

Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

January 19, 2006

This exam consists of 5 problems containing 21 subproblems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam. The exam consists of 9 numbered pages.

You can answer the questions in English or Danish.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make acceptable assumptions when a problem has been incompletely specified.

Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

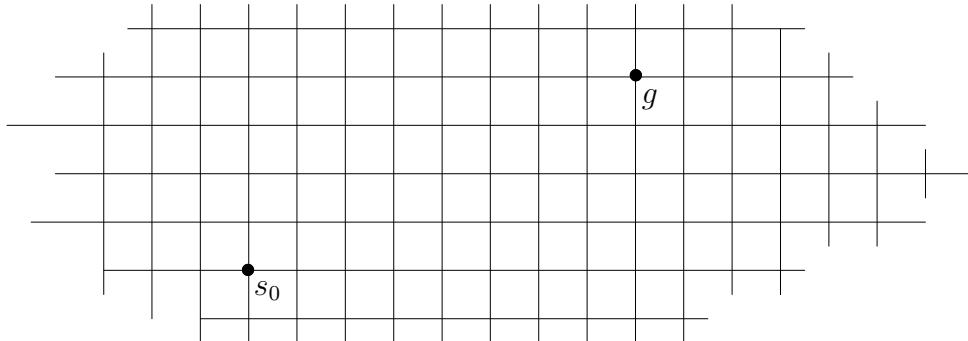
Good luck!

Problem 1 (25%)

Below is a simple search algorithm to determine whether a goal state g can be reached from an initial state s_0 . Q is a first-in-first-out queue. Thus, $\text{ENQUEUE}(Q, s)$ adds s to the end of Q while $\text{DEQUEUE}(Q)$ removes the first element on Q and returns it. The function $\text{SUCCESSOR}(s)$ returns the set of successor states of s .

```
SEARCH( $s_0, g$ )
1    $Q \leftarrow \emptyset$ 
2    $\text{ENQUEUE}(Q, s_0)$ 
3   loop do
4       if  $Q = \emptyset$  then return failure
5        $p \leftarrow \text{DEQUEUE}(Q)$ 
6       if  $p = g$  then return success
7       for each  $c \in \text{SUCCESSOR}(p)$ 
8           do  $\text{ENQUEUE}(Q, c)$ 
```

Assume that the state space forms an infinite grid and that the successors of a state s are the four states immediately left, right, under, and above s . Furthermore, assume that each search problem is defined by randomly selecting an initial state s_0 and goal state g such that there is a shortest path between them with length k . In the example below, we have $k = 12$.



a) Is **SEARCH** sound and complete when applied to the problems defined above? Argue for your claims.

a) **Answer:** Essentially **SEARCH** is BFS without a closed set. Thus, the algorithm is sound. It is also complete since the BFS frontier eventually will reach g .

Recall that the computation in lines 7 and 8 of **SEARCH** is called to *expand* a state.

b) What is the *exact* number of expansions of SEARCH as a function of k in the worst case on these problems? Are the problems hard or easy for SEARCH?

b) Answer: Since the branching factor is 4 for all states and we may find g as the last state at depth k , we get $1 + 4^1 + \dots + 4^k - 1$ expansions in the worst case. The subtraction of 1 in the end is due to the fact that g itself is not expanded. The problems are hard for SEARCH since the number of expanded states grows exponentially with k .

We want to improve the performance of SEARCH by reducing the number of expansions.

c) Write an extension of SEARCH called SEARCH' that significantly reduces the number of expanded states.

c) Answer: We add a closed set C to SEARCH to avoid re-expanding already expanded states.

```

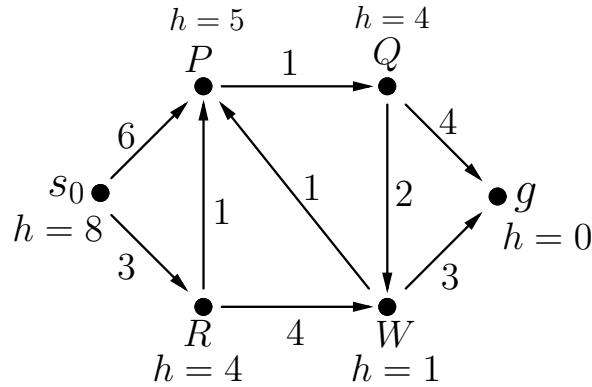
SEARCH'(s0, g)
1   Q ← ∅
2   C ← ∅
3   ENQUEUE(Q, s0)
4   loop do
5     if Q = ∅ then return failure
6     p ← DEQUEUE(Q)
7     if p ∉ C then
8       if p = g then return success
9       for each c ∈ SUCCESSOR(p)
10      do ENQUEUE(Q, c)
11      C ← C ∪ {p}

```

d) What is the *exact* number of expansions of your algorithm SEARCH' as a function of k in the worst case on these problems? Are the problems hard or easy for SEARCH'?

d) Answer: Since SEARCH' never re-expands a state, the states expanded before g in the worst case is all states with Manhattan distance less than or equal to k excluding g . Thus, we get $1 + 4 * 1 + 4 * 2 + 4 * 3 + \dots + 4 * k - 1$ expanded states in the worst case. The problems are easy for SEARCH' since the number of expanded states only grows quadratic with k .

Problem 2 (15%)



Consider the state space illustrated above. Each action edge is labeled with a positive cost and each state is labeled with the value of a heuristic function h estimating the cost of reaching g .

a) Is h admissible? Is h consistent? Argue briefly for your claims.

a) Answer: It is admissible, since the heuristic cost for each state is less than or equal to the actual cost of reaching g . It is not consistent. For instance we have $h(s_0) = 8 > \text{cost}(s_0, R) + h(R) = 7$.

Assume that the tree-search version of A* is applied to find a path from s_0 to g . Recall that this algorithm represents the search fringe by a priority queue Q where each node is a tuple (s, g, h, f) , where s is the state of the node, and g , h , and f are the g , h , and f -cost of the state. Nodes with lowest f -cost are given highest priority. In case of a tie, highest priority is given to the node with lowest h -cost.

b) Can it be concluded from your answer to 2.a that A* returns a shortest path from s_0 to g ?

b) Answer: Yes, h is admissible.

c) Can it be concluded from your answer to 2.a that A* can remove nodes of any previously expanded state from Q without compromising optimality?

c) Answer: No, this is only possible if h is consistent.

d) Finish the table below showing the content of Q in each iteration of A*. What is the path returned by A*?

iteration	Content of Q
0	$(s_0, 0, 8, 8)$
1	...
...	...

d) **Answer:** The path returned by A* is s_0RPQg . The content of Q is shown in the table below.

iteration	Content of Q
0	$(s_0, 0, 8, 8)$
1	$(R, 3, 4, 7), (P, 6, 5, 11)$
2	$(W, 7, 1, 8), (P, 4, 5, 9), (P, 6, 5, 11)$
3	$(P, 4, 5, 9), (g, 10, 0, 10), (P, 6, 5, 11), (P, 8, 5, 13)$
4	$(Q, 5, 4, 9), (g, 10, 0, 10), (P, 6, 5, 11), (P, 8, 5, 13)$
5	$(W, 7, 1, 8), (g, 9, 0, 9), (g, 10, 0, 10), (P, 6, 5, 11), (P, 8, 5, 13)$
6	$(g, 9, 0, 9), (g, 10, 0, 10), (g, 10, 0, 10), (P, 6, 5, 11), (P, 8, 5, 13), (P, 8, 5, 13)$

Problem 3 (20%)

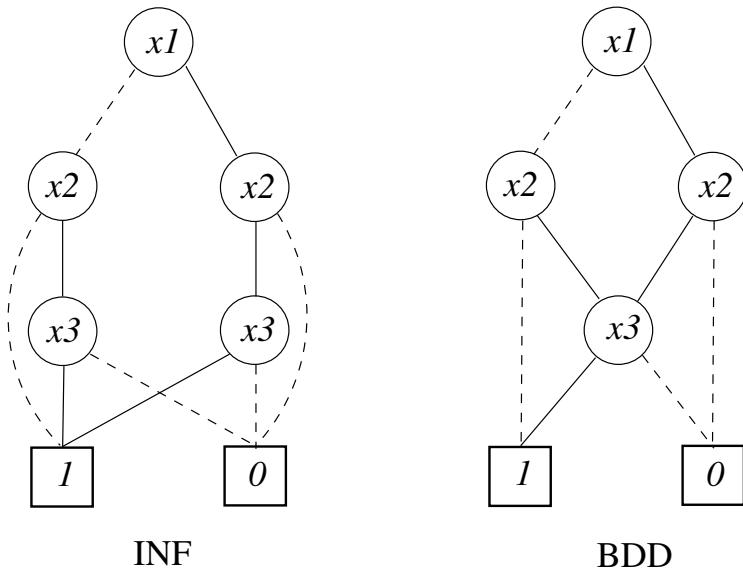
a) Translate $t \equiv (x_1 \Rightarrow x_2) \wedge (x_2 \Rightarrow x_3)$ to if-then-else normal form (INF) using the variable order x_1, x_2, x_3 .

a) Answer: The INF expression of t is shown below.

$$\begin{aligned} t &\equiv x_1 \rightarrow t_1, t_0 \\ t_0 &\equiv x_2 \rightarrow t_{01}, 1 \\ t_1 &\equiv x_2 \rightarrow t_{11}, 0 \\ t_{01} &\equiv x_3 \rightarrow 1, 0 \\ t_{11} &\equiv x_3 \rightarrow 1, 0 \end{aligned}$$

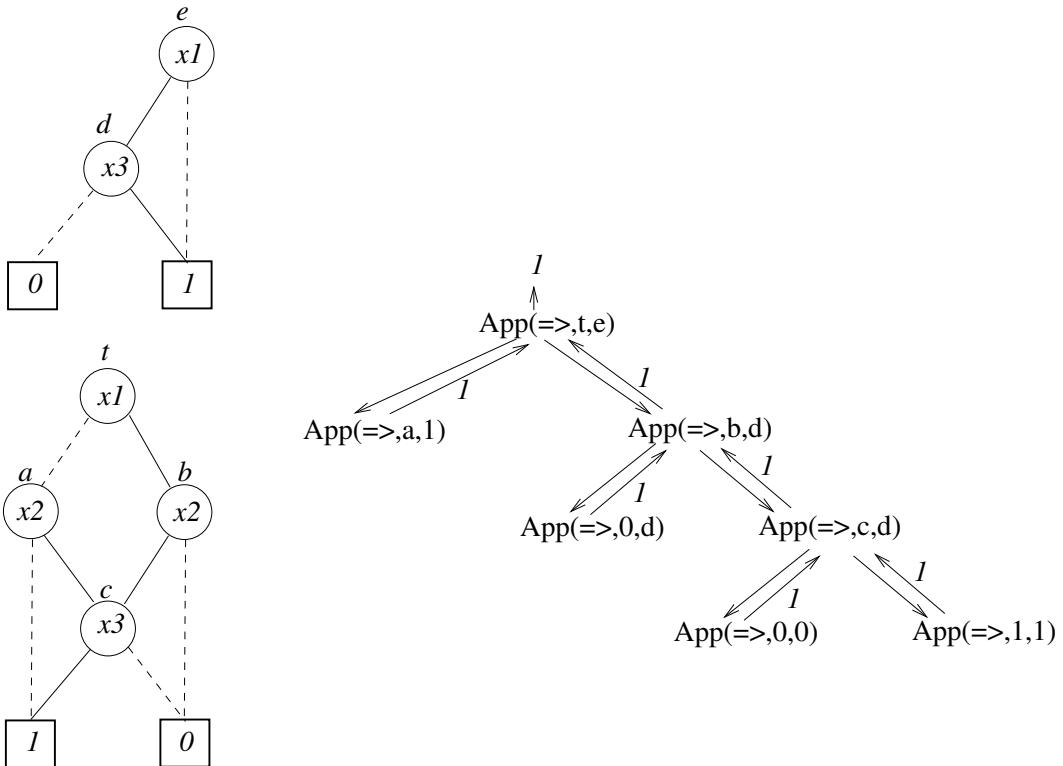
b) Show the binary decision tree of the result in 3.a and reduce it to a reduced ordered binary decision diagram (BDD).

b) Answer: The decision tree and corresponding BDD are shown in the figure below.



c) Build the BDD of $e \equiv x_1 \Rightarrow x_3$ using the same variable order as in question 3.a. Label the internal nodes of the BDD of t (from question 3.a) and e with unique identifiers and draw the call tree of $\text{APPLY}(\Rightarrow, t, e)$. You may assume that APPLY has been implemented with early termination. What is the BDD returned by APPLY ?

c) Answer: The BDD returned by APPLY is 1. The call tree is shown below.



d) What does your result in 3.c imply about the logical relation between t and e ?

d) Answer: It implies that e is a logical consequence of t ($t \models e$).

Problem 4 (25%)

Consider the crossword puzzle shown below.

x_3		x_4
x_1		
x_2		

The words are drawn from the dictionary

$$\{\text{ARH}, \text{NYS}, \text{HAT}, \text{SAM}, \text{MOM}, \text{WHEN}, \text{THIS}, \text{LAOS}, \text{ATOM}\}.$$

The variables x_1 , x_2 , x_3 , and x_4 denote words placed horizontally (x_1, x_2) and vertically (x_3, x_4) in the puzzle. The words can be placed in any space of correct length, but each position can only hold a single letter. Thus, $x_3 = \text{LAOS}$ is consistent with $x_2 = \text{SAM}$ but inconsistent with $x_2 = \text{MOM}$.

a) Write the domains D_1 , D_2 , D_3 , and D_4 of x_1 , x_2 , x_3 , and x_4 .

a) **Answer:** $D_1 = D_2 = \{\text{ARH}, \text{NYS}, \text{HAT}, \text{SAM}, \text{MOM}\}$, $D_3 = D_4 = \{\text{WHEN}, \text{THIS}, \text{LAOS}, \text{ATOM}\}$.

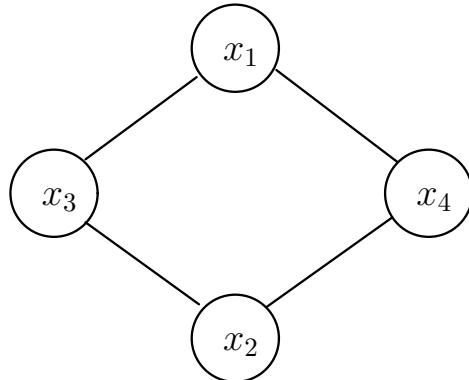
b) Draw the constraint graph of the problem.

b) **Answer:** The constraint graph is shown below.

c) Is the network arc-consistent? If not, write the resulting domains of enforcing arc-consistency.

c) **Answer:** No, it is not arc-consistent. The domains of an arc-consistent network are $D_1 = \{\text{ARH}, \text{HAT}\}$, $D_2 = \{\text{NYS}, \text{SAM}\}$, $D_3 = \{\text{WHEN}, \text{THIS}, \text{LAOS}\}$, and $D_4 = \{\text{THIS}, \text{ATOM}\}$.

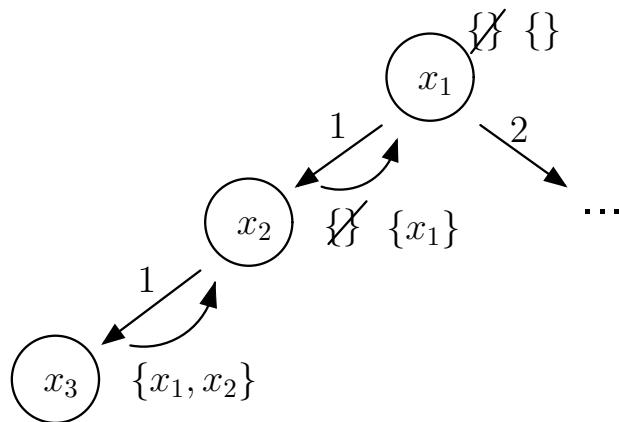
Assume that the variables are ordered (x_1, x_2, x_3, x_4) .



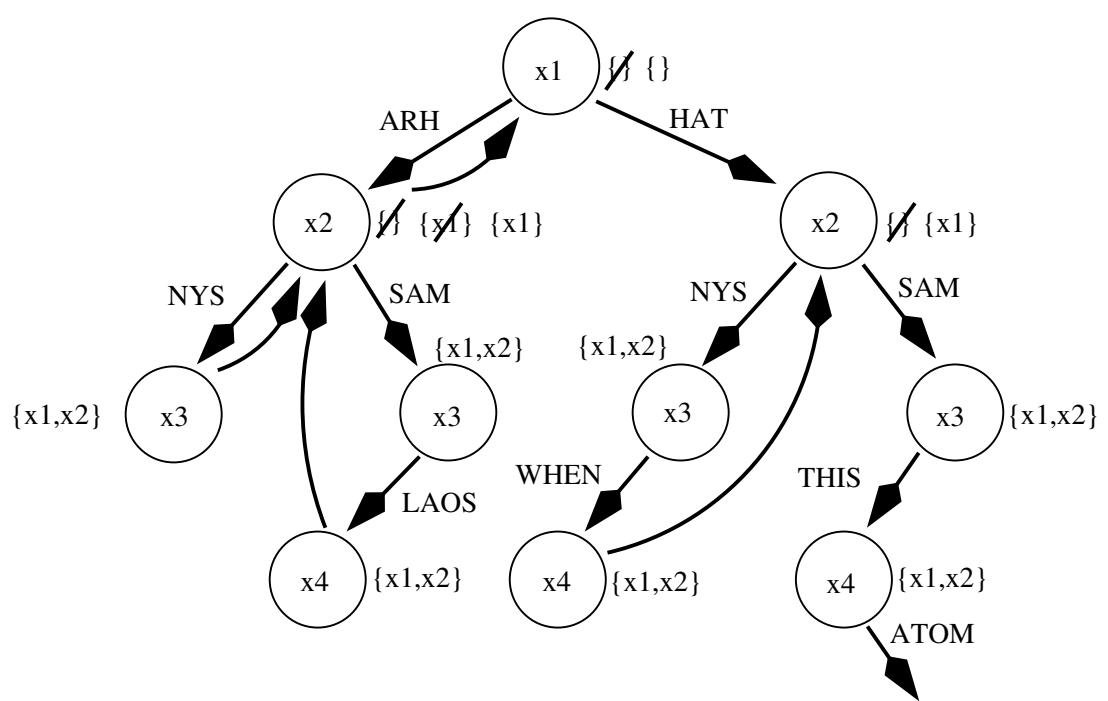
d) Recall that the ancestor set, denoted $anc(x)$, is the subset of the variables that precede and are connected to x . Write the ancestor set of x_1 , x_2 , x_3 , and x_4 .

d) Answer: $anc(x_1) = \{\}$, $anc(x_2) = \{\}$, $anc(x_3) = \{x_1, x_2\}$, and $anc(x_4) = \{x_1, x_2\}$.

e) Draw the search tree traversed by the graph-based backjumping algorithm (GRAPH-BASED-BACKJUMPING) applied to the resulting network in 4.c. Vertices of the tree are associated with variables, and edges denote assigned words. Track the induced ancestor set for each vertex and label each edge with the word assignment it represents. Domain values are assumed to be tried in the order they appear in the dictionary. An example search tree for another problem is shown below.

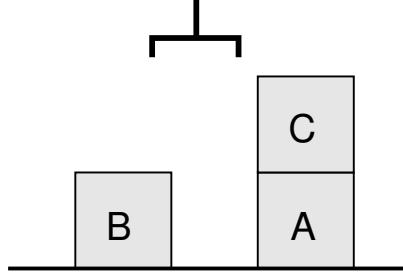


e) Answer: The search tree traversed by GRAPH-BASED-BACKJUMPING is shown below.



Problem 5 (15%)

In this problem, we study the Blocksworld planning domain. In the Blocksworld domain all objects are blocks. The blocks are placed on a table and may be stacked on top of each other by a gripper with a single hand. A state of the Blocksworld with three blocks A, B, and C is shown below.



The following predicates are used to characterize a state.

- $On(x, y)$: x is on top of y .
- $OnTable(x)$: x is on the table.
- $Clear(x)$: x has no block on top of it.
- $HandEmpty$: The hand of the gripper is free.
- $Holding(x)$: The hand of the gripper carries x .

The actions of the Blocksworld domain are defined by the following STRIPS action schemas.

ACTION($PickUp(x)$,
 PRECOND: $Clear(x) \wedge OnTable(x) \wedge HandEmpty$
 EFFECT: $\neg Clear(x) \wedge \neg OnTable(x) \wedge \neg HandEmpty \wedge Holding(x)$)

ACTION($PutDown(x)$,
 PRECOND: $Holding(x)$
 EFFECT: $\neg Holding(x) \wedge Clear(x) \wedge OnTable(x) \wedge HandEmpty$)

ACTION($Stack(x, y)$,
 PRECOND: $Holding(x) \wedge Clear(y)$
 EFFECT: $\neg Holding(x) \wedge \neg Clear(y) \wedge Clear(x) \wedge HandEmpty \wedge On(x, y)$)

ACTION($UnStack(x, y)$,
 PRECOND: $On(x, y) \wedge Clear(x) \wedge HandEmpty$
 EFFECT: $\neg On(x, y) \wedge \neg Clear(x) \wedge \neg HandEmpty \wedge Holding(x) \wedge Clear(y)$)

a) Write a first-order logic formula characterizing the state shown above. (You may want to study the actions to understand what constitutes a valid state.)

a) Answer: $OnTable(B) \wedge OnTable(A) \wedge On(C, A) \wedge Clear(B) \wedge Clear(C) \wedge Handempty$.

b) Write a first-order logic formula f that defines $Holding(x)$ in terms of the other predicates (i.e., $Holding(x) \equiv f$).

b) Answer: $Holding(x) \equiv \neg OnTable(x) \wedge \neg (\exists y On(x, y))$.

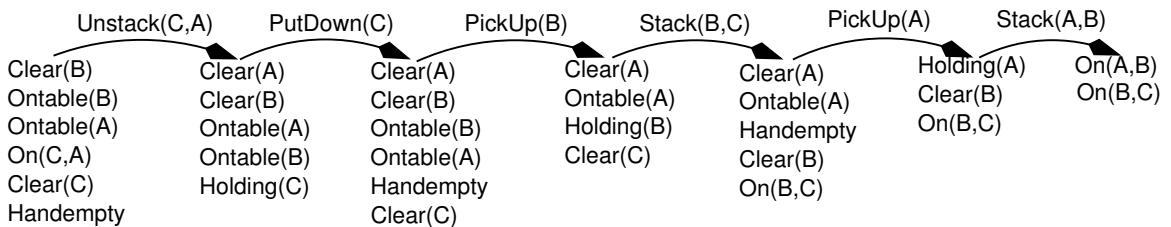
Consider a planning problem where the state defined in problem 5.a is the initial state and the goal states are given by $On(A, B) \wedge On(B, C)$.

c) Write a plan that achieves the goal.

c) Answer: $Unstack(C, A)$, $PutDown(C)$, $PickUp(B)$, $Stack(B, C)$, $PickUp(A)$, $Stack(A, B)$.

d) Draw a single path leading to a solution of the backward search tree of a regression planner applied to this planning problem. The vertices must show intermediate states and edges must be labeled with the applied actions.

d) Answer: A successful path of the search tree path is shown below.



Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

June 4, 2007

This exam consists of 4 problems containing 20 subproblems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam. The exam consists of 10 numbered pages.

You can answer the questions in English or Danish.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

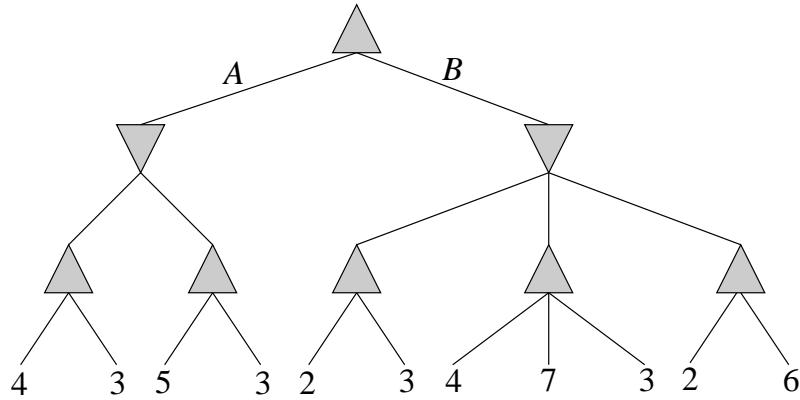
Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

Good luck!

Problem 1: Game Trees (10%)



In the game tree above, MAX can choose between two actions *A* and *B*.

a) What is the minimax value of MAX?

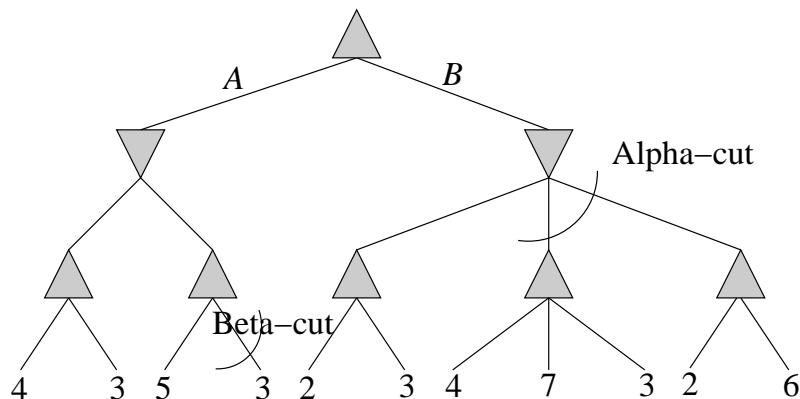
a) Answer: 4.

b) What is the minimax decision of MAX?

b) Answer: *A*.

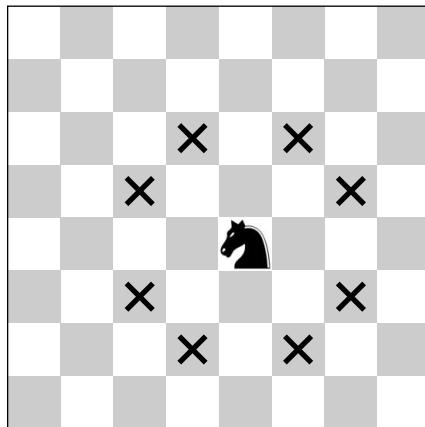
c) Which parts of the game tree are pruned by the ALPHA BETA SEARCH algorithm? Mark whether the pruning is due to α or β cuts.

c) Answer:

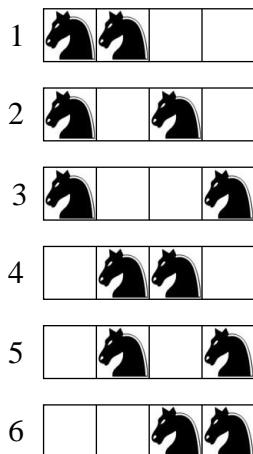


Problem 2: Constraint Programming (30%)

Recall that a Knight can move as shown in the figure below.



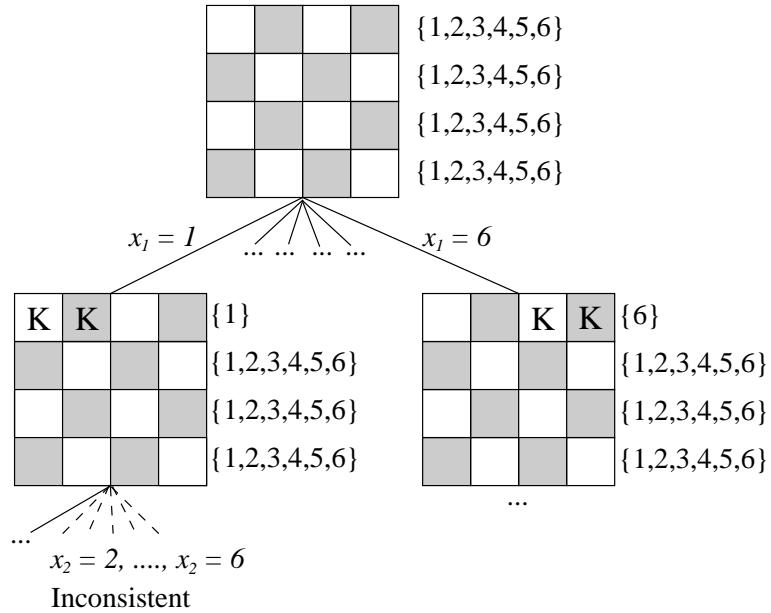
Consider placing eight Knights on a 4×4 chess board such that no Knights can capture each other and each row contains exactly two Knights. We will model this problem as a *Constraint Satisfaction Problem* (CSP). There are four variables x_1, x_2, x_3 , and x_4 , one for each row counted from the top. Each variable x_i has domain $D_i = \{1, 2, 3, 4, 5, 6\}$. The domain values denote the six different configurations of the Knights in the row as defined in the figure below.



- a) Assume that the domains have been pruned such that $D_1 = \{3, 5\}$, $D_2 = \{2, 3, 4\}$, $D_3 = \{3\}$, and $D_4 = \{1, 2, 3, 4, 5, 6\}$. Is this CSP arc consistent (why / why not)?

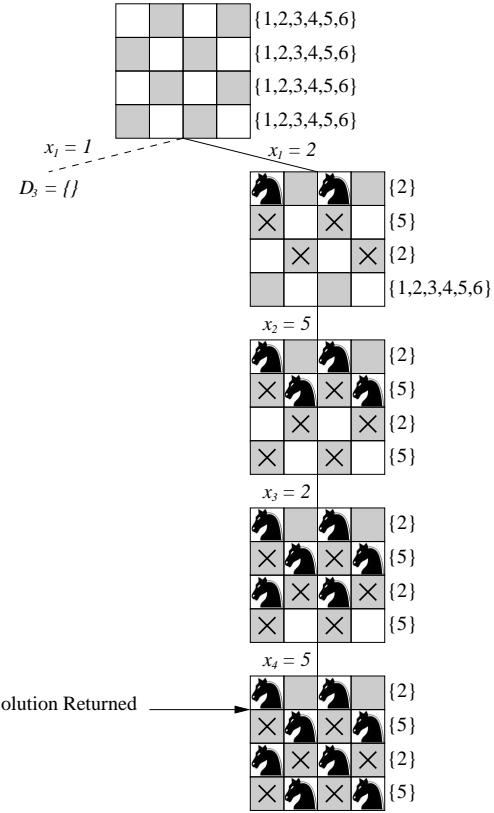
- a) **Answer:** the CSP is not arc consistent. For instance $x_2 = 2$ does not have any support in D_3 .

In the following, we will examine the search trees of different algorithms. We assume that each search node is drawn as a board with the configuration of Knights passed from the parent node and the remaining domain values written to the right of each row. We only draw a search node if the recursive call associated with it took place. Each edge is drawn as a line labeled with the assignment associated with it. We also draw edges for assignments that failed. We use dashed lines for these edges and give a brief explanation why the assignment failed. As an example, the figure below shows the root node and two children generated by the BACKTRACKING algorithm.



- b)** Draw the search tree of the FORWARD CHECKING algorithm. Assume that variables are assigned in the order $x_1 < x_2 < x_3 < x_4$ and that values are tried in the order $1 < 2 < \dots < 6$. Mark the node containing the solution returned.

b) Answer:



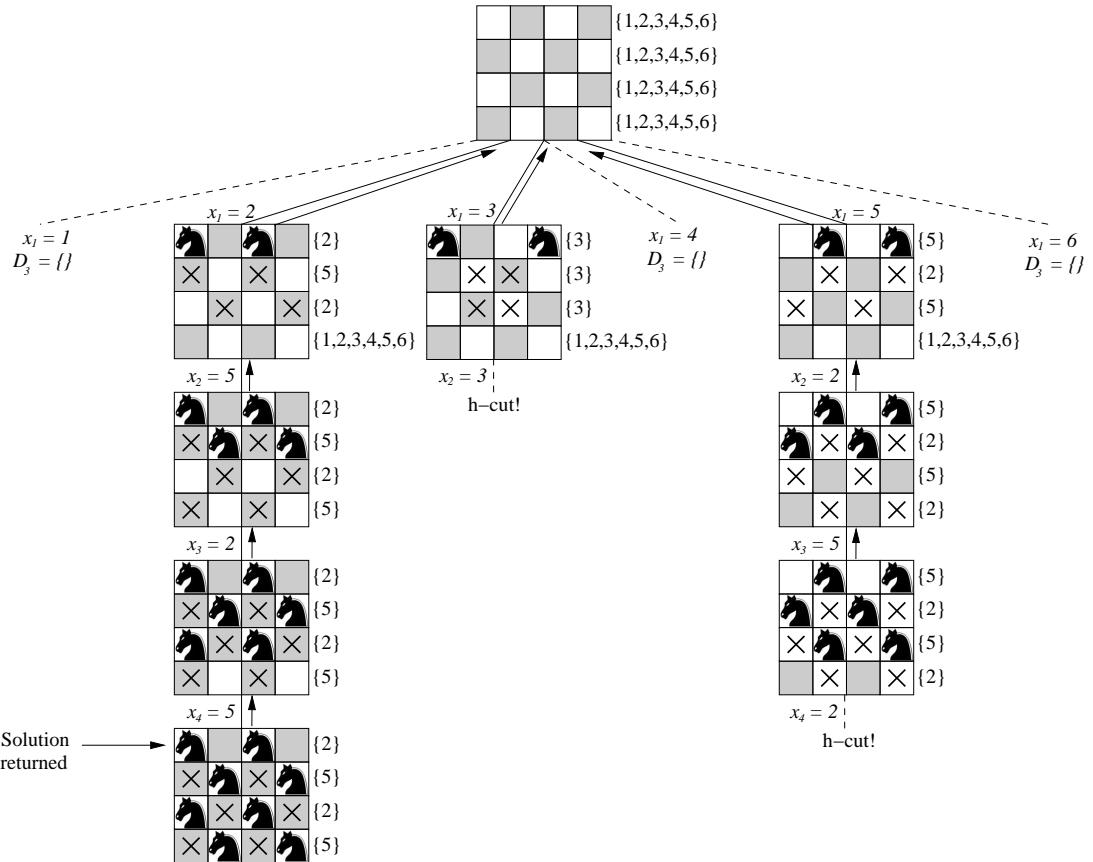
Assume that we want a solution with a minimum number of Knights in the left-most column. To solve this problem, we extend the CSP to a *Constraint Optimization Problem* (COP). Given a complete valid assignment of the variables, the objective function f is the number of Knights in the left-most column. The objective is to minimize f .

c) Define in words a heuristic function h that can be used to prune the search space by the BRANCH-AND-BOUND algorithm without compromising optimality.

c) Answer: h is the number of Knights in the left-most column in a partial assignment of the variables.

d) Draw the search tree of the BRANCH-AND-BOUND algorithm using FORWARD CHECKING as its underlying search algorithm. Again, assume that variables are assigned in the order $x_1 < x_2 < x_3 < x_4$ and that values are tried in the order $1 < 2 < \dots < 6$. Mark the node containing the solution returned.

d) Answer:



e) How many times is the search tree of the BRANCH-AND-BOUND algorithm pruned due to $Bound \leq h$?

e) Answer: 2 times.

Problem 3: Decision Trees (30%)

On a given day, Bob will decide to play tennis depending on

- whether it is a warm day (W),
- whether it is a dry day (D), and
- whether he feels fresh (F).

Bob will play tennis if it is a warm and dry day no matter how he feels, but if he feels fresh, he will play if it is just a warm or dry day, but not necessarily both. Bob will not play tennis on any other day.

a) Write a Boolean function $t(W, D, F)$ that evaluates to true if and only if the proposition symbols W , D , and F are assigned to truth-values corresponding to days where Bob will play tennis.

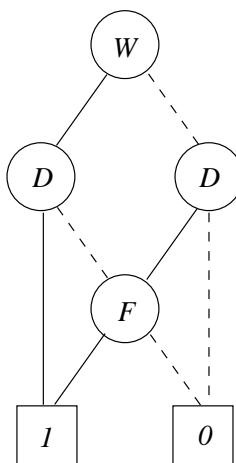
a) Answer: $t(W, D, F) = (W \wedge D) \vee (F \wedge (W \vee D))$

b) Write the expression of t in *If-then-else Normal Form* (INF) using the order $W < D < F$.

b) Answer: $W \rightarrow (D \rightarrow 1, (F \rightarrow 1, 0)), (D \rightarrow (F \rightarrow 1, 0), 0).$

c) Draw the *Reduced Ordered Binary Decision Diagram* (ROBDD) of t using the order $W < D < F$.

c) Answer:



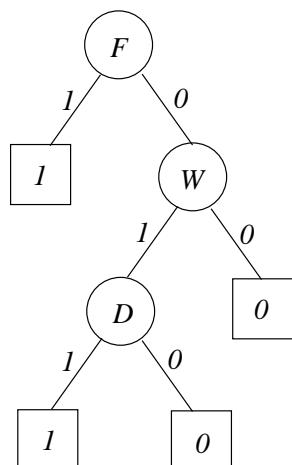
Example	W	D	F	$t(W, D, F)$
1	1	0	0	0
2	0	1	1	1
3	1	1	0	1
4	1	0	1	1
5	0	1	0	0

The table above shows negative and positive training examples for t . Consider using the DECISION TREE LEARNING algorithm (DTL) to generate a decision tree from the examples. You can assume that DTL picks the first attribute according to the order $W < D < F$ if several attributes have maximum information gain.

- d) Draw the decision tree returned by DTL. You may find the entropy table below helpful.

Examples		
Positive	Negative	Entropy
3	2	0.97
2	3	0.97
2	0	0.00
2	2	1.00
2	1	0.92
1	2	0.92
1	1	1.00
1	0	0.00
0	1	0.00

- d) Answer The decision tree returned by DTL is shown below.

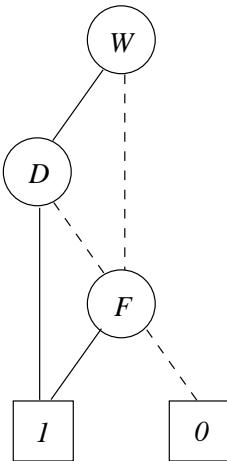


e) What is the information gain for each attribute W , D , and F for the root node?

e) **Answer** For the root node, we have $Gain(W) = 0.02$, $Gain(D) = 0.02$, and $Gain(F) = 0.42$.

f) Draw a ROBDD of the decision tree returned by DTL using variable order $W < D < F$. Can you from this ROBDD conclude that the decision tree returned by DTL is equal to t (why / why not)?

f) **Answer** The BDD of the decision tree is shown below. This BDD is different from the BDD of t , so due to the uniqueness of BDDs, we can conclude that the decision tree returned by DTL is unequal to t .



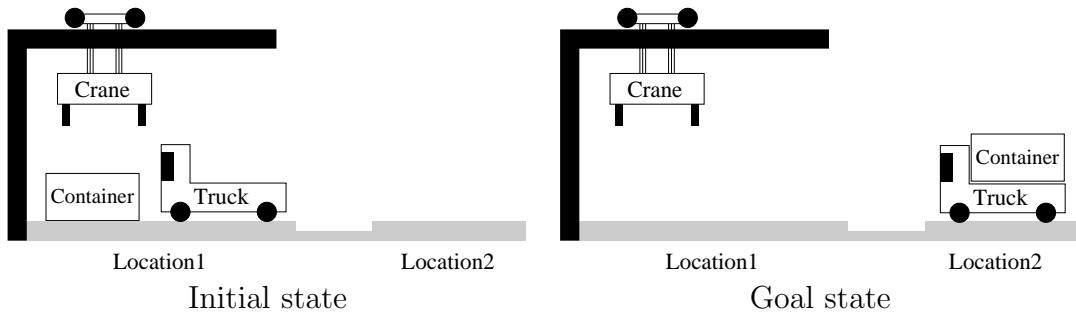
Problem 4: Heuristic Search (30%)

A *planning problem* is a special kind of search problem where each state is a set of true propositions and the successor function is given by a set of action descriptions. Formally, a planning problem is a 4-tuple (L, s_0, g, A) , where

- L is a finite set of proposition symbols,
- $s_0 \subseteq L$ is the initial state,
- $g \subseteq L$ is a goal state, and
- A is a finite set of actions, where each action $a \in A$ is a triple of subsets of L , which we write as $a = (pre(a), del(a), add(a))$. The set $pre(a)$ is called the *precondition* of the action, while the sets $del(a)$ and $add(a)$ together are called the *effect* of the action. An action a can only be applied in a state s if $pre(a) \subseteq s$. The result of applying a in s is a state s' given by $s' = (s \setminus del(a)) \cup add(a)$.

A plan of length n solving a planning problem (L, s_0, g, A) is an alternation of states and actions $s_0a_1s_1a_2s_2 \cdots a_{n-1}s_{n-1}a_ns_n$ where $s_n = g$ and $pre(a_i) \subseteq s_{i-1}$ for $i = 1, \dots, n$ and $s_i = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$ for $i = 1, \dots, n$.

We will consider a particular planning problem \mathcal{P} that involves a *truck*, a *crane*, and a *container*. The truck can move between two locations called *location1* and *location2*. The crane is permanently at *location1* and can be used to load the container onto the truck. Initially, the truck and container is at *location1* and the goal is to load the container onto the truck and drive it to *location2*. The initial state and goal state are shown below.



Formally, the problem is given by $\mathcal{P} = (L, s_0, g, A)$, where

$L = \{onground, ontruck, holding, at1, at2\}$, where
 $onground$ means that the container is on the ground,
 $ontruck$ means that the container is on the truck,
 $holding$ means that the crane is holding the container,
 $at1$ means that the truck is at *location1*, and

$at2$ means that the truck is at *location2*.

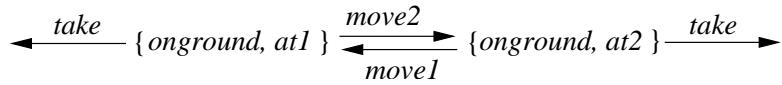
$$s_0 = \{onground, at1\},$$

$$g = \{ontruck, at2\}, \text{ and}$$

$A = \{take, put, load, unload, move1, move2\}$, where

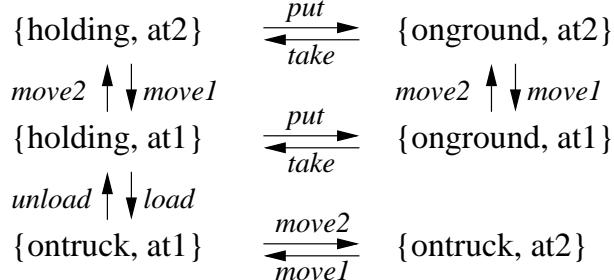
$$\begin{aligned} take &= (\{onground\}, \{onground\}, \{holding\}), \\ put &= (\{holding\}, \{holding\}, \{onground\}), \\ load &= (\{holding, at1\}, \{holding\}, \{ontruck\}), \\ unload &= (\{ontruck, at1\}, \{ontruck\}, \{holding\}), \\ move1 &= (\{at2\}, \{at2\}, \{at1\}), \text{ and} \\ move2 &= (\{at1\}, \{at1\}, \{at2\}). \end{aligned}$$

The figure below shows a part of the state space of \mathcal{P} .



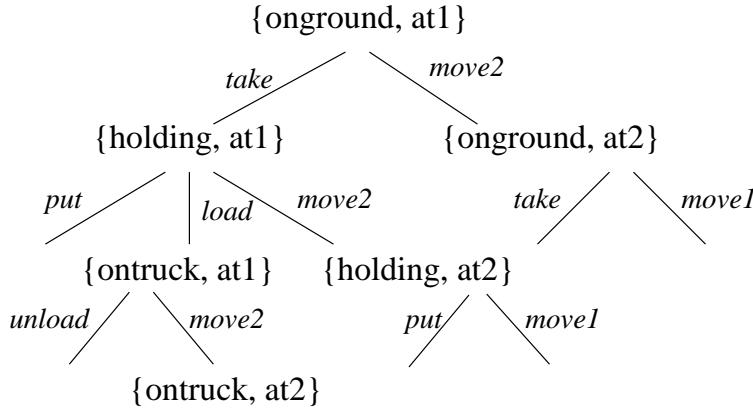
a) Draw the complete state space of \mathcal{P} .

a) Answer:



b) Draw the search tree of the graph-search version of the BREADTH-FIRST SEARCH algorithm (BFS) used to solve \mathcal{P} . Assume that actions are tried in the order $take < put < load < unload < move1 < move2$. What is the plan returned by BFS?

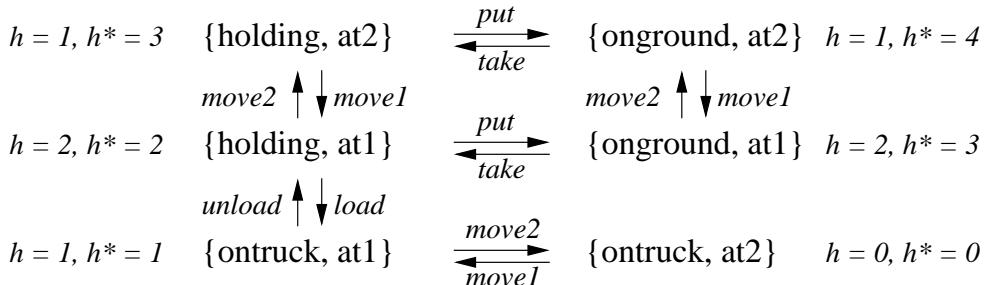
b) Answer: The plan returned by BFS is $\{onground, at1\}, take, \{holding, at1\}, load, \{ontruck, at1\}, move2, \{ontruck, at2\}$. The search tree of BFS is shown below.



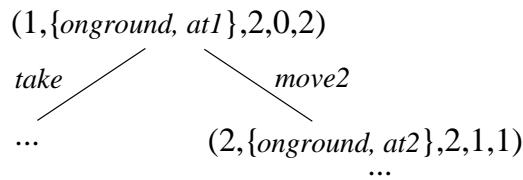
Assume that we want to find minimum length plans. Consider the heuristic function $h(s) = |g \setminus s|$. For the initial state we have $h(s_0) = |g \setminus s_0| = |\{ontruck, at2\} \setminus \{onground, at1\}| = |\{ontruck, at2\}| = 2$.

c) Show that h is an admissible heuristic for \mathcal{P} .

c) Answer: As shown below, we have $h(s) \leq h^*(s)$ for all states s , where $h^*(s)$ is the minimum length of a plan from s to the goal state g .

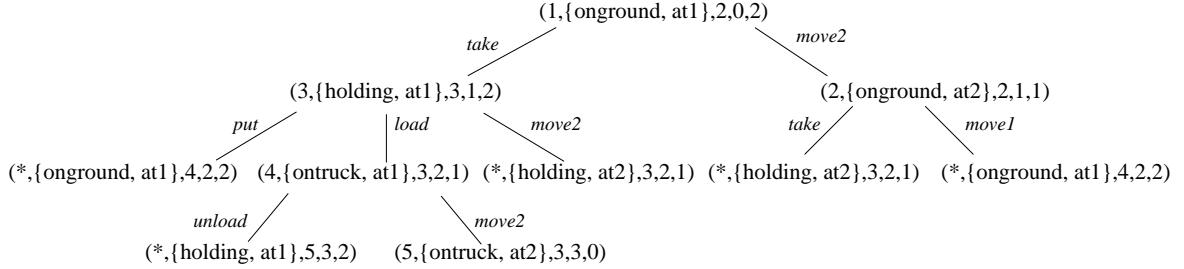


Consider using the tree-search version of A* to solve \mathcal{P} . Let every node n in the search tree be represented by a 5-tuple $(it, s, f(n), g(n), h(s))$, where it is the number of the iteration where n is expanded, s is the state associated with n , and $f(n)$ and $g(n)$ are the f -cost and g -cost of n , respectively. Let $it = *$ if the node is still on the fringe when A* terminates. As an example, the figure below shows the root node and one of its children.



d) Draw one of the possible search trees traversed by the tree-search version of A* used to solve \mathcal{P} .

d) Answer:



Alice claims that h remains admissible even if the goal state g in \mathcal{P} is substituted with a random state in the state space of \mathcal{P} .

e) Argue that Alice is correct or show a counter example.

e) Answer: Alice is correct. Each action can at most produce one goal proposition in g , so when reaching a state s' from a state s with action a , we always have $h(s) - h(s') \leq 1$ or equally $h(s) \leq c(s, a, s') + h(s')$. Thus, h is consistent and therefore admissible for any goal state.

Bob claims that actually h is *universally admissible*. That is, h is admissible for any correctly defined planning problem.

f) Argue that Bob is correct or show a counter example.

f) Answer: Bob is wrong. Counter example $L = \{a, b, c, d\}$, $s_0 = \{a, b\}$, $g = \{c, d\}$, and $A = \{\alpha\}$, where $\alpha = (\{a, b\}, \{a, b\}, \{c, d\})$. We have $h(s_0) = 2$, but $h^*(s_0) = 1$.

Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

4 June 2008

This exam consists of 5 numbered pages with 4 problems containing 13 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam (9:00-13:00).

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

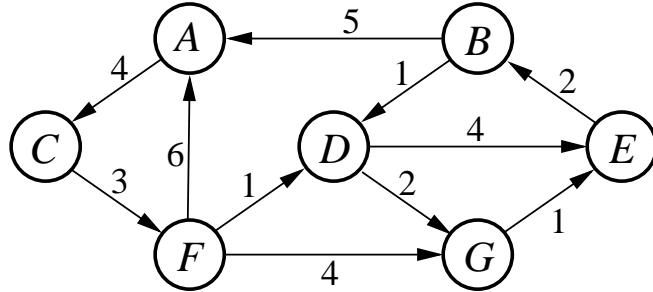
Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

Good luck!

Problem 1: Informed Search (30%)



In the graph above, vertices denote states and edges denote actions. More precisely, there is an edge $s \xrightarrow{c} s'$ between the vertices of two states s and s' if there exists some action a with cost c such that $(a, s') \in \text{SUCCESSOR-FN}(s)$.

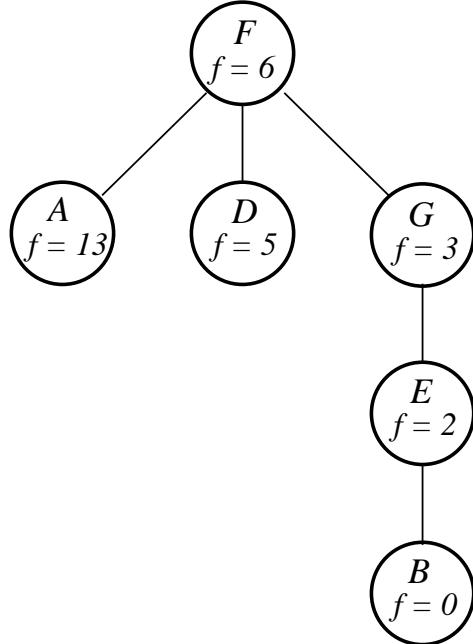
Consider a search problem where F is the initial state and B is the goal state.

