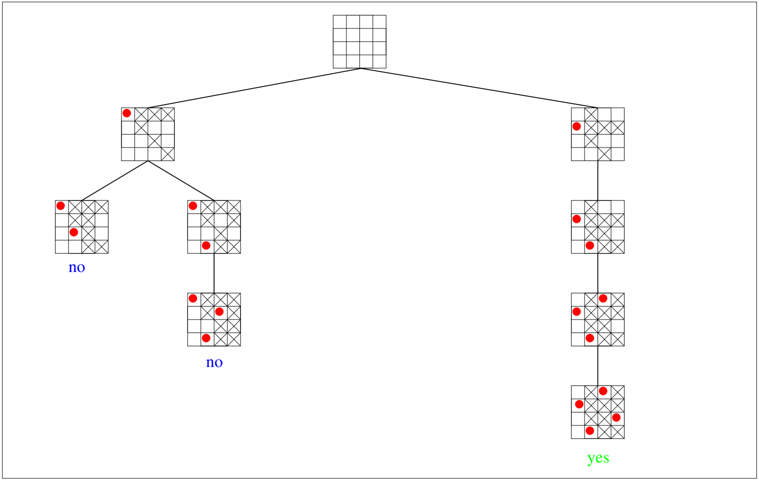
The queens problem: BT Algorithm with Node Consistency



The queens problem: BT Algorithm with Node Consistency



The queens problem: BT Algorithm with Node Consistency



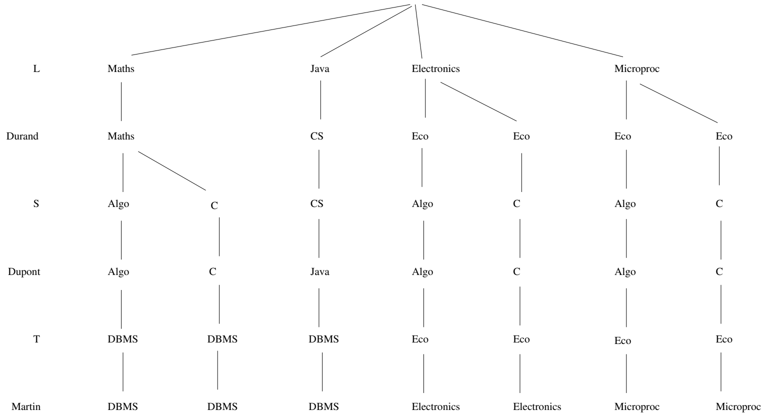
Exercice: Time Table

- 3 sections of students: S,L and T
- 3 professors: Durand, Dupont, Martin
- Durand teaches Computer Science to S, Maths to L and Economy to T
- Dupont teaches Algorithmics and C language to S and Java to L
- Martin teaches Electronics and Microprocessors to L and DBMS to T
- A section can only have one class at a time
- A professor can only teach one class at a time
- A class can only be taught by a professor who knows the subject

Exercice: Time Table

- Solve the time table problem for a class
- (X,D,C)
- X=S, L, T, Durand, Dupont, Martin
- $D_L = (Maths, Java, Electronics, Microproc)$, $D_S = (ComputerScience, Algo, C)$, $D_T = (Economy, DBMS)$, $D_{Durand} = (Maths, ComputerScience, Economy)$, $D_{Dupont} = (Algo, C, Java)$, $D_{Martin} = (Electronics, Microproc, DBMS)$
- $v_1 = (Durand, L)$, $r_1 = (Maths, Maths)$
- $v_2 = (Durand, S)$, $r_2 = (ComputerScience, ComputerScience)$
- $v_3 = (Durand, T)$, $r_3 = (Economy, Economy)$
- $v_4 = (Dupont, S)$, $r_4 = ((Algo, Algo), (C, C))$
- $v_5 = (Dupont, L)$, $r_5 = (Java, Java)$
- $v_6 = (Martin, L)$, $r_6 = ((Electronics, Electronics), (Microproc, Microproc))$
- $v_7 = (Martin, T)$, $r_7 = (DBMS, DBMS)$

Example


L
Maths
Java
Electronics
Microproc
Durand
Maths
CS
Eco
Eco
Eco
Eco
S
Algo
C
CS
Algo
C
Algo
C
Dupont
Algo
C
Java
Algo
C
Algo
C
T
DBMS
DBMS
DBMS
Eco
Eco
Eco
Eco
Martin
DBMS
DBMS
DBMS
Electronics
Electronics
Microproc
Microproc