- a) For each state, write the value of an admissible heuristic function. The zero function will not be accepted.

a) **Answer:** We choose the perfect heuristic that for each state is equal to the minimum distance to the goal. Since it never overestimates the goal distance, it is admissible. $h(A) = 13$, $h(B) = 0$, $h(C) = 9$, $h(D) = 5$, $h(E) = 2$, $h(F) = 6$, and $h(G) = 3$.

- b) Draw a possible search tree of the TREE-SEARCH version of greedy best-first search using your heuristic function. For each search node, write the f -value and state associated with the node. What is the returned solution? Is the returned solution optimal?

b) Answer: A possible search tree is shown below. The returned solution is (F, G, E, B) which is sub-optimal. The only optimal solution is (F, D, G, E, B) .



c) Would the GRAPH-SEARCH version of A* return an optimal solution using your heuristic? (why / why not)?

c) Answer: Yes, since the heuristic equals the minimum distance to the goal, it never overestimates the cost of a single action. The heuristic is therefore consistent, which means that the GRAPH-SEARCH version of A* returns an optimal solution.

Problem 2: Planning (15%)

Consider the propositional STRIPS planning domain given by the four actions shown below.

Action(A)
PRECOND: $v \wedge w$
EFFECT: $\neg v$

Action(B)
PRECOND:
EFFECT: p

Action(C)
PRECOND: w
EFFECT: $p \wedge \neg v$

Action(D)
PRECOND: $p \wedge q$
EFFECT: $v \wedge w$

Assume that a state-space regression planner is given the goal $p \wedge v$.

a) Which of the actions are relevant for this goal?

a) Answer: B , C , and D .

b) Which of the actions are consistent for this goal?

b) Answer: B and D .

c) For each backward applicable action, write the new goal achieved by regressing the current goal $p \wedge v$ through the action.

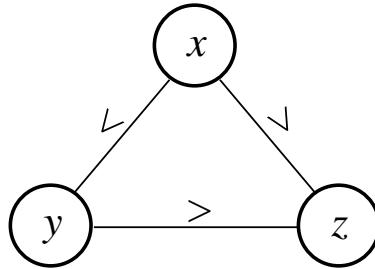
c) Answer: Since an action is backward applicable if it is relevant and consistent, we get that B and D are backward applicable. The new goal of B is v and the new goal of D is $p \wedge q$.

Problem 3: Constraint Programming (25%)

A Constraint Satisfaction Problem (CSP) has three variables x , y , and z each with domain $\{1, 2, 3, 4\}$. The constraints of the CSP are: $x > z$, $x > y$, and $y > z$.

- a) Draw the constraint graph of the CSP.

a) Answer:



Let D_x , D_y , and D_z denote the reduced domains of x , y and z after running the arc consistency algorithm AC-3 on the CSP.

- b) Write the value sets of D_x , D_y , and D_z .

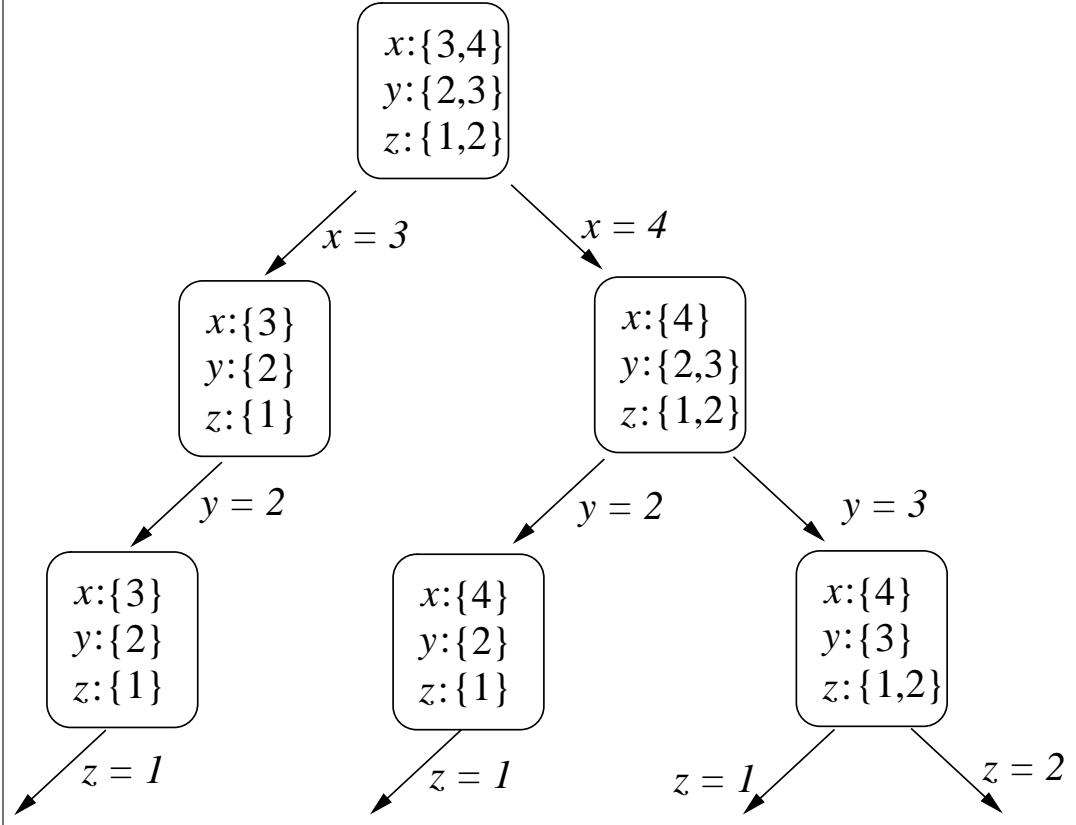
b) Answer: $D_x = \{3, 4\}$, $D_y = \{2, 3\}$, and $D_z = \{1, 2\}$

Recall that the Maintaining Arc Consistency algorithm (MAC) extends the forward checking algorithm by maintaining arc consistency between all variables before the beginning of the search process and after every variable assignment.

- c) Draw the complete search tree explored by the MAC algorithm. Assume that variables are assigned in the order x , y , and z . Each leaf of the tree should correspond to a valid assignment. List for each node the remaining domain of each variable after maintaining arc consistency. Label each edge with the variable assignment it is associated with. Thus, the root of the search tree should be drawn as shown below.

$x: D_x$
 $y: D_y$
 $z: D_z$

c) Answer:



Problem 4: Logic and ROBDDs (30%)

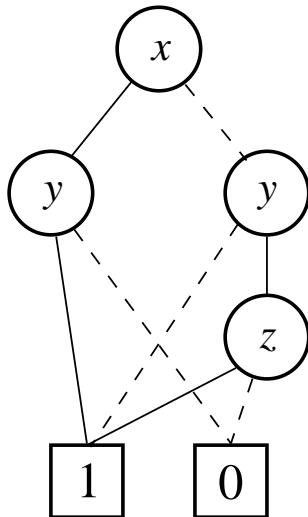
Consider the propositional sentence $(x \Leftrightarrow y) \vee (z \wedge y)$.

- a) Write an expression in If-then-else Normal Form (INF) that is logically equivalent to the sentence.

a) Answer: $x \rightarrow (y \rightarrow 1, 0), (y \rightarrow (z \rightarrow 1, 0), 1)$

- b) Draw the Reduced Ordered Binary Decision Diagram (ROBDD) using the variable order $x < y < z$ that is logically equivalent to the sentence.

b) Answer:



- c) List the variable assignments that satisfy the sentence.

c) Answer: $(x = 1, y = 1, z = 0), (x = 1, y = 1, z = 1), (x = 0, y = 0, z = 0), (x = 0, y = 0, z = 1), (x = 0, y = 1, z = 1)$

Recall that each BDD node is given a unique id and that the ids of the 0 and 1-terminal node are 0 and 1, respectively.

d) Assume that we are dealing with OBDDs (i.e., binary decision diagrams that are ordered, but not necessarily reduced). Write an algorithm that determines whether an OBDD is satisfiable. The input to your algorithm is the id of the root node of the OBDD and you are only allowed to use the functions *low()* and *high()* to traverse it. You achieve extra credit for your answer if the complexity of your algorithm is polynomial in the size of the OBDD.

d) Answer:

```
SAT( $u$ )
  init( $G$ )
  function SATrec( $u$ )
    if  $G(u) \neq \text{empty}$  then
      return  $G(u)$ 
    else
      if  $u \in \{0, 1\}$  then
         $r \leftarrow u$ 
      else
         $r \leftarrow \text{SATrec}(\text{low}(u)) \vee \text{SATrec}(\text{high}(u))$ 
         $G(u) \leftarrow r$ 
      return  $r$ 
  SATrec( $u$ )
```

Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

2 June 2009

This exam consists of 6 numbered pages with 4 problems containing 14 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam (9:00-13:00).

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

Good luck!

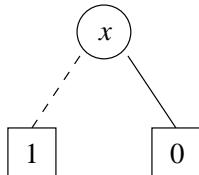
Problem 1: ROBDDs (25%)

Professor Smart has written the following algorithm `SIZE` that counts the number of internal (i.e., non-terminal) nodes of an ROBDD u :

```
SIZE( $u$ )
1: if  $u \in \{0, 1\}$  then
2:     return 0
3: else
4:     return  $1 + \text{SIZE}(\text{low}(u)) + \text{SIZE}(\text{high}(u))$ .
```

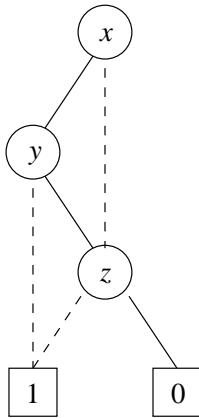
a) Draw an ROBDD where `SIZE` finds the correct number of internal nodes.

a) Answer: `SIZE` finds the correct number of internal nodes if the internal nodes of the ROBDD forms a tree e.g.:



b) Draw an ROBDD where `SIZE` finds an incorrect number of internal nodes.

b) Answer: `SIZE` finds an incorrect number of internal nodes if the internal nodes of the ROBDD does not form a tree e.g.:



c) Write a new version of $\text{SIZE}(u)$ that returns the correct number of internal nodes for an arbitrary ROBDD u . You get extra credit if your algorithm runs in $O(n)$, where n is the number of internal nodes. *Hint:* you may want to use a node set M with the following constant time operations:

$init(M)$: Initializes M to the empty set.
 $M(u)$: Returns *True* if ROBDD u is a member of M and *False* otherwise.
 $M \leftarrow u$: Inserts ROBDD node u into M .
 $size(M)$: Returns the number of nodes in M .

c) Answer:

$\text{SIZE}(u)$
1: $init(M)$
2: $\text{SIZEREC}(u)$
3: **return** $size(M)$

$\text{SIZEREC}(u)$
1: **if** $u \notin \{0, 1\} \wedge \neg M(u)$ **then**
2: $M \leftarrow u$
3: $\text{SIZEREC}(low(u))$
4: $\text{SIZEREC}(high(u))$

Problem 2: Logic (20%)

In this problem, we extend the Boolean expressions defined on page 6 of Henrik Reif Andersen's notes (A97) with existential quantification. The syntax of these quantified Boolean expressions is:

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \Rightarrow t \mid t \Leftrightarrow t \mid \exists x . t \mid (t),$$

where x ranges over the set of Boolean variables. The propositional connectives have their usual semantics. The semantics of the existential quantification is defined by the following logical equivalence:

$$\exists x_i . t(x_1, \dots, x_n) \equiv t(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \vee t(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n).$$

Notice that the resulting expression does not depend on the quantified variable x_i . It defines the set of assignments of the remaining variables for which t is true for some truth assignment of x_i .

a) Reduce the expression $\exists x . (x \Rightarrow y)$ using logical equivalences.

a) Answer: 1

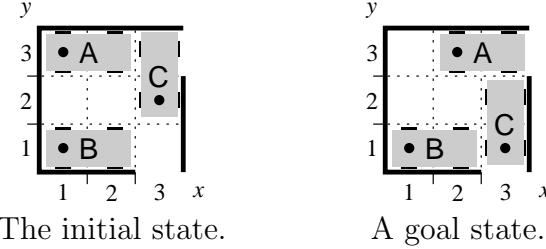
b) Reduce the expression $\exists x . (x \wedge y)$ using logical equivalences.

b) Answer: y

c) Write a quantified Boolean expression that is valid if and only if $t(x_1, \dots, x_n)$ is satisfiable.

c) Answer: $\exists x_1 . (\dots (\exists x_n . t(x_1, \dots, x_n)) \dots)$

Problem 3: Planning and Search (40%)



Consider the initial state of a simple rush hour puzzle shown in the left figure above. There are three cars A , B , and C that can move on a board with nine positions. Car A and B can only move left and right, while car C can only move up and down. The goal is to move car A and C to the two exits as shown in the right figure above. A planning domain description of the puzzle is shown below:

$\text{MoveRight}(v, x, y)$

Pre: $\text{at}(v, x, y) \wedge (x + 2 \leq 3) \wedge \neg\text{occ}(x + 2, y) \wedge \text{hor}(v)$

Eff: $\neg\text{at}(v, x, y) \wedge \text{at}(v, x + 1, y) \wedge \text{occ}(x + 2, y) \wedge \neg\text{occ}(x, y)$

$\text{MoveLeft}(v, x, y)$

Pre: $\text{at}(v, x, y) \wedge (x - 1 \geq 1) \wedge \neg\text{occ}(x - 1, y) \wedge \text{hor}(v)$

Eff: $\neg\text{at}(v, x, y) \wedge \text{at}(v, x - 1, y) \wedge \text{occ}(x - 1, y) \wedge \neg\text{occ}(x + 1, y)$

$\text{MoveUp}(v, x, y)$

Pre: $\text{at}(v, x, y) \wedge (y + 2 \leq 3) \wedge \neg\text{occ}(x, y + 2) \wedge \text{ver}(v)$

Eff: $\neg\text{at}(v, x, y) \wedge \text{at}(v, x, y + 1) \wedge \text{occ}(x, y + 2) \wedge \neg\text{occ}(x, y)$

$\text{MoveDown}(v, x, y)$

Pre: $\text{at}(v, x, y) \wedge (y - 1 \geq 1) \wedge \neg\text{occ}(x, y - 1) \wedge \text{ver}(v)$

Eff: $\neg\text{at}(v, x, y) \wedge \text{at}(v, x, y - 1) \wedge \text{occ}(x, y - 1) \wedge \neg\text{occ}(x, y + 1)$

The associated planning problem has initial state

$$\begin{aligned} & \text{hor}(A) \wedge \text{at}(A, 1, 3) \wedge \text{hor}(B) \wedge \text{at}(B, 1, 1) \wedge \text{ver}(C) \wedge \text{at}(C, 3, 2) \wedge \\ & \text{occ}(1, 1) \wedge \neg\text{occ}(1, 2) \wedge \text{occ}(1, 3) \wedge \text{occ}(2, 1) \wedge \neg\text{occ}(2, 2) \wedge \text{occ}(2, 3) \wedge \\ & \neg\text{occ}(3, 1) \wedge \text{occ}(3, 2) \wedge \text{occ}(3, 3) \end{aligned}$$

and goal

$$\text{at}(A, 2, 3) \wedge \text{at}(C, 3, 1).$$

Notice that the position of a car is defined by the position of the little black dot next to its identification letter.

a) List the representation rules of the STRIPS planning language that are violated by the domain description above and note briefly for each rule, why it is violated.

a) Answer:

- **Only positive literals in precondition.** Each action schema has negative literals in its precondition.
- **No arithmetic functions available.** Each action schema uses $+$, $-$.
- **No relational operations available.** Each action schema uses either \leq or \geq .

b) Write a planning domain and its associated initial state and goal for the described rush hour planning problem using the STRIPS planning language. You are allowed to define a completely new set of predicates and actions that fit your purpose.

b) Answer:

MoveRightA()

Pre: $at(A, 1) \wedge free(3, 3)$

Eff: $\neg at(A, 1) \wedge at(A, 2) \wedge \neg free(3, 3) \wedge free(1, 3)$

MoveLeftA()

Pre: $at(A, 2) \wedge free(1, 3)$

Eff: $\neg at(A, 2) \wedge at(A, 1) \wedge \neg free(1, 3) \wedge free(3, 3)$

MoveRightB()

Pre: $at(B, 1) \wedge free(3, 1)$

Eff: $\neg at(B, 1) \wedge at(B, 2) \wedge \neg free(3, 1) \wedge free(1, 1)$

MoveLeftB()

Pre: $at(B, 2) \wedge free(1, 1)$

Eff: $\neg at(B, 2) \wedge at(B, 1) \wedge \neg free(1, 1) \wedge free(3, 1)$

MoveUpC()

Pre: $at(C, 1) \wedge free(3, 3)$

Eff: $\neg at(C, 1) \wedge at(C, 2) \wedge \neg free(3, 3) \wedge free(3, 1)$

MoveDownC()

Pre: $at(C, 2) \wedge free(3, 1)$

Eff: $\neg at(C, 2) \wedge at(C, 1) \wedge \neg free(3, 1) \wedge free(3, 3)$

Initial state

$at(A, 1) \wedge at(B, 1) \wedge at(C, 2) \wedge free(1, 2) \wedge free(2, 2) \wedge free(3, 1)$

and goal

$at(A, 2) \wedge at(C, 1).$

Let the cost of a plan be its number of actions. Assume that we want to find an optimal plan with minimum cost. For this purpose, we define a heuristic function $h(s)$ that given a state s returns the cost of an optimal solution to a relaxed version of the planning problem. The relaxation, we will consider, is to ignore collisions by allowing a car to move to a position occupied by another car.

c) What is the value of h in the initial state?

c) Answer: 2

d) Is h admissible? (why/why not)

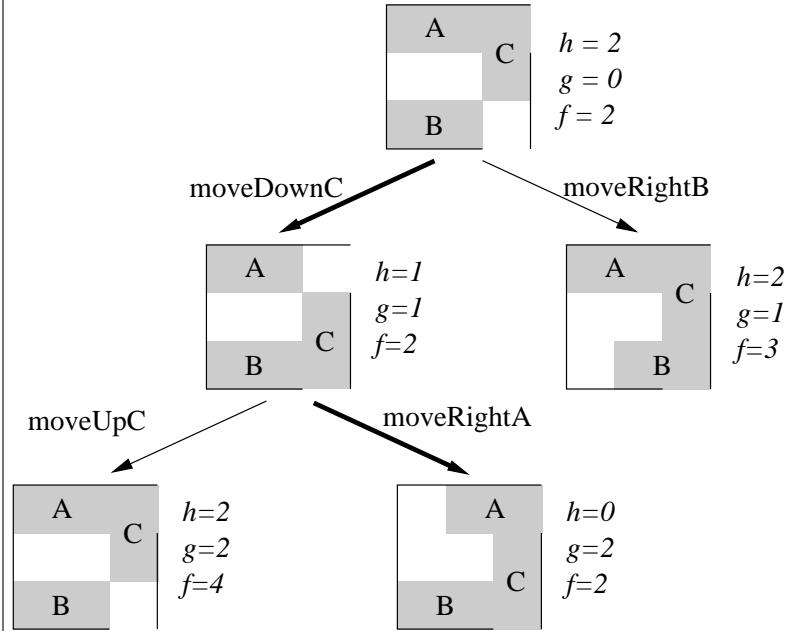
d) **Answer:** yes, any heuristic derived from a relaxed problem is admissible.

e) Is h consistent? (why/why not)

e) **Answer:** yes, any heuristic derived from a relaxed problem satisfies the triangle inequality and is therefore consistent.

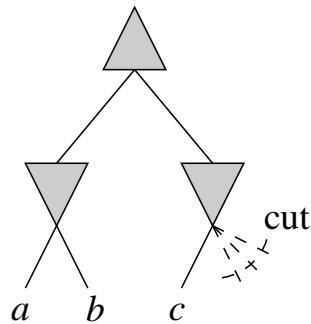
f) Draw the search tree of the tree search version of A* applied to the rush hour planning problem using h . Draw each search node as the board state, it contains, and note the value of h , g , and f . Indicate the path in the search tree corresponding to the found optimal plan. Include unexpanded nodes in the search tree.

f) **Answer:** The path of the found optimal plan is drawn with bold edges.



Problem 4: Adversarial Search (15%)

Consider running the alpha-beta search algorithm on the game tree below:



Assume that a , b , and c denote the minimax values computed by the alpha-beta search algorithm for their associated subtrees.

- a) Give a single assignment of a , b , and c that leads to the cut depicted in the figure.

a) **Answer:** for instance $a = 3$, $b = 3$, $c = 1$.

- b) Write an expression that defines all the assignments of a , b , and c that lead to the cut depicted in the figure.

b) **Answer:** $c \leq \text{Min}(a, b)$.

Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

2 June 2010

This exam consists of 7 numbered pages with 4 problems containing 15 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam (9:00-13:00).

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

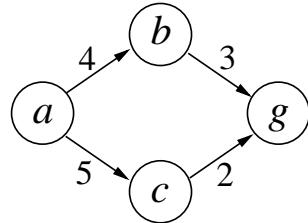
Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

Good luck!

Problem 1: Heuristic Search (10%)



In the search domain shown above, g is assumed to be the only goal state. Your task is to define heuristic functions h for the domain by defining the values of $h(a)$, $h(b)$, $h(c)$, and $h(g)$.

a) Define h such that h is a non-admissible heuristic function for the domain.

a) **Answer:** $h(g) = 0$, $h(b) = 4$ and arbitrary non-negative values for the remaining states.

b) Define h such that h is an admissible heuristic function for the domain.

b) **Answer:** E.g. $h(\cdot) = 0$ for all states.

c) Define h such that h is an inconsistent admissible heuristic function for the domain.

c) **Answer:** E.g. $h(g) = 0$, $h(b) = 2$, $h(a) = 7$, and $h(c) = 0$.

Problem 2: Propositional Logic (30%)

Consider the set of clauses

$$\mathcal{S} = \{(A \vee \neg B), (A \vee B \vee \neg C), (\neg D \vee E), (\neg A)\}.$$

- a) Compute the resolution closure $RC(\mathcal{S})$ of \mathcal{S} . You only have to list the clauses that are not already in \mathcal{S} .

a) Answer: $(A \vee \neg C), (\neg B), (B \vee \neg C), (\neg C)$.

We now turn to the unit clause heuristic applied by the DPLL algorithm.

- b) Which forced assignments are made when performing unit propagation on \mathcal{S} ?

b) Answer: $A = \text{false}, B = \text{false}, C = \text{false}$.

Assume that DPLL reduces the clause set *clauses* every time an assignment is made. This is done by

1. removing clauses that become logically equivalent to *true* and
2. removing the literals of the assigned symbol from the remaining clauses.

Thus, if A is assigned to *false* in the clause set \mathcal{S} , it is reduced to $\{(\neg B), (B \vee \neg C), (\neg D \vee E)\}$.

- c) Given this assumption, write the pseudocode of the FIND-UNIT-CLAUSE(*clauses, model*) function.

c) Answer:

```
FIND-UNIT-CLAUSE(clauses, model)
1: foreach  $c \in \text{clauses}$ 
2:   if  $c = (P)$  then return  $(P, \text{true})$ 
3:   if  $c = (\neg P)$  then return  $(P, \text{false})$ 
4: return  $(\text{Null}, \_)$ 
```

- d) What is the time complexity of your function?

d) Answer: $O(|\text{clauses}|)$ given a reasonable representation of clauses where the tests $c = (P)$ and $c = (\neg P)$ can be done in $O(1)$.

Problem 3: Constraint Programming (40%)

FORWARD-CHECKING-SEARCH(csp) **returns** a solution or failure

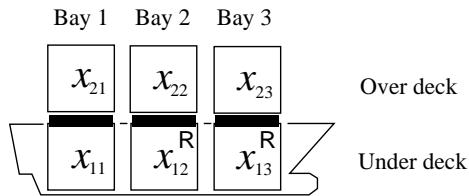
```
return RECURSIVE-FORWARD-CHECKING( $\{\}$ , $csp$ )
```

RECURSIVE-FORWARD-CHECKING($assignment,csp$) **returns** a solution or failure

```
if  $assignment$  is complete then return  $assignment$ 
 $var \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(\text{VARIABLES}[csp], assignment, csp)$ 
for each  $value$  in ORDER-DOMAIN-VALUES( $var, assignment, csp$ ) do
    if  $value$  is consistent with  $assignment$  do
        add  $\{var = value\}$  to  $assignment$ 
         $inferences \leftarrow \text{remove all domain values of remaining variables}$ 
        inconsistent with  $\{var = value\}$ 
    if  $inferences \neq \text{failure}$  then
        update  $csp$  with  $inferences$ 
         $result \leftarrow \text{RECURSIVE-FORWARD-CHECKING}(assignment, csp)$ 
        if  $result \neq \text{failure}$  then
            return  $result$ 
        remove  $\{var = value\}$  from  $assignment$  and remove  $inferences$  from  $csp$ 
    return  $\text{failure}$ 
```

In this problem, we will use the FORWARD-CHECKING-SEARCH algorithm shown above to solve a cost minimization problem. To do this, we change the algorithm to explore the complete search tree and return a found solution with minimum cost.

The minimization problem is to find a feasible stowage plan for a small container vessel with a minimum number of overstowing containers. The vessel consists of three bays that each has an over and under deck storage area separated by a removable metal cover (called a hatch lid). Each storage area can hold at most one container. The containers under deck stand on the keel while the containers over deck stand on the hatch lid. The vessel is shown in the figure below.



The vessel is empty initially and must be stowed with a set of containers C containing six containers. Five attribute functions are used to characterize a container. $h : C \rightarrow \mathcal{B}$ is Boolean function indicating whether or not the container

is heavy, $l : C \rightarrow \{20, 40, 45\}$ is the length of the container in feet, $r : C \rightarrow \mathcal{B}$ indicates whether or not the container is reefer, $i : C \rightarrow \mathcal{B}$ indicates whether or not the container is IMO, and $d : C \rightarrow \{1, 2, 3, 4\}$ is the number of ports downstream the container must be discharged. The following stowage rules must be satisfied

1. 45-foot containers can only be stored over deck.
2. Reefer containers need external power. Reefer containers can only be stored in storage areas with reefer plugs (indicated by “R” in the figure) and storage areas with reefer plugs can only hold reefer containers.
3. IMO containers contain explosives. A bay can at most store one IMO container and bays adjacent to a bay with an IMO container cannot store IMO containers.
4. To keep the vessel in balance a bay must store exactly one heavy container and one non-heavy container.

A feasible stowage plan assigns all the containers in C to a storage area without breaking any stowage rules. We assume that a CSP representing the problem of finding feasible stowage plans has six variables $X = \{x_{11}, x_{21}, x_{12}, x_{22}, x_{13}, x_{23}\}$ representing the six storage areas as shown in the figure. The domain of each variable is the set of containers C . Given this representation, we can formalize the first stowage rule as three constraints

$$l(x_{1i}) \neq 45 \text{ for } i \in \{1, 2, 3\}$$

and the second stowage rule as six constraints

$$\neg r(x_{11}), \neg r(x_{21}), r(x_{12}), \neg r(x_{22}), r(x_{13}), \neg r(x_{23}).$$

a) Write constraints of the remaining stowage rules and the fact that a container only can be stored in one storage area.

a) Answer: The third rule

$$\begin{aligned} & \neg(i(x_{1i}) \wedge i(x_{2i})) \text{ for } i \in \{1, 2, 3\}, \\ & \neg((i(x_{11}) \vee i(x_{21})) \wedge (i(x_{12}) \vee i(x_{22}))), \\ & \neg((i(x_{12}) \vee i(x_{22})) \wedge (i(x_{13}) \vee i(x_{23}))) \end{aligned}$$

The fourth rule

$$\neg(h(x_{1i}) \Leftrightarrow h(x_{2i})) \text{ for } i \in \{1, 2, 3\}.$$

The all different rule

$$x_{ij} \neq x_{vw} \text{ for } (i, j) \neq (v, w) \text{ and } i, v \in \{1, 2\}, j, w \in \{1, 2, 3\}.$$

The cost of a stowage plan is the number of overstows in the plan. An optimal stowage plan is a feasible stowage plan with minimum cost. Assume that a bay stores c_1 under deck and c_2 over deck. An overstay happens when c_1 must be discharged *before* c_2 ($d(c_1) < d(c_2)$). In this case, extra crane moves must be used to first unload c_2 to be able to remove the hatch lid and discharge c_1 and then load c_2 again.

- b)** Write the cost function of the problem (i.e., a function $cost(X)$ that for a complete assignment to X is equal to the number of overstows in the corresponding stowage plan).

b) Answer: Let

$$O(x_1, x_2) = \begin{cases} 1 & : d(x_2) > d(x_1) \\ 0 & : \text{Otherwise.} \end{cases}$$

We then have

$$cost(X) = \sum_{i=1}^3 O(x_{1i}, x_{2i}).$$

Consider stowing the set of containers $C = \{1, 2, 3, 4, 5, 6\}$ defined in the table below.

c	$h(c)$	$l(c)$	$r(c)$	$i(c)$	$d(c)$
1	0	20	1	1	1
2	1	45	0	0	4
3	0	40	0	1	3
4	1	20	1	0	3
5	0	45	0	0	2
6	1	40	0	0	3

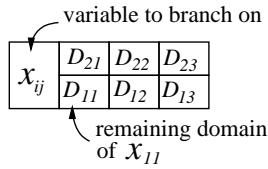
- c)** Write the reduced domains of each variable in X (i.e., remove containers from the domains that do not satisfy unary constraints of the problem).

c) Answer:

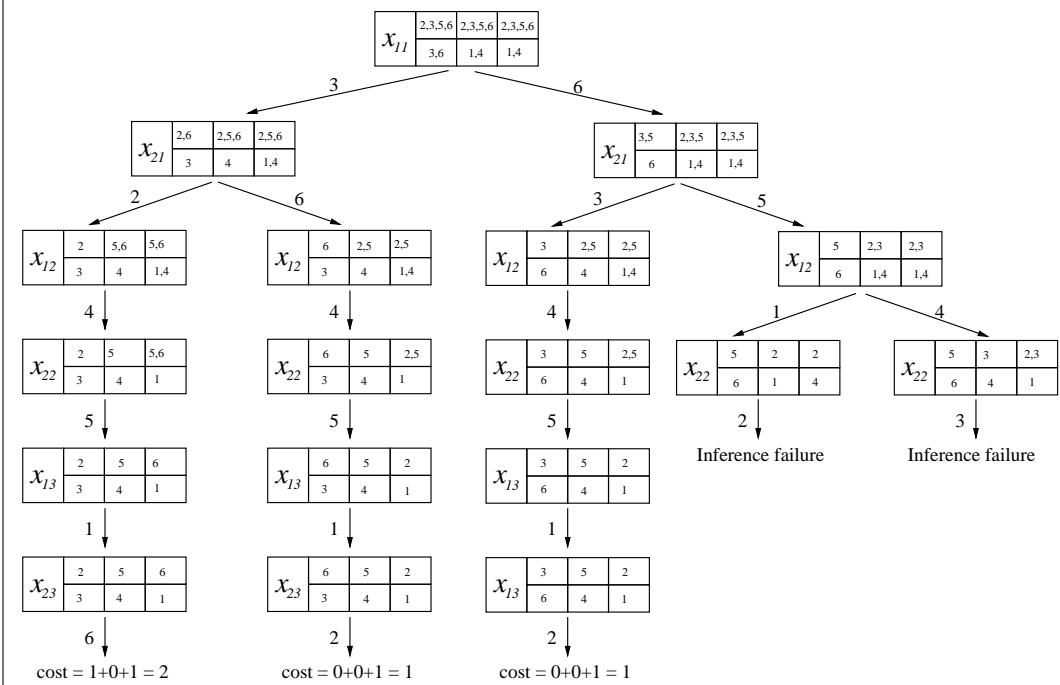
$$\begin{aligned} D(x_{21}) &= \{2, 3, 5, 6\} & D(x_{22}) &= \{2, 3, 5, 6\} & D(x_{23}) &= \{2, 3, 5, 6\} \\ D(x_{11}) &= \{3, 6\} & D(x_{12}) &= \{1, 4\} & D(x_{13}) &= \{1, 4\} \end{aligned}$$

Assume that our minimization version of FORWARD-CHECKING-SEARCH is initialized with the reduced domains of the variables. Further assume that it assigns the variables in the order $x_{11} \prec x_{21} \prec x_{12} \prec x_{22} \prec x_{13} \prec x_{23}$ and chooses values in domains in increasing order.

d) Draw the search tree explored by the minimization version of the FORWARD-CHECKING-SEARCH algorithm. Represent each search node by a box with remaining domain values for each variable as shown below. As usual draw an outgoing arc from the search node for each possible value assignment of the variable.



d) Answer:



e) What is a minimum cost solution to the problem?

e) Answer: There are two minimum cost solutions to the problem $x_{11} = 6$, $x_{21} = 3$, $x_{12} = 4$, $x_{22} = 5$, $x_{13} = 1$, $x_{23} = 2$ and $x_{11} = 3$, $x_{21} = 6$, $x_{12} = 4$, $x_{22} = 5$, $x_{13} = 1$, $x_{23} = 2$.

Problem 4: Games (20%)

Rock-paper-scissors is a hand game played by two people. The players count aloud to three each time raising one hand in a fist and swinging it down on the count. On the third count, the players change their hands into one of three gestures, which they then "throw" by extending it towards their opponent. The gestures are: Rock (clenched fist), Paper (an open hand), and Scissors (index and middle finger extended and separated). The objective is to select a gesture which defeats that of the opponent. Gestures are resolved as follows:

- Rock blunts or breaks scissors: that is, rock defeats scissors.
- Paper covers or captures rock: paper defeats rock.
- Scissors cut paper: scissors defeats paper.

If both players choose the same gesture, the game is tied and the players throw again.

a) Mention at least one fundamental assumption of the MINIMAX and EXPECTIMINIMAX algorithms that is not fulfilled by this game.

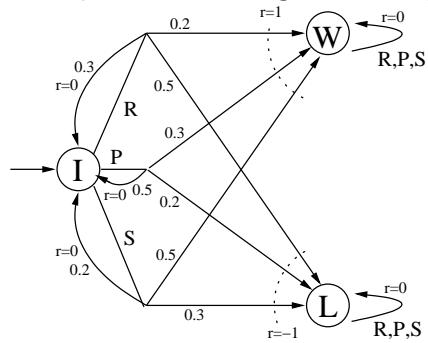
a) **Answer:** Both algorithms assume that at most one player can move in a state.

It is often possible to recognize and exploit the non-random behavior of an opponent. Assume that an opponent called Felix consistently plays Rock, Paper, and Scissors with probability 0.3, 0.5, and 0.2, respectively. Assume that winning is rewarded by 1 and losing is rewarded by -1 , while a reward of 0 is given for any other transition of the game.

Recall from the lectures that a Markov Decision Process (MDP) can be drawn as a graph where the vertices represent states and the edges represent transitions.

b) Draw an MDP that represents playing rock-paper-scissors against Felix and satisfies the assumptions of the Q LEARNING algorithm.

b) Answer: Notice that the Q LEARNING algorithm requires all actions to be



applicable in each state.

c) Argue which version of the Q LEARNING algorithm to apply for your MDP of the game.

c) Answer: The non-deterministic version of the Q LEARNING algorithm must be used since the MDP is non-deterministic.

Efficient Artificial Intelligence Programming (IAIP)

IT University of Copenhagen

8 June 2011

This exam consists of 8 numbered pages with 3 problems containing 15 subproblems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam (9:00-13:00).

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Begin a problem on a new page.

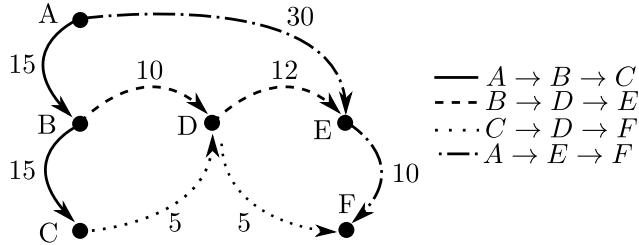
Write only on one side of the paper.

Write clearly.

Good luck!

Problem 1: Heuristic Search (25%)

The city has just hired you to create a system to help bus passengers find the fastest route to their destination. Consider the example below with four different bus lines.



The customer starts at A at 9:00 and wants to find the fastest route to F . Edges are labeled with the amount of time it takes for the bus to travel between nodes. For example, it takes 15 minutes to get from A to B and 30 minutes to get from A to E . The customer can transfer buses at any node and will transfer as many times as is necessary. Transfers are assumed to be instantaneous, i.e. a transfer can be made between a bus arriving at 9:02 and a bus leaving at 9:02 at the same node. Buses run on regular schedules that can be described by the pair $\langle a, b \rangle$, where a is the frequency of the bus line and b is the time offset from the current hour. For example, the pair $\langle 15, 5 \rangle$ describes a bus that comes every 15 minutes with an offset of 5 minutes, i.e. at :05, :20, :35, and :50. Each schedule is computed from the first node on the bus line. The schedules are shown below.

$A \rightarrow B \rightarrow C$	$\langle 15, 0 \rangle$
$B \rightarrow D \rightarrow E$	$\langle 20, 18 \rangle$
$A \rightarrow E \rightarrow F$	$\langle 30, 10 \rangle$
$C \rightarrow D \rightarrow F$	$\langle 10, 3 \rangle$

Thus, a customer may get on the ABC line at 9:00 or 9:15, the AEF line at 9:10 or 9:40, and so on. For example, the shortest time to get to D from A is to take the ABC line from A to B and transfer to the BDE line, arriving at D at 9:28 (15 minutes to B , 3 minutes waiting for the BDE line, and 10 minutes from B to D).

- a) What is the fastest route to F from A starting at 9:00? What time will the customer arrive at F ?

- a) **Answer:** ABC from A to B , transfer to the BDE , arriving at D as in the example. Then transfer immediately at 9:28 to the CDF and arrive at F at 9:33.

b) Consider a heuristic h that at any time at node A gives the value 15 and at node B has the value 19, and at all other nodes gives the value 0. Is this heuristic admissible? Why or why not?

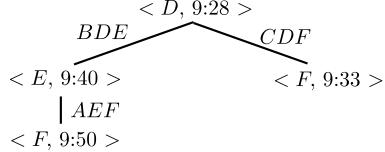
b) Answer: No, we can leave node B at 9:18 and arrive at node F 15 minutes later at 9:33. So the heuristic is not admissible at node B . It is, however, admissible at all other nodes.

c) When solving this problem as a search problem as described in section 3.1.1 of RN10, suggest what states and actions would represent for such a problem definition.

c) Answer: States can be represented by a pair $\langle n, t \rangle$ where n represents a node and t denotes the time we can arrive at that node. The actions that can be executed given a particular state are all of the bus connections leaving from that node at a time later than the arrival time.

d) Draw the search tree at node D at 9:28 assuming that the customer will not wait more than 20 minutes for a transfer.

d) Answer:



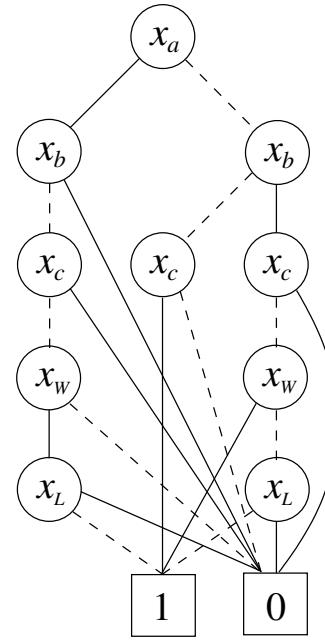
e) In reality, bus lines are circular. What happens to the size of the search tree if we add an edge from E to B on the BDE line?

e) Answer: The size of the graph becomes infinite, both in the number of branches and the depth, because bus lines are now infinitely long.

Problem 2: Configuration (35%)

An insurance company is using BDD-based configuration to help customers fill out an online car insurance form. Customers can choose from three insurance plans a , b , and c , where a is the cheapest, and c is the most expensive. The company has rules about which customers can buy which plans. It needs to know whether the customer is a woman and whether the customer's car drives more than 10000 km per year. Basically, women can get cheaper plans than men, and to get a cheap plan, the car must drive less than 10000 km per year. The company has implemented the form shown to the left below.

Woman	<input type="checkbox"/> (x_w)	
More than 10000 km per year	<input type="checkbox"/> (x_L)	
Insurance Plan	<input type="checkbox"/> (x_a) <input type="checkbox"/> (x_b) <input type="checkbox"/> (x_c)	
a	b	c



A customer fills out the form by writing a 0 or 1 in the blank fields. Thus, the customer must write 0 in the first field, if he is a man, and must write a 1 in the left-most bottom field to choose insurance plan a , and so on. The configurator guides the customer by showing the possible remaining values for each field. It writes 1, if only 1 is possible, 0 if only 0 is possible, and 0/1 if both 0 and 1 are possible for some solution. The configurator computes these possible values initially, when the customer has made no choices, and each time the customer makes a choice.

To this end, each field is associated with the Boolean variable shown in parentheses to the right of the field. The legal assignments of these variables according to the rules of the company are represented by the BDD shown to the right. As usual, low and high edges are drawn with dashed and solid lines, respectively.

- a) Is the BDD a reduced and ordered BDD (ROBDD)? Why / Why not?

a) Answer: It is not, the two x_L nodes are structurally identical. The BDD can be turned into an ROBDD by e.g. removing the right-most of these nodes and redirecting the low edge into this node to the left-most of the x_L nodes.

Recall that the paths of a BDD from the root node to the 1-terminal defines all the possible assignments of the variables.

b) What does the configurator initially write in the fields? Argue in terms of the paths of the BDD.

b) Answer: It writes 0/1 in each field since for each variable and truth-value, a path exists in the BDD, where the variable can be assigned the truth-value.

c) Which of the three statements below are true:

1. A customer can choose more than one plan.
2. A customer can choose at most one plan or none.
3. A customer can choose exactly one plan.

Argue for your answer in terms of the paths of the BDD.

c) Answer: The last choice, since this is the only choice that is true for all paths to the 1-terminal.

d) Assume that a customer initially chooses plan a . What does the configurator write as possible values in the remaining fields after this choice?

d) Answer: $x_W = 1$, $x_L = 0$, $x_b = 0$, and $x_c = 0$.

e) Let E denote the propositional sentence represented by the BDD. Can you conclude $E \models \neg x_W \Rightarrow \neg x_a$ from your answer to d)? Why / Why not?

e) Answer: Yes, the answer to d) shows that for all assignments where x_a is true, we have that x_W is true. So, $E \models x_a \Rightarrow x_W$, which by contraposition gives $E \models \neg x_W \Rightarrow \neg x_a$.

Problem 3: Constraint Programming (40%)

In this problem, we will use the FORWARD-CHECKING-SEARCH algorithm shown at the end to solve a constraint satisfaction problem. This version of the algorithm runs AC-3 in a pre-processing stage.

The constraint satisfaction problem is to find a feasible schedule for a set of container vessels that will arrive to a terminal to load/unload containers (see Figure 1(a)). The terminal has two cranes available, therefore at most two vessels can be stowed at the same time. For simplicity, we assume that each vessel takes always one unit of time for being stowed by the terminal. Figure 1(b) shows a schedule for three vessels using two time units. Each column represents a time slot and each row a crane. There are some rules about when container vessels can be stowed that makes the generation of the schedule a bit complicated, e.g., some vessels carrying hazardous materials need to be stowed alone.

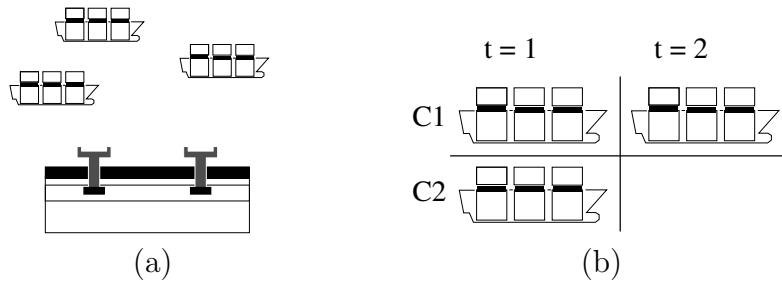


Figure 1: (a) Terminal with two cranes. (b) Schedule for 3 vessels being stowed by the terminal. Each column represents a time slot, each row a crane.

Today there is a set of five vessels coming into the terminal that need to be stowed. An ID number from 1 to 5 is assigned to each vessel. The terminal has only three consecutive time slots available to process the vessels, and there are some specific rules that the schedule must satisfy:

- (a) Vessel 4 will unload hazardous materials, therefore no other vessel can be stowed at the same time.
- (b) Vessels 1 and 2 must be stowed either at the same time or in two consecutive time slots.
- (c) Vessel 2 must be stowed before vessel 5.
- (d) Vessel 3 and 5 cannot be stowed at the same time.

We formulate our CSP as follows:

Variables	:	$X = \{x_1, \dots, x_5\}$	
Domains	:	$\forall x_i \in X, x_i \in \{1, 2, 3\}$	
Constraints	:	$x_2 < x_5,$	(1)
		$x_4 \neq x_1, x_4 \neq x_2, x_4 \neq x_3, x_4 \neq x_5,$	(2)
		$x_3 \neq x_5,$	(3)
		$ x_1 - x_2 \leq 1,$	(4)
		$\text{atmost}(\{x_1, \dots, x_5\}, 1, 2),$	(5)
		$\text{atmost}(\{x_1, \dots, x_5\}, 2, 2),$	(6)
		$\text{atmost}(\{x_1, \dots, x_5\}, 3, 2),$	(7)

where each variable represents a vessel, and their domain is the set of time slots when the vessels can be stowed by the terminal. The “atmost” constraint, $\text{atmost}(\text{scope}, v, n)$, receives as parameters a set of variables, scope , and two integer numbers, v and n . It constraints the domain of all variables in scope such that $|\{x_i | x_i \in \text{scope}, x_i = v\}| \leq n$. This constraint is used to limit the number of vessels stowed by the terminal at each time slot.

- a) Fill up the table below by matching each rule a feasible schedule must satisfy with their corresponding constraint from the CSP definition. As an example, the last row of the table matches rule d to constraint 3.

Rule	Constraint
a	
b	
c	
d	3

- a) Answer:

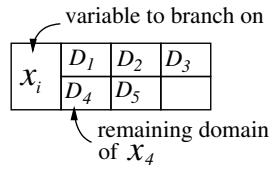
Rule	Constraint
a	2
b	4
c	1
d	3

- b) Write the reduced domains of each variable in X after running the AC-3 algorithm. You may assume that AC-3 only is given constraint 1-4 as input.

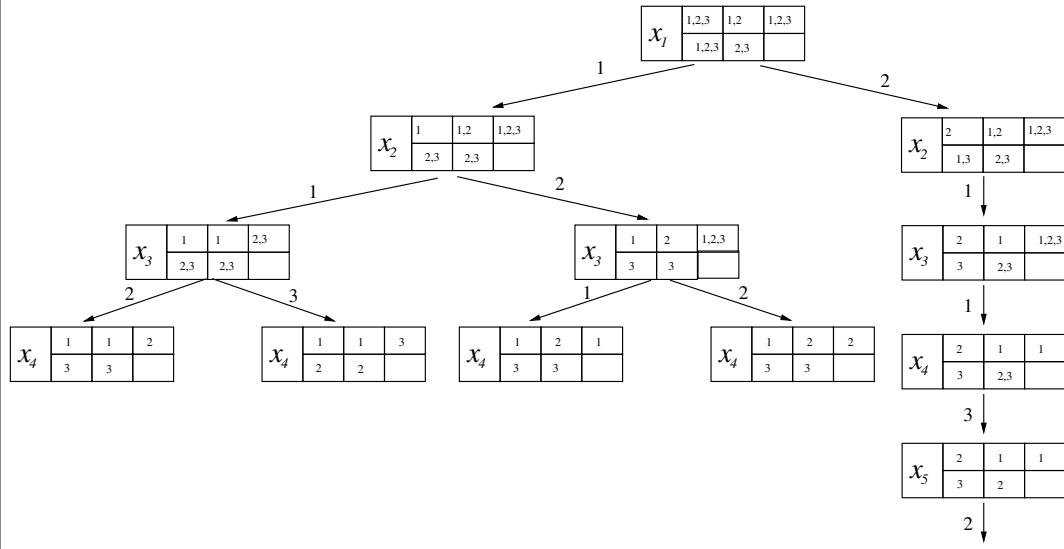
b) Answer:

Variable	Domain
x_1	$\{1, 2, 3\}$
x_2	$\{1, 2\}$
x_3	$\{1, 2, 3\}$
x_4	$\{1, 2, 3\}$
x_5	$\{2, 3\}$

c) Draw the search tree explored by the FORWARD-CHECKING-SEARCH algorithm. Represent each search node by a box with remaining domain values for each variable as shown below. As usual, draw an outgoing arc from the search node for each possible value assignment of the variable. The variables are assigned in the order x_1, x_2, x_3, x_4, x_5 , and the values from the domains of the variables are selected in increasing order.

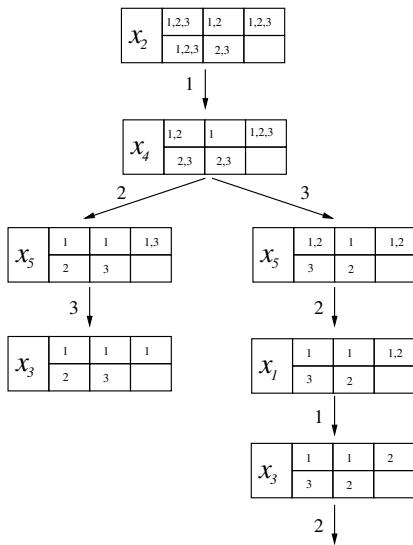


c) Answer:



d) Draw the search tree explored by the FORWARD-CHECKING-SEARCH algorithm using the *Minimum remaining values* heuristic to select a variable. Break ties using the *Degree* heuristic, and select the variable with the lowest index when ties persist. Select values from the domains of the variables in increasing order. Represent each search node as in question c).

d) Answer:



e) Do the heuristics applied in d) have any positive impact on the asymptotic complexity of FORWARD-CHECKING-SEARCH?

e) Answer: No, the heuristic enhances the performance of FORWARD-CHECKING-SEARCH in practice, in several situations, but it does not work in all cases. The asymptotic complexity remains the same.

FORWARD-CHECKING-SEARCH(csp) **returns** a solution or failure

run AC-3(csp)

return RECURSIVE-FORWARD-CHECKING($\{\}$, csp)

RECURSIVE-FORWARD-CHECKING($assignment, csp$) **returns** a solution or failure

if $assignment$ is complete **then return** $assignment$

$var \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(\text{VARIABLES}[csp], assignment, csp)$

for each $value$ **in** ORDER-DOMAIN-VALUES($var, assignment, csp$) **do**

if $value$ is consistent with $assignment$ **do**

add $\{var = value\}$ to $assignment$

$inferences \leftarrow \text{remove all domain values of remaining variables}$

inconsistent with $\{var = value\}$

```
if inferences  $\neq$  failure then
    update csp with inferences
    result  $\leftarrow$  RECURSIVE-FORWARD-CHECKING(assignment, csp)
    if result  $\neq$  failure then
        return result
    remove  $\{var = value\}$  from assignment and remove inferences from csp
return failure
```

Intelligent Systems Programming (ISP)

IT University of Copenhagen

4 June 2012

This exam consists of 7 numbered pages with 4 problems containing 18 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

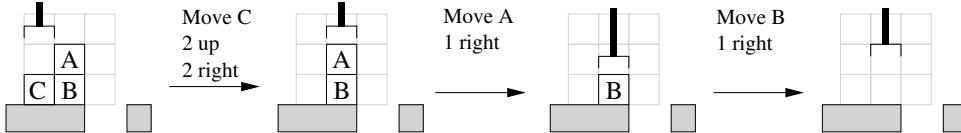
Good luck!

Problem 1: Search and Optimization (40%)

A crane is used to drop three blocks A , B , and C into a hole that is situated to the right of the blocks. The objective is to devise a way for the crane to drop all the blocks with minimum cost. The order in which the blocks are dropped is unimportant. The crane operations are governed by the following rules:

1. The movements of the crane gripper follow a grid.
2. The gripper can only reach a block positioned at the top of a stack.
3. A block cannot be moved into a position occupied by another block.
4. When the gripper holds a block, each up and right move of the gripper takes one time unit and has a cost of 1. All other moves can be ignored.
5. Cost is saved when extending the time for moving a block. The cost saving is a linear function of the extra time given to move the block. A saving of 1 is achieved when the total time to move the block is doubled. The total extra time given to drop the blocks can at most be one time unit.

Initially, the blocks are stacked as shown to the left in the figure below.



The figure illustrates dropping the blocks in the order C , A , B . Without extra time, the total time and cost of moving C , A , and B in this example is $4 + 1 + 1 = 6$. Let x_1 , x_2 , and x_3 denote the extra time used for moving block A , B , and C , respectively. The maximum cost that can be saved by using extra time for moving the blocks in this order can then be expressed by the following Linear Programming (LP) problem:

$$\begin{aligned} & \text{Maximize } x_1 + x_2 + \frac{1}{4}x_3 \\ & \text{Subject to } x_1 + x_2 + x_3 \leq 1 \\ & \quad x_1, x_2, x_3 \geq 0. \end{aligned}$$

a) Write the LP problem in slack form.

a) Answer:

$$\begin{aligned} & \text{Maximize } z \\ & \text{Subject to } x_4 = 1 - x_1 - x_2 - x_3 \\ & \quad z = x_1 + x_2 + \frac{1}{4}x_3 \\ & \quad x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

b) Solve the LP problem using the simplex method and show all the dictionaries computed (including the initial dictionary).

b) Answer: Since the slack form is feasible, it can be used as the initial dictionary:

$$\begin{aligned}x_4 &= 1 - x_1 - x_2 - x_3 \\z &= x_1 + x_2 + \frac{1}{4}x_3\end{aligned}$$

The maximum increase of x_1 is 1. This gives the first dictionary:

$$\begin{aligned}x_1 &= 1 - x_2 - x_3 - x_4 \\z &= 1 - \frac{3}{4}x_3 - x_4\end{aligned}$$

This dictionary is optimal.

c) Give an optimal primal and dual solution to the LP problem.

c) Answer: An optimal primal solution can be read directly from the optimal dictionary: $x_1 = 1$, $x_2 = 0$, $x_3 = 0$. The corresponding optimal dual solution is $y_1 = -\bar{c}_4 = 1$.

d) What is the minimum total cost of dropping the blocks in the order C, A, B given that extra time is used? What is the total time of the activity?

d) Answer: The total cost and time without using extra time is $4 + 1 + 1 = 6$. The maximum saving is 1 and is achieved by using 1 extra time unit. Thus, the total time used is 7 and the total cost is 5.

e) If more than one extra time unit is available for dropping the blocks in this order, then how much does the total minimum cost decrease per such additional time unit? (Hint: use the shadow price of the constraint.)

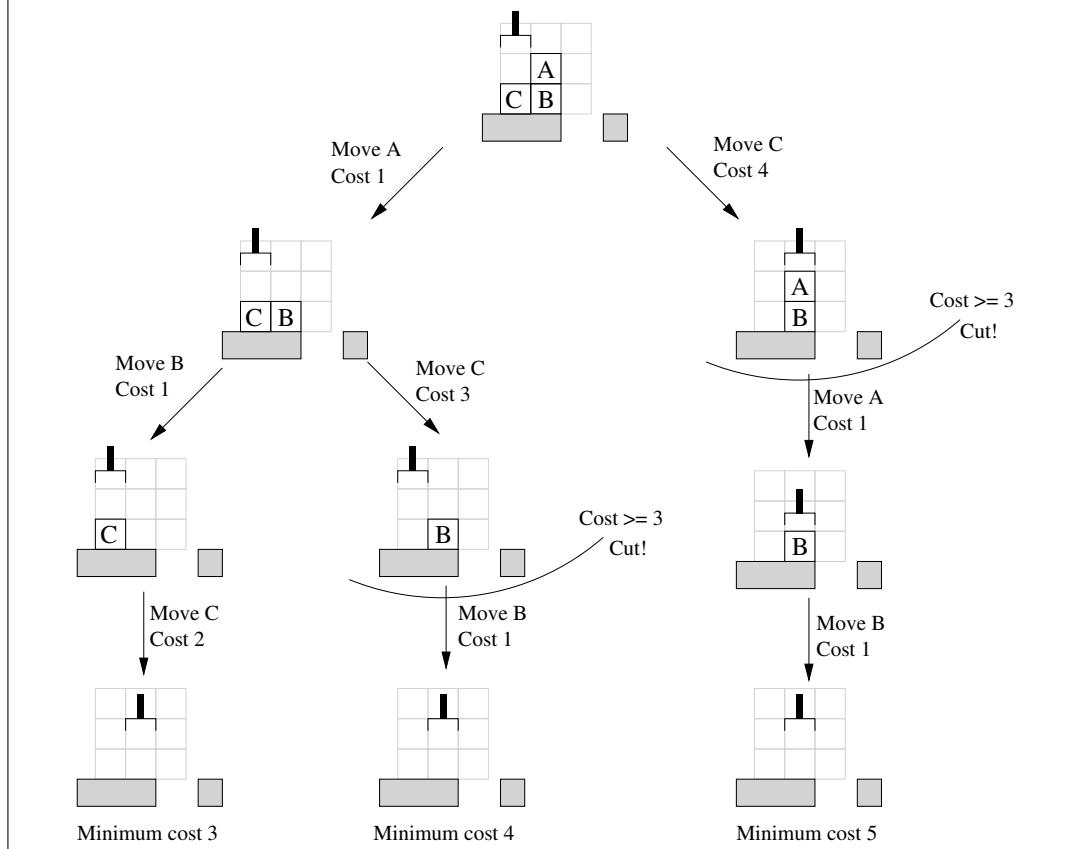
e) Answer: The minimum cost decreases an amount equal to the shadow price $y_1 = 1$ of the constraint.

At this point we know the minimum cost of dropping the blocks in the order C, A, B . We now want to compute the minimum cost of dropping the blocks in any order. To this end, we define a search problem. The initial state of the search problem is the initial configuration of the blocks. An action is to drop a single block. An action is applicable if the associated block can be reached by the gripper. Its cost is the number of up and right moves needed to drop the block. Assume that the complete state-space is explored by a Depth-First Search

(DFS) algorithm (i.e., the DFS algorithm continues searching until all solutions are found). Assume that the DFS algorithm applies the actions in the order A , B , C of the associated blocks.

- f) Draw the search tree explored by the DFS algorithm, where left-most nodes are explored first.

f) Answer:



- g) Write the minimum cost of each solution (leaf in the tree) by using the same approach as in question b) to d). What is the minimum cost of dropping the blocks and in what order should they be dropped?

g) Answer: We only need to compute the minimum cost of the first two solutions since we already have computed the minimum cost of the last solution.

First solution The cost without extra time is $1+1+2 = 4$. The LP problem in slack form associated with this solutions is:

$$\begin{aligned}x_4 &= 1 - x_1 - x_2 - x_3 \\z &= x_1 + x_2 + \frac{1}{2}x_3\end{aligned}$$

Since it is feasible, it can be used as the initial dictionary of the simplex method. We can at most increase x_1 to 1 and get the first dictionary:

$$\begin{aligned}x_1 &= 1 - x_2 - x_3 - x_4 \\z &= 1 - \frac{1}{2}x_3 - x_4\end{aligned}$$

This dictionary is optimal, which means that we at most can save 1 cost unit. So the minimum cost with extra time is 3.

Second solution The cost without extra time is $1+3+1 = 5$. The LP problem on slack form associated with this solutions is:

$$\begin{aligned}x_4 &= 1 - x_1 - x_2 - x_3 \\z &= x_1 + x_2 + \frac{1}{3}x_3\end{aligned}$$

Since it is feasible, it can be used as the initial dictionary of the simplex method. We can at most increase x_1 to 1 and get the first dictionary:

$$\begin{aligned}x_1 &= 1 - x_2 - x_3 - x_4 \\z &= 1 - \frac{2}{3}x_3 - x_4\end{aligned}$$

This dictionary is optimal, which means that we at most can save 1 cost unit. So the minimum cost with extra time is 4.

The minimum cost for each solution is written under its corresponding leaf in the search tree given in the answer to question f) above. The minimum overall cost is 3 and is achieved for the ordering A, B, C .

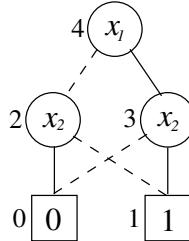
Assume that the DFS algorithm updates the minimum cost of a solution found so far every time a leaf node is reached.

h) Argue that this makes it possible for the DFS algorithm to terminate the exploration of sub-optimal paths early and indicate in the search tree which search nodes can be cut. (Hint: notice that at most 1 cost unit can be saved by using extra time).

h) Answer: If one plus the cost of a partial solution associated with a search node is larger than or equal to the cost of the solution with minimum cost found so far, the DFS algorithm can skip expanding that search node and backtrack. The reason is that no complete solution produced from the search node will have a cost that is less than the current solution with minimum cost even if the maximum saving of 1 cost unit is achieved. The resulting cuts are shown in the search tree given in the answer to question f) above.

Problem 2: Logic and BDDs (25%)

A multi-rooted unique table contains the following BDD nodes:



The unique identifier of a node is written to the left of the node. The variable ordering of the unique table is $x_1 \prec x_2$.

- a) Write a sentence f in propositional logic that is logically equivalent to the BDD represented by the node with identifier 4.

a) Answer: $x_1 \Leftrightarrow x_2$.

Let g be a sentence in propositional logic that is logically equivalent to the BDD represented by the node with identifier 3.

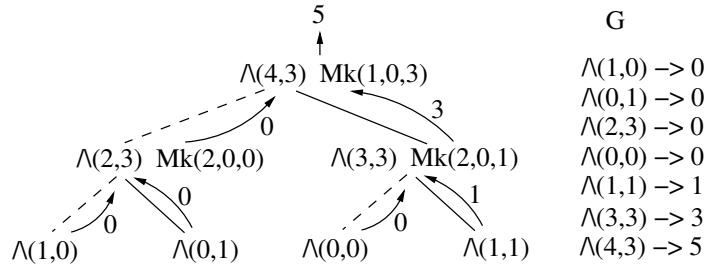
- b) Use logical equivalences to reduce the number of logical connectives in the sentence $f \wedge g$ as much as possible.

b) Answer: We have:

$$\begin{aligned}
 f \wedge g &\equiv (x_1 \Leftrightarrow x_2) \wedge x_2 \\
 &\equiv (x_1 \Rightarrow x_2) \wedge (x_2 \Rightarrow x_1) \wedge x_2 \\
 &\equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1) \wedge x_2 \\
 &\equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \wedge x_2 \vee x_1 \wedge x_2) \\
 &\equiv (\neg x_1 \vee x_2) \wedge (x_1 \wedge x_2) \\
 &\equiv (\neg x_1 \wedge x_1 \wedge x_2) \vee (x_2 \wedge x_1 \wedge x_2) \\
 &\equiv x_2 \wedge x_1 \wedge x_2 \\
 &\equiv x_1 \wedge x_2
 \end{aligned}$$

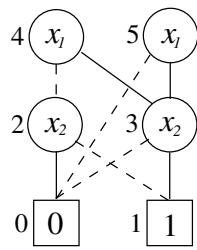
- c) Draw the call tree of $\text{APPLY}[T,H](\wedge,4,3)$, where T and H are assumed initially to store the unique table shown in the figure. The cache G is assumed to be empty initially. Indicate cache hits and show the final content of G .

c) Answer:



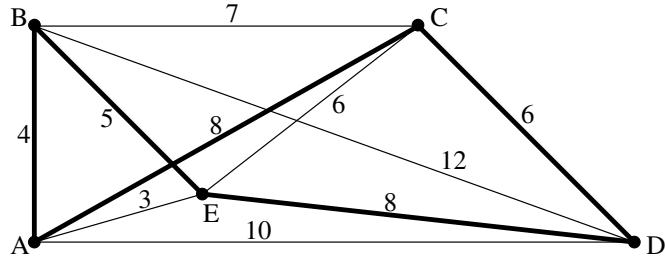
d) Draw the BDD-nodes of the resulting unique table.

d) Answer:



Problem 3: Local Search (20%)

Given a fully connected graph $G(V, E)$ composed of a set of vertices V and a set of edges E , we define a *cyclic path* to be a path that, starting from a vertex v_1 , visits all vertices exactly once before going back to the initial vertex v_1 . A cyclic path p can be represented as a tuple of vertices $\langle v_1, v_2, \dots, v_{|V|} \rangle$. The edges connecting the vertices in the tuple and an edge from $v_{|V|}$ to v_1 define the path. A cost $C(i, j)$ is associated to each edge connecting vertex v_i with vertex v_j . Given a cyclic path $p = \langle v_1, v_2, \dots, v_{|V|} \rangle$, the total cost of the path is thus $COST(p) = \sum_{i=1}^{|V|-1} C(i, i+1) + C(|V|, 1)$.



Consider the above depicted instance of such a graph. The thick lines indicate a cyclic path. Using the tuple representation, we can describe this path as $p = \langle B, E, D, C, A \rangle$ with $COST(p) = C(B, E) + C(E, D) + C(D, C) + C(C, A) + C(A, B) = 5 + 8 + 6 + 8 + 4 = 31$. We will consider the problem of finding a minimum cyclic path using local search techniques. A state of the local search is represented by a cyclic path and for this example, we will use a neighborhood that allows the swap of two adjacent vertices in the tuple representation of the path. Given $p_0 = \langle B, E, D, C, A \rangle$ as initial state, the neighbors of p_0 are:

- $\langle E, B, D, C, A \rangle$
- $\langle B, D, E, C, A \rangle$
- $\langle B, E, C, D, A \rangle$
- $\langle B, E, D, A, C \rangle$

We call this neighborhood the *adjacent-swap neighborhood*.

- a) Given $\langle B, E, C, A, D \rangle$ as the initial state, simulate a Tabu search for the next 3 moves assuming the following initial tabu list:

$$\begin{pmatrix} \langle B, E, C, A, D \rangle \\ \langle B, E, C, D, A \rangle \\ \langle E, B, A, C, D \rangle \\ \langle B, E, A, D, C \rangle \end{pmatrix}$$

Do so by showing for each move the updated tabu list, the list of neighboring states, their cost, and whether or not they have been selected for the next move.

a) Answer:

1st move, current state: $\langle B, E, C, A, D \rangle$

Tabu list	Neighborhood			
	Neighbors	Cost	Selected	Tabu
$\langle B, E, C, A, D \rangle$	$\langle E, B, C, A, D \rangle$	38		
$\langle B, E, C, D, A \rangle$	$\langle B, C, E, A, D \rangle$	38		
$\langle E, B, A, C, D \rangle$	$\langle B, E, A, C, D \rangle$	34	X	
$\langle B, E, A, D, C \rangle$	$\langle B, E, C, D, A \rangle$	31		X

2nd move, current state: $\langle B, E, A, C, D \rangle$

Tabu list:	Neighborhood			
	Neighbors	Cost	Selected	Tabu
$\langle B, E, C, A, D \rangle$	$\langle E, B, A, C, D \rangle$	31		X
$\langle B, E, C, D, A \rangle$	$\langle B, A, E, C, D \rangle$	31	X	
$\langle E, B, A, C, D \rangle$	$\langle B, E, C, A, D \rangle$	41		X
$\langle B, E, A, D, C \rangle$	$\langle B, E, A, D, C \rangle$	31		X
$\langle B, E, A, C, D \rangle$				

3rd move, current state: $\langle B, A, E, C, D \rangle$

Tabu list:	Neighborhood			
	Neighbors	Cost	Selected	Tabu
$\langle B, E, C, A, D \rangle$	$\langle A, B, E, C, D \rangle$	31		
$\langle B, E, C, D, A \rangle$	$\langle B, E, A, C, D \rangle$	34		X
$\langle E, B, A, C, D \rangle$	$\langle B, A, C, E, D \rangle$	38		
$\langle B, E, A, D, C \rangle$	$\langle B, A, E, D, C \rangle$	28	X	
$\langle B, E, A, C, D \rangle$				
$\langle B, A, E, C, D \rangle$				

b) Does the adjacent-swap neighborhood always generate feasible states for this problem? and why?

b) Answer: Yes. Since the graph is fully connected, it is always possible to find an edge that connects any two vertices. Thus, the order of the vertices does not impact the feasibility of the state. Since the adjacent-swap neighborhood only swaps the order of the vertices, it is not possible to add or remove vertices from the state. Thus the generated states are always feasible.

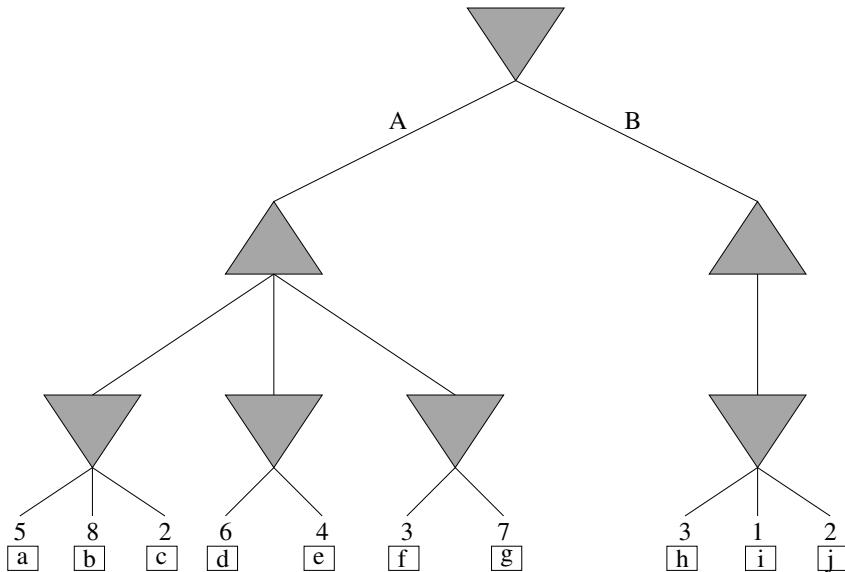
c) Assume you wrote a Simulated Annealing search that uses a fixed temperature schedule. During your experiments, you noticed that the search converges very fast to low-quality local optimums. You now want to change the algorithm so that it has a better chance of converging to high-quality solutions. In order to do that you:

- a) Speed-up the schedule such that the temperature decreases faster.
- b) Slow-down the schedule such that the temperature decrease slower.
- c) Let the search accept only improving solutions.

Briefly argue for your answer.

c) Answer: Slowing-down the schedule will give the search a higher chance of accepting states with a smaller improvement. This in turn results in a higher degree of diversification and thus improving the chance of finding better solution since a larger state space is sampled. The opposite effect would be obtained using a) or c).

Problem 4: Game Trees (15%)



In the game tree above, MIN can choose between actions A and B. Leaves are labelled with small letters from a to j .

- a)** Fill up the box below with the labels of the leaves pruned by the ALPHA-BETA-SEARCH algorithm. Write the labels in the row corresponding to the kind of cut that prunes the leaves. Assume that the left branches are always explored first.

α -cut	
β -cut	

- a) Answer:**

α -cut	g
β -cut	

- b)** Fill up the box below with the labels of the leaves pruned by the ALPHA-BETA-SEARCH algorithm as in a). This time assume that the right branches are explored first.

α -cut	
β -cut	

- b) Answer:**

α -cut	
β -cut	a, b, c, d, e

c) What is the minimax value of MIN when exploring the left branches first?
What is the minimax value when exploring the right branches first?

c) **Answer:** The minimax value is independent of the exploration strategy and is 1 in both cases.

Intelligent Systems Programming (ISP)

IT University of Copenhagen

6 June 2013

This exam consists of 5 numbered pages with 4 problems containing 11 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

You can write your answers using a pencil.

You are allowed to use a scientific pocket calculator.

The exam is open book.

Number the pages and write your name and CPR number on every page.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Begin a problem on a new page.

Write only on one side of the paper.

Write clearly.

Good luck!

Problem 1: Linear Programming (35%)

Consider the following LP problem written in slack form:

$$\begin{aligned} & \text{Maximize } z \\ & \text{Subject to } x_3 = -2 + 2x_1 + x_2 \\ & \quad x_4 = 3 - x_1 - x_2 \\ & \quad z = x_1 \\ & \quad x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

a) Which variables are decision variables and which variables are slack variables?

a) Answer: The decision variables are x_1 and x_2 . The slack variables are x_3 and x_4 .

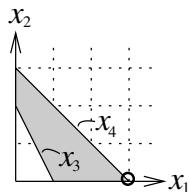
b) Write the LP problem in standard form.

b) Answer:

$$\begin{aligned} & \text{Maximize } z \\ & \text{Subject to } -2x_1 - x_2 \leq -2 \\ & \quad x_1 + x_2 \leq 3 \\ & \quad x_1, x_2 \geq 0. \end{aligned}$$

c) Draw the geometric interpretation of the feasible set. Indicate an optimal solution and show the slack variable associated with each constraint.

c) Answer The feasible set is the shaded polyhedron. There is only one optimal solution which is the corner point indicated by the circle.



d) Explain why the slack form is infeasible and write the auxiliary problem for the two-phase simplex method.

d) **Answer:** The slack form is infeasible since x_3 is negative. The auxiliary problem is:

$$\begin{array}{lll} \text{Maximize} & -x_0 \\ \text{Subject to} & -2x_1 - x_2 - x_0 \leq -2 \\ & x_1 + x_2 - x_0 \leq 3 \\ & x_0, x_1, x_2 \geq 0. \end{array}$$

e) Solve the LP problem using the two-phase simplex method. Show the pivots and dictionaries computed in each phase (including the slack form of the auxiliary problem). Write the optimal primal and dual solution found.

e) Answer:

First phase. The slack form of the auxiliary problem is:

$$\begin{aligned} \text{Maximize } & w \\ \text{Subject to } & x_3 = -2 + 2x_1 + x_2 + x_0 \\ & x_4 = 3 - x_1 - x_2 + x_0 \\ & w = -x_0 \\ & x_0, x_1, x_2, x_3, x_4 \geq 0 \end{aligned} .$$

First we perform a pivot to make the dictionary feasible (leaving x_3 , entering x_0):

$$\begin{aligned} x_0 &= 2 - 2x_1 - x_2 + x_3 \\ x_4 &= 5 - 3x_1 - 2x_2 + x_3 \\ w &= -2 + 2x_1 + x_2 - x_3 \end{aligned}$$

Since we prefer to pivot x_0 out of the basis, we perform the pivot: entering x_1 , leaving x_0 .

$$\begin{aligned} x_1 &= 1 - \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_0 \\ x_4 &= 2 - \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_0 \\ w &= -x_0 \end{aligned}$$

Second phase. The initial dictionary is:

$$\begin{aligned} x_1 &= 1 - \frac{1}{2}x_2 + \frac{1}{2}x_3 \\ x_4 &= 2 - \frac{1}{2}x_2 - \frac{1}{2}x_3 \\ z &= 1 - \frac{1}{2}x_2 + \frac{1}{2}x_3 \end{aligned}$$

Pivot: entering x_3 , leaving x_4 .

$$\begin{aligned} x_1 &= 3 - x_2 - x_4 \\ x_3 &= 4 - x_2 - 2x_4 \\ z &= 3 - x_2 - x_4 \end{aligned}$$

Since all coefficients in z are negative, this dictionary is optimal. The found primal solution is: $x_1 = 3$, $x_2 = 0$. The found optimal dual solution is $y_1 = -\bar{c}_3 = 0$, $y_2 = -\bar{c}_4 = 1$.

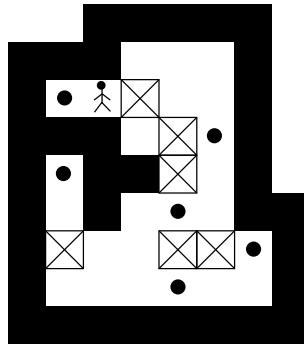
f) Given the economical interpretation of dual variables and their relation to constraints, is it surprising that the optimal value of one of the dual variables is zero (why/why not)?

f) Answer: No, the economical interpretation of y_1 is the amount we can gain per unit extra resource of the constraint, it is associated with. But this constraint is not binding in the optimal solution, since its slack variable x_3 is positive. For that reason it is not surprising that y_1 is zero.

Problem 2: Heuristics (20%)

The Sokoban puzzle is a game that is played on a square grid, where each square is a floor or a wall. Some floor squares are marked as storage locations (filled circles), and some of them have boxes (squares with crosses). The player is confined to the grid and may move horizontally or vertically onto empty squares (never through walls or boxes). The player can also move into a box, which pushes it into the square beyond. Boxes may not be pushed into other boxes or walls, and they cannot be pulled. The puzzle is solved when all boxes are at storage locations. An optimal solution of the game uses a minimum number of moves of the player.

The drawing below shows an initial state of Sokoban with six boxes. If the player moves right in this state, the adjacent box will be pushed one square to the right.



- a) Alice defines a heuristic for this problem to be the number of boxes currently not at a storage location. Explain carefully why Alice's heuristic is valid and admissible.

a) Answer: Alice's heuristic is valid because it is non-negative in all states of the game and because it is zero for all goal states. The heuristic is also admissible because it at least requires one move of the player to move a box to a storage location. Thus, as required, for any state, the heuristic is a lower bound of the remaining number of moves needed to solve the game.

- b) Define a new admissible heuristic that dominates Alice's heuristic. Explain carefully why your heuristic is valid, admissible, and dominating. The more states your heuristic strictly dominates Alice's heuristic, the more credit you earn for your answer.

b) Answer: Assume that the boxes are numbered 1 to 6. Let d_i denote the minimum Manhattan distance to a storage location for box i . The heuristic value is then defined to be $\sum_{i=1}^6 d_i$. The heuristic is valid because it is non-negative in all states of the game and because it is zero for all goal states. The heuristic is admissible since the player at least needs d_i moves to move box i to a storage location. The heuristic dominates Alice's heuristic because $d_i \geq 1$ if box i is not at a storage location. The heuristic strictly dominates Alice's heuristic in all states where at least one box is more than one move away from a storage location.

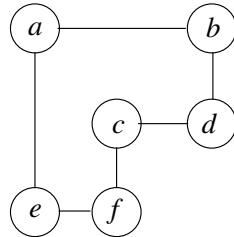
Problem 3: Constraint Programming (30%)

a		b	5
	c	d	5
e	f		5
5	4	6	

In the grade school puzzle above, the students must assign values to the variables a, b, \dots, f such that the row and column sums are correct (i.e., $a+b = 5$, $b+d = 6$ etc.). The domain of each variable is $\{1, 2, 3, 4\}$.

- a) Draw the constraint graph of this problem.

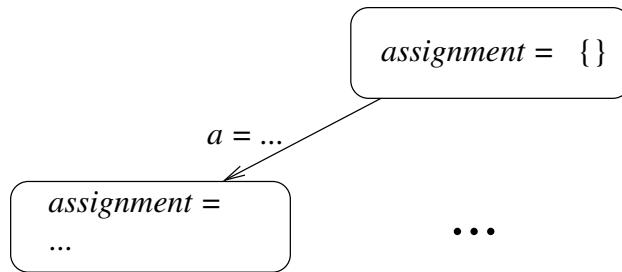
a) Answer:



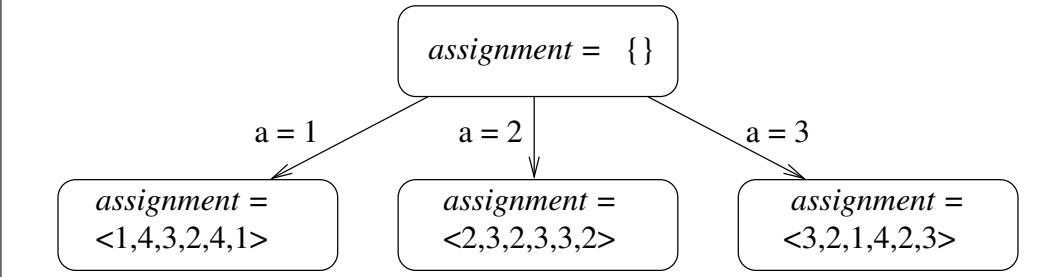
- b) Write the arc-consistent domain of each variable.

b) Answer: $a = \{1, 2, 3\}$, $b = \{2, 3, 4\}$, $c = \{1, 2, 3\}$, $d = \{2, 3, 4\}$, $e = \{2, 3, 4\}$, $f = \{1, 2, 3\}$

- c) Use the MAC algorithm to find all solutions to this problem. Select variables in the order $a \prec b \prec \dots \prec f$ and assign values in increasing order. Continue the search after the first solution is found to find all solutions. Show the call tree, where each node corresponds to a call to RECURSIVE-MAC. For each node, write the value of the *assignment* parameter passed to the node. You can draw your tree using the format shown below:



c) **Answer:** In the drawing, assignments are written as 6-tuples with the values assigned to a, b, \dots, f .



Problem 4: True or False (15%)

Indicate the correctness of each statement in the table below.

Statement	True or False
Without using the cache G , the APPLY algorithm has exponential time complexity	
The Tree Search version of A* is optimal when using a consistent heuristic	
MAX makes α -cuts in ALPHA-BETA-SEARCH	
$\neg C$ entails $\neg F \Rightarrow \neg C$	

Answer:

Statement	True or False
Without using the cache G , the APPLY algorithm has exponential time complexity	True
The Tree Search version of A* is optimal when using a consistent heuristic	True
MAX makes α -cuts in ALPHA-BETA-SEARCH	False
$\neg C$ entails $\neg F \Rightarrow \neg C$	True

Intelligent Systems Programming (ISP)

IT University of Copenhagen

6 June 2014

This exam consists of 5 numbered pages with 3 problems containing 17 subproblems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

You are allowed to use a scientific pocket calculator.

The exam is open book.

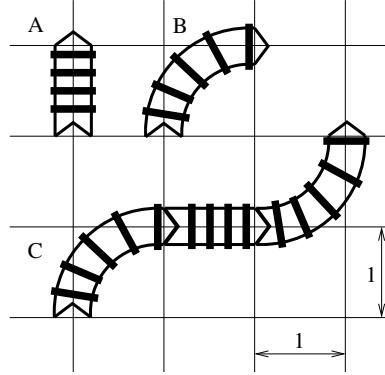
Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Heuristic Search (30%)

A model railway set contains a large number of curved and straight track pieces. The figure below shows a straight piece (A), a curved piece (B), and an example track made by connecting three pieces (C). Notice that a curved piece can be flipped such that one piece can be used to turn both left and right.



The tracks are made on a planar unit grid such that the driving length of a curved and straight piece are $\pi/2$ and 1, respectively. We assume that a new piece only can be added to the “arrow head” of a track (i.e., the example track (C) was build from left to right).

In the following, we use heuristic tree search to extend a given track into a closed loop track with **minimum remaining driving length**.

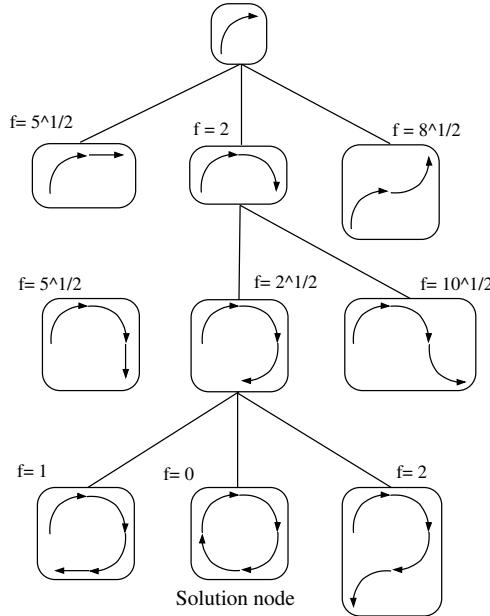
For this purpose, we use the Straight Line Distance (SLD) as heuristic. For the example track (C), the total driving length is $\pi + 1$ and the SLD between the two end-points is $\sqrt{3^2 + 2^2} = \sqrt{13}$.

We extend a track consisting of a single curved piece. The root-node of the search tree is shown below:



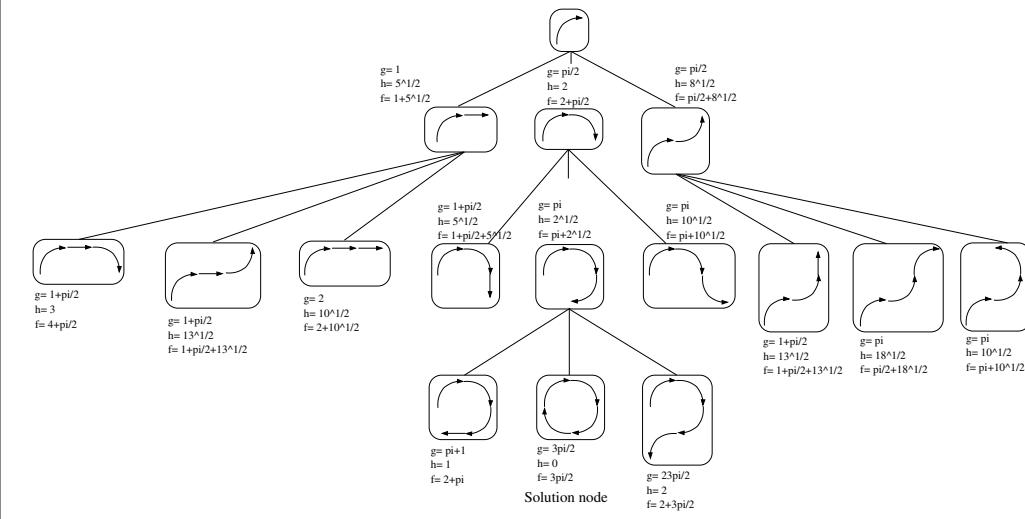
- a) Draw the search tree explored by the BEST-FIRST algorithm (including f -values). Mark the solution node.

a) Answer:



b) Draw the search tree explored by the A* algorithm (including g , h , and f -values). Mark the solution node.

b) Answer:

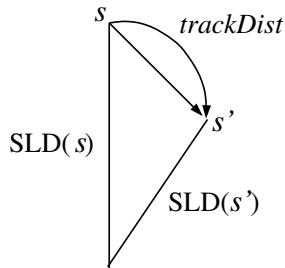


c) Why is the SLD heuristic admissible for this problem?

c) **Answer:** Because any track is longer than or equal to the SLD.

d) Is the SLD heuristic also consistent (why / why not)?

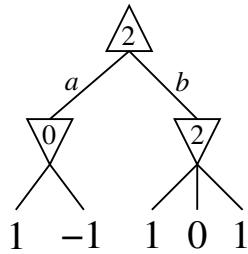
d) **Answer:** Yes, since the tracks are placed on a planar surface, the usual triangle inequality holds for straight track pieces. As shown in the figure below, it must also hold for curved track pieces. Thus, we have $trackDist + SLD(s') \geq SLD(s)$. But this is exactly the consistency requirement: $c(s, a, s') + h(s') \geq h(s)$.



Problem 2: Tic-Tac-Toe (25%)

Professor Smart is skeptical about using the MINIMAX decision for playing Tic-Tac-Toe. Instead he wants to base the decision of which action to play on the utility sum of the game tree.

For this purpose, he has defined the function $\text{UTILITYSUM}(s)$ that for a game state s returns the sum of utility of terminal nodes in the game tree of s . As an example, the UTILITYSUM value is shown for each state of the following game tree:



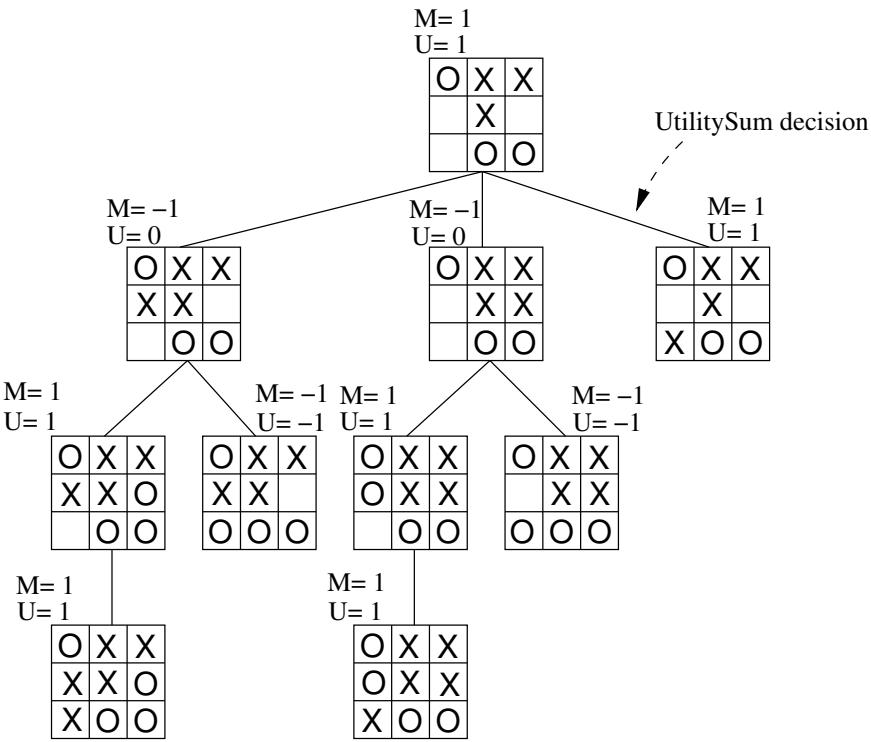
The UTILITYSUM decision of MAX is to choose an action which leads to a state with maximum UTILITYSUM value. Thus, the UTILITYSUM decision of MAX in the example above is to take action b .

Consider the game state q of Tic-Tac-Toe shown below. MAX plays crosses, and it is MAX's turn.

O	X	X
	X	
	O	O

- a) Draw the complete game tree of q and write the UTILITYSUM (U) and MINIMAX (M) value of each state. Mark the UTILITYSUM decision of MAX.

a) Answer:



b) Write the pseudo-code for UTILITYSUM(s).

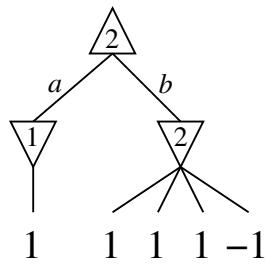
b) Answer:

$$UtilitySum(s) = \begin{cases} Utility(s) & : TerminalTest(s) \\ \sum_{a \in Actions(s)} UtilitySum(Result(s, a)) & : \text{Otherwise} \end{cases}$$

Professor Smart claims that the UTILITYSUM decision is at least as good as the MINIMAX decision for any turn-taking two-player game.

c) Prove that Professor Smart is wrong by drawing a game tree that forms a counter example.

c) **Answer:** The UTILITYSUM decision is b in the counter example shown below. If playing against an optimal opponent, however, this will lead to a terminal state with utility -1 . If playing the MINIMAX decision a on the other hand, a higher utility of 1 is guaranteed.



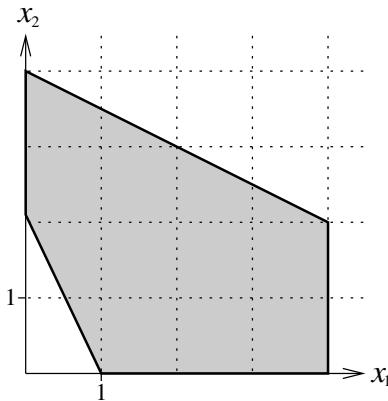
Problem 3: Linear Programming (45%)

Consider the following LP problem written in standard form:

$$\begin{aligned}
 & \text{Maximize} && 2x_1 + x_2 \\
 & \text{Subject to} && \frac{1}{2}x_1 + x_2 \leq 4 \\
 & && x_1 \leq 4 \\
 & && -2x_1 - x_2 \leq -2 \\
 & && x_1, x_2 \geq 0.
 \end{aligned}$$

a) Draw the polyhedron of the feasible set in a coordinate system.

a) Answer:



b) Write the LP problem in slack form.

b) Answer:

$$\begin{aligned}
 & \text{Maximize} && z \\
 & \text{Subject to} && x_3 = 4 - \frac{1}{2}x_1 - x_2 \\
 & && x_4 = 4 - x_1 \\
 & && x_5 = -2 + 2x_1 + x_2 \\
 & && z = 2x_1 + x_2 \\
 & && x_1, x_2, x_3, x_4, x_5 \geq 0
 \end{aligned}$$

c) Why is the dictionary of the slack form infeasible?

c) Answer: The dictionary associated with the slack form is infeasible because x_5 is negative.

- d) Show that the result of the first phase of the two-phase simplex method is the following initial dictionary for the second phase (notice that we assume that simplex always chooses a variable to enter the basis with largest coefficient in the objective expression):

$$\begin{aligned}x_3 &= \frac{7}{2} - \frac{3}{4}x_2 - \frac{1}{4}x_5 \\x_4 &= 3 + \frac{1}{2}x_2 - \frac{1}{2}x_5 \\x_1 &= 1 - \frac{1}{2}x_2 + \frac{1}{2}x_5 \\z &= 2 + x_5\end{aligned}$$

d) Answer:

The dictionary of the auxiliary problem is:

$$\begin{aligned}x_3 &= 4 - \frac{1}{2}x_1 - x_2 + x_0 \\x_4 &= 4 - x_1 + x_0 \\x_5 &= -2 + 2x_1 + x_2 + x_0 \\w &= -x_0\end{aligned}$$

First we perform a pivot to make the dictionary feasible (entering x_0 , leaving x_5):

$$\begin{aligned}x_3 &= 6 - \frac{5}{2}x_1 - 2x_2 + x_5 \\x_4 &= 6 - 3x_1 - x_2 + x_5 \\x_0 &= 2 - 2x_1 - x_2 + x_5 \\w &= -2 + 2x_1 + x_2 - x_5\end{aligned}$$

Since x_1 has largest coefficient, we perform the pivot: entering x_1 , leaving x_0 .

$$\begin{aligned}x_3 &= \frac{7}{2} - \frac{3}{4}x_2 - \frac{1}{4}x_5 + \frac{5}{4}x_0 \\x_4 &= 3 + \frac{1}{2}x_2 - \frac{1}{2}x_5 + \frac{3}{4}x_0 \\x_1 &= 1 - \frac{1}{2}x_2 + \frac{1}{2}x_5 - \frac{1}{2}x_0 \\w &= -x_0\end{aligned}$$

Since all coefficients in w are negative, the dictionary of the first phase is optimal. By removing the x_0 terms in the expression for x_3 , x_4 , and x_1 , we get the required result except that z must be expressed in terms of the non-basic variables. We have $z = 2x_1 + x_2 = 2(1 - \frac{1}{2}x_2 + \frac{1}{2}x_5) + x_2 = 2 + x_5$ as required.

- e) The initial dictionary for the second phase of the two-phase simplex method corresponds to a corner point of the polyhedron of the feasible set. Which?

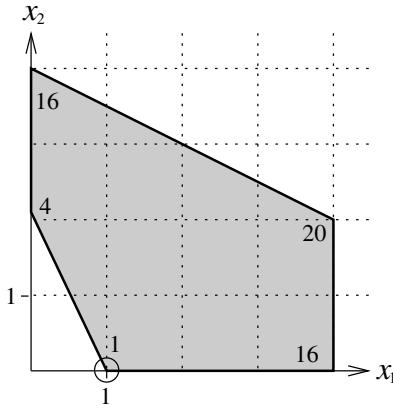
- e) **Answer:** The corner point corresponds to the feasible solution of the dictionary: $\langle 1, 0 \rangle$.

Instead of the objective $z = 2x_1 + x_2$, we want to maximize $z' = x_1^2 + x_2^2$.

The simplex method cannot handle this non-linear objective directly, but it can be shown that z' has maximum value in a corner point of the polyhedron of the feasible set. For that reason, we can define a HILL-CLIMBING algorithm for maximizing z' , where the neighborhood of a point p is defined to be all the corner points that it is possible to pivot to from p .

f) Write the value of z' for each corner point of the polyhedron of the feasible set.

f) Answer:



g) Which corner points belongs to the neighborhood of the corner point of the initial dictionary found in sub-problem e)?

g) Answer: $\langle 0, 2 \rangle$ and $\langle 4, 0 \rangle$.

h) Which variable must enter the basis to pivot to each of the corner points in this neighborhood?

h) Answer: x_2 for $\langle 0, 2 \rangle$ and x_5 for $\langle 4, 0 \rangle$.

i) Which corner point in this neighborhood does the HILL-CLIMBING algorithm choose?

i) Answer: The one with largest z' value: $\langle 4, 0 \rangle$.

j) Write the sequence of corner-points that the HILL-CLIMBING algorithm visits.

j) Answer: $\langle 1, 0 \rangle$, $\langle 4, 0 \rangle$, $\langle 4, 2 \rangle$.

Intelligent Systems Programming (ISP)

IT University of Copenhagen

27 May 2015

This exam consists of 7 numbered pages with 4 problems containing 18 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Linear Programming (25%)

You are in charge of n employees $\{e_1, e_2, \dots, e_n\}$ that must be assigned to n tasks $\{t_1, t_2, \dots, t_n\}$. Each employee is assigned to exactly one task and each task must have exactly one employee assigned to it. The cost of assigning employee e_i to task t_j is c_{ij} . Your objective is to find a valid assignment of your employees with minimum cost.

a) How many valid assignments of employees to tasks are there?

a) **Answer:** The first employee can be assigned to n tasks, the second to $n - 1$ tasks, and so on. In total we have $n!$ different valid assignments.

b) Based on your answer to 1a), would you recommend to use exhaustive search to find an optimal solution (why / why not)?

b) **Answer:** I would not recommend to use exhaustive search since the number of valid solutions $n!$ grows faster than 2^n .

Let $x_{ij} \geq 0$ be a continuous decision variable, where $x_{ij} = 1$ means that e_i is assigned to task t_j and $x_{ij} = 0$ means that e_i is not assigned to task t_j . It can be shown that your assignment problem can be solved optimally by a linear program (LP) using these decision variables.

Consider an instance of your assignment problem with $n = 2$, $c_{11} = 8$, $c_{12} = 6$, $c_{21} = -2$, and $c_{22} = 6$. The LP solving this problem is shown below.

$$\text{Minimize } 8x_{11} + 6x_{12} - 2x_{21} + 6x_{22}$$

$$\text{Subject to } x_{11} + x_{12} \leq 1 \quad (1)$$

$$x_{11} + x_{12} \geq 1 \quad (2)$$

$$x_{21} + x_{22} \leq 1 \quad (3)$$

$$x_{21} + x_{22} \geq 1 \quad (4)$$

$$x_{11} + x_{21} \leq 1 \quad (5)$$

$$x_{11} + x_{21} \geq 1 \quad (6)$$

$$x_{12} + x_{22} \leq 1 \quad (7)$$

$$x_{12} + x_{22} \geq 1 \quad (8)$$

$$x_{11}, x_{12}, x_{21}, x_{22} \geq 0$$

c) Constraints (1-4) can be written as $\sum_{j=1}^2 x_{ij} = 1$ for all $i \in \{1, 2\}$ and constraints (5-8) can be written as $\sum_{i=1}^2 x_{ij} = 1$ for all $j \in \{1, 2\}$. Explain in your own words what aspects of the problem these two constraint groups represent.

c) **Answer:** The first group says that each employee is assigned to exactly one task. The second group says that each task has exactly one employee assigned to it.

d) Translate the LP into standard form. Do not change the order of the constraints.

d) **Answer:** (Obs. we maximize the negative of the objective of the original problem)

$$\begin{aligned}
 \text{Maximize} \quad & -8x_{11} - 6x_{12} + 2x_{21} - 6x_{22} \\
 \text{Subject to} \quad & x_{11} + x_{12} \leq 1 \\
 & -x_{11} - x_{12} \leq -1 \\
 & x_{21} + x_{22} \leq 1 \\
 & -x_{21} - x_{22} \leq -1 \\
 & x_{11} + x_{21} \leq 1 \\
 & -x_{11} - x_{21} \leq -1 \\
 & x_{12} + x_{22} \leq 1 \\
 & -x_{12} - x_{22} \leq -1 \\
 & x_{11}, x_{12}, x_{21}, x_{22} \geq 0
 \end{aligned}$$

e) Write the LP of the dual problem.

e) **Answer:**

$$\begin{aligned}
 \text{Minimize} \quad & y_1 - y_2 + y_3 - y_4 + y_5 - y_6 + y_7 - y_8 \\
 \text{Subject to} \quad & y_1 - y_2 + y_5 - y_6 \geq -8 \\
 & y_1 - y_2 + y_7 - y_8 \geq -6 \\
 & y_3 - y_4 + y_5 - y_6 \geq 2 \\
 & y_3 - y_4 + y_7 - y_8 \geq -6 \\
 & y_1, y_2, \dots, y_8 \geq 0
 \end{aligned}$$

Your colleague claims to have found an optimal primal and dual solution to

this problem. Her primal solution is $x_{11} = 0$, $x_{12} = 1$, $x_{21} = 1$, $x_{22} = 0$. Her dual solution is $y_1 = 0$, $y_2 = 6$, $y_3 = 2$, $y_4 = 0$, $y_5 = 0$, $y_6 = 0$, $y_7 = 0$, and $y_8 = 0$, where y_i is the dual variable of constraint (i) in the primal problem.

f) Show that your colleague is right. What is the assignment of the solution and what is the cost of it?

f) Answer:

- By insertion into the constraints of the primal and dual problem, we see that both solutions are feasible.
- By insertion into the objective expression of the primal and dual problem, we get -4 in both cases.

Since the dual solution is an upper bound of the primal solution, this shows that both solutions are feasible and optimal.

The solution assigns e_1 to t_2 and e_2 to t_1 . The cost of the assignment is 4, since we maximize the negative value of the original objective.

Problem 2: Logic and BDDs (40%)

Let α denote the sentence $(\neg R \Rightarrow \neg(P \vee Q)) \wedge P$ in propositional logic.

a) Transform α to conjunctive normal form.

a) Answer:

$$\begin{aligned}
 (\neg R \Rightarrow \neg(P \vee Q)) \wedge P &\equiv ((P \vee Q) \Rightarrow R) \wedge P \\
 &\equiv (\neg(P \vee Q) \vee R) \wedge P \\
 &\equiv ((\neg P \wedge \neg Q) \vee R) \wedge P \\
 &\equiv (\neg P \vee R) \wedge (\neg Q \vee R) \wedge (P)
 \end{aligned}$$

b) Use the PL-RESOLUTION() algorithm to show $\alpha \models R \wedge P$. Write the resolvents found in each iteration of the algorithm.

b) Answer: The CNF representation of $\alpha \wedge \neg(R \wedge P)$ is $(\neg P \vee R) \wedge (\neg Q \vee R) \wedge (P) \wedge (\neg R \vee \neg P)$. The resolvents of each iteration of PL-RESOLUTION() are:

1. $(R), (\neg P), (\neg Q \vee \neg P), (\neg R)$
2. $(R), (\neg P), (\neg Q \vee \neg P), (\neg Q), (\neg R), ()$

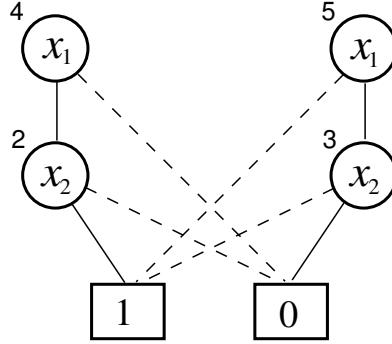
PL-RESOLUTION() returns true in second iteration as the empty clause is found. This proves the entailment. Notice that we assume that PL-RESOLUTION() finds the empty clause as the last resolvent such that it produces all resolvents in second iteration. It might equally well have found the empty clause as the first resolvent and only produce this resolvent in second iteration.

c) Are α and $R \wedge P$ logically equivalent?

c) Answer: Yes, they are as shown below.

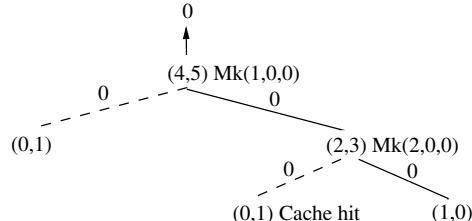
$$\begin{aligned}
 (\neg R \Rightarrow \neg(P \vee Q)) \wedge P &\equiv ((P \vee Q) \Rightarrow R) \wedge P \\
 &\equiv (\neg(P \vee Q) \vee R) \wedge P \\
 &\equiv ((\neg P \wedge \neg Q) \vee R) \wedge P \\
 &\equiv (\neg P \wedge \neg Q \wedge P) \vee (R \wedge P) \\
 &\equiv False \vee (R \wedge P) \\
 &\equiv R \wedge P
 \end{aligned}$$

The multi-rooted unique table shown below contains BDDs on the variables x_1 , x_2 , and x_3 in the order $x_1 \prec x_2 \prec x_3$.



- d) Draw the call tree of $\text{APPLY}(\wedge, 4, 5)$. Show the associated call to Mk for each node. Write the cache entries generated during this call and show the cache hits in the call tree. Describe the resulting multi-rooted unique table.

d) **Answer:** The call tree is shown below.



The new cache entries are $(0, 1) \rightarrow 0$, $(1, 0) \rightarrow 0$, $(2, 3) \rightarrow 0$, and $(4, 5) \rightarrow 0$. Since no new BDD was generated, the multi-rooted unique table is unchanged.

- e) What is returned by $\text{APPLY}(\wedge, 4, 5)$? What is the relation between this result and the result by $\text{PL-RESOLUTION}()$ in your answer to 2b) if you assume that $x_1 \equiv R$, $x_2 \equiv P$, and $x_3 \equiv Q$? Explain your answer well.

e) **Answer:** As shown in the call tree above, $\text{APPLY}(\wedge, 4, 5)$ returns 0 which represents *False*. With the given mapping, node 4 is logically equivalent to $R \wedge P$ since only $x_1 = 1$ and $x_2 = 1$ lead to the terminal 1 node. Using a similar argument, node 5 is seen to be logically equivalent to $\neg(R \wedge P)$. From the answer to 2c), this means that node 4 is logically equivalent to α . Thus, $\text{APPLY}(\wedge, 4, 5)$ corresponds to the proof by contradiction $\alpha \wedge \neg(R \wedge P)$ computed by PL-RESOLUTION in 2b). As for this algorithm, the result of APPLY shows that $\alpha \wedge \neg(R \wedge P) \equiv \text{False}$ which means that $\alpha \models R \wedge P$ as expected.

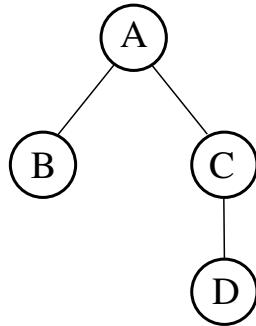
Problem 3: Constraint Programming (25%)

Given a tree T with $m + 1$ vertices and m edges, the *graceful labeling problem* is to find the possible labels in $\{0, 1, \dots, m\}$ for the vertices such that the absolute value of the difference of the labels of the vertices related to each edge are all different.

Formally, let f be a function from V to $\{0, 1, \dots, m\}$, where V is the set of the vertices of T . The function f is said to be a graceful labeling of T if and only if

- the labels of vertices are all different, and
- $|f(v_i) - f(v_j)|$ are all different for any edge (v_i, v_j) in T .

Consider the tree shown below.



Your task is to find a graceful labeling for this tree using constraint programming. To this end, assume that there are four variables $\{A, B, C, D\}$ each with a domain $\{0, 1, 2, 3\}$ to represent the labeling of the vertices in the tree.

a) Write the constraints of the problem.

a) Answer:

- AllDiff(A, B, C, D)
- AllDiff($|A - B|, |A - C|, |C - D|$)

Assume that the problem is solved with BACKTRACKING-SEARCH using a fixed variable ordering.

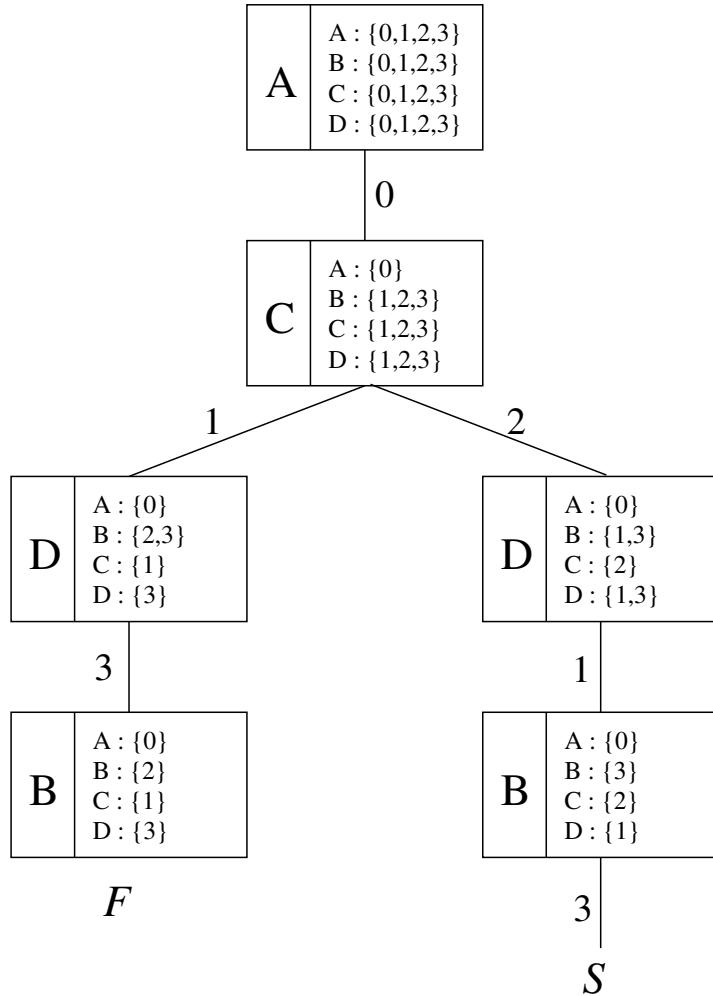
b) Which variable order would you choose and why?

b) Answer: Following the degree heuristic, I apply the fixed ordering $A \prec C \prec D \prec B$.

C) Show the call tree of FORWARD-CHECKING-SEARCH using your fixed variable ordering. Each call node is represented by the table shown below. V is the name of the variable assigned in the call. The right side of the table shows the remaining domains of the variables in the beginning of the call. An edge in the call tree is labeled with the value assigned to the variable of the call node it goes from. Apply the values in ascending order. Mark a failed node with F . Mark the node finding a complete solution with S . What is the solution found?

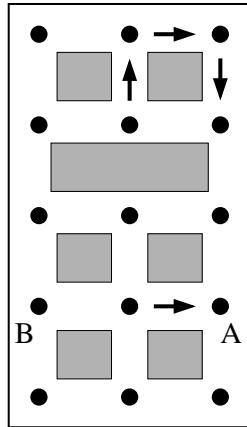
V	A : {0,1,2,3} B : {0,1,2,3} C : {0,1,2,3} D : {0,1,2,3}
-----	--

c) Answer:



The solution is $A = 0$, $B = 3$, $C = 2$, and $D = 1$.

Problem 4: Heuristics (10%)



The drawing above shows a city map. The arrows indicate one way streets. The vertices (black dots) are points where turn decisions can be made.

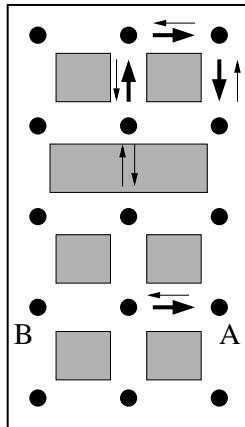
Assume that you use the Manhattan distance between two vertices as a heuristic estimate h of the length of a shortest path between them.

- a) If B is the goal vertex, what is then $h(A)$ and $h^*(A)$, where h^* denotes the actual length of a shortest path between A and B ?

a) **Answer:** $h(A) = 2$ (two left moves). $h^*(A) = 4$ (e.g., up, left, left, down).

- b) Make arrows in the drawing corresponding to the extra streets that must exist in the city for $h^* = h$ for any choice of goal vertex.

b) **Answer:**



c) Is h admissible for any choice of goal vertex?

c) **Answer:** Yes, the answer to 4b) shows that h is a relaxation of the problem.

d) Is h consistent for any choice of goal vertex?

c) **Answer:** Yes, the answer to 4b) shows that h is a relaxation of the problem.

Intelligent Systems Programming (ISP)

IT University of Copenhagen

1 June 2016

This exam consists of 7 numbered pages with 4 problems containing 20 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

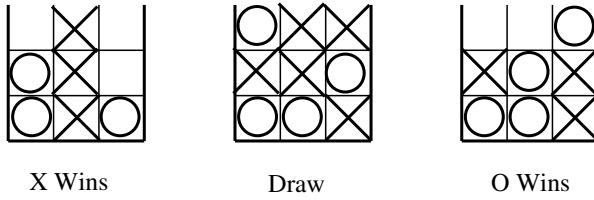
Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Adversarial Search (25%)

Connect Three is a simplified version of Connect Four. It is a two-player game, where the players take turns dropping “O” and “X” marked discs from the top into a three-column, three-row vertically suspended grid. The pieces fall straight down occupying the next available space within the column. The game terminates when three discs with the same mark are next to each other either vertically, horizontally, or diagonally and the player playing that mark wins. Otherwise the game terminates when all positions are occupied, and it is a draw. Three examples of terminal states are shown below.

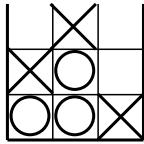


In the following, assume that the O-player is MAX and the X-player is MIN. Further assume that the utility of a terminal state is +2 for MAX and -2 for MIN, if MAX wins; -2 for MAX and +2 for MIN, if MIN wins; and 0 for MAX and 0 for MIN, if the game is a draw.

- a) Is Connect Three a zero-sum game (why/why not)?

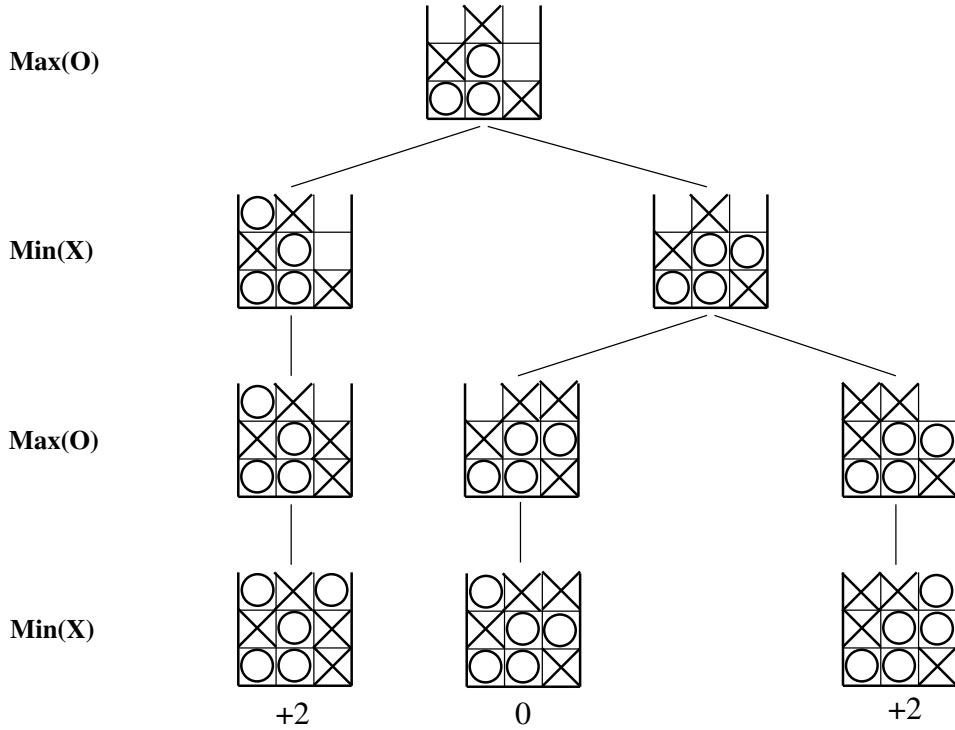
a) Answer: Yes, it is a zero-sum game, since the sum of utility for MAX and MIN is a constant, 0, in all terminal states.

In the game state shown below, it is MAX’s turn to drop an O-disc.



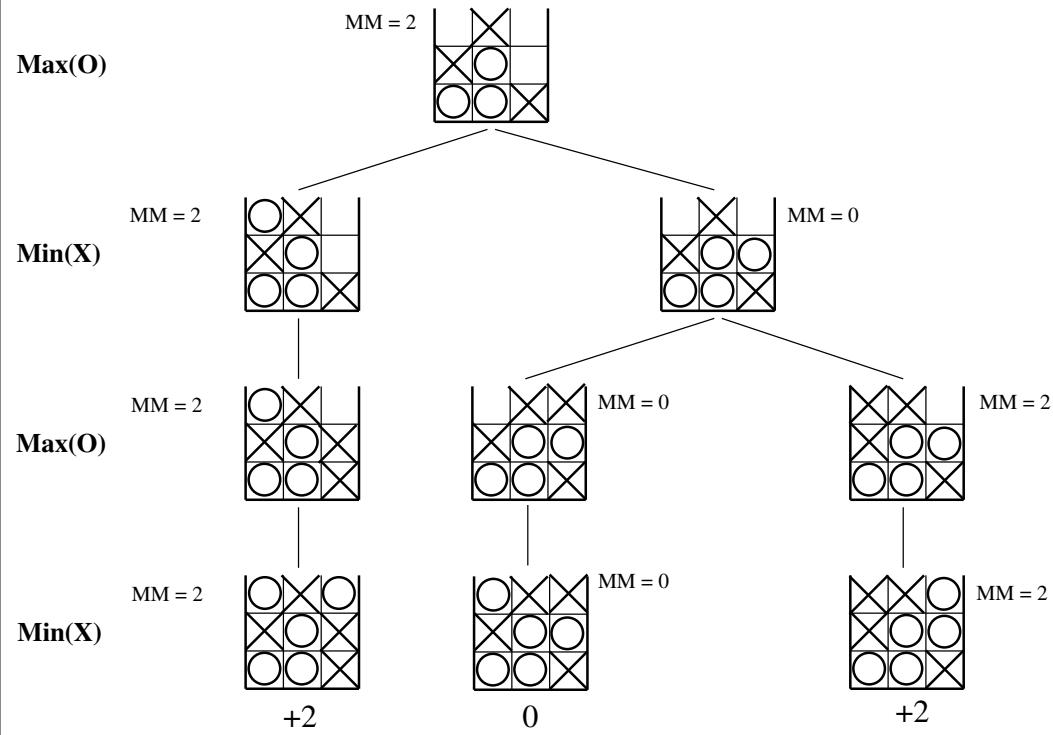
- b) Draw the complete game tree for this state. When building the game tree, you must assume that MAX drops discs into columns in the order from left to right, while MIN drops discs into columns in the order from right to left. Note the utility of MAX for each terminal state in the tree.

b) Answer:



c) Write the minimax value next to each node in the game tree. Specifically, write $MM = v$ next to a node, if its minimax value is v .

c) Answer:

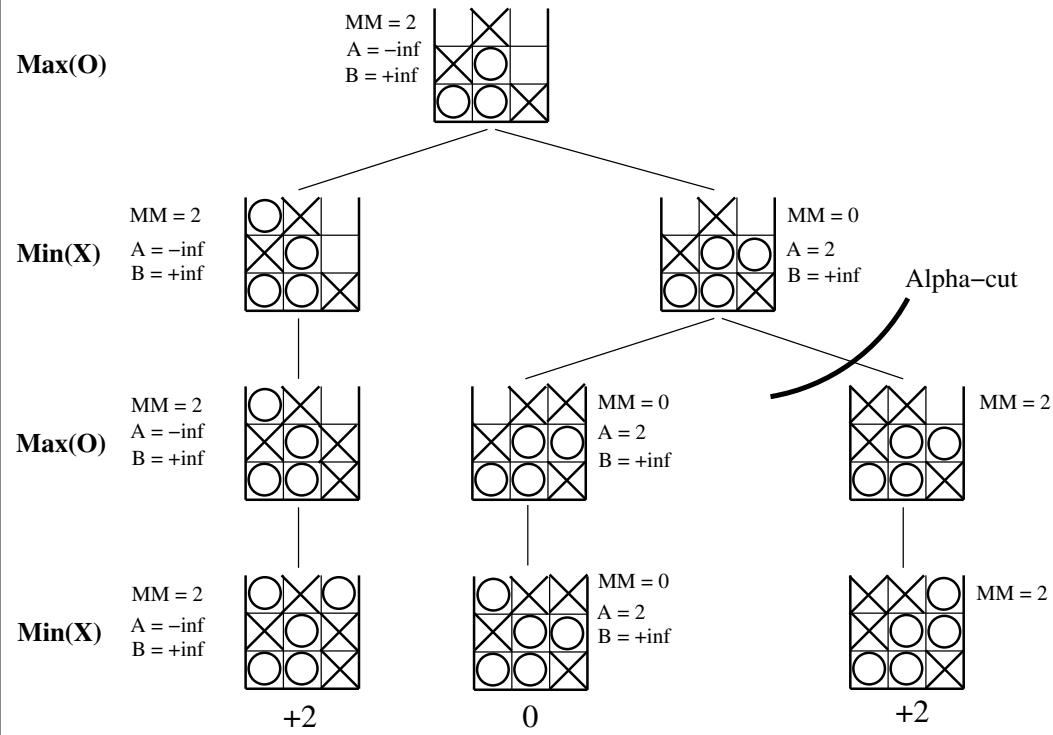


d) What action does MAX choose? You must refer to the minimax values of the game tree in your answer.

d) Answer: MAX chooses to drop a disc into the left-most column as the corresponding child node in the game tree has a maximum minimax value.

e) Assume that ALPHA-BETA-SEARCH is used to explore the game tree. For each node visited, write $A = p$ and $B = q$ next to the node, if the alpha and beta value passed down from its parent node is p and q , respectively. Show any cut in the tree and indicate whether it is an alpha-cut or beta-cut.

e) Answer:



f) Is it possible to increase the number of nodes pruned by ALPHA-BETA-SEARCH in the game tree by changing the order that MAX and MIN drops discs into the columns (why/why not)? Make a careful argument.

f) Answer: No, in the first layer of the tree MAX can swap its decision and try the right-most column first. But this way the winning solution of MAX will not have been found in the left most subtree, making it impossible for MIN to do an alpha-cut. So MAX needs to play the left-most column first. In this situation, MIN has a choice in the second layer of the game tree to try the left-most column first. This will also lead to an alpha-cut, but it prunes only two nodes as before.

Problem 2: Logic (25%)

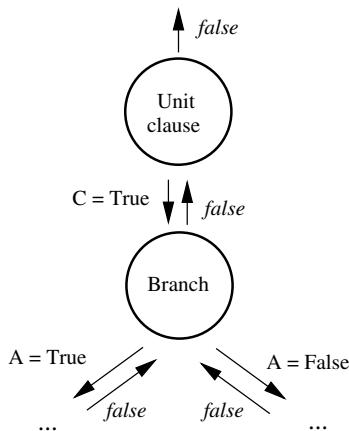
a) Convert each of the following sentences to CNF form:

$$\begin{aligned} S1 : & A \Leftrightarrow (B \vee E) \\ S2 : & E \Rightarrow D \\ S3 : & C \wedge F \Rightarrow \neg B \\ S4 : & E \Rightarrow B \\ S5 : & B \Rightarrow F \\ S6 : & B \Rightarrow C \end{aligned}$$

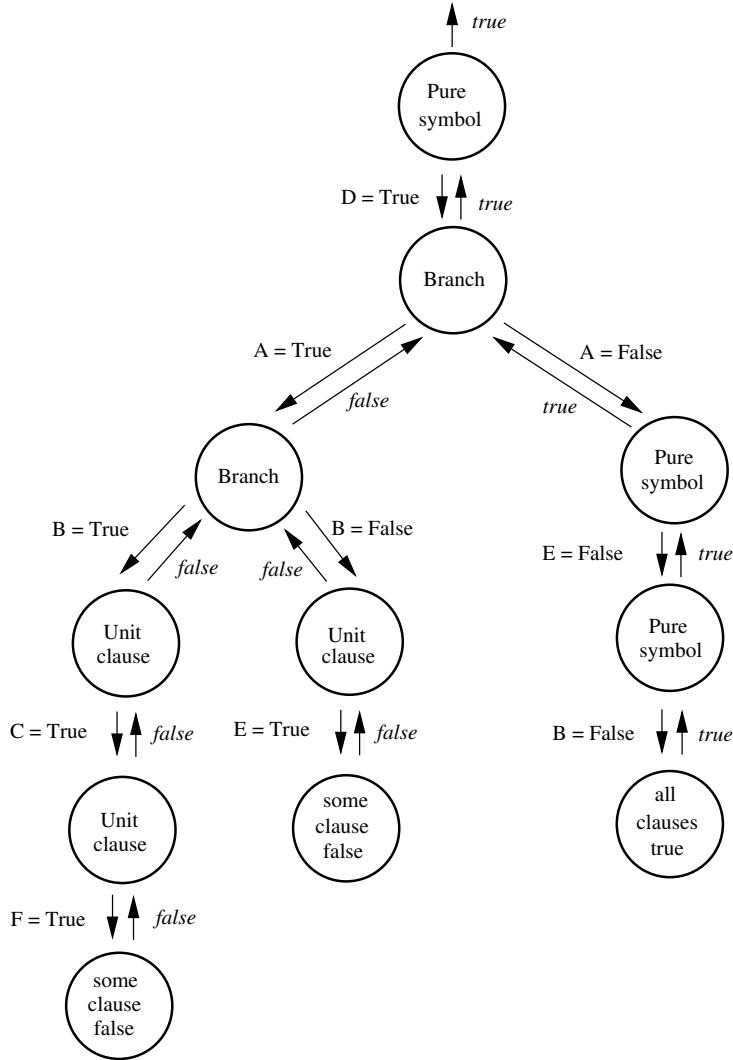
a) Answer:

$$\begin{aligned} S1 : & (\neg A \vee B \vee E) \wedge (\neg B \vee A) \wedge (\neg E \vee A) \\ S2 : & \neg E \vee D \\ S3 : & \neg C \vee \neg F \vee \neg B \\ S4 : & \neg E \vee B \\ S5 : & \neg B \vee F \\ S6 : & \neg B \vee C \end{aligned}$$

b) Draw the call tree of the execution of the DPLL algorithm on the conjunction of these sentences. If a recursive call is made at a node, then write inside it what caused the recursive call (i.e., *Pure symbol*, *Unit clause*, or *Branch*). Assume that DPLL searches through the propositional symbols in lexicographical order when scanning for pure symbols and unit clauses and when branching. Also assume that DPLL first tries the edge assigning a symbol to *True* when branching on it. Write next to each edge in the tree, the symbol assignment associated with it. Finally, draw back-edges with return values. The drawing below shows an example of a part of a call tree.



b) Answer:

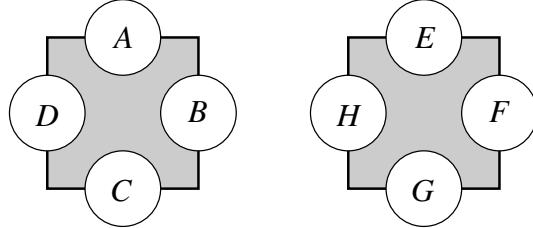


c) Write a satisfying truth assignment corresponding to the result returned by DPLL, if any exists.

c) Answer: $A = \text{False}$, $B = \text{False}$, $C = \text{True}$, $D = \text{True}$, $E = \text{False}$, and $F = \text{True}$.

Problem 3: Local Search (20%)

Claire is a wedding planner and must assign eight guests G_1, G_2, \dots, G_8 to eight chairs A, B, \dots, H placed around two tables as shown below.



The assignment must fulfill that

- the same number of men and women are assigned to each table, and
- men and women alternate (i.e., if a man is assigned to A , a woman must be assigned to B etc.).

Each guest may like or dislike a number of other guests. Let (g, v) denote such a preference, where g is a guest and v is a positive or negative integer. When two guests sit at the same table, any preferences they have towards each other are active.

Claire's objective is to find a feasible assignment that maximizes the total sum of the active preferences. She knows the following about the guests: guests with even ID are women; guests with odd ID are men; the preferences of G_8 are $(G_2, -8)$ and $(G_6, -6)$; the preference of G_2 is $(G_8, -2)$; the preference of G_4 is $(G_5, +4)$; the preference of G_5 is $(G_4, +4)$; no other guests have preferences.

For these guests the assignment ($A = G_1, B = G_2, C = G_5, D = G_8, E = G_3, F = G_4, G = G_7, H = G_6$) or in short $(1, 2, 5, 8, 3, 4, 7, 6)$ is feasible and the total sum of the active preferences is $-8 + (-2) = -10$.

Claire claims that among the approaches *local search*, *constraint programming*, and *linear programming*, it is only local search that directly can be used to solve this problem.

a) Is Claire correct (why/why not)?

a) Answer: Yes, while constraint programming can model the constraints, we cannot represent the objective. Linear programming cannot represent discrete variables. Local search methods can both represent the constraints and the objective.

Claire considers a swapping neighborhood, where a pair of guests only can exchange chairs if:

- the guests are of the same sex, and
- the guests currently are assigned to different tables.

b) Does this neighborhood maintain feasibility (i.e., if the current assignment is feasible then all assignments in its neighborhood are also feasible)? Argue for your answer.

b) Answer: Yes, the swap does not change the number of guests of each sex around the two tables and the alternation between men and women.

c) Is this neighborhood complete in the sense that an arbitrary feasible assignment can be reached from any initial feasible assignment through a number of swaps (why/why not)?

c) Answer: No, we cannot change the initial alternation between women and men.

d) Can Claire use the neighborhood to solve the assignment problem (why/why not)?

d) Answer: Yes, the constraints do not require a particular alternation between men and women, so she can constraint the problem to only consider the one given in an initial feasible solution.

e) Write the complete neighborhood for the assignment $(1, 2, 5, 8, 3, 4, 7, 6)$. Write the total sum of the active preferences for each neighbor. What neighbor would the HILL-CLIMBING algorithm choose as the next state?

e) Answer:

$$(3, 2, 5, 8, 1, 4, 7, 6) : -2 + (-8) = -10$$

$$(7, 2, 5, 8, 3, 4, 1, 6) : -2 + (-8) = -10$$

$$(1, 4, 5, 8, 3, 2, 7, 6) : 4 + 4 = 8$$

$$(1, 6, 5, 8, 3, 4, 7, 2) : -6$$

$$(1, 2, 7, 8, 3, 4, 5, 6) : -2 + (-8) + 4 + 4 = -2$$

$$(1, 2, 3, 8, 5, 4, 7, 6) : -2 + (-8) + 4 + 4 = -2$$

$$(1, 2, 5, 6, 3, 4, 7, 8) : 0$$

$$(1, 2, 5, 4, 3, 8, 7, 6) : 4 + 4 - 6 = 2$$

The HILL-CLIMBING algorithm would choose $(1, 4, 5, 8, 3, 2, 7, 6)$ as the next state.

Problem 4: Linear Programming (30%)

Refinery Thailand (RT) converts crude oil into gasoline and diesel oil. RT has 100 tons of crude oil available at a cost of 30 thousand baht per tons [a]. The minimum mass of crude oil needed is the sum of the total mass of gasoline and diesel oil produced minus 8 tons due to some additives that we will ignore here [b]. RT can sell one ton of gasoline and diesel oil for 100 thousand and 60 thousand bath, respectively [c]. Gasoline is the more valuable product, but unfortunately due to the refinery process, the mass of produced diesel oil must be larger than the mass of produced gasoline minus 6 tons [d]. RT wants to maximize its profit which is the income from sales of produced gasoline and diesel oil subtracted the cost of the crude oil needed for the production [e]. Let x_1 , x_2 , and x_3 denote the mass of gasoline, diesel oil, and crude oil in tons, respectively.

- a) Explain why the above optimization problem can be mathematically formalized as follows: maximize $100x_1 + 60x_2 - 30x_3$ (1) subject to $x_3 \geq x_1 + x_2 - 8$ (2), $x_2 \geq x_1 - 6$ (3), $x_3 \leq 100$ (4), and $x_1, x_2, x_3 \geq 0$ (5). Refer to all sentences [a-e] and expressions (1-5) in your explanation.

- a) **Answer:** (1) represents the profit in thousands bath according to the profit definition [e] using the sale prices and costs mentioned in [a] and [c]. (2) expresses the amount of crude oil needed for producing gasoline and diesel oil as defined by [b]. (3) expresses that the amount of diesel oil is lower bounded by the amount of gasoline as described in [d]. (4) limits the amount of crude oil as defined by [a]. Finally, (5) expresses the physical fact that the masses used in the production cannot be negative.

- b) Show that the optimization problem is a linear program (LP) by converting it to standard form.

- b) **Answer:**

$$\begin{array}{lllll} \text{Maximize} & 100x_1 & + & 60x_2 & - 30x_3 \\ \text{Subject to} & x_1 & + & x_2 & - x_3 \leq 8 \\ & x_1 & - & x_2 & \leq 6 \\ & & & & x_3 \leq 100 \\ & & & & x_1, x_2, x_3 \geq 0 \end{array}$$

- c) Write the slack form of the LP. Is the slack form a feasible initial dictionary for the SIMPLEX algorithm (why/why not)?

c) Answer:

$$\begin{aligned}
 & \text{Maximize } z \\
 \text{Subject to } & x_4 = 8 - x_1 - x_2 + x_3 \\
 & x_5 = 6 - x_1 + x_2 \\
 & x_6 = 100 - x_3 \\
 & z = 100x_1 + 60x_2 - 30x_3 \\
 & x_1, x_2, \dots, x_6 \geq 0
 \end{aligned}$$

d) Solve the problem using the SIMPLEX algorithm. Write all dictionaries produced by the algorithm.

d) Answer:

Initial dictionary : entering x_1 , leaving x_5

Second dictionary : entering x_2 , leaving x_4

$$\begin{aligned}
 & \text{Maximize } z \\
 \text{Subject to } & x_4 = 2 - 2x_2 + x_3 + x_5 \\
 & x_1 = 6 + x_2 - x_5 \\
 & x_6 = 100 - x_3 \\
 & z = 600 + 160x_2 - 30x_3 - 100x_5
 \end{aligned}$$

Third dictionary : entering x_3 , leaving x_6

$$\begin{aligned}
 & \text{Maximize } z \\
 \text{Subject to } & x_2 = 1 + 0.5x_3 - 0.5x_4 + 0.5x_5 \\
 & x_1 = 7 + 0.5x_3 - 0.5x_4 - 0.5x_5 \\
 & x_6 = 100 - x_3 \\
 & z = 760 + 50x_3 - 80x_4 - 20x_5
 \end{aligned}$$

Fourth dictionary : optimal

$$\begin{aligned}
 & \text{Maximize } z \\
 \text{Subject to } & x_2 = 51 - 0.5x_4 + 0.5x_5 - 0.5x_6 \\
 & x_1 = 57 - 0.5x_4 - 0.5x_5 - 0.5x_6 \\
 & x_3 = 100 - x_6 \\
 & z = 5760 - 80x_4 - 20x_5 - 50x_6
 \end{aligned}$$

e) What is RT's maximum profit and how much gasoline and diesel oil is produced to achieve this profit?

e) **Answer:** The objective expression of the optimal dictionary proves the maximum profit of RT to be 5760 thousand bath achieved by producing 57 tons of gasoline and 51 tons of diesel oil.

f) RT has an option for buying more crude oil before the production. What is the highest price that RT can accept without losing money?

f) **Answer:** The limit on crude oil is 100 tons. The optimal dual value for this constraint can be read from the optimal dictionary as the negative value of the coefficient in the objective expression for x_6 . Thus, the dual value for the constraint is 50. In other words, RT can expect an increase in profit of 50 thousand bath per extra tons of crude oil, which then defines the highest crude oil price that RT can accept without losing money.

Intelligent Systems Programming (ISP)

BSc and MSc Exam

IT University of Copenhagen

7 June 2017

This exam consists of 6 numbered pages with 3 problems containing 17 sub-problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam.

You can answer the questions in English or Danish.

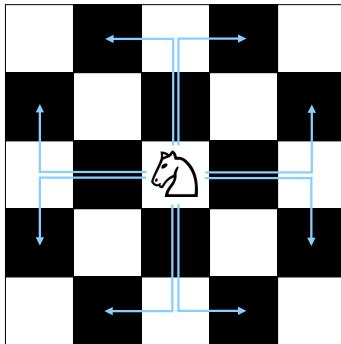
Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Constraint Programming and Heuristic Search (40%)

Recall that a knight in the game of chess can move and attack as shown by the arrows in the figure below.



In the following, we want to place five knights on a 3x3 board such that none of them can attack each other. Let A, B, C, D , and E denote the five knights. Initially, the knights are placed as shown below.

3	A	B
2	C	D
1		E

a b c

In this state, A can attack E by moving to position $2c$, A can also move to position $1b$ without attacking another knight.

Just as for the 8-queens problem, let the number of conflicts of a knight equal the number of knights that can attack it.

- a) For the initial state of the board shown above, write the number of conflicts of each knight next to its letter.

a) Answer:

3	A ¹		B ¹
2	C ¹	D ⁰	E ¹
1			

a b c

b) Use the MIN-CONFLICTS constraint programming algorithm to solve the problem. Assume that MIN-CONFLICTS instead of choosing a conflicted knight at random chooses the one that comes first in the lexicographical order $A \prec B \prec C \prec D \prec E$. Further, assume that a knight can be moved to any position that is not currently occupied by another knight (i.e., not just the positions that can be reached by legal moves). Draw the state of a board in each iteration. Draw the solution that MIN-CONFLICTS returns.

b) Answer: The number written in each empty position is the number of violations that the knight to move has at that position.

3	A ¹	0	B ¹
2	C ¹	D ⁰	E ¹
1	1	1	1

Iteration 1:
Min-Conflicts moves A
from 3a to 3b

3	1	A ⁰	B ¹
2	C ¹	D ⁰	E ⁰
1	1	0	1

Iteration 2:
Min-Conflicts moves B
from 3c to 1b

3		A	
2	C	D	E
1		B	

Iteration 3:
No conflicts!
Min-Conflicts returns the shown solution

We now want to show that it also is possible to solve the problem if we only make legal moves of the knights. To that end, we define the problem as a heuristic search problem, where: the initial state is the initial configuration of the board shown above; the applicable actions are legal moves of the knights; a goal state is one, where no knights can attack each other; and the cost of applying an action is 1.

Consider the GRAPH-SEARCH version of greedy best-first search using the heuristic function $h(n)$, where the value of $h(n)$ is defined to be the total sum of conflicts of the knights in the state of n (e.g., in a state of n , where A, B, C and D each has 1 conflict and E each has 0 conflicts, we have $h(n) = 4$).

c) Why is $h(n)$ a valid heuristic function for the search problem?

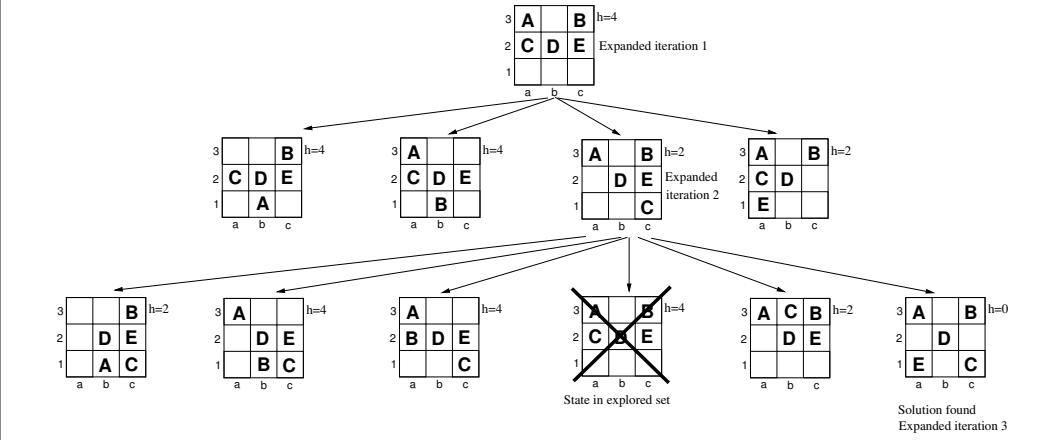
c) **Answer:** Because it is zero in any goal state and never negative.

d) Is $h(n)$ also an admissible heuristic? (why / why not)

d) **Answer:** No, it may overestimate the remaining cost to reach a goal state. For instance, MIN-CONFLICTS' last move of B in the answer to b) above is a legal move to a goal state, but $h(n)$ is 2 in the parent state, where it at most should be 1.

e) Draw the search tree of the GRAPH-SEARCH version of greedy best-first search using the heuristic function $h(n)$. Write each search node as a board state and note its h -value. Indicate the expanded nodes.

e) **Answer:**



f) What is the solution returned by the algorithm?

f) **Answer:** Move C from 2a to 1c, Move E from 2c to 1a.

g) Prof. Smart claims that a consistent heuristic to reach this solution is the number of misplaced knights (i.e., knights not yet at their goal position). Is Prof. Smart right? (why/why not)

f) **Answer:** He is right, since his heuristic corresponds to a relaxation of the rules of how knights can move, it is both admissible and consistent.

Problem 2: Propositional Logic (45%)

A group of people at the faculty where Prof. Smart works has arranged a movie festival at the university, featuring lesser known AI-themed movies. In the following week, two movies will be shown each day from Wednesday to Friday as listed below:

	Wednesday	Thursday	Friday
4-6 p.m.	Identity Matrix	SimplEx Machina	Identity Matrix: Transposed
6-8 p.m.	Need for Speedy Algorithms	$\neg X\text{-Men}$	U, Robot

Prof. Smart wants to join and makes a list of criteria for his attendance:

- (i) He wants to watch at least one movie per day.
- (ii) “Identity Matrix: Transposed” is the sequel to “Identity Matrix”, so he wants to either watch both, or none, of these.
- (iii) Although he has flexible working hours, Prof. Smart will not be able to attend the early screening two days in a row.

At home, Prof. Smart tells his wife about his plan to attend the movie festival. She wants to join him, adding that:

- (iv) If he is going to watch either “ $\neg X\text{-Men}$ ” or “SimplEx Machina” (which she doesn’t find that interesting), then she wants to watch “U, Robot” in return.

a) Translate each of the statements (i) - (iv) above into a sentence in propositional logic. Use variables named x_1, \dots, x_6 , where x_i states that Prof. Smart and his wife watch the i 'th movie in the screening order (you should assume that Prof. Smart and his wife will watch the same movies).
 Translate your expressions into CNF; be careful to write your intermediate results.

a) Answer:

- (i) $(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6)$
- (ii) $(x_1 \wedge x_5) \vee (\neg x_1 \wedge \neg x_5)$ (or, equivalently, $x_1 \Leftrightarrow x_5$).
- (iii) $(x_1 \Rightarrow \neg x_3) \wedge (x_3 \Rightarrow \neg x_5)$
- (iv) $(x_4 \vee x_3) \Rightarrow x_6$

The expressions in CNF are:

$$(i) \ (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6)$$

(ii)

$$\begin{aligned} (x_1 \wedge x_5) \vee (\neg x_1 \wedge \neg x_5) &\equiv ((x_1 \wedge x_5) \vee \neg x_1) \wedge ((x_1 \wedge x_5) \vee \neg x_5) \\ &\equiv (x_1 \vee \neg x_1) \wedge (x_5 \vee \neg x_1) \wedge (x_1 \vee \neg x_5) \wedge (x_5 \vee \neg x_5) \\ &\equiv \text{true} \wedge (x_5 \vee \neg x_1) \wedge (x_1 \vee \neg x_5) \wedge \text{true} \\ &\equiv (x_5 \vee \neg x_1) \wedge (x_1 \vee \neg x_5). \end{aligned}$$

Or, $x_1 \Leftrightarrow x_5 \equiv (x_1 \Rightarrow x_5) \wedge (x_5 \Rightarrow x_1) \equiv (\neg x_1 \vee x_5) \wedge (\neg x_5 \vee x_1)$.

$$(iii) \ (x_1 \Rightarrow \neg x_3) \wedge (x_3 \Rightarrow \neg x_5) \equiv (\neg x_1 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_5)$$

$$(iv) \ (x_4 \vee x_3) \Rightarrow x_6 \equiv \neg(x_4 \vee x_3) \vee x_6 \equiv (\neg x_4 \wedge \neg x_3) \vee x_6 \equiv (\neg x_4 \vee x_6) \wedge (\neg x_3 \vee x_6)$$

At work the next day, Prof. Smart finds out that he had seen an early version of the program, but now the final program is ready. Since the order of the movies has changed, as well as the screening times, the CNF-expression of Prof. Smart and his wife's rules are now:

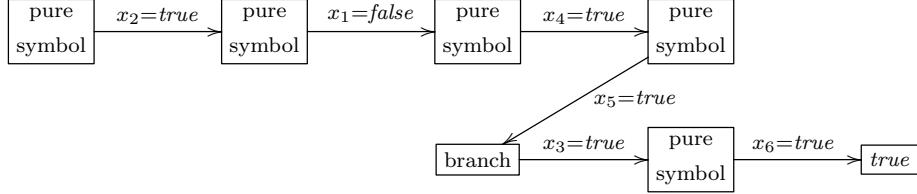
$$\begin{aligned} &(\neg x_5 \vee x_2) \wedge (x_5 \vee x_6) \wedge (\neg x_3 \vee x_4) \wedge (x_1 \vee x_2) \wedge (x_6 \vee \neg x_3) \\ &\quad \wedge (\neg x_1 \vee \neg x_4) \wedge (x_3 \vee \neg x_6) \wedge (x_3 \vee x_4) \wedge (\neg x_1 \vee x_2) \quad R \end{aligned}$$

Prof. Smart is relieved to see that their problem is still satisfiable.

b) Confirm Prof. Smart's insight by using DPLL to find a satisfying assignment for R .

Use the following instructions: Draw the call tree of DPLL, where each node correspond to a call of DPLL. Indicate at each non-leaf node what caused the recursive call corresponding to its child(ren) (i.e. “Pure symbol”, “Unit clause”, or “Branch”), and write on the edge to each of its children the assignment associated with it (e.g. “ $x_1 = \text{true}$ ”). Label leafs with the value (“ true ” or “ false ”) that is returned at that node. Assume that the algorithm uses the natural ordering of the variables when searching for pure symbols and unit clauses.

b) Answer: The DPLL call tree is:



The solution found by DPLL is $x_1 = \text{false}$, $x_2 = x_3 = x_4 = x_5 = x_6 = \text{true}$.

c) Prof. Smart wonders whether his rules imply that he must watch the last movie, i.e. whether R entails x_6 . He considers using forward chaining (i.e., the PL-FC-ENTAILS algorithm) for finding out. Is that a good idea? (why/why not)

c) Answer: Forward chaining only works on definite clauses, and several of the clauses from R are not definite, e.g. $x_1 \vee x_2$. Therefore, forward chaining cannot be used to check whether R entails x_6 (and hence it is not a good idea).

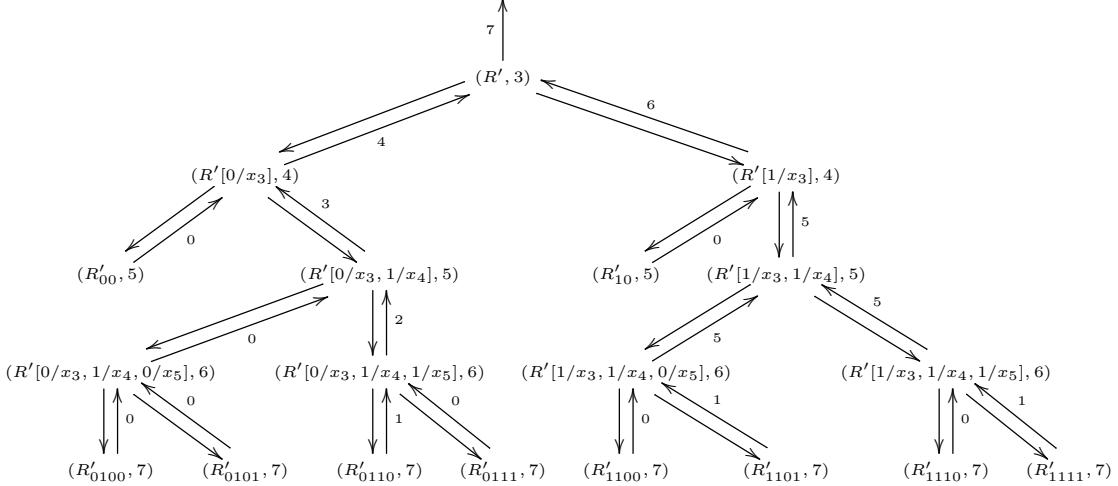
The following Monday, Prof. Smart realizes that he has a meeting that means that he cannot make it to the screening of the first movie. Thus, he must let $x_1 = \text{false}$. On recommendation from a colleague, he also decides to let $x_2 = \text{true}$.

d) Write the expression R' where $R' = R[0/x_1, 1/x_2]$. Make the binary decision diagram of R' using the order $x_3 \prec x_4 \prec x_5 \prec x_6$. Show the call tree of $\text{BUILD}'(R', 3)$; you are allowed to assume that the algorithm uses early termination.

d) Answer:

$$\begin{aligned}
 R' &= (\neg x_5 \vee 1) \wedge (x_5 \vee x_6) \wedge (\neg x_3 \vee x_4) \wedge (0 \vee 1) \wedge (x_6 \vee \neg x_3) \\
 &\quad \wedge (1 \vee \neg x_4) \wedge (x_3 \vee \neg x_6) \wedge (x_3 \vee x_4) \wedge (1 \vee 1) \\
 &\equiv (x_5 \vee x_6) \wedge (\neg x_3 \vee x_4) \wedge (x_6 \vee \neg x_3) \wedge (x_3 \vee \neg x_6) \wedge (x_3 \vee x_4)
 \end{aligned}$$

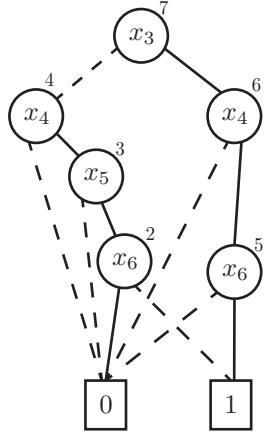
The call tree of $\text{BUILD}'(R', 3)$ is:



where

$$\begin{aligned}
 R'_{00} &\stackrel{\text{def.}}{=} R'[0/x_3, 0/x_4] = (x_5 \vee x_6) \wedge (1 \vee 0) \wedge (x_6 \vee 1) \wedge (0 \vee \neg x_6) \wedge (0 \vee 0) \equiv 0, \\
 R'_{10} &\stackrel{\text{def.}}{=} R'[1/x_3, 0/x_4] = (x_5 \vee x_6) \wedge (0 \vee 0) \wedge (x_6 \vee 0) \wedge (1 \vee \neg x_6) \wedge (1 \vee 0) \equiv 0, \\
 R'_{0100} &\stackrel{\text{def.}}{=} R'[0/x_3, 1/x_4, 0/x_5, 0/x_6] = (0 \vee 0) \wedge (1 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 1) \equiv 0, \\
 R'_{0101} &\stackrel{\text{def.}}{=} R'[0/x_3, 1/x_4, 0/x_5, 1/x_6] = (0 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (0 \vee 0) \wedge (0 \vee 1) \equiv 0, \\
 R'_{0110} &\stackrel{\text{def.}}{=} R'[0/x_3, 1/x_4, 1/x_5, 0/x_6] = (1 \vee 0) \wedge (1 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 1) \equiv 1, \\
 R'_{0111} &\stackrel{\text{def.}}{=} R'[0/x_3, 1/x_4, 1/x_5, 1/x_6] = (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (0 \vee 0) \wedge (0 \vee 1) \equiv 0, \\
 R'_{1100} &\stackrel{\text{def.}}{=} R'[1/x_3, 1/x_4, 0/x_5, 0/x_6] = (0 \vee 0) \wedge (0 \vee 1) \wedge (0 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \equiv 0, \\
 R'_{1101} &\stackrel{\text{def.}}{=} R'[1/x_3, 1/x_4, 0/x_5, 1/x_6] = (0 \vee 1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \equiv 1, \\
 R'_{1110} &\stackrel{\text{def.}}{=} R'[1/x_3, 1/x_4, 1/x_5, 0/x_6] = (1 \vee 0) \wedge (0 \vee 1) \wedge (0 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \equiv 0, \\
 R'_{1111} &\stackrel{\text{def.}}{=} R'[1/x_3, 1/x_4, 1/x_5, 1/x_6] = (1 \vee 1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \equiv 1.
 \end{aligned}$$

The corresponding BDD is seen below.



- e) How many possible movie programs (i.e. solutions) can Prof. Smart and his wife choose between? (Use the answer from d).

e) **Answer:** There are two paths in the BDD from d) leading to the node [1]. One path corresponds to the solution where $x_3 = \text{false}$, $x_4 = \text{true}$, $x_5 = \text{true}$, and $x_6 = \text{false}$. The other path corresponds to the solutions where $x_3 = \text{true}$, $x_4 = \text{true}$, $x_6 = \text{true}$, while x_5 can be either *true* or *false*. Thus there are 3 solutions.

Problem 3: True or False? (15%)

Write whether the following claims are true or false. If a claim is true, it suffices to write “True” in the answer. If a claim is false, however, you must argue why (e.g., by writing the true version of the statement).

a) MIN makes α -cuts and updates β .

a) **Answer:** True.

b) The degree heuristic is normally stronger than the minimum-remaining-values heuristic for selecting variables in BACKTRACKING-SEARCH.

b) **Answer:** False, it is the opposite way around.

c) If MAX plays the action found by the MINIMAX-DECISION algorithm, MAX gets at least the minimax value even if MIN plays sub-optimally.

c) **Answer:** True.

d) A satisfying solution to a DNF expression can be found in polynomial time.

d) **Answer:** True.

e) Global constraints are practical for modeling a problem, but do not increase the propagation power of constraint programming solvers compared to using their equivalent binary constraints.

e) False, on the contrary, constraint propagation for a global constraint is normally stronger than constraint propagation for its binary constraint representation.

Intelligent Systems Programming (ISP)

BSc and MSc Exam

IT University of Copenhagen

29 May 2018

This exam consists of 8 numbered pages with 3 problems. Each problem is marked with the weight in percent it is given in the evaluation. You have 4 hours to complete the exam. **Notice that the first version of problem 3 only is to be solved by Bachelor (BSc) students, while the second version only is to be solved by Master (MSc) students.**

You can answer the questions in English or Danish.

Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Constraint Programming (35%)

A letter puzzle consists of a board with $n \times n$ squares plus some clues appearing immediately next to the board at the beginning or end of a row or column (see figure (a) below). The clues are letters, and to solve the puzzle, each square must either be left empty or filled with a letter, such that:

- (i) In each row and each column, the $n - 1$ first letters in the alphabet appear exactly once in a square, while one square is left blank.
- (ii) A clue next to a column/row specifies the first letter to appear in that column/row when the column/row is read in the direction of the clue. That is, if e.g. an A appears to the right of the third row, then the first letter in the third row when read from right to left, has to be an A.

An example of a letter puzzle (a) and a possible solution (b) is shown below.

	B			
B				
	B			B

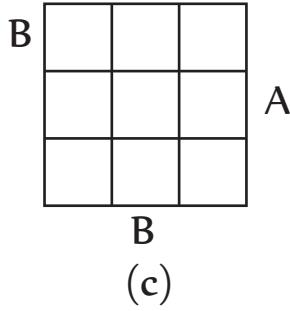
(a)

	B			C
B		C		A
C	B	A		
A		C	B	

(b)

In the following we will look at the 3×3 letter puzzle shown in figure (c) below and formulate it as a CSP with 9 variables, x_1, \dots, x_9 , where x_i denotes the letter or blank space that should be put in the corresponding square shown in figure (d) below. Each variable x_i for $i \in \{1, 2, \dots, 9\}$ has the domain $D_i = \{\circ, A, B\}$, where \circ denotes an empty space.

- a) Write binary constraints corresponding to the rules in (i) and (ii) above applied to the puzzle shown in figure (c) below. Note that AllDiff constraints should be rephrased as a number of binary constraints.



B

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

A

B

(d)

a) Answer:

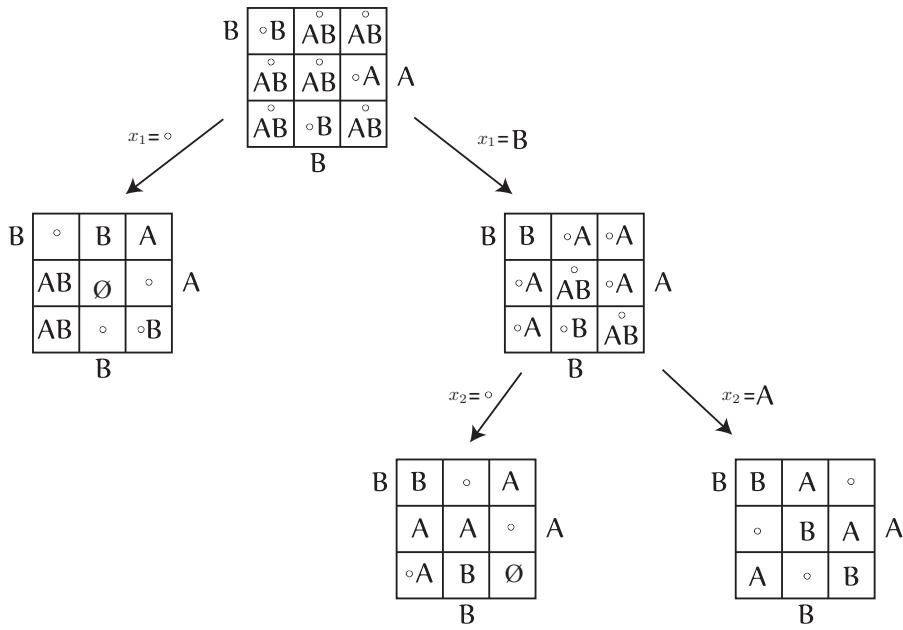
$$\begin{aligned}
 & x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3, \\
 & x_4 \neq x_5, x_4 \neq x_6, x_5 \neq x_6, \\
 & x_7 \neq x_8, x_7 \neq x_9, x_8 \neq x_9, \\
 & x_1 \neq x_4, x_1 \neq x_7, x_4 \neq x_7, \\
 & x_2 \neq x_5, x_2 \neq x_8, x_5 \neq x_8, \\
 & x_3 \neq x_6, x_3 \neq x_9, x_6 \neq x_9, \\
 & x_1 = B \vee x_1 = \circ \wedge x_2 = B, \\
 & x_6 = A \vee x_6 = \circ \wedge x_5 = A, \\
 & x_8 = B \vee x_8 = \circ \wedge x_5 = B.
 \end{aligned}$$

b) Make the CSP node consistent and write the domains for the variables, whose domains changed (if any). Make the CSP arc-consistent and write the domains for the variables, whose domains changed (if any).

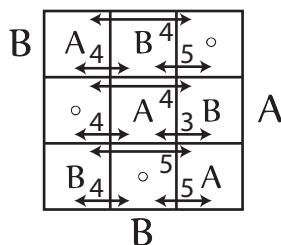
b) Answer: The CSP is already node-consistent since there are no unary constraints. Making the graph arc-consistent changes the following domains: $D_1 = \{\circ, B\}$, $D_6 = \{\circ, A\}$, $D_8 = \{\circ, B\}$.

c) Use the MAC algorithm to solve the puzzle. Draw the search tree with a node for each variable branched on. For each arc, indicate which variable is branched on and which value is assigned. In each node, write the remaining domains of each unassigned variable after arc-consistency is applied. Indicate failures and empty domains. You should use the MRV (minimum-remaining value) heuristic to choose the variable to branch on. If there is a tie, take the variables in numerical order of the subscript. Assign the domain values in the order $\circ \prec A \prec B$.

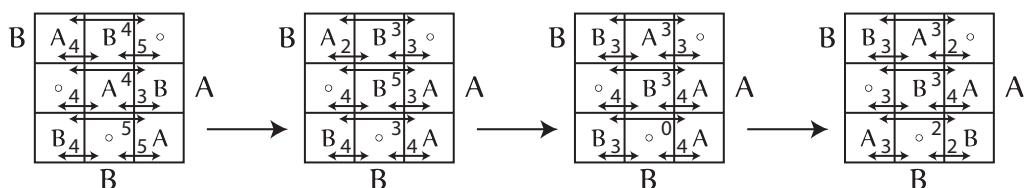
c) Answer:



d) Instead of MAC, use hill climbing to solve the problem. The initial state is shown below. The neighborhood of a state appears by swapping the content of two squares in the same row (thus, each state has a neighborhood of 9 other states). The cost of a state is the number of conflicts, where a conflict is a clue that is not satisfied or a column where A and B do not appear exactly once. You are allowed to take side-steps, and if so, you should not return immediately to the previous state. Indicate possible swaps for a state by arrows and write the cost of the resulting state above the arrow as shown for the initial state below.



d) Answer:



Problem 2: Heuristic Search and Logic (35%)

Prof. Smart is working on using search to prove logical equivalences in propositional logic. The problem has five rules. Each rule is defined by a precondition and an effect as shown in the table below:

Rule	Precondition	Effect
R1	$x \Rightarrow \text{False}$	$\neg x \vee \text{False}$
R2	$\neg x \vee \text{False}$	$\neg x$
R3	$\neg x \wedge y$	$y \wedge \neg x$
R4	$x \wedge \neg y$	$\neg y \wedge x$
R5	$\neg x \wedge x$	False

False represents the propositional constant *False*, while x and y represent propositional symbols. A rule is applicable if its precondition matches a sub-sentence of a sentence in propositional logic. The result of applying a rule is that the matched sub-sentence is substituted with the effect of the rule. Notice that we have required that x and y only can match a single propositional symbol. They cannot match general sentences. Four examples of rule application are shown below. Notice in the last example that a parenthesis around a sub-sentence disappear if it no longer is needed.

$$\begin{array}{ll}
 (A \Rightarrow \text{False}) \wedge \neg B & \xrightarrow{\text{Rule R1}} (\neg A \vee \text{False}) \wedge \neg B \\
 \neg B \wedge A \wedge \neg A & \xrightarrow{\text{Rule R3}} A \wedge \neg B \wedge \neg A \\
 \neg B \wedge A \wedge \neg A & \xrightarrow{\text{Rule R4}} \neg B \wedge \neg A \wedge A \\
 (\neg A \vee \text{False}) \wedge B & \xrightarrow{\text{Rule R2}} \neg A \wedge B
 \end{array}$$

Assume that you are given a sentence S_0 in propositional logic and that you search for new sentences by using actions that correspond to applying these five rules.

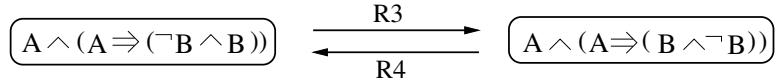
a) Would this approach be sound in the sense that all new sentences are logically equivalent to S_0 (why/why not)?

a) **Answer:** Yes, the approach is sound. The sub-sentences that the rules substitute are logically equivalent since the precondition and effect of each rule is a well-known logical equivalences (e.g., we can prove them with a truth table).

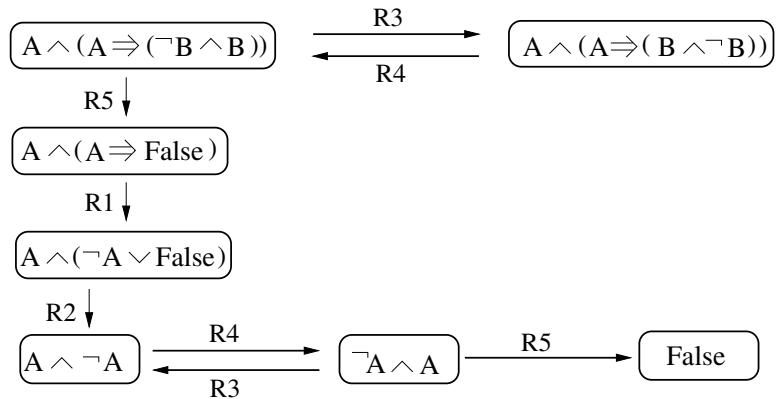
b) Would this approach be complete in the sense that any sentence that is logically equivalent to S_0 can be found in this way (why/why not)?

b) Answer: No, say S_0 is $\neg x \vee x$. No rule can be applied to this sentence, so for instance the sentence *True* that is logically equivalent to S_0 will never be found as required by completeness.

c) Draw the complete state space that can be reached by applying the rules from the initial sentence $A \wedge (A \Rightarrow (\neg B \wedge B))$. Annotate each edge with the rule it corresponds to. A part of this state space is shown below.



c) Answer:



Prof. Smart wants to use his search approach to determine if a sentence is logically equivalent to *False*. He claims that if the cost of all rule applications is one, a valid heuristic for this search problem is to count the number of propositional symbols and constants and subtract one. As an example, the heuristic value of $A \wedge (A \Rightarrow \text{False})$ is two since there are two propositional symbols in the sentence (A and A) and one constant (False).

d) Is the heuristic admissible (why/why not)?

d) Answer: Yes, a rule application can at most remove one propositional symbol or constant from a sentence. So the number of rule applications needed to reach the goal is larger than or equal to the heuristic value of the state.

e) Is the heuristic consistent (why/why not)?

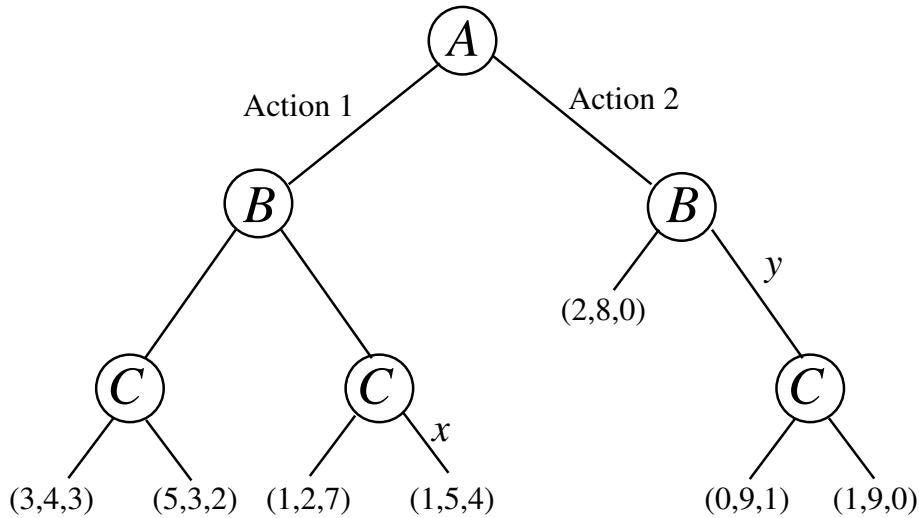
e) **Answer:** Yes, we need $h(s') + c(s, a, s') \geq h(s)$, where s' is the state resulting from applying a rule a in s and $c(s, a, s')$ is the cost of applying a . In our case this is equivalent to $h(s) - h(s') \leq 1$ since all rule applications have a cost of one. This inequality must hold, since a rule application at most can remove a single propositional symbol or constant.

f) Which sequence of rule applications corresponds to the solution returned by A* Tree Search given an initial state of the sentence $A \wedge (A \Rightarrow (\neg B \wedge B))$ and using Prof. Smarts heuristic?

f) **Answer:** $R5, R1, R2, R4, R5$, since this is the shortest path to False as seen in the state space graph.

Problem 3: FOR BSC STUDENTS ONLY (30%)

The figure below shows a game tree of a turn-taking game with three players A , B , and C .



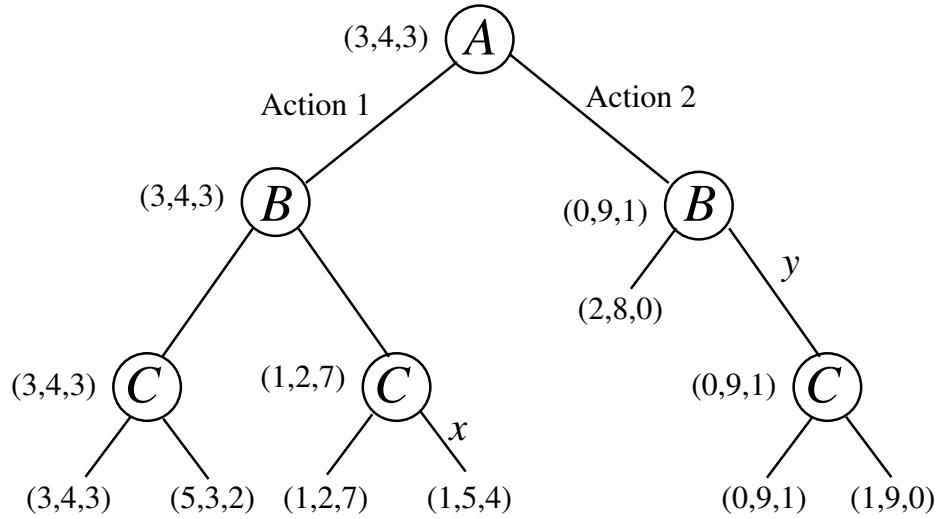
Each leaf-node is a terminal state. Its utility is given as a triple (a, b, c) , where a , b , and c are the utility of player A , B , and C , respectively.

- a) The game is zero-sum. How do the utilities in the game tree show this?

- a) **Answer:** The sum of the utilities in each triple is a constant 10.

- b) Write the multi-player minimax value next to each internal node of the tree.

b) Answer:



c) What is the minimax decision for player A in the root node and why?

c) Answer: It is action 1, since this action leads to the maximum utilization 3 for player A.

The MINIMAX function below can be used to calculate the minimax value of a state of this 3-player game.

```

function MINIMAX(state) returns the Minimax value of state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow (-\infty, -\infty, -\infty)$ 
  for each a in ACTIONS(state) do
    w  $\leftarrow$  MINIMAX(RESULT(state,a))
    if VALUE(w,PLAYER(state))  $>$  VALUE(v,PLAYER(state)) then
      v  $\leftarrow$  w
  return v

```

TERMINAL-TEST(*s*) returns true if *s* is a terminal state of the game. UTILITY(*s*) returns the utility triple of a terminal state *s*. ACTIONS(*s*) returns the applicable actions of the player that is to take a turn in state *s*. RESULT(*s,a*) returns the state resulting from executing action *a* in state *s*. VALUE(*u,p*) returns the utility of player *p* in utility triple *u*. Finally, PLAYER(*s*) returns the player that is to take a turn in state *s*.

d) Write pseudo-code of a function MINIMAX-DECISION(*s*) that returns the minimax decision of the player that is to take a turn in state *s*.

d) **Answer:**

```
function MINIMAX-DECISION(state) returns an action
    return argmaxa ∈ Actions(state) VALUE(MINIMAX(RESULT(state,a)),PLAYER(state))
```

e) Prof. Smart claims that if the MINIMAX function explores actions from left to right in the tree, it can make cuts similar to α and β cuts at edge x and y . Is this correct (why/why not)?

e) **Answer:** Prof. smart is right. For edge x , C just learned that it can get 7. This means that A and B at most can get 3 at this node, and since B already has 4 in the node above, there is no need for further exploration. For edge y , B just learned that it can get 8. But this means that A and C at most can get 2 at this node, and since A already has 3 in the node above, there is no need for further exploration.

Problem 3: FOR MSC STUDENTS ONLY (30%)

Linear Programming

Assume we have an LP problem, where the dual LP problem is shown below.

$$\begin{array}{ll} \text{Minimize} & -y_1 \\ \text{Subject to} & y_1 - y_2 \geq 0 \\ & -y_2 \geq -4 \\ & y_1, y_2 \geq 0 \end{array}$$

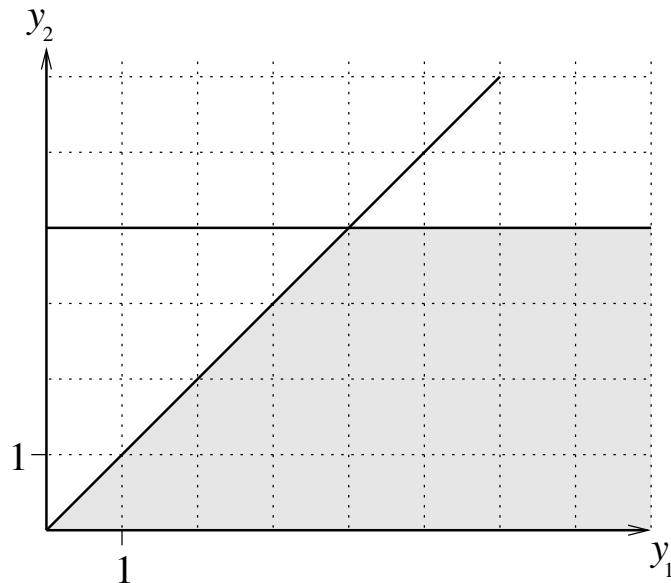
- a) Write the LP problem shown above on standard form.

- a) Answer:

$$\begin{array}{ll} \text{Maximize} & y_1 \\ \text{Subject to} & -y_1 + y_2 \leq 0 \\ & y_2 \leq 4 \\ & y_1, y_2 \geq 0 \end{array}$$

- b) Draw the geometric interpretation of the LP problem shown above.

- b) Answer:



- c) Determine based on your answer to b) alone, whether the primal LP problem is feasible, infeasible, or unbounded. Explain your answer.

c) **Answer:** Since the dual LP problem is unbounded, the primal LP must be infeasible.

d) Write the primal LP problem.

d) **Answer:**

$$\begin{array}{lll} \text{Maximize} & -4x_2 \\ \text{Subject to} & x_1 & \leq -1 \\ & -x_1 - x_2 & \leq 0 \\ & x_1, x_2 & \geq 0 \end{array}$$

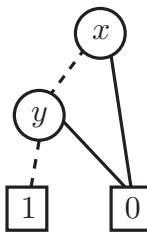
Binary Decision Diagrams

Consider the following Boolean expression

$$(x \Rightarrow y) \wedge (\neg x \wedge \neg y)$$

e) Draw the BDD for the expression using variable order $x \prec y$.

e) **Answer:**



f) Is the expression valid, unsatisfiable or neither? Give a reason for your answer.

f) **Answer:** Neither. If the expression is valid, the BDD is 1. If it is unsatisfiable, the BDD would be 0.

Intelligent Systems Programming (ISP)

BSc and MSc Exam

IT University of Copenhagen

24 May 2019

This exam consists of 9 numbered pages with 3 problems. Each problem is marked with the weight in percent it is given in the evaluation. **Notice that the first version of problem 3 only is to be solved by Bachelor (BSc) students, while the second version only is to be solved by Master (MSc) students.**

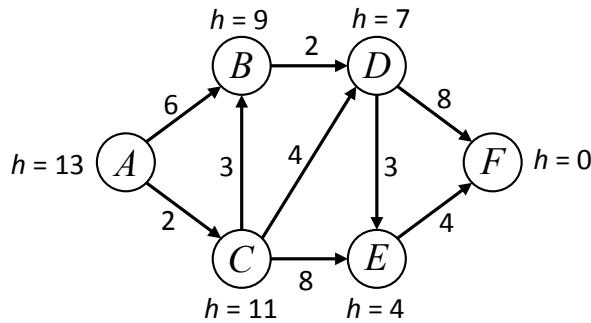
Use notations and methods that have been discussed in the course.

Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Heuristic Search (25%)

The state-space graph shown below represents a heuristic search problem. The vertices A, B, C, D, E , and F denote states and the edges denote actions. The step cost of each action is written next to its edge. For example, the step cost of taking the action from C to B is 3. The initial state is A and there is only one goal state F . The value of a heuristic function (h) for this search problem is written next to each state.



- a) Argue that h is a valid and admissible heuristic function.

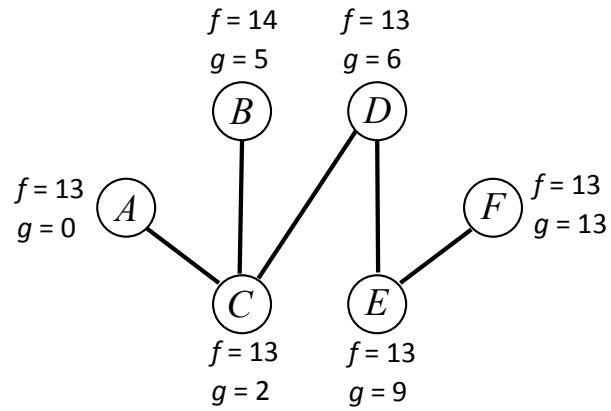
a) Answer: h is valid since h is zero for the goal state F and non-negative for all other states. Inspecting the graph, we see that h for each state is equal to the minimum path-cost to reach the goal from the state. Thus, h is never larger than this minimum path-cost and for that reason h is admissible by definition.

- b) Argue that h also is a consistent heuristic function (hint: there is a simple reason for this).

b) Answer: Since $h(s)$ for any state s is the cost of an actual path in the graph from s to F with minimum cost, we have that $h(s) \leq \text{cost}(s, s') + h(s')$. The reason is that only when the edge (s, s') is on an optimal path to F , do we have $h(s) = \text{cost}(s, s') + h(s')$. In all other cases, (s, s') is on a sub-optimal path to F such that $h(s) < \text{cost}(s, s') + h(s')$. Thus, h fulfills the definition of consistency.

c) Draw the search tree grown by A* graph search to solve the search problem. Write the final g and f -values associated with each node in the tree when the algorithm terminates.

c) Answer:



Problem 2: Propositional Logic and BDDs (40%)

Prof. Smart is planning a weekend trip with his wife. There are four sights that they consider seeing, namely X, Y, Z and W. The rest of the time, they will just enjoy the atmosphere of the city. Smart's wife makes the following statements about their trip.

1. "I'm fine whether we see X or not, but if we choose to see both X and Y, we don't have time for neither Z nor W."
2. "However, if we leave out either X or Y (or both), we do have time for W and then we should definitely go see it."

a) Write logical expressions for the statements (1) and (2) above.

- Use the variables x, y, z and w to write the two statements. The variable x should denote whether Prof. Smart and his wife will see X, i.e. x is true iff they go see X. Similarly for the other variables.
- Translate the expressions to CNF.

a) Answer: The two statements can be expressed in propositional logic as follows.

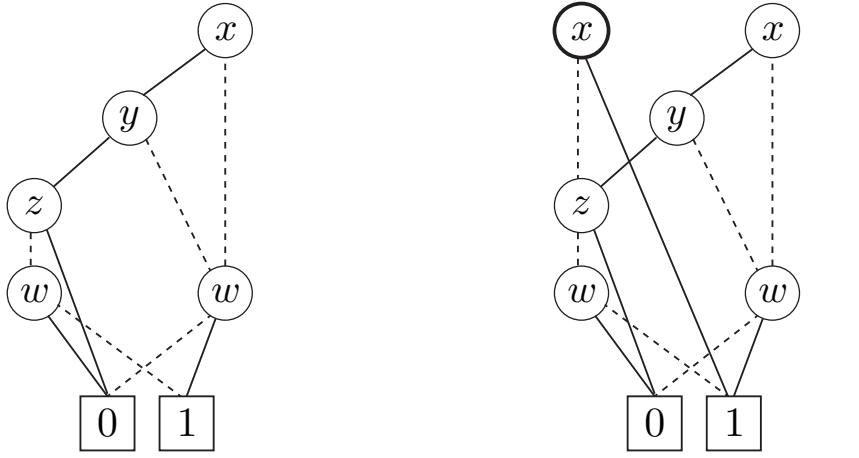
$$\begin{aligned} 1 : \quad & (x \vee \neg x) \wedge ((x \wedge y) \Rightarrow (\neg z \wedge \neg w)) \\ 2 : \quad & (\neg y \vee \neg x) \Rightarrow w \end{aligned}$$

The expressions translate to CNF as follows.

$$\begin{aligned} (x \vee \neg x) \wedge ((x \wedge y) \Rightarrow (\neg z \wedge \neg w)) &\equiv (x \wedge y) \Rightarrow (\neg z \wedge \neg w) \\ &\equiv \neg(x \wedge y) \vee (\neg z \wedge \neg w) \\ &\equiv (\neg x \vee \neg y) \vee (\neg z \wedge \neg w) \\ &\equiv (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg w) \end{aligned}$$

$$(\neg y \vee \neg x) \Rightarrow w \equiv \neg(\neg y \vee \neg x) \vee w \equiv (y \wedge x) \vee w \equiv (y \vee w) \wedge (x \vee w)$$

Prof. Smart makes a BDD corresponding to the conjunction of the two requirements of Mrs. Smart using the the variable ordering $x \prec y \prec z \prec w$, see Figure 1a. After thinking for a while, he then makes another BDD representing his own requirement - an implication. This results in the multi-rooted BDD in Figure 1b. That is, Prof. Smart's requirement is represented by the BDD with the root that has a bolder outline in Figure 1b.



(a) Conjunction of (1) and (2).

(b) Multi-rooted BDD.

Figure 1: BDDs

b) What does Prof. Smart's requirement correspond to?

- Write a logical expression that is equivalent to his requirement. You can use the if-then-else operator if you want to, but it is not required.
- Write the expression on a form, where the main connective is an implication.
- Further, write a sentence in English natural language that this requirement could correspond to.

b) Answer: Prof. Smart's requirement corresponds to the following expression.

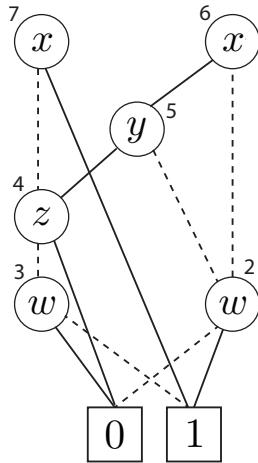
$$\begin{aligned}
 x \rightarrow 1, (z \rightarrow 0, \neg w) &\equiv x \rightarrow 1, ((z \wedge 0) \vee (\neg z \wedge \neg w)) \\
 &\equiv x \rightarrow 1, (\neg z \wedge \neg w) \\
 &\equiv (x \wedge 1) \vee (\neg x \wedge \neg z \wedge \neg w) \\
 &\equiv x \vee (\neg x \wedge \neg z \wedge \neg w) \\
 &\equiv (x \vee \neg x) \wedge (x \vee (\neg z \wedge \neg w)) \\
 &\equiv x \vee (\neg z \wedge \neg w) \\
 &\equiv x \vee \neg(z \vee w) \\
 &\equiv (z \vee w) \Rightarrow x
 \end{aligned}$$

The requirement corresponds to the following statement: "If we go see Z or W, then we should also go see X".

c) Write the unique table T corresponding to the multi-rooted BDD in Figure 1b. You should assume that the variables are represented by their number in the ordering. Thus the first variable (x) has variable index 1 while the second variable (y) has variable index 2 and so on.

- Make sure that the nodes are numbered as they would when made through a number of calls to MK.
- Copy the BDD from Figure 1b next to T on the same page and write the IDs next to the nodes in the figure.

c) **Answer:** The figure below shows the BDD with IDs next to the nodes. The unique table T for this multi-rooted BDD is given below.



T

ID	var	low	high
0	5	-	-
1	5	-	-
2	4	0	1
3	4	1	0
4	3	3	0
5	2	2	4
6	1	2	5
7	1	4	1

Now Prof. Smart wants to see what happens when both his own and his wife's requirements are taken into account. That is, he wants to make the conjunction of the two BDDs with roots in the multi-rooted BDD.

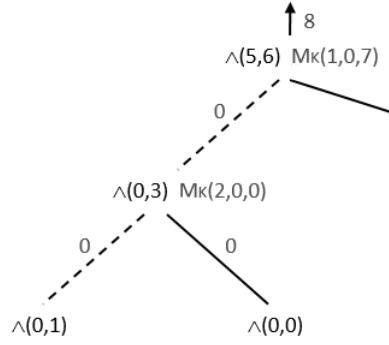
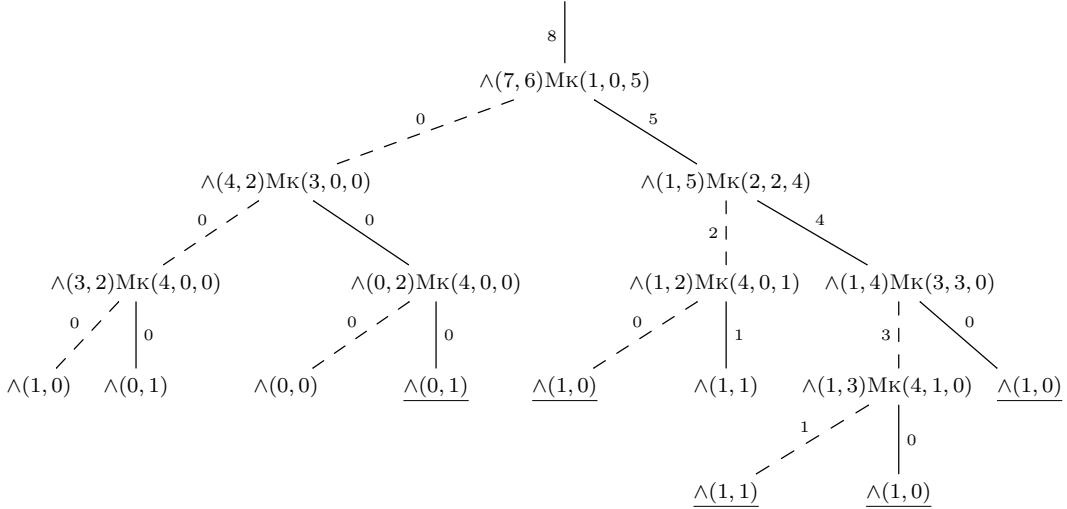


Figure 2: Call tree of APPLY of unrelated BDDs.

d) Use APPLY to make the conjunction (\wedge) of the BDDs corresponding to the two nodes testing on x in Figure 1b.

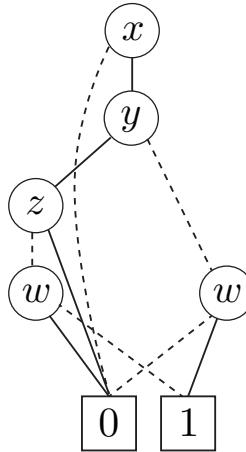
- Write the call tree. When doing this, you have to use the following guidelines. For each call to APP, you have to write the two arguments (two node identifiers), and you have to write which call to MK is made. On the arcs, write what the method call returns. That is, you follow the notation in the slides giving an example of APPLY, part of which can be seen in Figure 2.
- You also need to have a table showing G , and you have to underline the nodes of the call tree in which there is a cache hit. The call tree, G , and T has to be on the same page. If necessary, turn your paper to “landscape” mode.
- Draw the resulting BDD.

d) **Answer:** The call tree of $\text{APPLY}(\wedge, 7, 6)$ is as follows.



G and the resulting BDD is given below.

G	
(1,0)	0
(0,1)	0
(3,2)	0
(0,0)	0
(0,2)	0
(4,2)	0
(1,1)	1
(1,2)	2
(1,3)	3
(1,4)	4
(1,5)	5
(7,6)	8



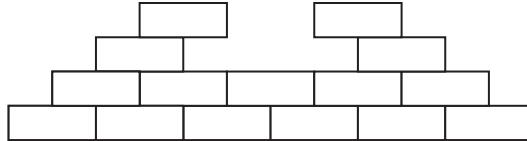
Looking at his result, Prof. Smart exclaims: "Oh, then it is not possible for us to see both Z and W!"

e) Is Prof. Smart right? Give an explanation why/why not.

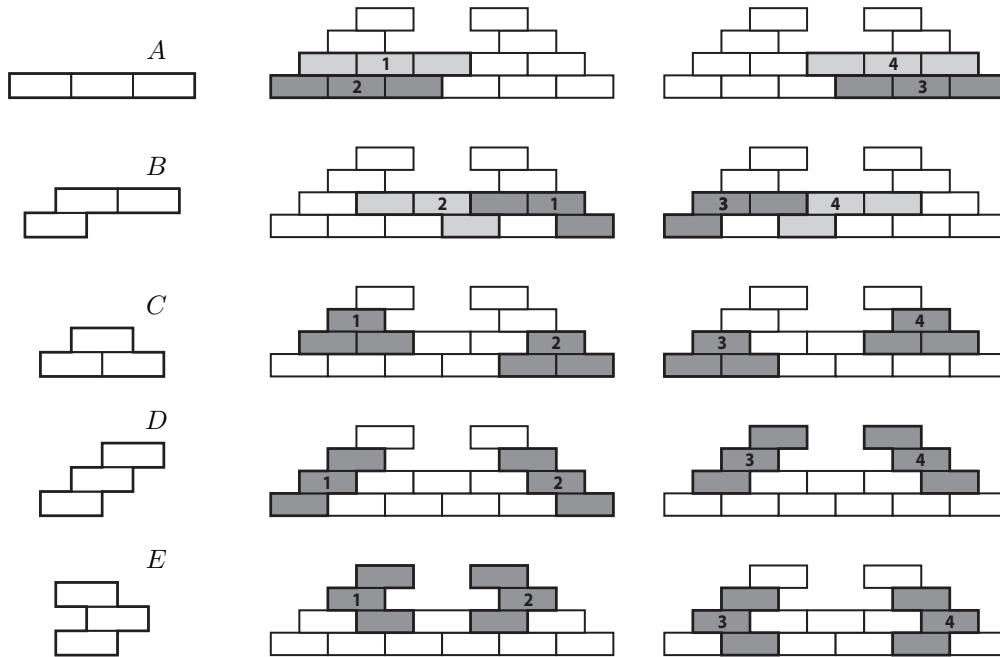
e) **Answer:** He is not right. If Prof. Smart and his wife choose to see X, Z and W, but not Y, the expression given by the BDD is still satisfied.

Problem 3: FOR BSC STUDENTS ONLY (35%)

In the *Brick by Brick* puzzle, the goal is to use each of five different blocks to build the wall shown below. Each block must be used exactly one time.

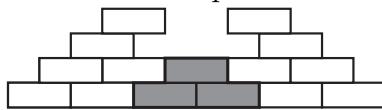


In a CSP formulation of the problem, we use five variables A , B , C , D , and E to represent the position of each block. Each variable has domain $\{1, 2, 3, 4\}$ corresponding to four different positions of the block in the wall. It is allowed to flip the blocks horizontally and vertically. The only constraint is that the blocks cannot overlap each other. The blocks and the possible positions that we model are defined in the figure shown below.



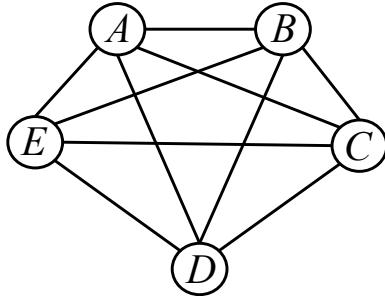
- a) Is the CSP formulation complete in the sense that all configurations of the blocks can be represented (why/why not)?

- a) **Answer:** No, it is not complete. For instance it is not possible to represent configurations, where block C is in the position shown below.



b) Draw the constraint graph of the CSP.

b) **Answer:** Since there exists positions for each pair of blocks, where they overlap each other, the constraint graph is complete as shown below.



c) Assume that $A \in \{1, 2, 3, 4\}$ and $B \in \{1, 2\}$. What are the $A \rightarrow B$ arc-consistent domains of A and B ?

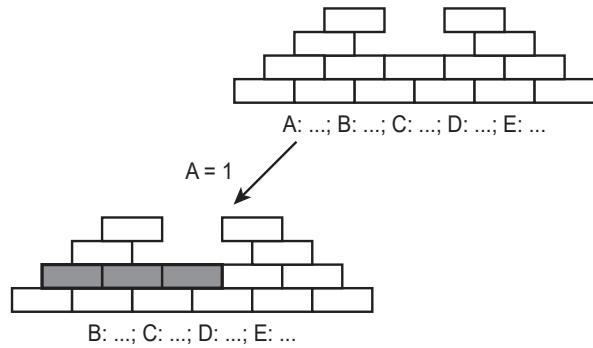
c) **Answer:** For $A \rightarrow B$ arc-consistency, only the domain of A is reduced. In this case, it is reduced to $\{1, 2\}$ as position 3 and 4 of overlap with both position 1 and 2 of B .

d) Recall that the initial domain of all variables in the CSP is $\{1, 2, 3, 4\}$. Why is an arc consistency algorithm like AC-3 unable to remove any domain values from this CSP?

d) **Answer:** Since for each position of a block there are positions for the remaining blocks that do not overlap it, no domain values can be removed in arc-consistency revisions.

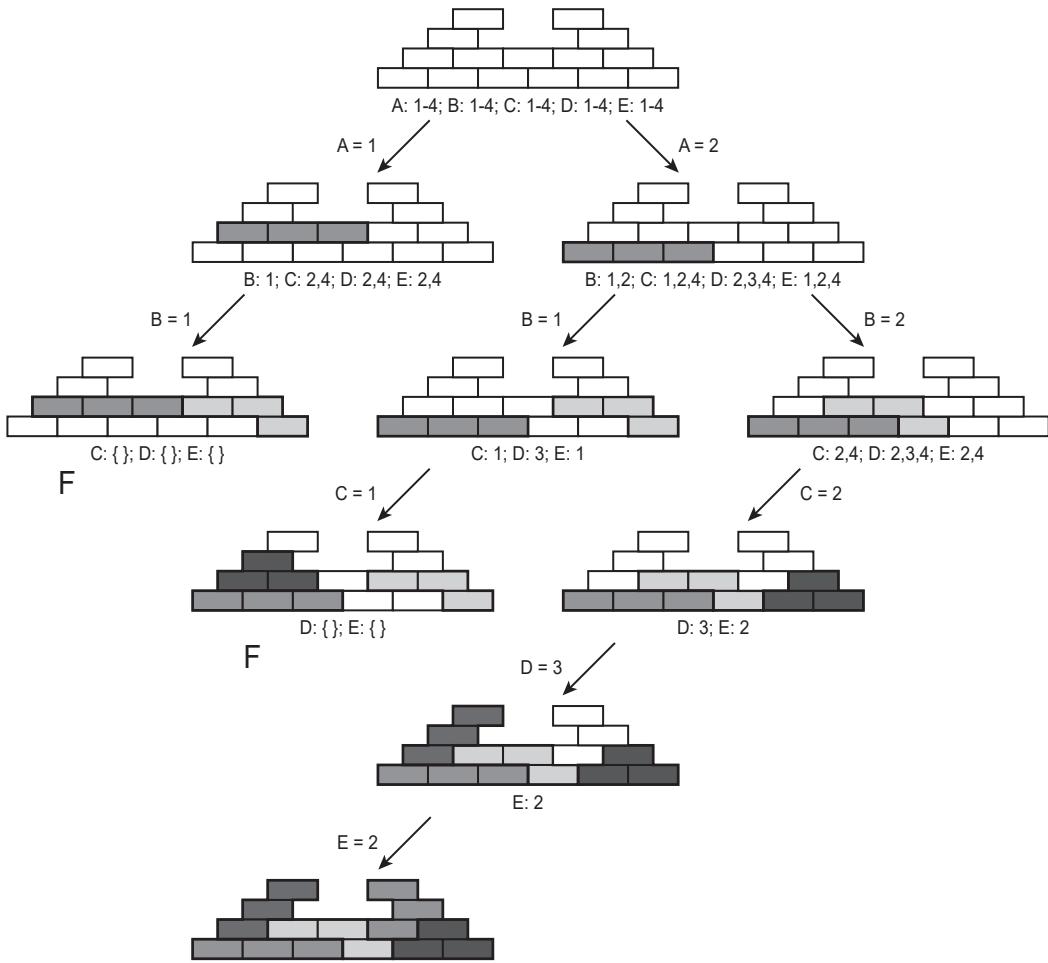
We will use FORWARD-CHECKING-SEARCH to try to solve the puzzle. We use the MRV heuristic to select an unassigned variable, where ties are broken by choosing the variable that comes first in the alphabet. Domain values are tried in ascending order. Each call to RECURSIVE-FORWARD-CHECKING is drawn as a puzzle state given by assigned variables, with the domains of unassigned variables

written under it. The initial call and a child call corresponding to the assignment $A = 1$ are illustrated below (notice that the domains have not been written in the example, but you will have to in your answer).



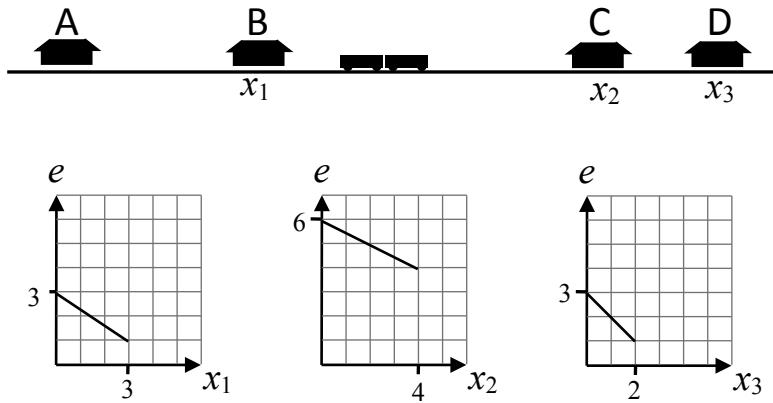
- e) Draw the complete call tree of FORWARD-CHECKING-SEARCH using the approach described above. Indicate a *failure* result with an F in the tree. We recommend that you use squared paper in "landscape" orientation.

e) Answer:



Problem 3: FOR MSC STUDENTS ONLY (35%)

A metro train drives between four stations A , B , C , and D . The management wants to minimize the total amount of energy that the train uses to drive between the stations. It is currently 12 units, but can be reduced by delaying the arrival time at station B , C , and D . Let x_1 , x_2 , and x_3 denote the arrival delay in minutes at each of these three stations. The graphs below shows the energy e needed to drive from station A to B , B to C , and C to D as a function of x_1 , x_2 , and x_3 , respectively. Thus, if the management decides the delays $x_1 = 0$, $x_2 = 2$, and $x_3 = 1$, the total needed energy is reduced from 12 to $3 + 5 + 2 = 10$ units.



The delays have to fulfil the following rules:

- The sum of all delays is at most 4 minutes.
- The maximum delay at station B , C , and D is 3, 4, and 2 minutes, respectively.

a) Write a linear program (LP) in standard form that can be used to solve the management's optimization problem. You can assume that the objective is to maximize the amount of saved energy. The only decision variables should be x_1 , x_2 , and x_3 .

a) Answer:

$$\begin{aligned}
 & \text{Maximize} && \frac{2}{3}x_1 + \frac{1}{2}x_2 + x_3 \\
 & \text{Subject to} && x_1 + x_2 + x_3 \leq 4 \\
 & && x_1 \leq 3 \\
 & && x_2 \leq 4 \\
 & && x_3 \leq 2 \\
 & && x_1, x_2, x_3 \geq 0
 \end{aligned}$$

- b) Write the slack form of the LP. Is the slack form a feasible initial dictionary for the SIMPLEX algorithm (why/why not)?

b) Answer:

$$\begin{aligned}
 & \text{Maximize} && z \\
 & \text{Subject to} && x_4 = 4 - x_1 - x_2 - x_3 \\
 & && x_5 = 3 - x_1 \\
 & && x_6 = 4 - x_2 \\
 & && x_7 = 2 - x_3 \\
 & && z = \frac{2}{3}x_1 + \frac{1}{2}x_2 + x_3 \\
 & && x_1, x_2, \dots, x_7 \geq 0
 \end{aligned}$$

The slack form is a feasible initial dictionary for the Simplex algorithm since all variables are assigned to non-negative values.

- c) Solve the problem using the SIMPLEX algorithm. Write all dictionaries produced by the algorithm.

c) **Answer:** Initial dictionary (slack form) : entering x_3 , leaving x_7 .

Second dictionary : entering x_1 , leaving x_4 .

$$\begin{aligned} \text{Maximize } & z \\ \text{Subject to } & x_4 = 2 - x_1 - x_2 + x_7 \\ & x_5 = 3 - x_1 \\ & x_6 = 4 - x_2 \\ & x_3 = 2 - x_7 \\ & z = 2 + \frac{2}{3}x_1 + \frac{1}{2}x_2 - x_7 \end{aligned}$$

Third dictionary : optimal.

$$\begin{aligned} \text{Maximize } & z \\ \text{Subject to } & x_1 = 2 - x_2 - x_4 + x_7 \\ & x_5 = 1 + x_2 + x_4 - x_7 \\ & x_6 = 4 - x_2 \\ & x_3 = 2 - x_7 \\ & z = 3\frac{1}{3} - \frac{1}{6}x_2 - \frac{2}{3}x_4 - \frac{1}{3}x_7 \end{aligned}$$

d) Based on your answer to c), what is the minimum energy needed by the train and what are the corresponding delays at station B , C , and D ?

d) **Answer:** From the optimal dictionary, we get that the maximum energy reduction is $3\frac{1}{3}$, which means that the minimum energy needed by the train is $12 - 3\frac{1}{3} = 8\frac{2}{3}$ units. The corresponding delays in station B (x_1), C (x_2), and D (x_3) are 2, 0, and 2 minutes, respectively.

e) Based on your answer to c), what is the minimum energy needed by the train, if the total allowed delay is increased from 4 to $4\frac{1}{2}$ minutes, but the maximum delay in station D is reduced from 2 to $1\frac{1}{2}$ minutes?

e) **Answer:** We answer this question using shadow prices. From the z expression in the optimal dictionary, we get the following optimal dual solution $y_1 = \frac{2}{3}$, $y_2 = 0$, $y_3 = 0$, and $y_4 = \frac{1}{3}$. Since the new rules correspond to changing the right side of constraint 1 and 4 with $+\frac{1}{2}$ and $-\frac{1}{2}$, the change in maximum energy reduction is $\frac{2}{3} * \frac{1}{2} - \frac{1}{3} * \frac{1}{2} = \frac{1}{6}$. Hence, the minimum energy needed is reduced by $\frac{1}{6}$ from $8\frac{2}{3}$ to $8\frac{1}{2}$.

Introduction to Artificial Intelligence (IAI)

BSc and MSc Exam

IT University of Copenhagen

29 May 2020

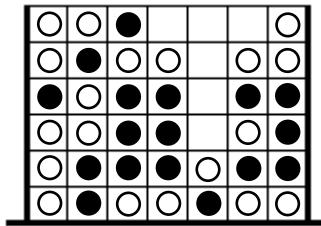
This exam consists of 8 numbered pages with 3 problems. Each problem is marked with the weight in percent it is given in the evaluation. **Notice that Bachelor (BSc) and Master (MSc) students must solve the same problems.**

- Ensure that you have a quiet, comfortable, and undisturbed work environment.
- Use notations and methods that have been discussed in the course.
- Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Adversarial Search (30%)

Connect 4 is a two-player board game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The discs fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. An example of a game state is shown below.

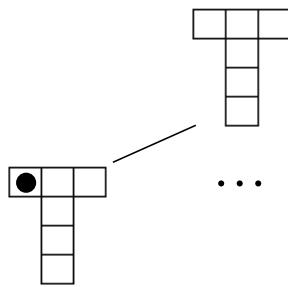


- a) Consider building a complete game tree from the initial state of Connect 4, where the grid is empty. How many nodes are there at depth six in this tree?

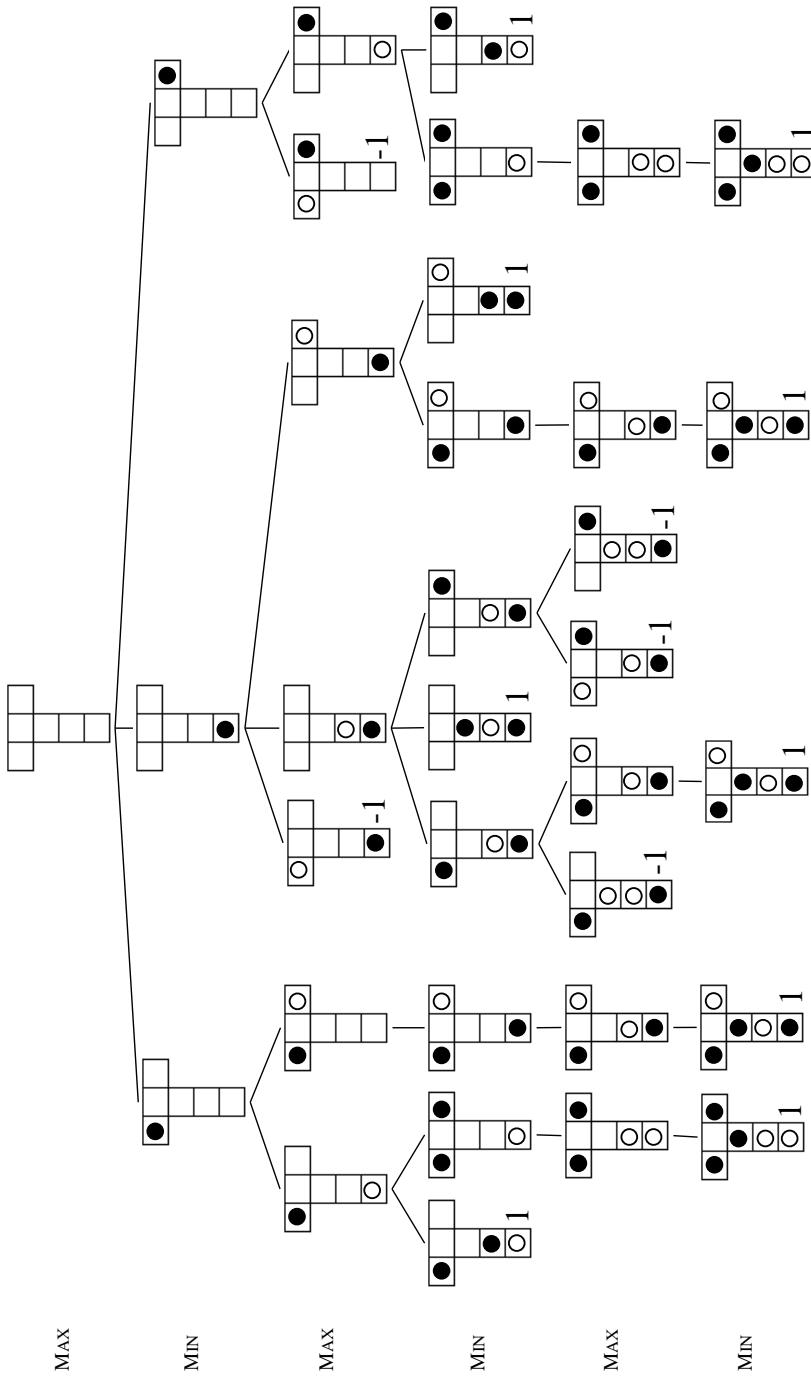
a) **Answer:** The tree has $7^6 = 117649$ nodes at depth six. The reason is that the game can not terminate before depth eight and each player can choose among all seven columns at least until depth six.

Assume that MAX plays the black discs and that it is MAX's turn in the game state shown above. MAX's utility of winning, losing, and drawing is 1, -1, and 0, respectively.

- b) Draw the complete game tree for MAX in this state. Assume in the tree that each player chooses the columns from left to right. Write the utility next to each terminal node. Turn the paper in "landscape mode" and simplify the representation of each state by only drawing the free slots as shown below.

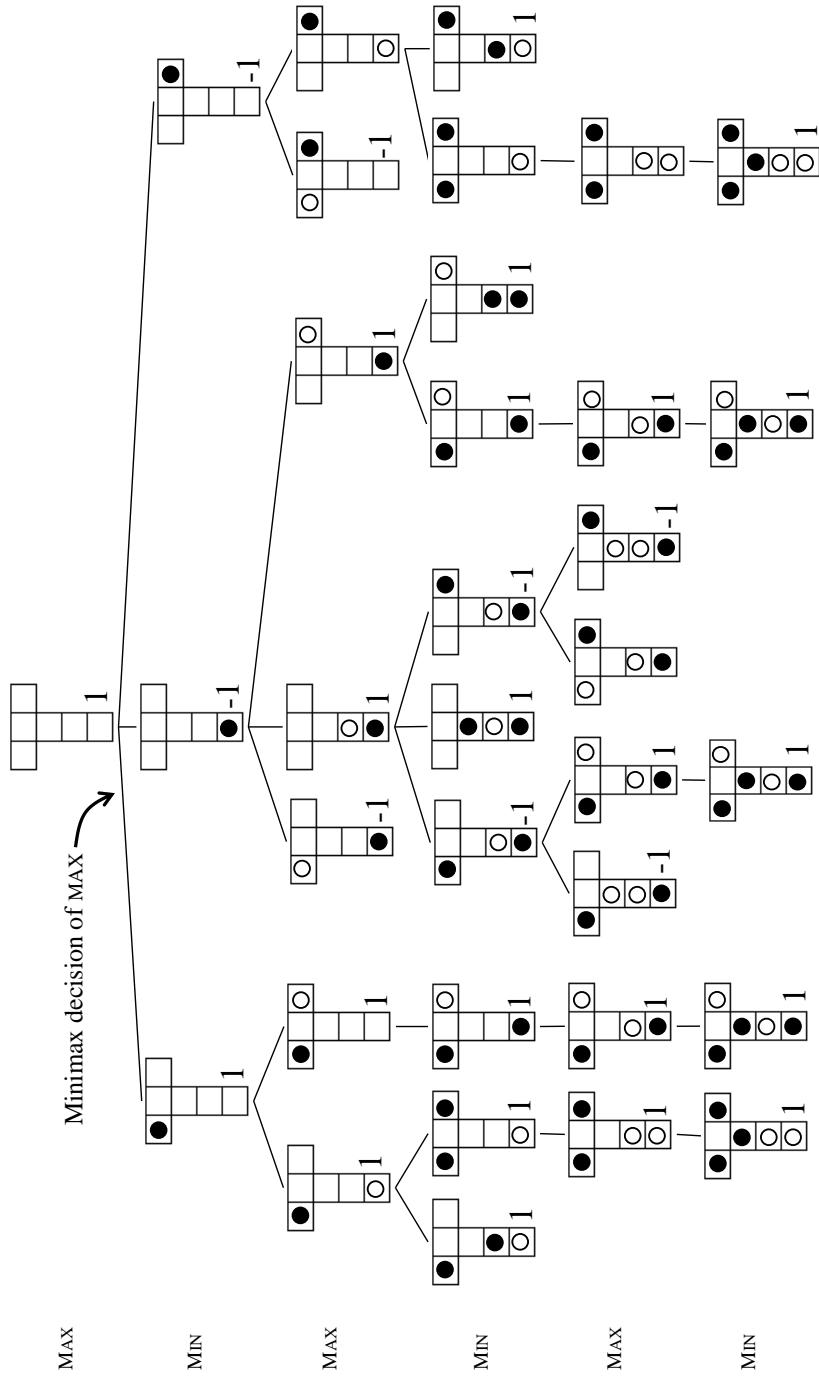


b) Answer:



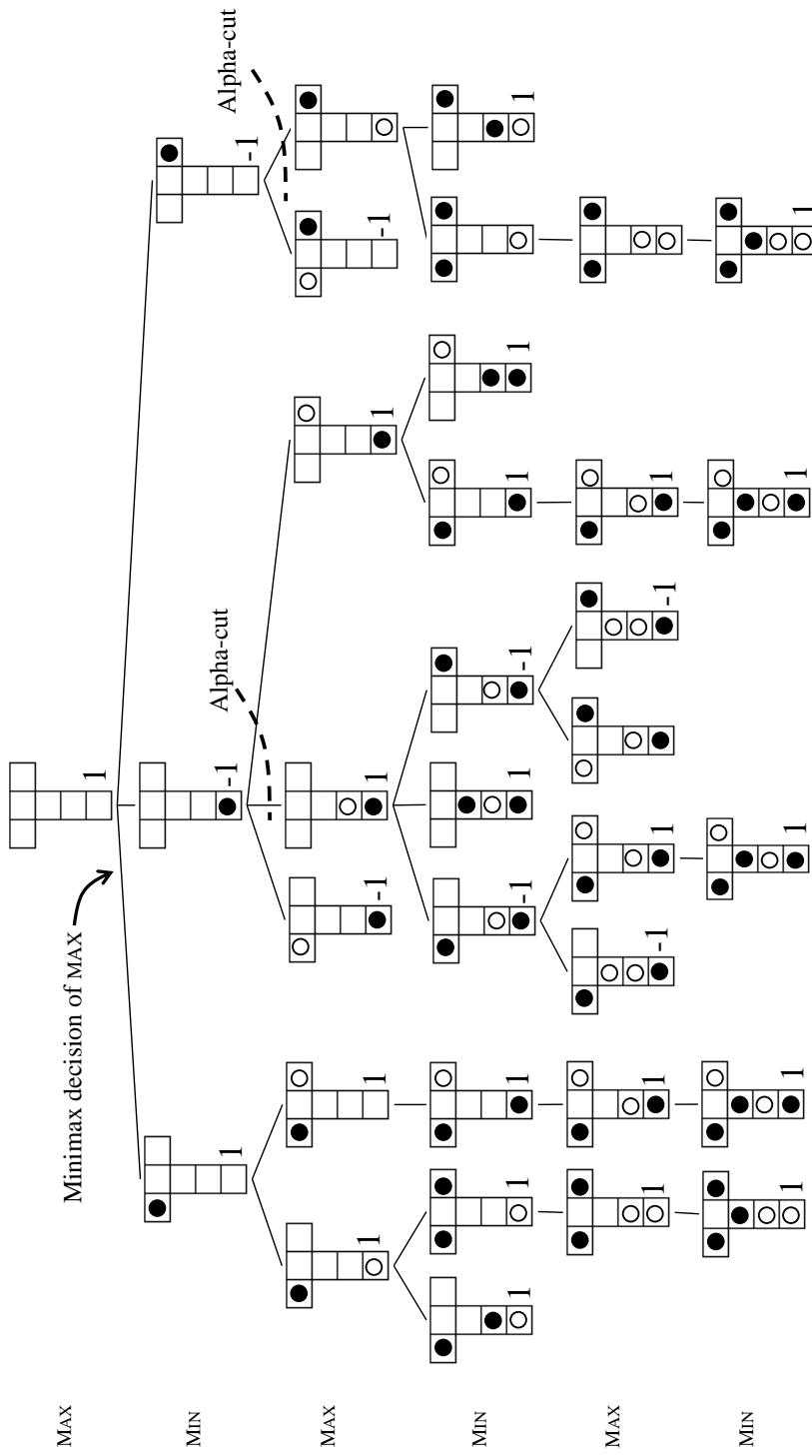
c) Write the minimax value next to each node in the game tree that you made in your answer to sub-problem b) and indicate the minimax decision of MAX (you do not have to re-draw the tree).

c) Answer:



d) Indicate the alpha and beta cuts that the MINIMAX algorithm with α - β pruning would have made in the tree that you made in your answer to subproblem b) (again, you do not have to re-draw the tree). Is it possible to prune more nodes if MAX in the root node chooses columns in another order than left to right (why/why not)?

d) **Answer:** The tree has two alpha-cuts as indicated below. Any re-order of MAX's actions in the root node would lead to the best action being found later and thus fewer nodes cut in the tree.



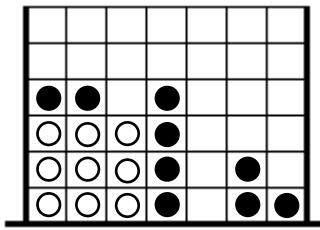
Since the game trees of Connect 4 are very large, the MINIMAX algorithm must apply an evaluation function and cutoff test to be computationally feasible. Consider two evaluation functions f_1 and f_2 . They are both defined in terms of lines of connected discs with same color in a game state. Each line must contain at least two discs in the horizontal, vertical or diagonal direction and each line must have maximum length. As an example, the game state shown above has three vertical black lines of length three and one vertical black line of length two. The evaluation functions differ by the value they assign to a line of length n . For f_1 , the value is $2n$, while it for f_2 is $n!$. Both functions evaluate a game state as the total value of black lines minus the total value of white lines.

e) Consider two agents playing Connect 4 against each other using the same MINIMAX algorithm but one applying evaluation function f_1 and the other f_2 . Which of the agents do you think would win and why?

e) **Answer:** According to f_1 , it is better to have two lines of length two than one line of length three. Since this is seldom the case in the game, it is reasonable to assume that the agent using f_2 has a higher chance of winning.

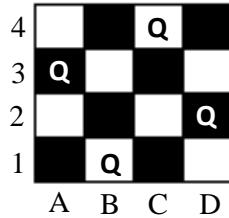
f) Positive values of f_1 and f_2 indicate that the game state is strongest for black. For that reason we expect terminal states won by black to have positive f_1 and f_2 values. Is that always the case (why/why not)?

f) **Answer:** No, it is not always the case. Consider the game state below, where black has won. White has eight and four lines of length three and two, respectively. Black has one and three lines of length four and two, respectively. This means that $f_1 = 20 - 64 = -44$ and $f_2 = 30 - 56 = -26$, which both are negative numbers.



Problem 2: Constraint Satisfaction (40%)

Recall that the 4-Queens Problem consists in placing four queens on a 4×4 chessboard so that no two queens can capture each other. The figure below shows a solution.



We will use four variables $\{A, B, C, D\}$ with domain $\{1, 2, 3, 4\}$ to represent the positions of the queens. Each variable is associated with a column of the chessboard. The value of a variable is the row number of a queen placed in the column. Hence, the variable assignment representing the solution shown above is $A = 3$, $B = 1$, $C = 4$, and $D = 2$.

a) Prof. Smart claims that this is a bad representation of the problem since it is incomplete. It is possible to place more than one queen in a column, but the representation does not allow that. Do you agree (why/why not)?

a) **Answer:** Prof. Smart is right about that the representation is incomplete. However, since queens in the same column can capture each other, it just discards infeasible solutions. In other words, the representation takes into account the constraints of the problem which is a clever way to reduce the search space. It is complete in the space of feasible solutions.

We will use a simple version of backtracking search called **BASIC-BACKTRACK-SEARCH** to find solutions to the 4-Queens Problem. The pseudocode of **BASIC-BACKTRACK-SEARCH** is shown below.

```
function BASIC-BACKTRACK-SEARCH(csp) returns a solution, or failure
    return BASIC-BACKTRACK( {}, csp )

function BASIC-BACKTRACK(assignment,csp) returns a solution, or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var,assignment,csp) do
        if value is consistent with assignment then
            add { var = value } to assignment
```

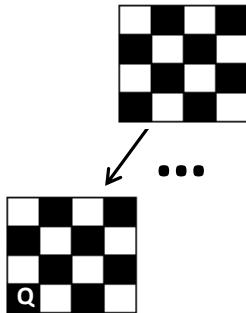
```

result  $\leftarrow$  BASIC-BACKTRACK(assignment,csp)
if result  $\neq$  failure then
    return result
remove { var = value } from assignment
return failure

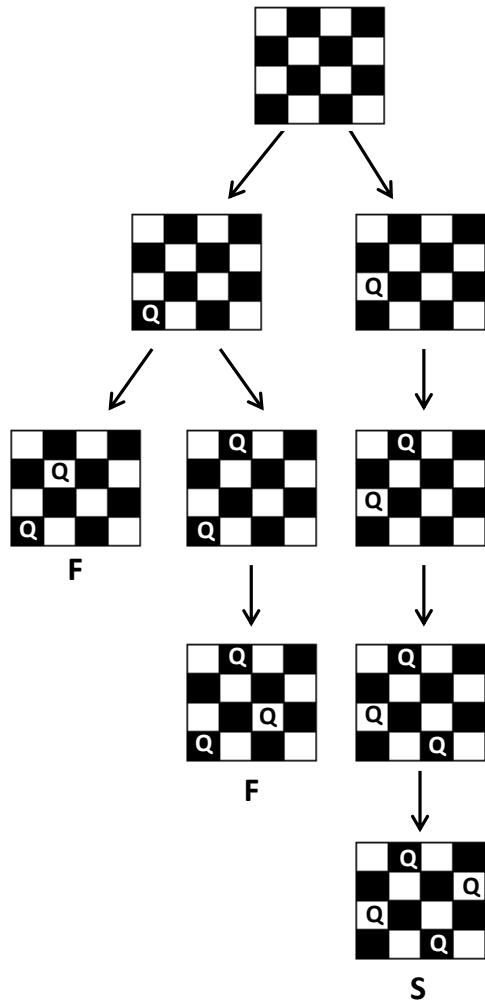
```

Assume that SELECT-UNASSIGNED-VARIABLE chooses the variables in the order A, B, C, D and that ORDER-DOMAIN-VALUES orders domain values 1, 2, 3, 4.

- b) Draw the search tree traversed by BASIC-BACKTRACK-SEARCH for the 4-Queens Problem. Each call to BASIC-BACKTRACK is a node in the tree and is drawn as the state of the chessboard given by assigned variables. The initial node and a child node corresponding to the assignment $A = 1$ are illustrated below. Mark failed leaf nodes with the letter "F" and mark the solution node with the letter "S".



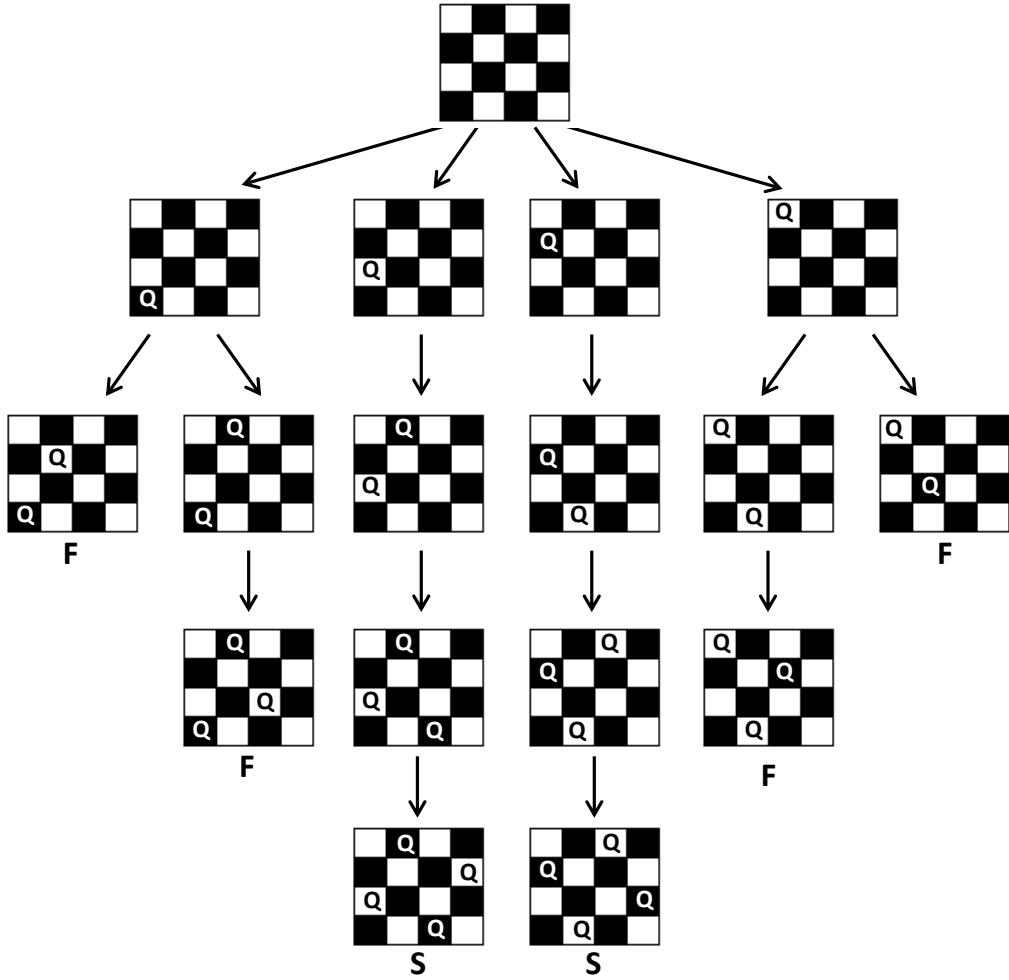
b) Answer:



BASIC-BACKTRACK-SEARCH returns a single solution or failure if no solution exists. We now would like to modify it to find all solutions.

- c) Assume that BASIC-BACKTRACK-SEARCH continues searching after finding the first solution to the 4-Queens Problem. Draw the complete search tree it then explores. Mark failed leaf nodes with the letter "F" and mark solution nodes with the letter "S".

c) Answer:



Let BACKTRACK-SEARCH-ALL refer to this new version of BASIC-BACK-TRACK-SEARCH.

- d) Write pseudocode for the function $\text{BACKTRACK-SEARCH-ALL}(csp)$ that returns the set of all solutions to a given constraint satisfaction problem, csp . Notice that the function simply returns the empty set if csp has no solutions. Hint: use the pseudocode for BASIC-BACK-TRACK-SEARCH as a template.

d) Answer:

```

function BACKTRACK-SEARCH-ALL(csp) returns the set of all solutions to csp
  return BACKTRACK-ALL({},csp)

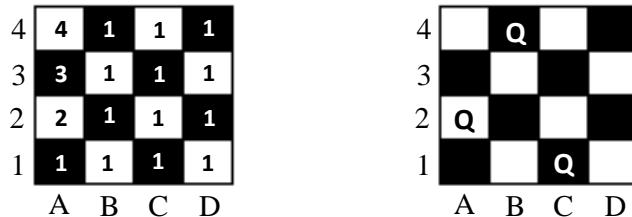
function BACKTRACK-ALL(assignment,csp) returns a set of solutions
  if assignment is complete then return {assignment}
  result  $\leftarrow \{\}$ 
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var,assignment,csp) do
    if value is consistent with assignment then
      add { var = value } to assignment
      result  $\leftarrow$  result  $\cup$  BACKTRACK-ALL(assignment,csp)
      remove { var = value } from assignment
  return result

```

e) Can the *least constraining value heuristic* for ordering domain values be used in order to reduce the number of search nodes explored by BACKTRACK-SEARCH-ALL (why/why not)?

e) Answer: No, it has no impact. We always need to explore all value assignments for a variable, so the order it happens does not matter.

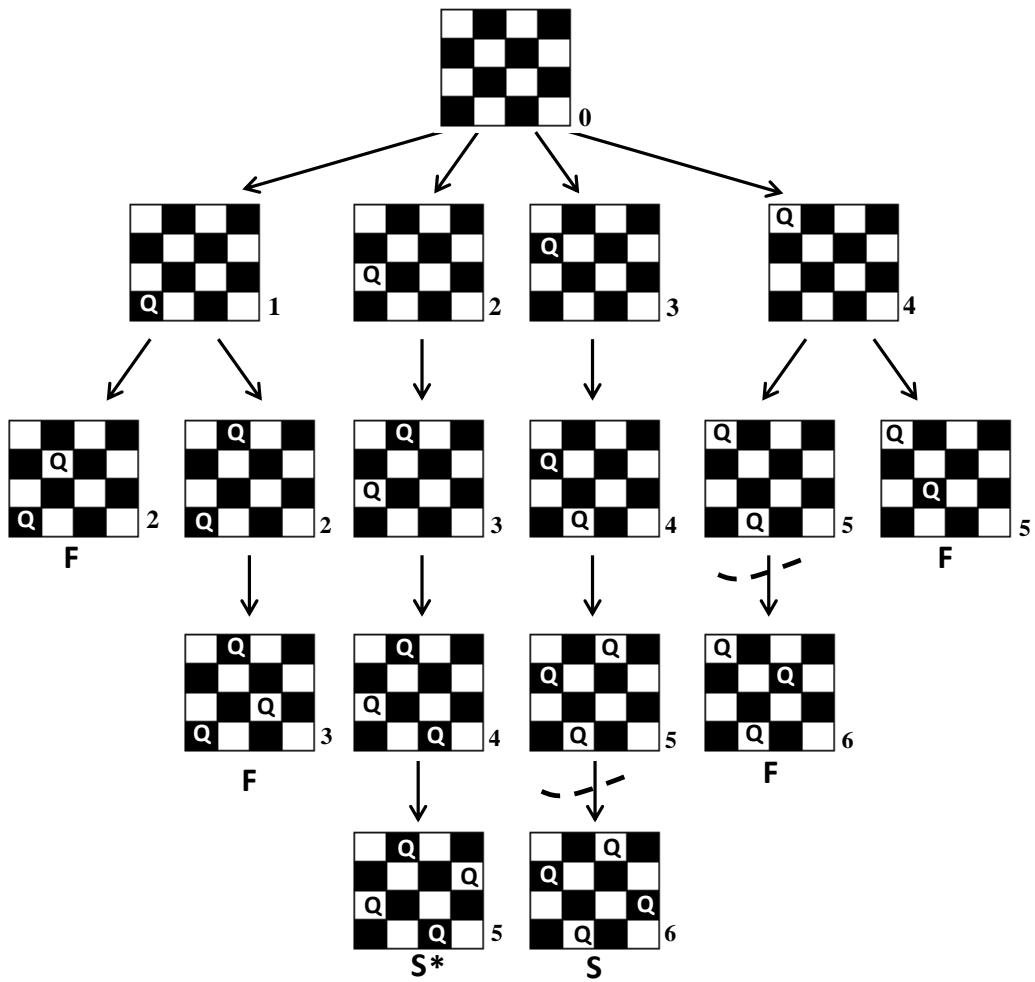
Assume that each domain value of the variables in a constraint satisfaction problem is associated with a cost. Further, define the cost of a variable assignment to be the sum of the costs of the assigned values. For our representation of the 4-Queens Problem, this means that each position on the chessboard has a cost. Assume that these costs are defined as shown in the left figure below. Thus, the cost of the assignment $A = 2$, $B = 4$, and $C = 1$ shown in the right figure is $2 + 1 + 1 = 4$.



An algorithm that returns all solutions like BACKTRACK-SEARCH-ALL can be used to find a solution with minimum cost of its assignment.

f) Calculate the costs of the assignments of the nodes in the complete search tree of the 4-Queens Problem that you made in your answer to sub-problem c). Write the costs next to the nodes and mark nodes with minimum cost solutions with a star "*" (you do not have to re-draw the tree). Assume that BACKTRACK-SEARCH-ALL finds a minimum cost solution by keeping track of a least cost solution found so far during the search. How can the cost of this solution be used to prune branches of the search tree that only can contain sub-optimal solutions? Show which branches are cut in the tree.

f) **Answer:** Since the cost of an assignment grows monotonically with the number of variables assigned, BACKTRACK-SEARCH-ALL does not have to expand children of a node with an assignment that has a cost that is equal to or larger than the cost of the current least cost solution. The reason is that these children only can lead to solutions with the same or higher cost than the least cost solution. The branches cut in the complete search tree of the 4-Queens Problem are shown below.



Problem 3: Logic (30%)

Patients often infect each other and it is important for hospitals to know whether an outbreak is COVID-19 or just flu or cold. We will develop a small knowledge base to determine this. Let the propositional symbols X , Y , and Z identify whether a group of patients have COVID-19, flu, or cold, respectively. Thus, if X is *true*, the group has COVID-19. The table below describes two relevant symptoms of these diseases.

Symptom	COVID-19 (X)	Flu (Y)	Cold (Z)
Fever (F)	Common (<i>true</i>)	Common (<i>true</i>)	Rare (<i>false</i>)
Loss of Appetite (L)	Common (<i>true</i>)	Sometimes (<i>false</i>)	Common (<i>true</i>)

We use the propositional symbols F and L to denote the observed symptoms of the patient group. F denotes fever and is *true* if fever is common in the group and *false* if it is rare. L denotes loss of appetite and is *true* if it is common and *false* if it sometimes happens in the group. Assume that we know that all patients in the group have one and only one disease and that it is either COVID-19, flu, or cold. Moreover, assume that we can draw conclusions from the symptoms that are certain. For instance, if fever is rare in the patient group, we can conclude that they have cold. The table below shows three different sentences (1), (2), and (3) representing the fever symptom.

(1)	(2)	(3)
$(F \Rightarrow X \wedge Y \wedge \neg Z)$	$(F \Rightarrow X \vee Y) \wedge$ $(F \Rightarrow \neg Z)$	$(F \Rightarrow X \vee Y) \wedge$ $(F \Rightarrow \neg Z) \wedge$ $(\neg F \Rightarrow Z) \wedge$ $(\neg F \Rightarrow \neg X \wedge \neg Y)$

- a) Explain why sentence (1) is incorrect and sentence (2) does not represent all information about the fever symptom according to our assumptions.

- a) **Answer:** Sentence (1) claims that if fever is common, the patients have COVID-19 *and* flu. This is impossible according to our assumptions. The first conjunct of sentence (2) says that the patients either have COVID-19 or flu. The second conjunct says that they can not have cold. This is correct according to our assumptions, but the derived knowledge is incomplete. The last two conjuncts of sentence (3) add the missing knowledge. They say that if fever is rare, the patients have cold and can not have COVID-19 or flu.

b) Prof. Smart claims that sentence (3) is logically equivalent to sentence (2) if the implication operator in each conjunct of (2) is substituted with a bi-implication operator. Nevertheless, he prefers sentence (3), since each of the four conjuncts of (3) is a definite clause. Do you agree about these claims (why/why not)?

b) Answer: Prof. Smart is correct about his first claim. We have

$$(F \Rightarrow X \vee Y) \wedge (\neg F \Rightarrow \neg X \wedge \neg Y) \equiv (F \Rightarrow X \vee Y) \wedge (F \Leftarrow X \vee Y) \\ \equiv F \Leftrightarrow X \vee Y$$

and

$$(F \Rightarrow \neg Z) \wedge (\neg F \Rightarrow Z) \equiv (F \Rightarrow \neg Z) \wedge (F \Leftarrow \neg Z) \\ \equiv F \Leftrightarrow \neg Z.$$

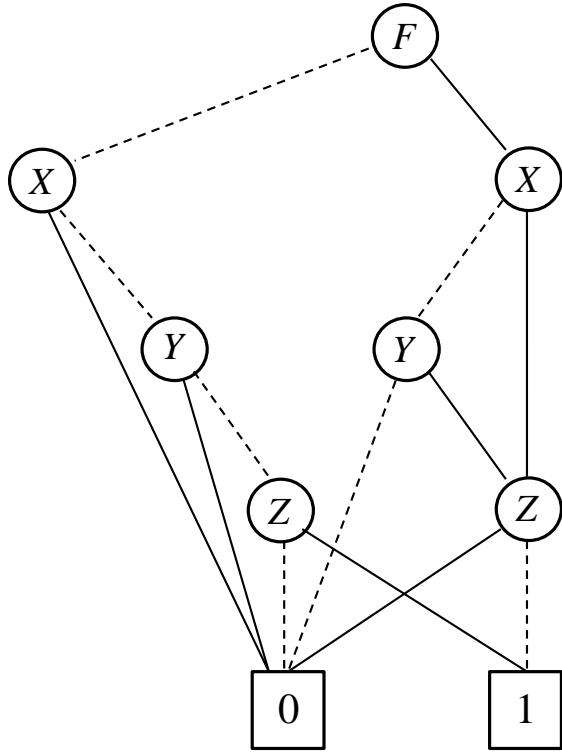
He is wrong, however, about his second claim. For instance, we have for the third conjunct of (3)

$$\neg F \Rightarrow Z \equiv F \vee Z.$$

This is not a definite clause since it has two positive literals and not exactly one as required.

c) Draw the BDD of (3) using variable order $F \prec X \prec Y \prec Z$.

c) Answer:



d) Write a sentence of the loss of appetite symptom (L) in the same format as (3).

d) Answer:

$$\begin{aligned} L &\Rightarrow X \vee Z \wedge \\ L &\Rightarrow \neg Y \wedge \\ \neg L &\Rightarrow Y \wedge \\ \neg L &\Rightarrow \neg X \wedge \neg Z \end{aligned}$$

e) Write the conjunction of (3) and the sentence for loss of appetite you gave in your answer to sub-problem d) in conjunctive normal form (CNF). Write all intermediate results of your derivation.

e) **Answer:** We first translate each conjunct of the two sentences to CNF

$$\begin{aligned}
 F \Rightarrow X \vee Y &\equiv (\neg F \vee X \vee Y) \\
 F \Rightarrow \neg Z &\equiv (\neg F \vee \neg Z) \\
 \neg F \Rightarrow Z &\equiv (F \vee Z) \\
 \neg F \Rightarrow \neg X \wedge \neg Y &\equiv F \vee (\neg X \wedge \neg Y) \equiv (F \vee \neg X) \wedge (F \vee \neg Y) \\
 L \Rightarrow X \vee Z &\equiv (\neg L \vee X \vee Z) \\
 L \Rightarrow \neg Y &\equiv (\neg L \vee \neg Y) \\
 \neg L \Rightarrow Y &\equiv (L \vee Y) \\
 \neg L \Rightarrow \neg X \wedge \neg Z &\equiv L \vee (\neg X \wedge \neg Z) \equiv (L \vee \neg X) \wedge (L \vee \neg Z).
 \end{aligned}$$

We then conjoin all these CNF sentences

$$\begin{aligned}
 &(\neg F \vee X \vee Y) \wedge \\
 &(\neg F \vee \neg Z) \wedge \\
 &(F \vee Z) \wedge \\
 &(F \vee \neg X) \wedge \\
 &(F \vee \neg Y) \wedge \\
 &(\neg L \vee X \vee Z) \wedge \\
 &(\neg L \vee \neg Y) \wedge \\
 &(L \vee Y) \wedge \\
 &(L \vee \neg X) \wedge \\
 &(L \vee \neg Z).
 \end{aligned}$$

f) Assume that a knowledge base (KB) contains the CNF sentence that you gave in your answer to sub-problem e). Show that $KB \wedge F \wedge L \models X$ using resolution. That is, show that the patients have COVID-19 if fever and loss of appetite is common.

f) **Answer:** We need to show that the empty clause () (i.e., *false*) can be derived from (F) , (L) , $(\neg X)$, and the set of clauses in KB . Resolution on $(\neg X)$ and $(\neg F \vee X \vee Y)$ gives $(\neg F \vee Y)$. Resolution on $(\neg F \vee Y)$ and (F) gives (Y) . Resolution on (L) and $(\neg L \vee \neg Y)$ gives $(\neg Y)$. Resolution on $(\neg Y)$ and (Y) gives () as required.

Introduction to Artificial Intelligence (IAI)

BSc and MSc Exam

IT University of Copenhagen

17 May 2021

This exam consists of 13 numbered pages with 5 problems. Each problem is marked with the weight in percent it is given in the evaluation. **Notice that the first version of problem 4 only is to be solved by Bachelor (BSc) students, while the second version only is to be solved by Master (MSc) students.**

- Ensure that you have a quiet, comfortable, and undisturbed work environment.
- Use notations and methods that have been discussed in the course.
- Make appropriate assumptions when a problem has been incompletely specified.

Good luck!

Problem 1: Heuristic Search (20%)

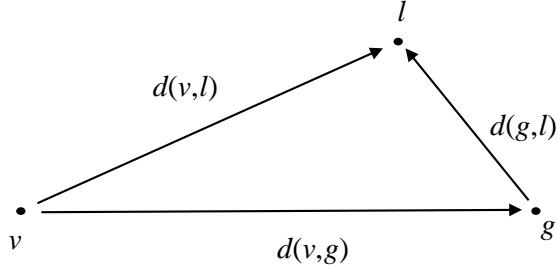
In route planning it is possible to compute strong heuristics based on landmarks. Consider a road network represented by a directed graph (V, E) . The set of vertices V is the intersections in the road network. The set of edges E is the road sections. Each edge $e \in E$ is represented by a triple $e = (v, v', c)$, where $v, v' \in V$ are the two intersections the road section connects and $c > 0$ is its length.

The states of the route planning problem is the set of vertices V . The actions is the set of edges E . An action of an edge (v, v', c) is to drive from v to v' and it has cost c . The initial state is some intersection $v_0 \in V$ and the goal is to find a route to some intersection $g \in V$.

The landmarks L is a subset of intersections $L \subset V$ that are geographically evenly distributed in the road network. An initial one-time effort is spent on computing the shortest distance in the road network $d(v, l)$ from any intersection $v \in V$ to any landmark $l \in L$. The result is used to compute heuristics for general route planning problems.

Consider the shortest distance in the road network $d(v, g)$ from some intersection v to the goal intersection g . Notice that unless g is a landmark, this distance has not been precomputed.

- a) Use the triangle below to show that $d(v, g) \geq d(v, l) - d(g, l)$ for any landmark $l \in L$.



- a) **Answer:** Due to the triangle inequality, we have $d(v, g) + d(g, l) \geq d(v, l)$. From this it follows $d(v, g) \geq d(v, l) - d(g, l)$.

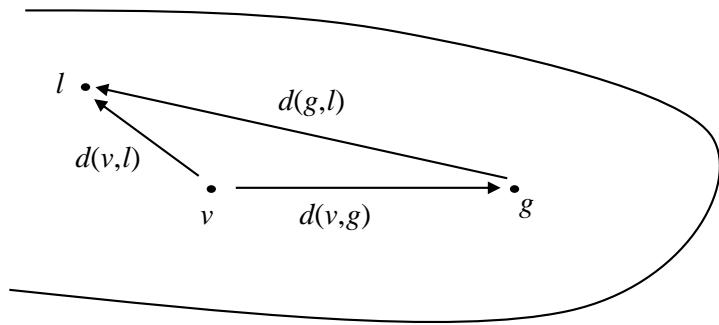
Let a heuristic function $h(v)$ be defined by $h(v) = \max_{l \in L} \{d(v, l) - d(g, l)\}$.

- b) A valid heuristic function requires $h(g) = 0$. Show that this requirement is fulfilled.

- b) **Answer:** We have $h(g) = \max_{l \in L} \{d(g, l) - d(g, l)\} = \max_{l \in L} \{0\} = 0$.

c) A valid heuristic function also requires $h(v) \geq 0$. Argue that even for a large set of evenly distributed landmarks it is possible for this requirement to be broken.

c) Answer:



As shown above, if g is at an edge of the roadmap such that v is closer than g to any landmark then $d(v, l) - d(g, l) \leq 0$ for all $l \in L$ such that $h(v) < 0$.

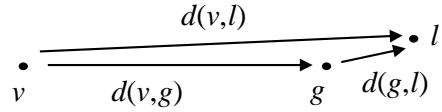
In the remainder we assume that $h(v) \geq 0$.

d) Why is $h(v)$ an admissible heuristic?

d) Answer: We have $d(v, g) \geq d(v, l) - d(g, l)$ for any landmark $l \in L$. For that reason $h(v) \leq d(v, g)$ as required.

e) Do you expect $h(v)$ to be a stronger heuristic than the straight line heuristic $h_{SLD}(v)$ (why/why not)?

e) Answer:



Since the landmarks are evenly distributed, we often will have a landmark behind and fairly close to the goal as shown above. In this case $d(v, l) - d(g, l)$ will be close to $d(v, g)$. The straight line distance on the other hand can be expected to be significantly less than $d(v, g)$ on most road networks and therefore a weaker heuristic.

Problem 2: Local Search (25%)¹

Assume that a student during a work week has seven lectures A, B, C, D, E, F , and G . Each lecture requires a single preparation activity that the student must plan. The name of a preparation activity is the lowercase letter of the lecture. Your task is to make a timetable of the preparation activities.

A timetable is feasible if the following constraints are fulfilled:

1. There are five days in a work week: Mon, Tue, Wed, Thu, and Fri.
2. Each day has four consecutive time slots.
3. The duration of each activity is one time slot.
4. The student can at most work on one activity in a time slot.
5. The time slots of the lectures are given and fixed.
6. A preparation activity must take place before the lecture and in the same week.

Assume that the student has found the feasible timetable FT shown below. Notice that the slots that can hold preparation activities are numbered 1 to 13.

Mon	Tue	Wed	Thu	Fri
a^1	B		e^3	g^4
A		C		F
c^7	d^8	D	E	G
b^9			f^{11}	I^{12}
				I^{13}

A feasible timetable has a cost that reflects the student's preferences. The cost is defined as the sum of the following terms:

1. Work early: there is a cost of two for each activity scheduled in the last time slot of a day (e.g., b causes an additional cost of two in FT).
2. Consolidate activities: a day has a cost of one if it has a period of inactivity between activities (e.g., Tue causes an additional cost of one in FT).
3. Prepare close to lectures: there is a cost of one for each day a preparation activity is before its lecture (e.g., c causes an additional cost of two in FT).

¹This problem is inspired by a BSc thesis project by M.R. Almind and R.K. Petersen, 2021.

a) What is the cost of FT ?

a) **Answer:** The "work early" cost is $2 + 2 = 4$, due to b and f . The "consolidate activities" cost is $1 + 1 = 2$, due to slot 5 and 6 being unoccupied. The "prepare close to lectures" cost is $1 + 2 + 1 + 1 = 5$, due to b , c , d , and f , respectively. Thus, the total cost of FT is $4 + 2 + 5 = 11$.

We will improve FT with local search using a swap neighborhood. A swap can happen between any pair of the 13 time slots that can hold preparation activities. The time slots must be different and at least one of them must currently hold a preparation activity. The result of a swap is that the time slots exchange assigned activities. A swap is only possible if the resulting timetable is feasible. Let $x \leftrightarrow y$ denote a swap between time slot x and y . The result of $2 \leftrightarrow 3$ in FT is that time slot 2 is assigned to e and time slot 3 becomes unoccupied. Swap $1 \leftrightarrow 7$ is impossible since a must happen before A . The swap neighborhood of a timetable is defined by all the possible swaps that can be done on it.

b) Write the neighborhood of FT as a list of pairs $(i \leftrightarrow j, cost)$, where $cost$ is the cost of the resulting timetable.

b) **Answer:** $(2 \leftrightarrow 3, 11), (2 \leftrightarrow 4, 13), (2 \leftrightarrow 7, 10), (2 \leftrightarrow 8, 9), (2 \leftrightarrow 12, 10), (3 \leftrightarrow 5, 11), (3 \leftrightarrow 6, 10), (3 \leftrightarrow 10, 14), (3 \leftrightarrow 11, 13), (4 \leftrightarrow 5, 13), (4 \leftrightarrow 6, 11), (4 \leftrightarrow 10, 16), (4 \leftrightarrow 11, 15), (4 \leftrightarrow 12, 11), (5 \leftrightarrow 7, 10), (5 \leftrightarrow 8, 10), (5 \leftrightarrow 12, 10), (6 \leftrightarrow 12, 8), (7 \leftrightarrow 8, 11), (7 \leftrightarrow 9, 11), (7 \leftrightarrow 10, 13), (8 \leftrightarrow 10, 13), (10 \leftrightarrow 12, 13), (11 \leftrightarrow 12, 12)$.

Recall the pseudocode of the simulated annealing algorithm for a maximization problem shown below.

1. **function** SIMULATED-ANNEALING($problem, schedule$) **returns** a solution
2. $current \leftarrow \text{MAKE-NODE}(problem.\text{INITIAL-STATE})$
3. **for** $t = 1$ to ∞ **do**
4. $T \leftarrow schedule(t)$
5. **if** $T = 0$ **then return** $current$
6. $next \leftarrow$ a random selected neighbor of $current$
7. $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
8. **if** $\Delta E \geq 0$ **then** $current \leftarrow next$
9. **else** $current \leftarrow next$ only with probability $e^{\Delta E/T}$

c) Re-write the pseudocode of the simulated annealing algorithm to change it to solve a minimization problem. You only have to re-write the lines that change.

c) **Answer:** The only change required is to make ΔE positive for an improving solution. Hence, line 7 becomes $\Delta E \leftarrow current.VALUE - next.VALUE$.

Assume that your corrected algorithm uses a temperature schedule defined by $schedule(1) = 3$, $schedule(2) = 2$, $schedule(3) = 2$, and $schedule(4) = 0$. Further, assume that the random selected neighbor of $current$ in line 6 is the result of swap $10 \leftrightarrow 12$, $6 \leftrightarrow 12$, $2 \leftrightarrow 3$ in iteration 1, 2, and 3 of the algorithm, respectively. Finally, assume that only neighbors with $e^{\Delta E/T}$ larger than 0.55 are accepted.

d) Write the value of T , $next$, and $\Delta E/T$ for each iteration of your corrected simulated annealing algorithm. Write also the value of $e^{\Delta E/T}$ and $current$ when relevant. Assume that $problem.INITIAL-STATE$ is the feasible timetable FT found by the student.

d) Answer:

Iteration 1

$$T = 3$$

next = result of swap $10 \leftrightarrow 12$

$$\Delta E = 11 - 13 = -2$$

$$e^{-2/3} \approx 0.51$$
 not accepted

Iteration 2

$$T = 2$$

next = result of swap $6 \leftrightarrow 12$

$$\Delta E = 11 - 8 = 3$$
 accepted

current =

Mon	Tue	Wed	Thu	Fri
a^1	B	2	e^3	g^4
A	5	C	f^6	F
c^7	d^8	D	E	G
b^9	10	11	12	13

Iteration 3

$$T = 2$$

next = result of swap $2 \leftrightarrow 3$

$$\Delta E = 8 - 9 = -1$$

$$e^{-1/2} \approx 0.61$$
 accepted

current =

Mon	Tue	Wed	Thu	Fri
a^1	B	e^2	3	g^4
A	5	C	f^6	F
c^7	d^8	D	E	G
b^9	10	11	12	13

Iteration 4

$$T = 0$$

return *current*

Prof. Smart wants to show that the swap neighborhood is complete. Assume that FT_1 and FT_2 are two arbitrarily chosen feasible timetables for the same set of lectures with fixed time slots. Prof Smart must show that there is a series of possible swaps that transforms FT_1 to FT_2 .

e) Assume that preparation activity p is at time slot s_1 in FT_1 and at time slot s_2 in FT_2 , where s_1 happens chronologically before s_2 . Explain why the swap $s_1 \leftrightarrow s_2$ always is possible in FT_1 .

e) Answer: There are two cases to consider. The first case is that s_2 is unoccupied in FT_1 . The only change is that p will be moved to a later slot, but this slot is feasible in FT_2 it also will be in FT_1 . The second case is that s_2 holds preparation activity q in FT_1 . The additional change is that q will be swapped to an earlier time slot, but this will not break any constraints since it still happens before its lecture.

The question is what to do if preparation activity p is at time slot s_1 in FT_1 and at time slot s_2 in FT_2 , where s_1 happens chronologically after s_2 . Prof. Smart claims that since the swap $s_2 \leftrightarrow s_1$ is feasible in FT_2 , the series of swaps that transforms FT_1 to FT_2 can be found as follows. Start by transforming FT_2 by iteratively swapping any preparation activity that has an earlier time slot than in FT_1 to the time slot it has in FT_1 . Call the resulting timetable FT'_2 . Every preparation activity in FT_1 must now be at the same or an earlier time slot than in FT'_2 . Transform FT_1 to FT'_2 by iteratively swapping any preparation activity that has an earlier time slot than in FT'_2 to the time slot it has in FT'_2 . The series of swaps transforming FT_1 to FT_2 is the ones just performed on FT_1 followed by the ones done on FT_2 in reverse order and with every swap reversed.

f) Is Prof. Smart's claim correct (why/why not)?

f) Answer: Yes. If preparation activity p is at time slot s_1 in FT_1 and at time slot s_2 in FT_2 , where s_1 happens chronologically after s_2 , the swap $s_2 \leftrightarrow s_1$ is possible in FT_2 for the reasons given in the answer to question e). Since the argument holds for any such p , it is possible to transform FT_2 to FT'_2 . Swaps can be reversed. If swap $i \leftrightarrow j$ is possible on timetable T and results in T' , swap $j \leftrightarrow i$ is possible on timetable T' and results in timetable T . Thus, it is possible to transform FT'_2 to FT_2 by reversing the swaps and applying them in reverse order. The swaps performed on FT_1 results FT'_2 as every swap only will affect preparation activities that currently are not at their right slot in FT'_2 .

Problem 3: Logic and ROBDDs (30%)

Mastermind is a code-breaker game for two players. One player is the *codemaker*, the other is the *codebreaker*. The game is played using a decoding board with a shield at one end covering a code. In the version of the game that we will study, the code is made with black and white *code pegs*. The code consists of three of these placed in a row of holes behind the shield. The game starts by the codemaker making a code behind the shield. The codebreaker is not allowed to see it. The codebreaker tries to guess the code. Each guess is made by placing a row of code pegs on the decoding board. Once placed, the codemaker provides feedback by placing small black and white *key pegs* in the three holes to the left of the row with the guess. A black key peg is placed for each code peg from the guess which is correct in both color and position. A white key peg indicates the existence of a correct color of a code peg but placed in the wrong position. Consider the state of the game shown below after the codebreaker has made two guesses.

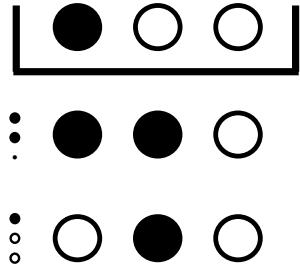


Figure 1: Example game state

The code is the top row behind the indicated shield. The first guess is the next row. It gets two black key pegs shown to the left. The reason is that the left and right code peg in the guess have the correct color. The middle black code peg, on the other hand, has a wrong color. The second guess gets one black key peg since the right white code peg in the guess have the correct color. It also gets two white key pegs. The reason is that the left and middle code peg by swapping position can get the right color without using a position currently associated with a black key peg. Notice that the ordering of the key pegs is irrelevant.

We will use propositional logic to reason about this game. Let the Boolean variables x_1 , x_2 , and x_3 represent the left, middle, and right code peg of the code made by the codemaker. Similarly, let the Boolean variables y_1 , y_2 , and y_3 represent the left, middle, and right code peg of a code guess made by the codebreaker. Assume that a peg variable is assigned to *True* if the peg is black and *False* if the peg is white.

a) Write a Boolean expression on x_1 , x_2 , and x_3 that is true exactly for the code made by the codemaker in the game state shown above.

a) Answer: $x_1 \wedge \neg x_2 \wedge \neg x_3$.

Now, consider any code made by the codemaker and any guess made by the codebreaker.

b) Write a Boolean expression on x_1 , x_2 , x_3 , y_1 , y_2 , and y_3 that is true if and only if the codebreaker has guessed the code (use $p \Leftrightarrow q$ to express that two Boolean variables have the same truth value).

b) Answer: $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3)$.

c) Write a Boolean expression E on x_1 , x_2 , x_3 , y_1 , y_2 , and y_3 that is true if and only if the feedback from the codemaker is two black key pegs.

c) Answer:

$$\begin{aligned} E \equiv & \\ & (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge \neg(x_3 \Leftrightarrow y_3) \vee \\ & (x_1 \Leftrightarrow y_1) \wedge \neg(x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3) \vee \\ & \neg(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3). \end{aligned}$$

d) Formally prove that $(True \Leftrightarrow x) \equiv x$ and $(False \Leftrightarrow x) \equiv \neg x$.

d) Answer:

$$\begin{aligned}
 & (True \Leftrightarrow x) \\
 & \equiv (True \Rightarrow x) \wedge (x \Rightarrow True) \text{ // biconditional elim.} \\
 & \equiv (\neg True \vee x) \wedge (\neg x \vee True) \text{ // implication elim.} \\
 & \equiv (False \vee x) \wedge (\neg x \vee True) \text{ // semantics} \\
 & \equiv x \wedge True \text{ // simple equivalences} \\
 & \equiv x \text{ // simple equivalences}
 \end{aligned}$$

$$\begin{aligned}
 & (False \Leftrightarrow x) \\
 & \equiv (False \Rightarrow x) \wedge (x \Rightarrow False) \text{ // biconditional elim.} \\
 & \equiv (\neg False \vee x) \wedge (\neg x \vee False) \text{ // implication elim.} \\
 & \equiv (True \vee x) \wedge (\neg x \vee False) \text{ // semantics} \\
 & \equiv True \wedge \neg x \text{ // simple equivalences} \\
 & \equiv \neg x \text{ // simple equivalences.}
 \end{aligned}$$

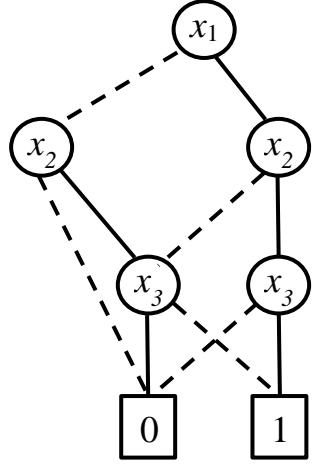
e) Assume that the codebreaker's guess is the first guess made in the game state shown in Figure 1. Use logical equivalences to simplify the expression $E[True/y_1, True/y_2, False/y_3]$ representing this situation. What codes in addition to black-white-white are possible?

e) Answer:

$$\begin{aligned}
 E[True/y_1, True/y_2, False/y_3] & \equiv \\
 & (x_1 \Leftrightarrow True) \wedge (x_2 \Leftrightarrow True) \wedge \neg(x_3 \Leftrightarrow False) \vee \\
 & (x_1 \Leftrightarrow True) \wedge \neg(x_2 \Leftrightarrow True) \wedge (x_3 \Leftrightarrow False) \vee \\
 & \neg(x_1 \Leftrightarrow True) \wedge (x_2 \Leftrightarrow True) \wedge (x_3 \Leftrightarrow False) \equiv \\
 & x_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge \neg x_2 \wedge \neg x_3 \vee \neg x_1 \wedge x_2 \wedge \neg x_3.
 \end{aligned}$$

This expression is only true for the assignments represented by the codes black-black-black, black-white-white, and white-black-white. Thus, the codes in addition to black-white-white that are possible are black-black-black and white-black-white.

Prof. Smart claims that the expression $E[True/y_1, True/y_2, False/y_3]$ is represented by the ROBDD below.



f) Show that Prof. Smart's claim is correct. Write and explain the expression that the ROBDD represents and prove that it is logically equivalent to $E[True/y_1, True/y_2, False/y_3]$.

f) Answer: The expression represented by the ROBDD is a disjunction of expressions for the paths leading to the 1-terminal: $x_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge \neg x_2 \wedge \neg x_3 \vee \neg x_1 \wedge x_2 \wedge \neg x_3$. As the answer to question e) shows, this expression is logically equivalent to $E[True/y_1, True/y_2, False/y_3]$.

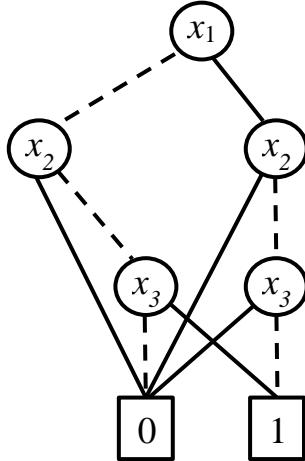
Consider the feedback of the second guess of the codebreaker shown in Figure 1.

g) Which codes of the codemaker can get this feedback?

g) Answer: One code peg must match and the rest must be possible to swap to a position, where they match. If left peg matches, such a code is white-white-black. If the middle peg matches there is no code that can get this feedback. If the right peg matches, we get black-white-white.

h) Draw an ROBDD with variable order $x_1 \prec x_2 \prec x_3$ that represents the set of codes you found in question g).

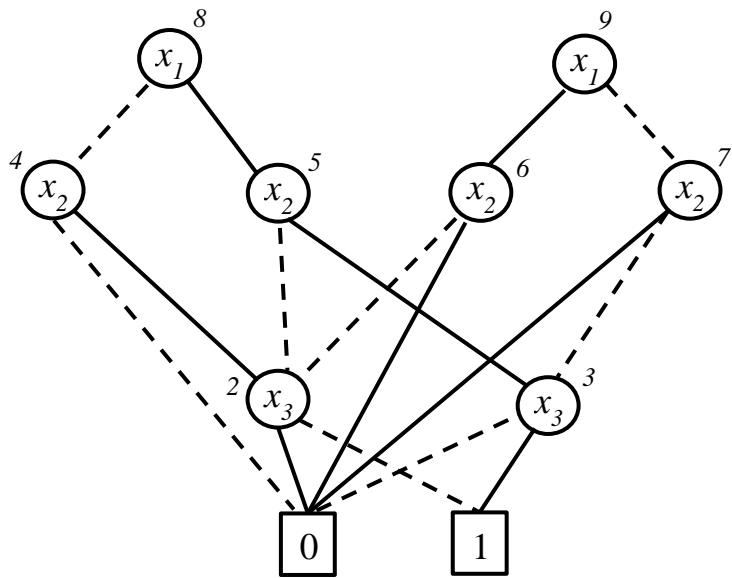
h) Answer: The ROBDD below with variable order $x_1 \prec x_2 \prec x_3$ has two satisfying assignments [$x_1 = \text{False}, x_2 = \text{False}, x_3 = \text{True}$] and [$x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{False}$] as required.



Assume that Prof. Smart's ROBDD and your ROBDD from question h) is represented by a multi-rooted ROBDD using variable order $x_1 \prec x_2 \prec x_3$.

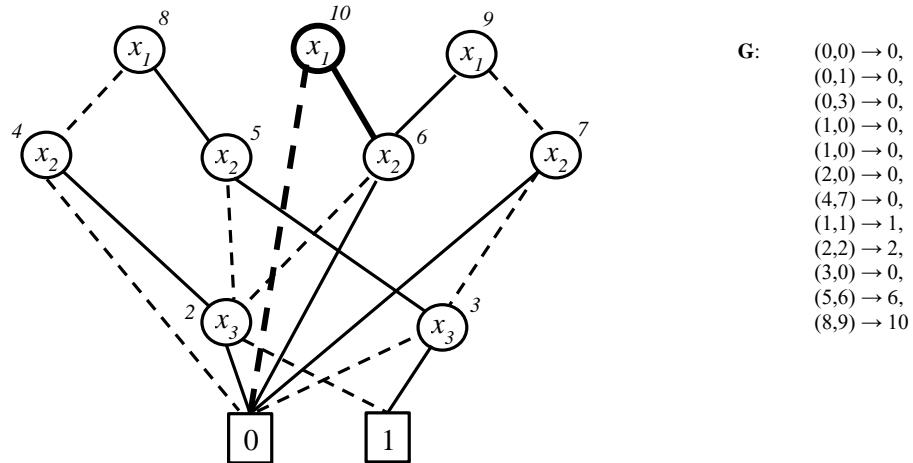
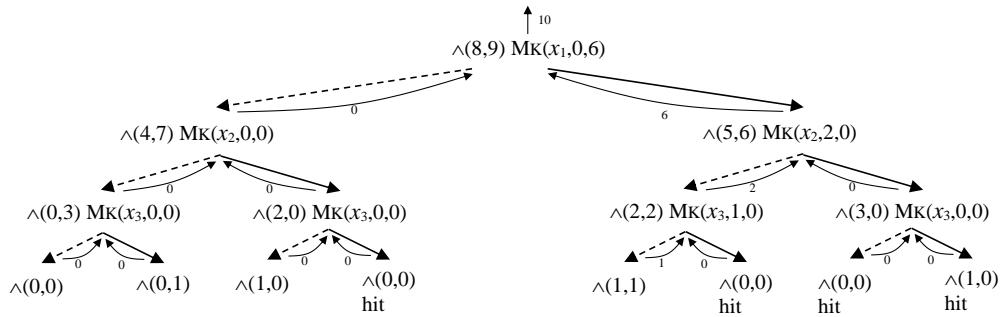
- i)** Draw the multi-rooted ROBDD and write identifiers next to each internal node that is consistent with the multi-rooted ROBDD being made by a series of calls to MK.

i) Answer:



j) Use APPLY to make the conjunction (\wedge) of the two ROBDDs using your multi-root representation. Draw the call tree of APP as in the APPLY example of Lecture 7. Indicate cache hits and draw the resulting multi-rooted ROBDD. How can your result be used by the codebreaker in the example game state shown in Figure 1?

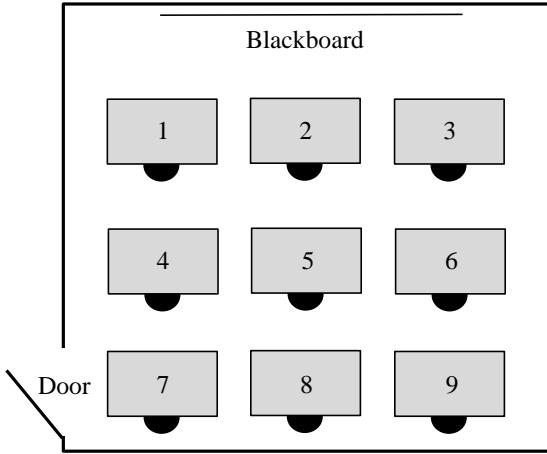
j) Answer:



The results show that the code must be black-white-white given the two feedbacks from the codemaker.

Problem 4: FOR BSC STUDENTS ONLY (25%)

After the COVID-19 reopening, a class teacher faces the challenge of assigning pupils to tables in the classroom. There are five students Adam, Ben, Connie, David, and Eve. The classroom has nine numbered tables in three rows as shown below.



The rules are:

1. At most one pupil can sit at a table.
2. Connie must be close to the door either at table 4,5,7, or, 8.
3. Eve is near-sighted and must sit near the blackboard in the front row.
4. David and Adam must be separated by at least a complete row.
5. There can not be more than one girl or one boy per row.
6. Due to COVID-19, a table directly above, below, to the left, or to the right of an occupied table must be empty.

We will use constraint programming to solve this problem. Let the variables A , B , C , D , and E denote the table assigned to Adam, Ben, Connie, David, and Eve, respectively. The domain of each variable is $\{1, 2, \dots, 9\}$. The constraints are defined by the six rules described above.

a) Which of the rules are unary constraints and why?

a) **Answer:** Unary constraints are constraints on single variables. They are the constraints of rule 2 and 3.

b) The remaining constraints can be formulated as binary constraints. Define each of these constraints formally and introduce notation as needed. As an example, a formalization of the last rule is:

$$(\alpha = i) \Rightarrow (\beta \neq j) \text{ forall } \alpha \in V, i \in D, \beta \in V \setminus \{\alpha\}, j \in Adj_i,$$

where $V = \{A, B, C, D, E\}$, $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and Adj_i is the set of any tables directly above, below, to the left, or to the right of table i .

b) Answer:

Rule 1

$$\alpha \neq \beta \text{ forall } \alpha \in V, \beta \in V \setminus \{\alpha\}.$$

Rule 4

$$(\alpha = i) \Rightarrow (\beta \neq j) \text{ forall } \alpha \in \{D, A\}, i \in D, \beta \in \{D, A\} \setminus \{\alpha\}, j \in RowAB_i,$$

where $RowAB_i$ is the set of tables in any row directly above and below the row of table i .

Rule 5

$$(\alpha = i) \Rightarrow (\beta \neq j) \text{ forall } \alpha \in \{C, E\}, i \in D, \beta \in \{C, E\} \setminus \{\alpha\}, j \in RowS_i$$

and

$$(\alpha = i) \Rightarrow (\beta \neq j) \text{ forall } \alpha \in \{A, B, D\}, i \in D, \beta \in \{A, B, D\} \setminus \{\alpha\}, j \in RowS_i,$$

where $RowS_i$ is the set of tables in the same row as table i .

We will use the following table to indicate possible values of the variables.

	1	2	3	4	5	6	7	8	9
A	✓	✓	✓	✓	✓	✓	✓	✓	✓
B	✓	✓	✓	✓	✓	✓	✓	✓	✓
C	✓	✓	✓	✓	✓	✓	✓	✓	✓
D	✓	✓	✓	✓	✓	✓	✓	✓	✓
E	✓	✓	✓	✓	✓	✓	✓	✓	✓

c) Remove check marks from the table corresponding to domain values that break node consistency.

c) Answer: Connie must sit at table 4,5,7, or 8, so all other values are impossible for variable C . Eve must sit in the front row, so all other value than 1, 2, and 3 are impossible for variable E . The resulting table is shown below.

	1	2	3	4	5	6	7	8	9
A	✓	✓	✓	✓	✓	✓	✓	✓	✓
B	✓	✓	✓	✓	✓	✓	✓	✓	✓
C				✓	✓		✓	✓	
D	✓	✓	✓	✓	✓	✓	✓	✓	✓
E	✓	✓	✓						

d) Continue by also removing check marks of values that break arc consistency. Explain why the removed values break arc consistency. Be careful there might be less than you think.

d) Answer: David and Adam must be separated by a complete row. If either of them sit in the middle row, there is no position possible for the other. Thus, these positions are not arc-consistent. No other values can be removed due to arc consistency.

	1	2	3	4	5	6	7	8	9
A	✓	✓	✓				✓	✓	✓
B	✓	✓	✓	✓	✓	✓	✓	✓	✓
C				✓	✓		✓	✓	
D	✓	✓	✓				✓	✓	✓
E	✓	✓	✓						

We will use FORWARD-CHECKING-SEARCH and MAC-SEARCH to solve the CSP. Assume that the CSP has been made node and arc consistent before calling the algorithms.

e) Draw the call tree of RECURSIVE-FORWARD-CHECKING. Draw each internal call node of the tree as the state of the table of domain values after the inference step of the algorithm. Draw leaf nodes that returns *failure* as <failure>. Draw leaf nodes that find a result as <success>. Next to each arc in the tree, write the assignment of the variable selected in the call of the parent node associated with it. Assume that variables are selected in ascending lexicographical order and that domain values are assigned in ascending numerical order.

e) Answer:

	1	2	3	4	5	6	7	8	9
A	✓	✓	✓				✓	✓	✓
B	✓	✓	✓	✓	✓	✓	✓	✓	✓
C				✓	✓		✓	✓	
D	✓	✓	✓				✓	✓	✓
E	✓	✓	✓						

↓ $A = 1$

	1	2	3	4	5	6	7	8	9
A	✓								
B					✓	✓	✓	✓	✓
C					✓		✓	✓	
D							✓	✓	✓
E			✓						

↓ $B = 5$

	1	2	3	4	5	6	7	8	9
A	✓								
B					✓				
C							✓		
D							✓		✓
E			✓						

↓ $D = 9$

	1	2	3	4	5	6	7	8	9
A	✓								
B					✓				
C							✓		
D								✓	
E			✓						

↓

< success >

f) Draw the call tree of RECURSIVE-MAC using the same approach.

f) Answer:

	1	2	3	4	5	6	7	8	9
A	✓	✓	✓			✓	✓	✓	
B	✓	✓	✓	✓	✓	✓	✓	✓	✓
C				✓	✓		✓	✓	
D	✓	✓	✓			✓	✓	✓	
E	✓	✓	✓						

↓ $A = 1$

	1	2	3	4	5	6	7	8	9
A	✓								
B					✓		✓	✓	✓
C					✓		✓	✓	
D							✓	✓	✓
E			✓						

↓ $B = 5$

	1	2	3	4	5	6	7	8	9
A	✓								
B					✓				
C							✓		
D								✓	
E			✓						

↓

< success >

Problem 4: FOR MSC STUDENTS ONLY (25%)

After being expelled from the Garden of Eden, Adam and Eve played Rock-Paper-Scissors about the hard tasks they faced. Rock-Paper-Scissors is played by each person simultaneously showing a hand sign for either Rock (R), Paper (P), or Scissors (S). The outcome of the game is win, draw, or lose, with utility 1, 0, and -1, respectively. From Adam's point of view, the different combinations have the utilities shown in the table below.

		Adam			
		R	P	S	
		R	0	1	-1
		Eve	P	-1	0
		S	1	-1	0

Let x_1 , x_2 , and x_3 denote the probabilities that Adam plays rock, paper, and scissors, respectively. The expected utility U of the game for Adam depends on what Eve plays as shown in the table below.

Eve plays	Rock	Paper	Scissors
U	$x_2 - x_3$	$-x_1 + x_3$	$x_1 - x_2$

After some time, Adam learns that Eve plays Paper with probability $\frac{1}{2}$ and Rock and Scissors with probability $\frac{1}{4}$. He wonders which probabilities x_1 , x_2 , and x_3 he should choose to maximize his expected utility of the game in this situation. He realizes that he needs to find an optimal solution to the problem

$$\begin{array}{ll} \text{Maximize} & \frac{1}{4}(x_2 - x_3) + \frac{1}{2}(-x_1 + x_3) + \frac{1}{4}(x_1 - x_2) \\ \text{Subject to} & x_1 + x_2 + x_3 = 1 \end{array}$$

$$x_1, x_2, x_3 \geq 0.$$

a) Explain why the problem is a linear program.

a) Answer: It is on non-negative real decision variables. The objective is a maximization of a linear function on the decision variables. The constraints are linear functions on the left side, a real number on the right side and one of the operators $=$, \leq , and \geq . The problem therefore fulfills the requirements to a linear program.

b) Write the problem in standard form.

b) Answer:

$$\begin{array}{lll} \text{Maximize} & -\frac{1}{4}x_1 & + \frac{1}{4}x_3 \\ \text{Subject to} & x_1 + x_2 + x_3 \leq 1 \\ & -x_1 - x_2 - x_3 \leq -1 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

c) Write the slack form of the linear program and explain why it is infeasible as an initial dictionary to the SIMPLEX algorithm.

c) Answer:

$$\begin{array}{lll} \text{Maximize} & z \\ \text{Subject to} & x_4 = 1 - x_1 - x_2 - x_3 \\ & x_5 = -1 + x_1 + x_2 + x_3 \\ & z = -\frac{1}{4}x_1 + \frac{1}{4}x_3 \\ & x_1, x_2, \dots, x_5 \geq 0 \end{array}$$

The slack form is infeasible since $x_5 < 0$ in the dictionary.

d) Write the slack form of the auxiliary problem used by the two phase SIMPLEX algorithm.

d) Answer:

$$\begin{array}{lll} \text{Maximize} & w \\ \text{Subject to} & x_4 = 1 - x_1 - x_2 - x_3 + x_0 \\ & x_5 = -1 + x_1 + x_2 + x_3 + x_0 \\ & w = -x_0 \\ & x_0, x_1, \dots, x_5 \geq 0 \end{array}$$

e) Solve the first phase of the two phase SIMPLEX algorithm. Write all dictionaries produced by the algorithm.

e) Answer:

Initial dictionary (slack form of auxiliary problem) : First pivot entering x_0 , leaving x_5 .

$$\begin{array}{ll} \text{Maximize} & w \\ \text{Subject to} & x_4 = 2 - 2x_1 - 2x_2 - 2x_3 + x_5 \\ & x_0 = 1 - x_1 - x_2 - x_3 + x_5 \\ & w = -1 + x_1 + x_2 + x_3 - x_5 \\ & x_0, x_1, \dots, x_5 \geq 0 \end{array}$$

First and final dictionary : Pivot entering x_1 , leaving x_0 .

$$\begin{array}{ll} \text{Maximize} & w \\ \text{Subject to} & x_4 = 0 - x_5 + 2x_0 \\ & x_1 = 1 - x_2 - x_3 + x_5 - x_0 \\ & w = -x_0 \\ & x_0, x_1, \dots, x_5 \geq 0 \end{array}$$

f) Solve the second phase of the two phase SIMPLEX algorithm. Write all dictionaries produced by the algorithm.

f) **Answer:**

Initial dictionary from first phase : Pivot entering x_3 , leaving x_1 .

$$\begin{aligned} \text{Maximize } z \\ \text{Subject to } x_4 &= 0 && - x_5 \\ x_1 &= 1 - x_2 - x_3 + x_5 \\ z &= -\frac{1}{4} + \frac{1}{4}x_2 + \frac{1}{2}x_3 - \frac{1}{4}x_5 \\ & & & x_1, x_2, \dots, x_5 \geq 0 \end{aligned}$$

First dictionary : Pivot entering x_5 , leaving x_4 .

$$\begin{aligned} \text{Maximize } z \\ \text{Subject to } x_4 &= 0 && - x_5 \\ x_3 &= 1 - x_1 - x_2 + x_5 \\ z &= \frac{1}{4} - \frac{1}{2}x_1 - \frac{1}{4}x_2 + \frac{1}{4}x_5 \\ & & & x_1, x_2, \dots, x_5 \geq 0 \end{aligned}$$

Final optimal dictionary.

$$\begin{aligned} \text{Maximize } z \\ \text{Subject to } x_5 &= 0 && - x_4 \\ x_3 &= 1 - x_1 - x_2 - x_4 \\ z &= \frac{1}{4} - \frac{1}{2}x_1 - \frac{1}{4}x_2 - \frac{1}{4}x_4 \\ & & & x_1, x_2, \dots, x_5 \geq 0 \end{aligned}$$

g) What are the optimal probabilities of x_1 , x_2 , and x_3 for Adam and what is Adam's expected utilization of the game when using these probabilities?

g) Answer: The optimal probabilities is the solutions of the optimal dictionary of the SIMPLEX algorithm $x_1 = 0$, $x_2 = 0$, and $x_3 = 1$. The expected utilization is the value of z in the optimal dictionary $\frac{1}{4}$.

Eve soon realizes that Adam has figured her out and changes her strategy. Adam reconsiders how to choose x_1 , x_2 , and x_3 . No matter what choice he makes, Eve eventually will learn his probabilities. When she has done that, she will simply play the token that gives him least utilization. For that reason he must find an

optimal solution to the problem

$$\text{Maximize } \min\{x_2 - x_3, -x_1 + x_3, x_1 - x_2\}$$

$$\text{Subject to } x_1 + x_2 + x_3 = 1$$

$$x_1, x_2, x_3 \geq 0.$$

h) Show that this problem can be formulated as a linear program. You do not have to write it in standard form (hint, you need to introduce an auxiliary variable).

h) Answer:

$$\text{Maximize } u$$

$$\text{Subject to } u \leq x_2 - x_3$$

$$u \leq -x_1 + x_3$$

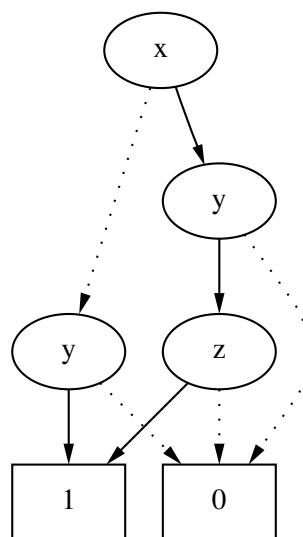
$$u \leq x_1 - x_2$$

$$x_1 + x_2 + x_3 = 1$$

$$x_1, x_2, x_3, u \geq 0.$$

An Introduction to Binary Decision Diagrams

Henrik Reif Andersen



Lecture notes for 49285 Advanced Algorithms E97, October 1997.
(Minor revisions, Apr. 1998)
E-mail: hra@it.dtu.dk. Web: <http://www.it.dtu.dk/~hra>

Department of Information Technology, Technical University of Denmark
Building 344, DK-2800 Lyngby, Denmark.

Preface

This note is a short introduction to Binary Decision Diagrams. It provides some background knowledge and describes the core algorithms. More details can be found in Bryant's original paper on Reduced Ordered Binary Decision Diagrams [Bry86] and the survey paper [Bry92]. A recent extension called Boolean Expression Diagrams is described in [AH97].

This note is a revision of an earlier version from fall 1996 (based on versions from 1995 and 1994). The major differences are as follows: Firstly, ROBDDs are now viewed as nodes of one global graph with one fixed ordering to reflect state-of-the-art of efficient BDD packages. The algorithms have been changed (and simplified) to reflect this fact. Secondly, a proof of the canonicity lemma has been added. Thirdly, the sections presenting the algorithms have been completely restructured. Finally, the project proposal has been revised.

Acknowledgements

Thanks to the students on the courses of fall 1994, 1995, and 1996 for helping me debug and improve the notes. Thanks are also due to Hans Rischel, Morten Ulrik Sørensen, Niels Marette, Jørgen Staunstrup, Kim Skak Larsen, Henrik Hulgaard, and various people on the Internet who found typos and suggested improvements.

Contents

1 Boolean Expressions	6
2 Normal Forms	7
3 Binary Decision Diagrams	8
4 Constructing and Manipulating ROBDDs	15
4.1 MK	15
4.2 BUILD	17
4.3 APPLY	19
4.4 RESTRICT	20
4.5 SATCOUNT, ANYSAT, ALLSAT	22
4.6 SIMPLIFY	25
4.7 Existential Quantification and Substitution	25
5 Implementing the ROBDD operations	27
6 Examples of problem solving with ROBDDs	27
6.1 The 8 Queens problem	27
6.2 Correctness of Combinational Circuits	29
6.3 Equivalence of Combinational Circuits	29
7 Verification with ROBDDs	31
7.1 Knights tour	33
8 Project: An ROBDD Package	35
References	36

1 Boolean Expressions

The classical calculus for dealing with *truth values* consists of *Boolean variables* x, y, \dots , the constants *true* 1 and *false* 0, the operators of *conjunction* \wedge , *disjunction* \vee , *negation* \neg , *implication* \Rightarrow , and *bi-implication* \Leftrightarrow which together form the Boolean expressions. Sometimes the variables are called *propositional variables* or *propositional letters* and the Boolean expressions are then known as *Propositional Logic*.

Formally, Boolean expressions are generated from the following grammar:

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \Rightarrow t \mid t \Leftrightarrow t,$$

where x ranges over a set of Boolean variables. This is called the *abstract syntax* of Boolean expressions. The concrete syntax includes parentheses to solve ambiguities. Moreover, as a common convention it is assumed that the operators bind according to their relative priority. The priorities are, with the highest first: \neg , \wedge , \vee , \Leftrightarrow , \Rightarrow . Hence, for example

$$\neg x_1 \wedge x_2 \vee x_3 \Rightarrow x_4 = (((\neg x_1) \wedge x_2) \vee x_3) \Rightarrow x_4.$$

A Boolean expression with variables x_1, \dots, x_n denotes for each assignment of truth values to the variables itself a truth value according to the standard truth tables, see figure 1. *Truth assignments* are written as sequences of assignments of values to variables, e.g., $[0/x_1, 1/x_2, 0/x_3, 1/x_4]$ which assigns 0 to x_1 and x_3 , 1 to x_2 and x_4 . With this particular truth assignment the above expression has value 1, whereas $[0/x_1, 1/x_2, 0/x_3, 0/x_4]$ yields 0.

	\neg	\wedge	\vee	\Rightarrow	\Leftrightarrow
0	1	0 1	0 0 0	0 1 1	0 1 0
1	0	1 0	1 1 1	1 0 1	1 0 1

Figure 1: Truth tables.

The set of truth values is often denoted $\mathbb{B} = \{0, 1\}$. If we fix an ordering of the variables of a Boolean expression t we can view t as defining a function from \mathbb{B}^n to \mathbb{B} where n is the number of variables. Notice, that the particular ordering chosen for the variables is essential for what function is defined. Consider for example the expression $x \Rightarrow y$. If we choose the ordering $x < y$ then this is the function $f(x, y) = x \Rightarrow y$, true if the first argument implies the second, but if we choose the ordering $y < x$ then it is the function $f(y, x) = x \Rightarrow y$, true if the second argument implies the first. When we later consider compact representations of Boolean expressions, such variable orderings play a crucial role.

Two Boolean expressions t and t' are said to be equal if they yield the same truth value for all truth assignments. A Boolean expression is a *tautology* if it yields true for all truth assignments; it is *satisfiable* if it yields true for at least one truth assignment.

Exercise 1.1 Show how all operators can be encoded using only \neg and \vee . Use this to argue that any Boolean expression can be written using only \vee , \wedge , variables, and \neg applied to variables.

Exercise 1.2 Argue that t and t' are equal if and only if $t \Leftrightarrow t'$ is a tautology. Is it possible to say whether t is satisfiable from the fact that $\neg t$ is a tautology?

2 Normal Forms

A Boolean expression is in *Disjunctive Normal Form (DNF)* if it consists of a disjunction of conjunctions of variables and negations of variables, i.e., if it is of the form

$$(t_1^1 \wedge t_2^1 \wedge \cdots \wedge t_{k_1}^1) \vee \cdots \vee (t_1^l \wedge t_2^l \wedge \cdots \wedge t_{k_l}^l) \quad (1)$$

where each t_i^j is either a variable x_i^j or a negation of a variable $\neg x_i^j$. An example is

$$(x \wedge \neg y) \vee (\neg x \wedge y)$$

which is a well-known function of x and y (which one?). A more succinct presentation of (1) is to write it using indexed versions of \wedge and \vee :

$$\bigvee_{j=1}^l \left(\bigwedge_{i=1}^{k_j} t_i^j \right).$$

Similarly, a *Conjunctive Normal Form (CNF)* is an expression that can be written as

$$\bigwedge_{j=1}^l \left(\bigvee_{i=1}^{k_j} t_i^j \right)$$

where each t_i^j is either a variable or a negated variable. It is not difficult to prove the following proposition:

Proposition 1 *Any Boolean expression is equal to an expression in CNF and an expression in DNF.*

In general, it is hard to determine whether a Boolean expression is satisfiable. This is made precise by a famous theorem due to Cook [Coo71]:

Theorem 1 (Cook) *Satisfiability of Boolean expressions is NP-complete.*

(For readers unfamiliar with the notion of NP-completeness the following short summary of the pragmatic consequences suffices. Problems that are NP-complete can be solved by algorithms that run in exponential time. No polynomial time algorithms are known to exist for any of the NP-complete problems and it is very unlikely that polynomial time algorithms should indeed exist although nobody has yet been able to prove their non-existence.)

Cook's theorem even holds when restricted to expressions in CNF. For DNFs satisfiability is decidable in polynomial time but for DNFs the tautology check is hard (co-NP complete). Although satisfiability is easy for DNFs and tautology check easy for CNFs,

this does not help us since the conversion between CNFs and DNFs is exponential as the following example shows.

Consider the following CNF over the variables $x_0^1, \dots, x_0^n, x_1^1, \dots, x_1^n$:

$$(x_0^1 \vee x_1^1) \wedge (x_0^2 \vee x_1^2) \wedge \dots \wedge (x_0^n \vee x_1^n).$$

The corresponding DNF is a disjunction which has a disjunct for each of the n -digit binary numbers from $000\dots000$ to $111\dots111$ — the i 'th digit representing a choice of either x_0^i (for 0) or x_1^i (for 1):

$$\begin{aligned} & (x_0^1 \wedge x_0^2 \wedge \dots \wedge x_0^{n-1} \wedge x_0^n) \quad \vee \\ & (x_0^1 \wedge x_0^2 \wedge \dots \wedge x_0^{n-1} \wedge x_1^n) \quad \vee \\ & \quad \vdots \\ & (x_1^1 \wedge x_1^2 \wedge \dots \wedge x_1^{n-1} \wedge x_0^n) \quad \vee \\ & (x_1^1 \wedge x_1^2 \wedge \dots \wedge x_1^{n-1} \wedge x_1^n). \end{aligned}$$

Whereas the original expression has size proportional to n the DNF has size proportional to $n2^n$.

The next section introduces a normal form that has more desirable properties than DNFs and CNFs. In particular, there are efficient algorithms for determining the satisfiability and tautology questions.

Exercise 2.1 Describe a polynomial time algorithm for determining whether a DNF is satisfiable.

Exercise 2.2 Describe a polynomial time algorithm for determining whether a CNF is a tautology.

Exercise 2.3 Give a proof of proposition 1.

Exercise 2.4 Explain how Cook's theorem implies that checking in-equivalence between Boolean expressions is NP-hard.

Exercise 2.5 Explain how the question of tautology and satisfiability can be decided if we are given an algorithm for checking equivalence between Boolean expressions.

3 Binary Decision Diagrams

Let $x \rightarrow y_0, y_1$ be the *if-then-else* operator defined by

$$x \rightarrow y_0, y_1 = (x \wedge y_0) \vee (\neg x \wedge y_1)$$

hence, $t \rightarrow t_0, t_1$ is true if t and t_0 are true or if t is false and t_1 is true. We call t the *test expression*. All operators can easily be expressed using only the if-then-else operator and the constants 0 and 1. Moreover, this can be done in such a way that all tests are performed only on (un-negated) variables and variables occur in no other places. Hence the operator gives rise to a new kind of normal form. For example, $\neg x$ is $(x \rightarrow 0, 1)$, $x \Leftrightarrow y$ is $x \rightarrow (y \rightarrow 1, 0), (y \rightarrow 0, 1)$. Since variables must only occur in tests the Boolean expression x is represented as $x \rightarrow 1, 0$.

An *If-then-else Normal Form (INF)* is a Boolean expression built entirely from the if-then-else operator and the constants 0 and 1 such that all tests are performed only on variables.

If we by $t[0/x]$ denote the Boolean expression obtained by replacing x with 0 in t then it is not hard to see that the following equivalence holds:

$$t = x \rightarrow t[1/x], t[0/x]. \quad (2)$$

This is known as the *Shannon expansion* of t with respect to x . This simple equation has a lot of useful applications. The first is to generate an INF from any expression t . If t contains no variables it is either equivalent to 0 or 1 which is an INF. Otherwise we form the Shannon expansion of t with respect to one of the variables x in t . Thus since $t[0/x]$ and $t[1/x]$ both contain one less variable than t , we can recursively find INFs for both of these; call them t_0 and t_1 . An INF for t is now simply

$$x \rightarrow t_1, t_0.$$

We have proved:

Proposition 2 *Any Boolean expression is equivalent to an expression in INF.*

Example 1 Consider the Boolean expression $t = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$. If we find an INF of t by selecting in order the variables x_1, y_1, x_2, y_2 on which to perform Shannon expansions, we get the expressions

$$\begin{aligned} t &= x_1 \rightarrow t_1, t_0 \\ t_0 &= y_1 \rightarrow 0, t_{00} \\ t_1 &= y_1 \rightarrow t_{11}, 0 \\ t_{00} &= x_2 \rightarrow t_{001}, t_{000} \\ t_{11} &= x_2 \rightarrow t_{111}, t_{110} \\ t_{000} &= y_2 \rightarrow 0, 1 \\ t_{001} &= y_2 \rightarrow 1, 0 \\ t_{110} &= y_2 \rightarrow 0, 1 \\ t_{111} &= y_2 \rightarrow 1, 0 \end{aligned}$$

Figure 2 shows the expression as a tree. Such a tree is also called a *decision tree*. \square

A lot of the expressions are easily seen to be identical, so it is tempting to identify them. For example, instead of t_{110} we can use t_{000} and instead of t_{111} we can use t_{001} . If we substitute t_{000} for t_{110} in the right-hand side of t_{11} and also t_{001} for t_{111} , we in fact see that t_{00} and t_{11} are identical, and in t_1 we can replace t_{11} with t_{00} .

If we in fact identify *all* equal subexpressions we end up with what is known as a *binary decision diagram* (a *BDD*). It is no longer a tree of Boolean expressions but a directed acyclic graph (DAG).

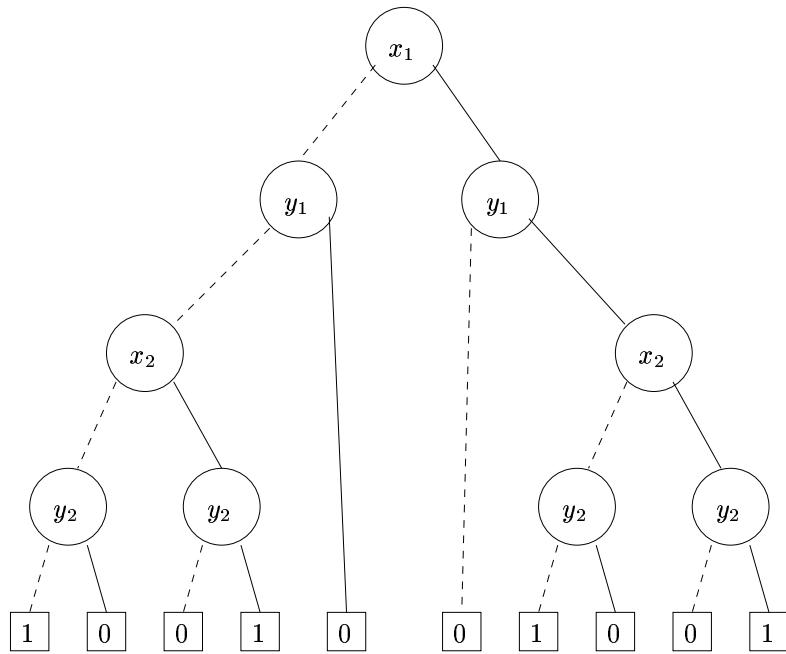


Figure 2: A decision tree for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$. Dashed lines denote low-branches, solid lines high-branches.

Applying this idea of sharing, t can now be written as:

$$\begin{aligned}
 t &= x_1 \rightarrow t_1, t_0 \\
 t_0 &= y_1 \rightarrow 0, t_{00} \\
 t_1 &= y_1 \rightarrow t_{00}, 0 \\
 t_{00} &= x_2 \rightarrow t_{001}, t_{000} \\
 t_{000} &= y_2 \rightarrow 0, 1 \\
 t_{001} &= y_2 \rightarrow 1, 0
 \end{aligned}$$

Each subexpression can be viewed as the node of a graph. Such a node is either *terminal* in the case of the constants 0 and 1, or *non-terminal*. A non-terminal node has a low-edge corresponding to the else-part and a high-edge corresponding to the then-part. See figure 3. Notice, that the number of nodes has decreased from 9 in the decision tree to 6 in the BDD. It is not hard to imagine that if each of the terminal nodes were other big decision trees the savings would be dramatic. Since we have chosen to consistently select variables in the same order in the recursive calls during the construction of the INF of t , the variables occur in the same orderings on all paths from the root of the BDD. In this situation the binary decision diagram is said to be *ordered* (an *OBDD*). Figure 3 shows a BDD that is also an OBDD.

Figure 4 shows four OBDDs. Some of the tests (e.g., on x_2 in b) are redundant, since both the low- and high-branch lead to the same node. Such unnecessary tests can be removed: any reference to the redundant node is simply replaced by a reference to

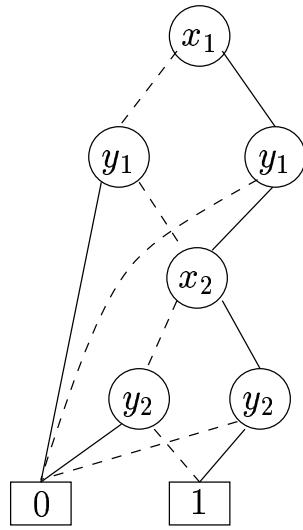


Figure 3: A BDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ with ordering $x_1 < y_1 < x_2 < y_2$. Low-edges are drawn as dotted lines and high-edges as solid lines.

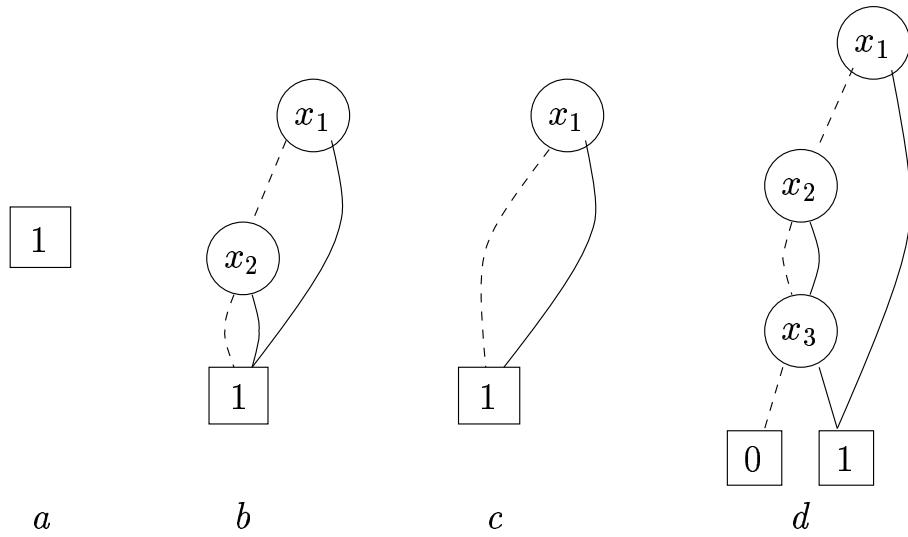


Figure 4: Four OBDDs: a) An OBDD for 1. b) Another OBDD for 1 with two redundant tests. c) Same as b) with one of the redundant tests removed. d) An OBDD for $x_1 \vee x_3$ with one redundant test.

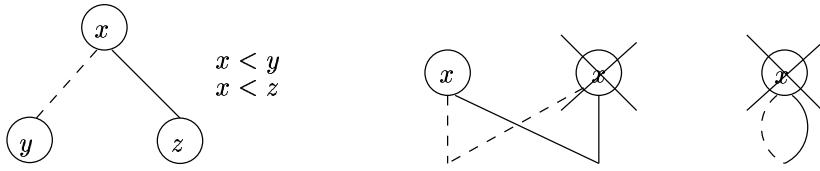


Figure 5: The ordering and reducedness conditions of ROBDDs. Left: Variables must be *ordered*. Middle: Nodes must be *unique*. Right: Only *non-redundant tests* should be present.

its subnode. If all identical nodes are shared and all redundant tests are eliminated, the OBDD is said to be *reduced* (an *ROBDD*). ROBDDs have some very convenient properties centered around the *canonicity lemma* below. (Often when people speak about BDDs they really mean *ROBDDs*.) To summarize:

A *Binary Decision Diagram (BDD)* is a rooted, directed acyclic graph with

- one or two terminal nodes of out-degree zero labeled 0 or 1, and
- a set of variable nodes u of out-degree two. The two outgoing edges are given by two functions $\text{low}(u)$ and $\text{high}(u)$. (In pictures, these are shown as dotted and solid lines, respectively.) A variable $\text{var}(u)$ is associated with each variable node.

A BDD is *Ordered* (OBDD) if on all paths through the graph the variables respect a given linear order $x_1 < x_2 < \dots < x_n$. An (O)BDD is *Reduced* (R(O)BDD) if

- (**uniqueness**) no two distinct nodes u and v have the same variable name and low- and high-successor, i.e.,

$$\text{var}(u) = \text{var}(v), \text{low}(u) = \text{low}(v), \text{high}(u) = \text{high}(v) \text{ implies } u = v,$$

and

- (**non-redundant tests**) no variable node u has identical low- and high-successor, i.e.,

$$\text{low}(u) \neq \text{high}(u).$$

The ordering and reducedness conditions are shown in figure 5.

ROBDDs have some interesting properties. They provide compact representations of Boolean expressions, and there are efficient algorithms for performing all kinds of logical operations on ROBDDs. They are all based on the crucial fact that for any function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ there is *exactly one ROBDD representing it*. This means, in particular, that there is *exactly one ROBDD* for the constant true (and constant false) function on \mathbb{B}^n : the terminal node 1 (and 0 in case of false). Hence, it is possible to *test in constant time whether an ROBDD is constantly true or false*. (Recall that for Boolean expressions this problem is NP-complete.)

To make this claim more precise we must say what we mean for an ROBDD to represent a function. First, it is quite easy to see how the nodes u of an ROBDD inductively defines Boolean expressions t^u : A terminal node is a Boolean constant. A non-terminal node marked with x is an if-then-else expression where the condition is x and the two branches are the Boolean expressions given by the low- or high-son, respectively:

$$\begin{aligned} t^0 &= 0 \\ t^1 &= 1 \\ t^u &= \text{var}(u) \rightarrow t^{\text{high}(u)}, t^{\text{low}(u)}, \quad \text{if } u \text{ is a variable node.} \end{aligned}$$

Moreover, if $x_1 < x_2 < \dots < x_n$ is the variable ordering of the ROBDD, we associate with each node u the function f^u that maps $(b_1, b_2, \dots, b_n) \in \mathbb{B}^n$ to the truth value of $t^u[b_1/x_1, b_2/x_2, \dots, b_n/x_n]$. We can now state the key lemma:

Lemma 1 (Canonicity lemma)

For any function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ there is exactly one ROBDD u with variable ordering $x_1 < x_2 < \dots < x_n$ such that $f^u = f(x_1, \dots, x_n)$.

Proof: The proof is by induction on the number of arguments of f . For $n = 0$ there are only two Boolean functions, the constantly false and constantly true functions. Any ROBDD containing at least one non-terminal node is non-constant. (Why?) Therefore there is exactly one ROBDD for each of these: the terminals 0 and 1.

Assume now that we have proven the lemma for all functions of n arguments. We proceed to show it for all functions of $n+1$ arguments. Let $f : \mathbb{B}^{n+1} \rightarrow \mathbb{B}$ be any Boolean function of $n+1$ arguments. Define the two functions f_0 and f_1 of n arguments by fixing the first argument of f to 0 respectively 1:

$$f_b(x_2, \dots, x_{n+1}) = f(b, x_2, \dots, x_{n+1}) \text{ for } b \in \mathbb{B}.$$

(Sometimes f_0 and f_1 are called the *negative* and *positive co-factors* of f with respect to x_1 .) These functions satisfy the following equation:

$$f(x_1, \dots, x_n) = x_1 \rightarrow f_1(x_2, \dots, x_n), f_0(x_2, \dots, x_n). \quad (3)$$

Since f_0 and f_1 take only n arguments we assume by induction that there are unique ROBDD nodes u_0 and u_1 with $f^{u_0} = f_0$ and $f^{u_1} = f_1$.

There are two cases to consider. If $u_0 = u_1$ then $f^{u_0} = f^{u_1}$ and $f_0 = f^{u_0} = f^{u_1} = f_1 = f$. Hence $u_0 = u_1$ is an ROBDD for f . It is also the only ROBDD for f since due to the ordering, if x_1 is at all present in the ROBDD rooted at u , x_1 would need to be the root node. However, if $f = f^u$ then $f_0 = f^u[0/x_1] = f^{\text{low}(u)}$ and $f_1 = f^u[1/x_1] = f^{\text{high}(u)}$. Since $f_0 = f^{u_0} = f^{u_1} = f_1$ by assumption, the low- and high-son of u would be the same, making the ROBDD violate the reducedness condition of non-redundant tests.

If $u_0 \neq u_1$ then $f^{u_0} \neq f^{u_1}$ by the induction hypothesis (using the names x_2, \dots, x_{n+1} in place of x_1, \dots, x_n). We take u to be the node with $\text{var}(u) = x_1$, $\text{low}(u) = u_0$, and $\text{high}(u) = u_1$, i.e., $f^u = x_1 \rightarrow f^{u_1}, f^{u_0}$ which is reduced. By assumption $f^{u_1} = f_1$ and $f^{u_0} = f_0$ therefore using (3) we get $f^u = f$. Suppose that v is some other node with $f^v = f$. Clearly, f^v must depend on x_1 , i.e., $f^v[0/x_1] \neq f^v[1/x_1]$ (otherwise also

$f_0 = f^v[0/x_1] = f^v[1/x_1] = f_1$, a contradiction). Due to the ordering this means that $\text{var}(v) = x_1 = \text{var}(u)$. Moreover, from $f^v = f$ it follows that $f^{\text{low}(v)} = f_0 = f^{u_0}$ and $f^{\text{high}(v)} = f_1 = f^{u_1}$, which by the induction hypothesis implies that $\text{low}(v) = u_0 = \text{low}(u)$ and $\text{high}(v) = u_1 = \text{high}(u)$. From the reducedness property of uniqueness it follows that $u = v$. \square

An immediate consequence is the following. Since the terminal 1 is an ROBDD for all variable orderings it is the only ROBDD that is constantly true. So in order to check whether an ROBDD is constantly true it suffices to check whether it is the terminal 1 which is definitely a constant time operation. Similarly, ROBDDs that are constantly false must be identical to the terminal 0. In fact, to determine whether two Boolean functions are the same, it suffices to construct their ROBDDs (in the same graph) and check whether the resulting nodes are the same!

The ordering of variables chosen when constructing an ROBDD has a great impact on the size of the ROBDD. If we consider again the expression $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ and construct an ROBDD using the ordering $x_1 < x_2 < y_1 < y_2$ the ROBDD consists of 9 nodes (figure 6) and not 6 nodes as for the ordering $x_1 < y_1 < x_2 < y_2$ (figure 3).

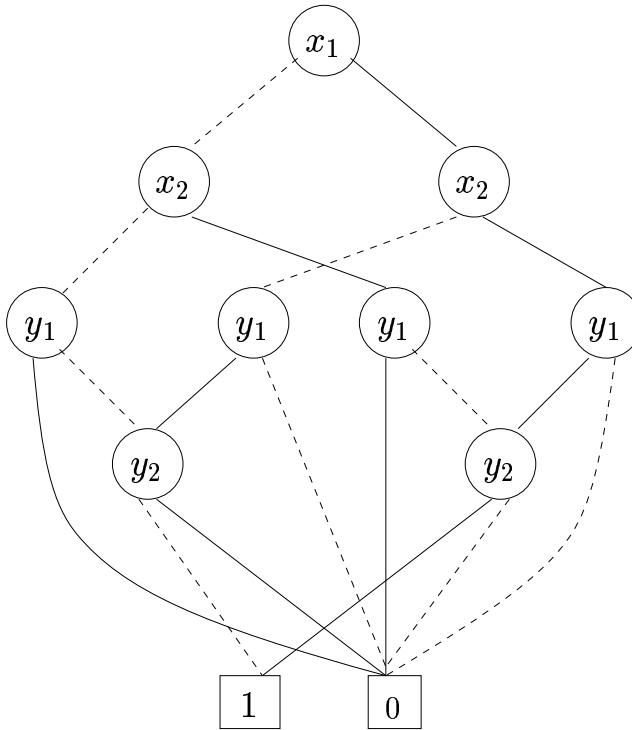


Figure 6: The ROBDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ with variable ordering $x_1 < x_2 < y_1 < y_2$.

Exercise 3.1 Show how to express all operators from the if-then-else operator and the constants 0 and 1.

Exercise 3.2 Draw the ROBDDs for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3)$ with orderings $x_1 < x_2 < x_3 < y_1 < y_2 < y_3$ and $x_1 < y_1 < x_2 < y_2 < x_3 < y_3$.

Exercise 3.3 Draw the ROBDDs for $(x_1 \Leftrightarrow y_1) \vee (x_2 \Leftrightarrow y_2)$ with orderings $x_1 < x_2 < y_1 < y_2$ and $x_1 < y_1 < x_2 < y_2$. How does it compare with the example in figures 3 and 6? Based on the examples you have seen so far, what variable ordering would you recommend for constructing a small ROBDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3) \wedge \dots \wedge (x_k \Leftrightarrow y_k)$?

Exercise 3.4 Give an example of a sequence of ROBDDs $u_n, 0 \leq n$ which induces exponentially bigger decision trees. I.e., if u_n has size $\Theta(n)$ then the decision tree should have size $\Theta(2^n)$.

Exercise 3.5 Construct an ROBDD of maximum size over six variables.

4 Constructing and Manipulating ROBDDs

In the previous section we saw how to construct an OBDD from a Boolean expression by a simple recursive procedure. The question arises now how do we construct a *reduced* OBDD? One way is to first construct an OBDD and then proceed by reducing it. Another more appealing approach, which we follow here, is to reduce the OBDD during construction.

To describe how this is done we will need an explicit representation of ROBDDs. Nodes will be represented as numbers $0, 1, 2, \dots$ with 0 and 1 reserved for the terminal nodes. The variables in the ordering $x_1 < x_2 < \dots < x_n$ are represented by their indices $1, 2, \dots, n$. The ROBDD is stored in a table $T : u \mapsto (i, l, h)$ which maps a node u to its three attributes $var(u) = i$, $low(u) = l$, and $high(u) = h$. Figure 7 shows the representation of the ROBDD from figure 3 (with the variable names changed to $x_1 < x_2 < x_3 < x_4$).

4.1 Mk

In order to ensure that the OBDD being constructed is reduced, it is necessary to determine from a triple (i, l, h) whether there exists a node u with $var(u) = i$, $low(u) = l$, and $high(u) = h$. For this purpose we assume the presence of a table $H : (i, l, h) \mapsto u$ mapping triples (i, l, h) of variable indices i , and nodes l, h to nodes u . The table H is the “inverse” of the table T , i.e., for variable nodes u , $T(u) = (i, l, h)$, if and only if, $H(i, l, h) = u$. The operations needed on the two tables are:

$T : u \mapsto (i, l, h)$	
$init(T)$	initialize T to contain only 0 and 1
$u \leftarrow add(T, i, l, h)$	allocate a new node u with attributes (i, l, h)
$var(u), low(u), high(u)$	lookup the attributes of u in T
$H : (i, l, h) \mapsto u$	
$init(H)$	initialize H to be empty
$b \leftarrow member(H, i, l, h)$	check if (i, l, h) is in H
$u \leftarrow lookup(H, i, l, h)$	find $H(i, l, h)$
$insert(H, i, l, h, u)$	make (i, l, h) map to u in H

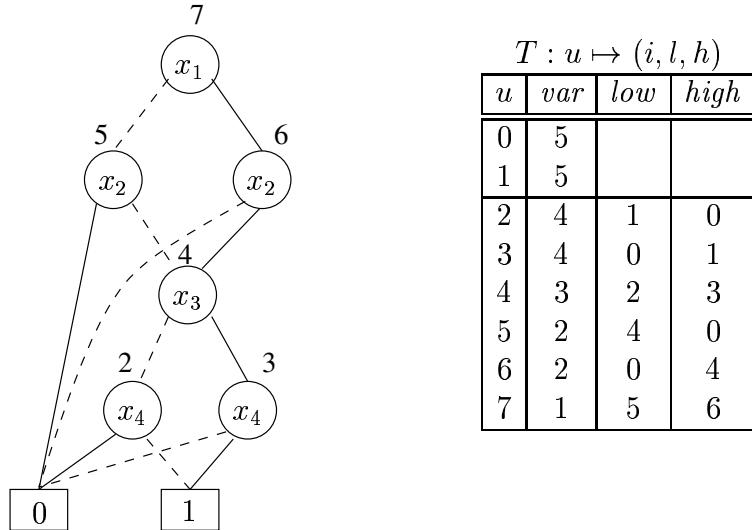


Figure 7: Representing an ROBDD with ordering $x_1 < x_2 < x_3 < x_4$. The numbers inside the vertices are the identities used in the representation. The numbers 0 and 1 are reserved for the terminal nodes. The numbers to the right of the ROBDD shows the index of the variables in the ordering. The constants are assigned an index which is the number of variables in the ordering plus one (here $4+1 = 5$). This makes some subsequent algorithms easier to present. The low- and high-fields are unused for the terminal nodes.

```

MK[T, H](i, l, h)
1:  if  $l = h$  then return  $l$ 
2:  else if member( $H, i, l, h$ ) then
3:      return lookup( $H, i, l, h$ )
4:  else  $u \leftarrow add(T, i, l, h)$ 
5:      insert( $H, i, l, h, u$ )
6:  return  $u$ 

```

Figure 8: The function $\text{MK}[T, H](i, l, h)$.

```

BUILD[ $T, H$ ]( $t$ )
1: function BUILD'( $t, i$ ) =
2:   if  $i > n$  then
3:     if  $t$  is false then return 0 else return 1
4:   else  $v_0 \leftarrow \text{BUILD}'(t[0/x_i], i + 1)$ 
5:      $v_1 \leftarrow \text{BUILD}'(t[1/x_i], i + 1)$ 
6:     return MK( $i, v_0, v_1$ )
7: end BUILD'
8:
9: return BUILD'( $t, 1$ )

```

Figure 9: Algorithm for building an ROBDD from a Boolean expression t using the ordering $x_1 < x_2 < \dots < x_n$. In a call $\text{BUILD}'(t, i)$, i is the lowest index that any variable of t can have. Thus when the test $i > n$ succeeds, t contains no variables and must be either constantly false or true.

We shall assume that all these operations can be performed in *constant time*, $O(1)$. Section 5 will show how such a low complexity can be achieved.

The function $\text{MK}[T, H](i, l, h)$ (see figure 8) searches the table H for a node with variable index i and low-, high-branches l, h and returns a matching node if one exists. Otherwise it creates a new node u , inserts it into H and returns the identity of it. The running time of MK is $O(1)$ due to the assumptions on the basic operations on T and H . The OBDD is ensured to be reduced if nodes are only created through the use of MK . In describing MK and subsequent algorithms, we make use of the notation $[T, H]$ to indicate that MK depends on the global data structures T and H , but we leave out the arguments when invoking it as part of other algorithms.

4.2 Build

The construction of an ROBDD from a given Boolean expression t proceeds as in the construction of an if-then-else normal form (INF) in section 2. An ordering of the variables $x_1 < \dots < x_n$ is fixed. Using the Shannon expansion $t = x_1 \rightarrow t[1/x_1], t[0/x_1]$, a node for t is constructed by a call to MK , after the nodes for $t[0/x_1]$ and $t[1/x_1]$ have been constructed by recursion. The algorithm is shown in figure 9. The call $\text{BUILD}'(t, i)$ constructs an ROBDD for a Boolean expression t with variables in $\{x_i, x_{i+1}, \dots, x_n\}$. It does so by first recursively constructing ROBDDs v_0 and v_1 for $t[0/x_i]$ and $t[1/x_i]$ in lines 4 and 5, and then proceeding to find the identity of the node for t in line 6. Notice that if v_0 and v_1 are identical, or if there already is a node with the same i , v_0 and v_1 , no new node is created.

An example of using BUILD to compute an ROBDD is shown in figure 10. The running time of BUILD is bad. It is easy to see that for a variable ordering with n variables there will always be generated on the order of 2^n calls .

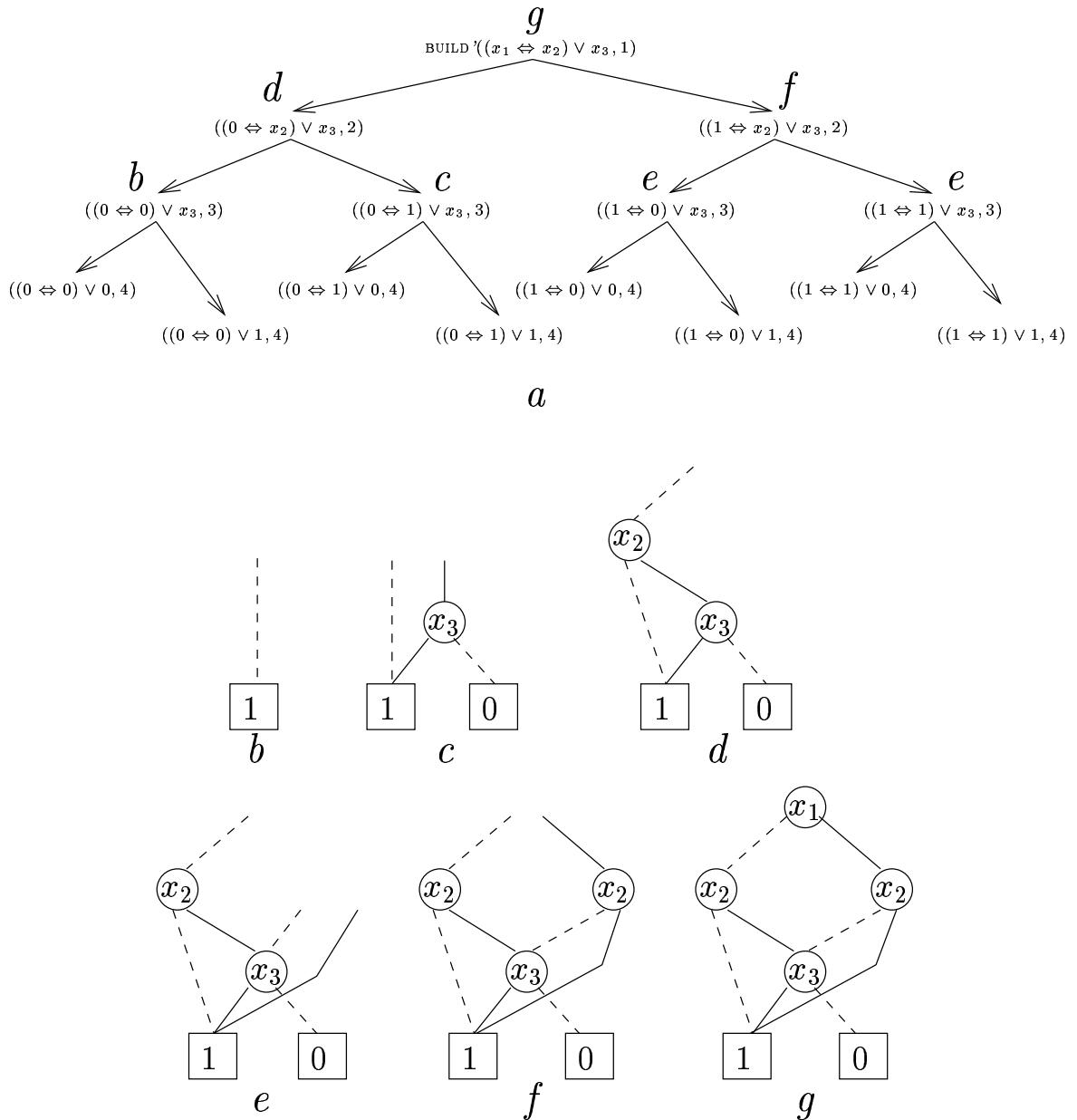


Figure 10: Using `BUILD` on the expression $(x_1 \Leftrightarrow x_2) \vee x_3$. (a) The tree of calls to `BUILD`. (b) The ROBDD after the call `BUILD'((0 \Leftrightarrow 0) \vee x_3, 3)`. (c) After the call `BUILD'((0 \Leftrightarrow 1) \vee x_3, 3)`. (d) After the call `BUILD'((0 \Leftrightarrow x_2) \vee x_3, 2)`. (e) After the calls `BUILD'((1 \Leftrightarrow 0) \vee x_3, 3)` and `BUILD'((1 \Leftrightarrow 1) \vee x_3, 3)`. (f) After the call `BUILD'((1 \Leftrightarrow x_2) \vee x_3, 2)`. (g) The final result.

4.3 Apply

```

APPLY[ $T, H$ ]( $op, u_1, u_2$ )
1:  $init(G)$ 
2:
3: function APP( $u_1, u_2$ ) =
4:   if  $G(u_1, u_2) \neq empty$  then return  $G(u_1, u_2)$ 
5:   else if  $u_1 \in \{0, 1\}$  and  $u_2 \in \{0, 1\}$  then  $u \leftarrow op(u_1, u_2)$ 
6:   else if  $var(u_1) = var(u_2)$  then
7:      $u \leftarrow MK(var(u_1), APP(low(u_1), low(u_2)), APP(high(u_1), high(u_2)))$ 
8:   else if  $var(u_1) < var(u_2)$  then
9:      $u \leftarrow MK(var(u_1), APP(low(u_1), u_2), APP(high(u_1), u_2))$ 
10:  else (*  $var(u_1) > var(u_2)$  *)
11:     $u \leftarrow MK(var(u_2), APP(u_1, low(u_2)), APP(u_1, high(u_2)))$ 
12:   $G(u_1, u_2) \leftarrow u$ 
13:  return  $u$ 
14: end APP
15:
16: return APP( $u_1, u_2$ )

```

Figure 11: The algorithm $APPLY[T, H](op, u_1, u_2)$.

All the binary Boolean operators on ROBDDs are implemented by the same general algorithm $APPLY(op, u_1, u_2)$ that for two ROBDDs computes the ROBDD for the Boolean expression $t^{u_1} op t^{u_2}$. The construction of $APPLY$ is based on the Shannon expansion (2):

$$t = x \rightarrow t[1/x], t[0/x].$$

Observe that for all Boolean operators op the following holds:

$$(x \rightarrow t_1, t_2) op (x \rightarrow t'_1, t'_2) = x \rightarrow t_1 op t'_1, t_2 op t'_2 \quad (4)$$

If we start from the root of the two ROBDDs we can construct the ROBDD of the result by recursively constructing the low- and the high-branches and then form the new root from these. Again, to ensure that the result is reduced, we create the node through a call to MK . Moreover, to avoid an exponential blow-up of recursive calls, *dynamic programming* is used. The algorithm is shown in figure 11.

Dynamic programming is implemented using a table of results G . Each entry (i, j) is either *empty* or contains the earlier computed result of $APP(i, j)$. The algorithm distinguishes between four different cases, the first of them handles the situation where both arguments are terminal nodes, the remaining three handle the situations where at least one argument is a variable node.

If both u_1 and u_2 are terminal, a new *terminal node* is computed having the value of op applied to the two truth values. (Recall, that terminal node 0 is represented by a node with identity 0 and similarly for 1.)

If at least one of u_1 and u_2 are non-terminal, we proceed according to the variable index. If the nodes have the same index, the two low-branches are paired and APP recursively computed on them. Similarly for the high-branches. This corresponds exactly to the case shown in equation (4). If they have different indices, we proceed by pairing the node with lowest index with the low- and high-branches of the other. This corresponds to the equation

$$(x_i \rightarrow t_1, t_2) \text{ op } t = x_i \rightarrow t_1 \text{ op } t, t_2 \text{ op } t \quad (5)$$

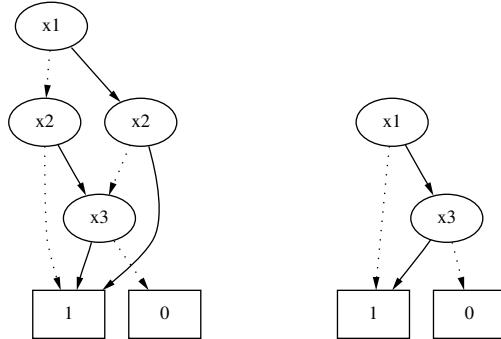
which holds for all t . Since we have taken the index of the terminals to be one larger than the index of the non-terminals, the last two cases, $\text{var}(u_1) < \text{var}(u_2)$ and $\text{var}(u_1) > \text{var}(u_2)$, take account of the situations where one of the nodes is a terminal.

Figure 12 shows an example of applying the algorithm on two small ROBDDs. Notice how pairs of nodes from the two ROBDDs are combined and computed.

To analyze the complexity of APPLY we let $|u|$ denote the number of nodes that can be reached from u in the ROBDD. Assume that G can be implemented with constant lookup and insertion times. (See section 5 for details on how to achieve this.) Due to the dynamic programming at most $|u_1||u_2|$ calls to APPLY are generated. Each call takes constant time. The total running time is therefore $O(|u_1||u_2|)$.

4.4 Restrict

The next operation we consider is the *restriction* of a ROBDD u . That is, given a truth assignment, for example $[0/x_3, 1/x_5, 1/x_6]$, we want to compute the ROBDD for t^u under this restriction, i.e., find the ROBDD for $t^u[0/x_3, 1/x_5, 1/x_6]$. As an example consider the ROBDD of figure 10(g) (repeated below to the left) representing the Boolean expression $(x_1 \Leftrightarrow x_2) \vee x_3$. Restricting it with respect to the truth assignment $[0/x_2]$ yields an ROBDD for $(\neg x_1 \vee x_3)$. It is constructed by replacing each occurrence of a node with label x_2 by its left branch yielding the ROBDD at the right:



The algorithm again uses MK to ensure that the resulting OBDD is reduced. Figure 13 shows the algorithm in the case where only singleton truth assignments ($[b/x_j]$, $b \in \{0, 1\}$) are allowed. Intuitively, in computing $\text{RESTRICT}(u, j, b)$ we search for all nodes with $\text{var} = j$ and replace them by their low- or high-son depending on b . Since this might force nodes above the point of replacement to become equal, it is followed by a reduction (through the calls to MK). Due to the two recursive calls in line 3, the algorithm has an exponential running time, see exercise 4.7 for an improvement that reduces this to linear time.

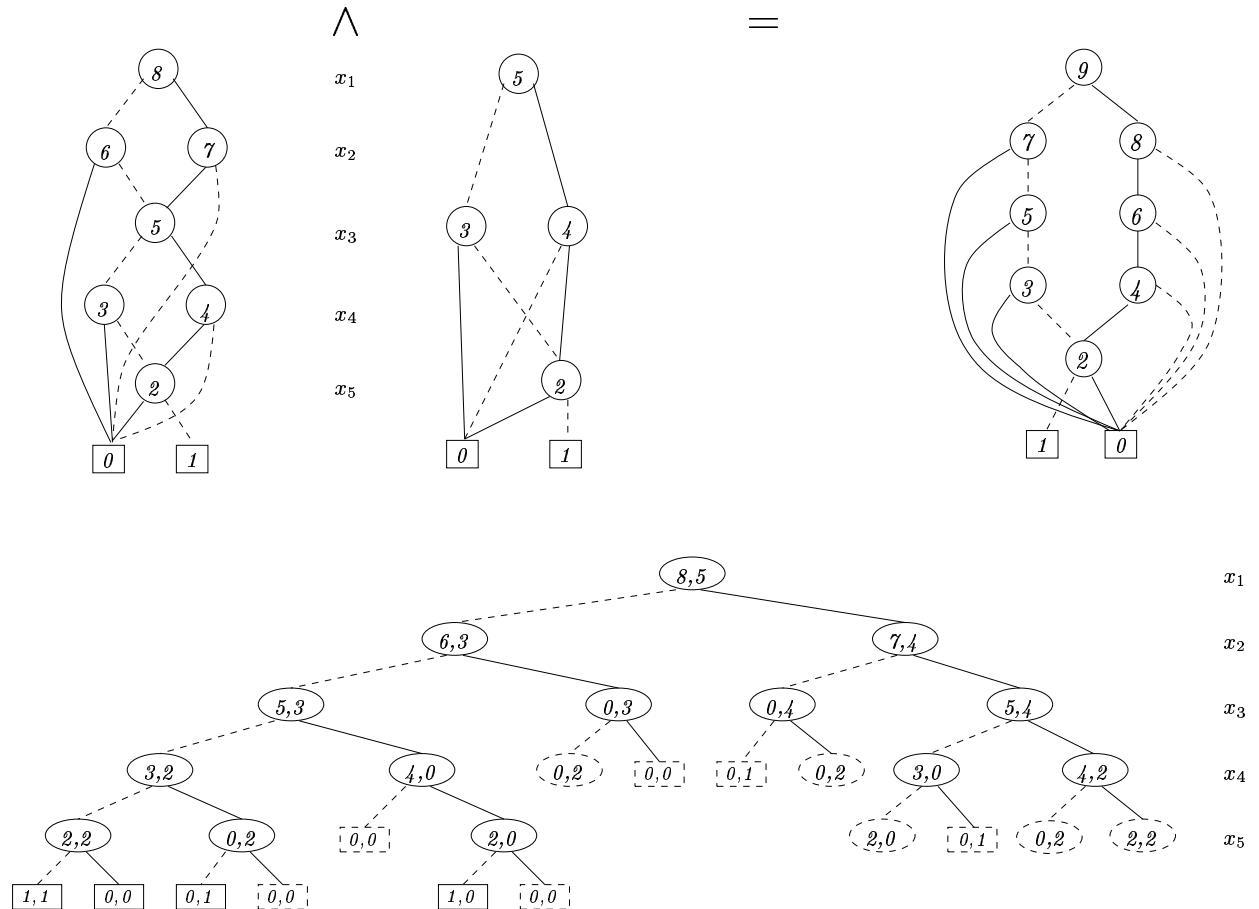


Figure 12: An example of applying the algorithm `APPLY` for computing the conjunction of the two ROBDDs shown at the top left. The result is shown to the right. Below the tree of arguments to the recursive calls of `APP`. Dashed nodes indicate that the value of the node has previously been computed and is not recomputed due to the use of dynamic programming. The solid ellipses show calls that finishes by a call to `MK` with the variable index indicated by the variables to the right of the tree.

```

RESTRICT[ $T, H](u, j, b) =$ 
1: function  $res(u) =$ 
2:   if  $var(u) > j$  then return  $u$ 
3:   else if  $var(u) < j$  then return  $MK(var(u), res(low(u)), res(high(u)))$ 
4:   else (*  $var(u) = j$  *) if  $b = 0$  then return  $res(low(u))$ 
5:   else (*  $var(u) = j, b = 1$  *) return  $res(high(u))$ 
6: end  $res$ 
7: return  $res(u)$ 

```

Figure 13: The algorithm $RESTRICT[T, H](u, j, b)$ which computes an ROBDD for $t^u[j/b]$.

4.5 SatCount, AnySat, AllSat

In this section we consider operations to examine the set of satisfying truth assignments of a node u . A truth assignment ρ satisfies a node u if $t^u[\rho]$ can be evaluated to 1 using the truth tables of the Boolean operators. Formally, the satisfying truth assignments is the set $sat(u)$:

$$sat(u) = \{\rho \in \mathbb{B}^{\{x_1, \dots, x_n\}} \mid t^u[\rho] \text{ is true}\},$$

where $\mathbb{B}^{\{x_1, \dots, x_n\}}$ denotes the set of all truth assignments for variables $\{x_1, \dots, x_n\}$, i.e., functions from $\{x_1, \dots, x_n\}$ to the truth values $\mathbb{B} = \{0, 1\}$. The first algorithm, SAT-COUNT, computes the size of $sat(u)$, see figure 14. The algorithm exploits the following fact. If u is a node with variable index $var(u)$ then two sets of truth assignments can make f^u true. The first set has var_u equal to 0, the other has var_u equal to 1. For the first set, the number is found by finding the number of truth assignments $count(low(u))$ making $low(u)$ true. All variables between $var(u)$ and $var(low(u))$ in the ordering can be chosen arbitrarily, therefore in the case of var_u being 0, a total of $2^{var(low(u))-var(u)-1} * count(low(u))$ satisfying truth assignments exists. To be efficient, dynamic programming should be applied in SATCOUNT (see exercise 4.10).

The next algorithm ANYSAT in figure 15 finds a satisfying truth assignment. Some irrelevant variables present in the ordering might not appear in the result and they can be assigned any value whatsoever. ANYSAT simply finds a path leading to 1 by a depth-first traversal, preferring somewhat arbitrarily low-edges over high-edges. It is particularly simple due to the observation that *if a node is not the terminal 0, it has at least one path leading to 1*. The running time is clearly linear in the result.

ALLSAT in figure 16 finds all satisfying truth-assignments leaving out irrelevant variables from the ordering. ALLSAT(u) finds all paths from a node u to the terminal 1. The running time is linear in the size of the result multiplied with the time to add the single assignments $[x_{var(u)} \mapsto 0]$ and $[x_{var(u)} \mapsto 1]$ in front of a list of up to n elements. However, the result can be exponentially large in $|u|$, so the running time is the poor $O(2^{|u|}n)$.

```

SATCOUNT[ $T$ ]( $u$ )
1: function  $count(u)$ 
2:   if  $u = 0$  then  $res \leftarrow 0$ 
3:   else if  $u = 1$  then  $res \leftarrow 1$ 
4:   else  $res \leftarrow 2^{var(low(u))-var(u)-1} * count(low(u))$ 
      +  $2^{var(high(u))-var(u)-1} * count(high(u))$ 
5:   return  $res$ 
6: end  $count$ 
7:
8: return  $2^{var(u)-1} * count(u)$ 

```

Figure 14: An algorithm for determining the number of valid truth assignments. Recall, that the “variable index” var of 0 and 1 in the ROBDD representation is $n+1$ when the ordering contains n variables (numbered 1 through n). This means that $var(0)$ and $var(1)$ always gives $n+1$.

```

ANYSAT( $u$ )
1:   if  $u = 0$  then Error
2:   else if  $u = 1$  then return []
3:   else if  $low(u) = 0$  then return [ $x_{var(u)} \mapsto 1$ , ANYSAT( $high(u)$ )]
4:   else return [ $x_{var(u)} \mapsto 0$ , ANYSAT( $low(u)$ )]

```

Figure 15: An algorithm for returning a satisfying truth-assignment. The variables are assumed to be x_1, \dots, x_n ordered in this way.

```

ALLSAT( $u$ )
1:   if  $u = 0$  then return  $\langle \rangle$ 
2:   else if  $u = 1$  then return  $\langle [ ] \rangle$ 
3:   else return
4:      $\langle$  add [ $x_{var(u)} \mapsto 0$ ] in front of all
5:       truth-assignments in ALLSAT( $low(u)$ ),
6:     add [ $x_{var(u)} \mapsto 1$ ] in front of all
7:       truth-assignments in ALLSAT( $high(u)$ ) $\rangle$ 

```

Figure 16: An algorithm which returns all satisfying truth-assignments. The variables are assumed to be x_1, \dots, x_n ordered in this way. We use $\langle \dots \rangle$ to denote sequences of truth assignments. In particular, $\langle \rangle$ is the empty sequence of truth assignments, and $\langle [] \rangle$ is the sequence consisting of the single empty truth assignment.

```

SIMPLIFY( $d, u$ )
1: function  $sim(d, u)$ 
2:   if  $d = 0$  then return 0
3:   else if  $u \leq 1$  then return  $u$ 
4:   else if  $d = 1$  then
5:     return MK( $var(u)$ ,  $sim(d, low(u))$ ,  $sim(d, high(u))$ )
6:   else if  $var(d) = var(u)$  then
7:     if  $low(d) = 0$  then return  $sim(high(d), high(u))$ 
8:     else if  $high(d) = 0$  then return  $sim(low(d), low(u))$ 
9:     else return MK( $var(u)$ ,
10:                   $sim(low(d), low(u))$ ,
11:                   $sim(high(d), high(u))$ )
12:   else if  $var(d) < var(u)$  then
13:     return MK( $var(d)$ ,  $sim(low(d), u)$ ,  $sim(high(d), u)$ )
14:   else
15:     return MK( $var(u)$ ,  $sim(d, low(u))$ ,  $sim(d, high(u))$ )
16: end sim
17:
18: return  $sim(d, u)$ 

```

Figure 17: An algorithm (due to Coudert et al [CBM89]) for simplifying an ROBDD b that we only care about on the domain d . Dynamic programming should be applied to improve efficiency (exercise 4.12)

$\text{MK}(i, u_0, u_1)$	$O(1)$	
$\text{BUILD}(t)$	$O(2^n)$	
$\text{APPLY}(op, u_1, u_2)$	$O(u_1 u_2)$	
$\text{RESTRICT}(u, j, b)$	$O(u)$	See note
$\text{SATCOUNT}(u)$	$O(u)$	See note
$\text{ANYSAT}(u)$	$O(p)$	$p = \text{AnySat}(u), p = O(u)$
$\text{ALLSAT}(u)$	$O(r * n)$	$r = \text{AllSat}(u), r = O(2^{ u })$
$\text{SIMPLIFY}(d, u)$	$O(d u)$	See note

Note: These running times only holds if dynamic programming is used (exercises 4.7, 4.10, and 4.12).

Table 1: Worst-case running times for the ROBDD operations. The running times are the expected running times since they are all based on a hash-table with expected constant time search and insertion operations.

4.6 Simplify

The final algorithm called `SIMPLIFY` is shown in figure 17. The algorithm is used to simplify an ROBDD by trying to remove nodes. The simplification is based on a domain d of interest. The ROBDD u is supposed to be of interest only on truth assignments that also satisfy d . (This occurs when using ROBDDs for *formal verification*. Section 7 shows how to do formal verification with ROBDDs, but contains no example of using `SIMPLIFY`.)

To be precise, given d and u , `SIMPLIFY` finds another ROBDD u' , typically smaller than u , such that $t^d \wedge t^u = t^d \wedge t^{u'}$. It does so by trying to identify sons, and thereby making some nodes redundant. A more detailed analysis is left to the reader.

The running time of the algorithms of the previous sections is summarized in table 1.

4.7 Existential Quantification and Substitution

When applying ROBDDs often *existential quantification* and *composition* is used. Existential quantification is the Boolean operation $\exists x.t$. The meaning of an existential quantification of a Boolean variable is given by the following equation:

$$\exists x.t = t[0/x] \vee t[1/x]. \quad (6)$$

On ROBDDs existential quantification can therefore be implemented using two calls to `RESTRICT` and a single call to `APPLY`.

Composition is the ROBDD operation performing the equivalent of substitution on Boolean expression. Often the notation $t[t'/x]$ is used to describe the result of substituting all free occurrences of x in t by t' . (An occurrence of a variable is free if it is not within the scope of a quantifier.)¹ To perform this substitution on ROBDDs we observe the

¹Since ROBDDs contain no quantifiers we shall not be concerned with the problems of free variables of t' being bound by quantifiers of t .

following equation, which holds if t contains no quantifiers:

$$t[t'/x] = t[t' \rightarrow 1, 0/x] = t' \rightarrow t[1/x], t[0/x]. \quad (7)$$

Since $(t' \rightarrow t[1/x], t[0/x]) = (t' \wedge t[1/x]) \vee (\neg t' \wedge t[0/x])$ we can compute this with two applications of RESTRICT and three applications of APPLY (with the operators \wedge , (\neg) , \wedge , \vee). However, by essentially generalizing APPLY to operators op with three arguments we can do better (see exercise 4.13).

Exercises

Exercise 4.1 Construct the ROBDD for $\neg x_1 \wedge (x_2 \Leftrightarrow \neg x_3)$ with ordering $x_1 < x_2 < x_3$ using the algorithm BUILD in figure 9.

Exercise 4.2 Show the representation of the ROBDD of figure 6 in the style of figure 7.

Exercise 4.3 Suggest an improvement $BUILDCONJ(t)$ of BUILD which generates only a linear number of calls for Boolean expressions t that are conjunctions of variables and negations of variables.

Exercise 4.4 Construct the ROBDDs for x and $x \Rightarrow y$ using whatever ordering you want. Compute the disjunction of the two ROBDDs using APPLY.

Exercise 4.5 Construct the ROBDDs for $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$ using BUILD with the ordering $x_1 < x_2 < x_3$. Use APPLY to find the ROBDD for $\neg(x_1 \wedge x_3) \vee (x_2 \wedge x_3)$.

Exercise 4.6 Is there any essential difference in running time between finding $RESTRICT(b, 1, 0)$ and $RESTRICT(b, n, 0)$ when the variable ordering is $x_1 < x_2 < \dots < x_n$?

Exercise 4.7 Use dynamic programming to improve the running time of RESTRICT.

Exercise 4.8 Generalise RESTRICT to arbitrary truth assignments $[x_{i_1} = b_{i_1}, x_{i_2} = b_{i_2}, \dots, x_{i_n} = b_{i_n}]$. It might be convenient to assume that $x_{i_1} < x_{i_2} < \dots < x_{i_n}$.

Exercise 4.9 Suggest a substantially better way of building ROBDDs for (large) Boolean expressions than BUILD.

Exercise 4.10 Change SATCOUNT such that dynamic programming is used. How does this change the running time?

Exercise 4.11 Explain why dynamic programming does not help in improving the running time of ALLSAT.

Exercise 4.12 Improve the efficiency of SIMPLIFY with dynamic programming.

Exercise 4.13 Write the algorithm $COMPOSE(u_1, x, u_2)$ for computing the ROBDD of $u_1[u_2/x]$ efficiently along the lines of APPLY. First generalize APPLY to operators op with three arguments (as for example the if-then-else operator), utilizing once again the Shannon expansion. Then use equation 7 to write the algorithm.

5 Implementing the ROBDD operations

There are many choices that have to be taken in implementing the ROBDD operations. There is no obvious best way of doing it. This section gives hints for some reasonable solutions.

First, the node table T is an array as shown in figure 7. The only problem is that the size of the array is not known until the full BDD has been constructed. Either a fixed upper bound could be assumed, or other tricks must be applied (for example *dynamic arrays* [CLR90, sec. 18.4]). The table H could be implemented as a hash-table using for instance the hash function

$$h(i, v_0, v_1) = \text{pair}(i, \text{pair}(v_0, v_1)) \bmod m$$

where pair is a pairing function that maps pairs of natural numbers to natural numbers and m is a prime. One choice for the pairing function is

$$\text{pair}(i, j) = \frac{(i + j)(i + j + 1)}{2} + i$$

which is a bijection, and therefore “perfect”: it produces no collisions. As usual with hash-tables we have to decide on the size as a prime m . However, since the size of H grows dynamically it can be hard to find a good choice for m . One solution would be to take m very large, for example $m = 15485863$ (which is the 1000000'th prime number), and then take as the hashing function

$$h'(i, v_0, v_1) = h(i, v_0, v_1) \bmod 2^k$$

using a table of size 2^k . Starting from some reasonable small value of k we could increase the table when it contains 2^k elements by adding one to k , construct a new table and rehash all elements into this new table. (Again, see for example [CLR90, sec. 18.4] for details.) For such a dynamic hash-table the amortized, expected cost of each operation is still $O(1)$.

The table G used in `APPLY` could be implemented as a two-dimensional array. However, it turns out to be very sparsely used – especially if we succeed in getting small ROBDDs – and it is better to use a hash-table for it. The hashing function used could be $g(v_0, v_1) = \text{pair}(v_0, v_1) \bmod m$ and as for H a dynamic hash-table could be used.

6 Examples of problem solving with ROBDDs

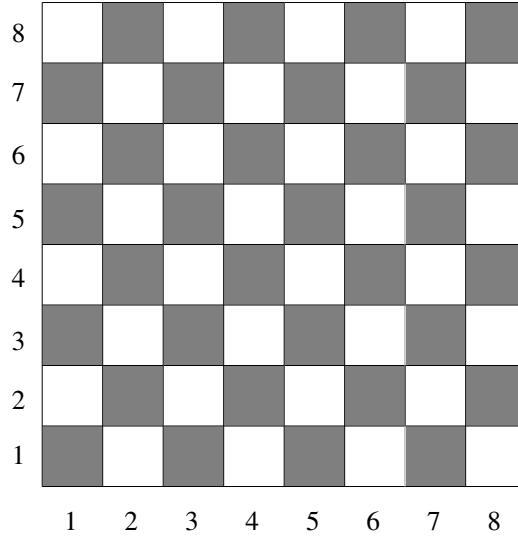
This section will describe various examples of problems that can be solved with an ROBDD-package. The examples are *not* chosen to illustrate when ROBDDs are the best choice, but simply chosen to illustrate the scope of potential applications.

6.1 The 8 Queens problem

A classical chess-board problem is the *8 queens problem*: Is it possible to place 8 queens on a chess board so that no queen can be captured by another queen? To be a bit more

general we could ask the question for arbitrary N : Is it possible to place N queens safely on a $N \times N$ chess board?

To solve the problem using ROBDDs we must encode it using Boolean variables. We do this by introducing a variable for each position on the board. We name the variables as x_{ij} , $1 \leq i, j \leq N$ where i is the row and j is the column. A variable will be 1 if a queen is placed on the corresponding position.



The capturing rules for queens require that no other queen can be positioned on the same row, column, or any of the diagonals. This we can express as Boolean expressions: For all i, j ,

$$\begin{aligned} x_{ij} &\Rightarrow \bigwedge_{1 \leq l \leq N, l \neq j} \neg x_{il} \\ x_{ij} &\Rightarrow \bigwedge_{1 \leq k \leq N, k \neq i} \neg x_{kj} \\ x_{ij} &\Rightarrow \bigwedge_{1 \leq k \leq N, 1 \leq j+k-i \leq N, k \neq i} \neg x_{k,j+k-i} \\ x_{ij} &\Rightarrow \bigwedge_{1 \leq k \leq N, 1 \leq j+i-k \leq N, k \neq i} \neg x_{k,j+i-k} \end{aligned}$$

Moreover, there must be a queen in each row: For all i ,

$$x_{i1} \vee x_{i2} \vee \cdots \vee x_{iN}$$

Taking the conjunction of all the above requirements, we get a predicate $Sol_N(\vec{x})$ true at exactly the configurations that are solutions to the N queens problem.

Exercise 6.1 (8 Queens Problem) Write a program that can find an ROBDD for $Sol_N(\vec{x})$ when given N as input. Make a table of the number of solutions to the N queens problem for $N = 1, 2, 3, 4, 5, 6, 7, 8, \dots$. When there is a solution, give one.

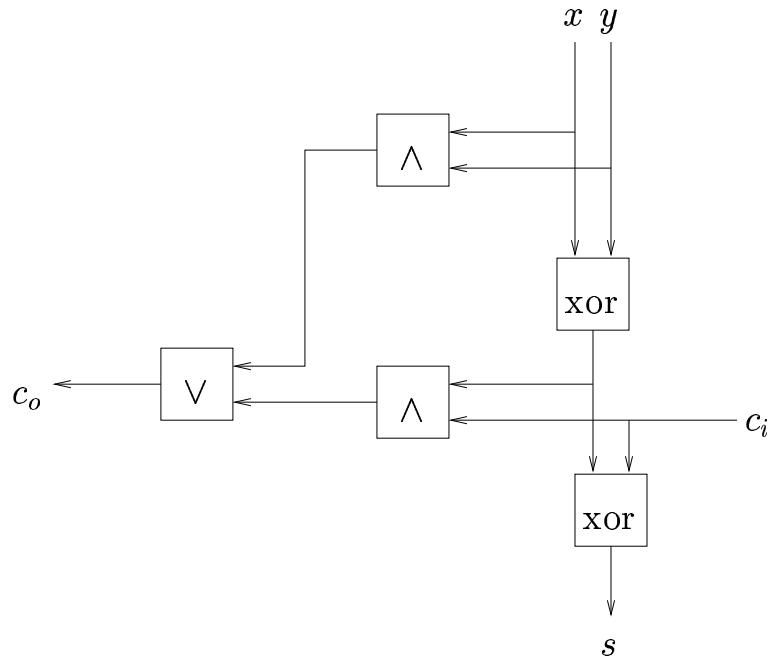


Figure 18: A full-adder

6.2 Correctness of Combinational Circuits

A *full-adder* takes as arguments two bits x and y and an incoming carry bit c_i . It produces as output a sum bit s and an outgoing carry bit c_o . The requirement is that $2 * c_o + s = x + y + c_i$, in other words c_o is the most significant bit of the sum of x , y , and c_i , and s the least significant bit. The requirement can be written down as a table for c_o and a table for s in terms of values of x , y , and c_i . From such a table it is easy to write down a DNF for c_o and s .

At the normal level of abstraction a combinational circuit is nothing else than a Boolean expression. It can be represented as an ROBDD, using `BUILD` to construct the trivial ROBDDs for the inputs and using a call to `APPLY` for each gate.

Exercise 6.2 Find DNFs for c_o and s . Verify that the circuit in figure 18 implements a one bit full-adder using the ROBDD-package and the DNFs.

6.3 Equivalence of Combinational Circuits

As above we can construct an ROBDD from a combinational circuit and use the ROBDDs to show properties. For instance, the equivalence with other circuits.

Exercise 6.3 Verify that the two circuits in figure 19 are *not* equivalent using ROBDDs. Find an input that returns different outputs in the two circuits.

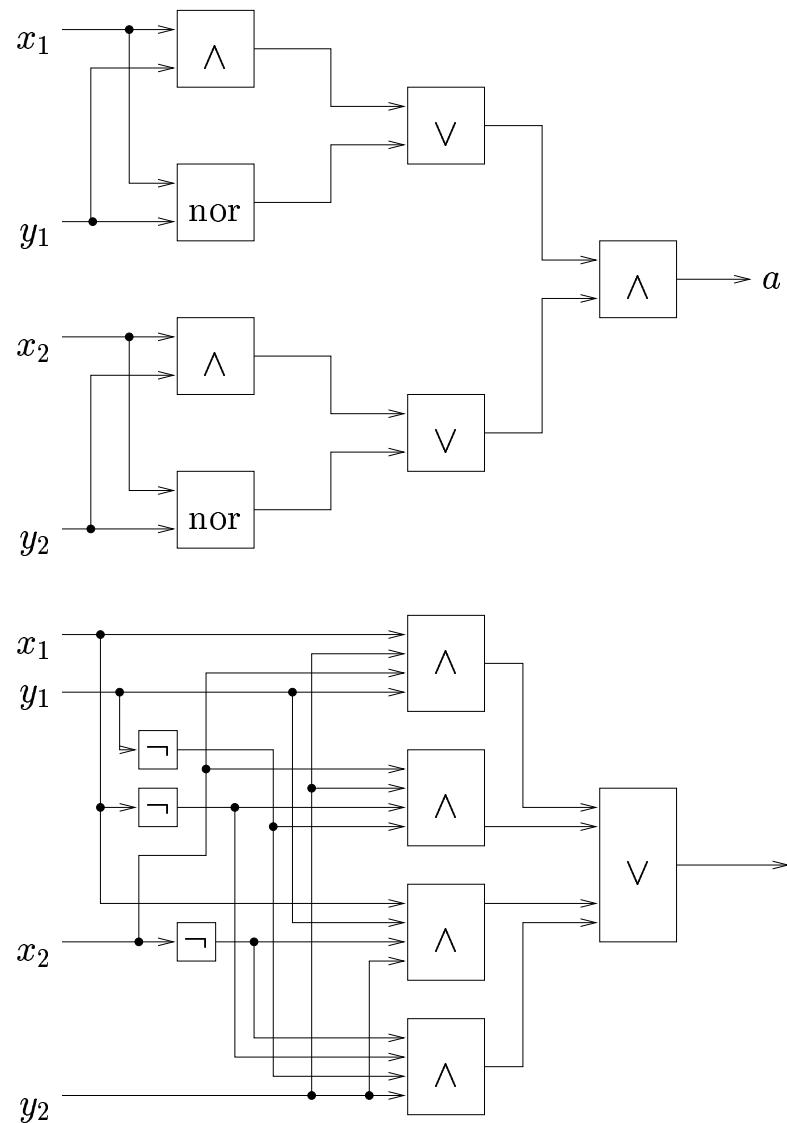


Figure 19: Two circuits used in exercise 6.3

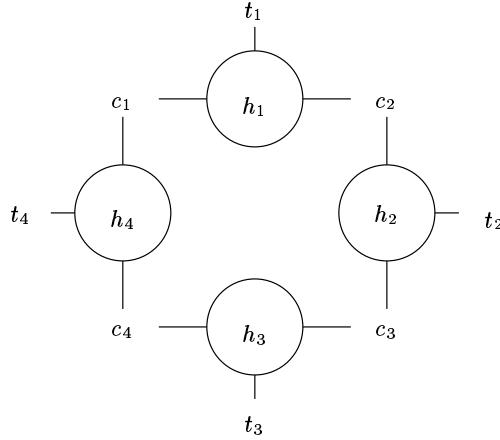


Figure 20: Milner’s Scheduler with 4 cyclers. The token is passed clockwise from c_1 to c_2 to c_3 to c_4 and back to c_1

7 Verification with ROBDDs

One of the major uses of ROBDDs is in *formal verification*. In formal verification a model of a system M is given together with some properties P supposed to hold for the system. The task is to determine whether indeed M satisfy P . The approach we take, in which we shall use an algorithm to answer the satisfaction problem, is often called *model checking*.

We shall look at a concrete example called *Milner’s Scheduler* (taken from Milner’s book [Mil89]). The model consists of N *cyclers*, connected in a ring, that co-operates on starting and detecting termination of N tasks that are not further described. The scheduler must make sure that the N tasks are always started in order but they are allowed to terminate in any order. This is one of the properties that has to be shown to hold for the model. The cyclers try to fulfill this by passing a token: the holder of the token is the only process allowed to start its task.

All cyclers are similar except that one of them has the token in the initial state. The cyclers cyc_i , $1 \leq i \leq N$ are described in a state-based fashion as small transition systems over the Boolean variables t_i , h_i , and c_i . The variable t_i is 1 when task i is running and 0 when it is terminated; h_i is 1 when cycler i has a token, 0 otherwise; c_i is 1 when cycler $i - 1$ has put down the token and cycler i not yet picked it up. Hence a cycler starts a task by changing t_i from 0 to 1, and detects its termination when t_i is again changed back to 0; and it picks up the token by changing c_i from 1 to 0 and puts it down by changing c_{i+1} from 0 to 1. The behaviour of cycler i is described by two transitions:

```

if  $c_i = 1 \wedge t_i = 0$  then  $t_i, c_i, h_i := 1, 0, 1$ 
if  $h_i = 1$  then  $c_{(i \bmod N)+1}, h_i := 1, 0$ 

```

The meaning of a transition “**if condition then assignment**” is that, if the *condition* is true in some state, then the system can evolve to a new state performing the (parallel) *assignment*. Hence, if the system is in a state where c_i is 1 and t_i is 0 then we can simultaneously set t_i to 1, c_i to 0 and h_i to 1.

The transitions are encoded by a single predicate over the value of the variables before the transitions (the *pre-state*) and the values after the transition (the *post-state*). The variables in the pre-state are the $t_i, h_i, c_i, 1 \leq i \leq N$ which we shall collectively refer to as \vec{x} and in the post-state $t'_i, h'_i, c'_i, 1 \leq i \leq N$, which we shall refer to as \vec{x}' . Each transition is an atomic action that excludes any other action. Therefore in the encoding we shall often have to say that a lot of variables are unchanged. Assume that S is a subset of the unprimed variables \vec{x} . We shall use a predicate $unchanged_S$ over \vec{x}, \vec{x}' which ensures that all variables in S are unchanged. It is defined as follows:

$$unchanged_S =_{\text{def}} \bigwedge_{x \in S} x = x'.$$

It is slightly more convenient to use the predicate $assigned_{S'} = unchanged_{\vec{x} \setminus S'}$ which express that every variable *not* in S' is unchanged. We can now define P_i , the transitions of cyller i over the variables \vec{x}, \vec{x}' as follows:

$$\begin{aligned} P_i =_{\text{def}} & (c_i \wedge \neg t_i \wedge t'_i \wedge \neg c'_i \wedge h'_i \wedge assigned_{\{c_i, t_i, h_i\}}) \\ \vee & (h_i \wedge c'_{(i \bmod N)+1} \wedge \neg h'_i \wedge assigned_{\{c_{(i \bmod N)+1}, h_i\}}) \end{aligned}$$

The signalling of termination of task i , by changing t_i from 1 to 0 performed by the environment is modeled by N transitions $E_i, 1 \leq i \leq N$:

$$E_i =_{\text{def}} t_i \wedge \neg t'_i \wedge assigned_{\{t_i\}},$$

expressing the transitions **if** $t_i = 1$ **then** $t_i := 0$. Now, at any given state the system can perform one of the transitions from one of the P_i 's or the E_i 's, i.e., all possible transitions are given by the predicate T :

$$T =_{\text{def}} P_1 \vee \cdots \vee P_n \vee E_1 \vee \cdots \vee E_n.$$

In the initial state we assume that all tasks are stopped, no cyller has a token and only place 1 (c_1) has a token. Hence the initial state can be characterized by the predicate I over the unprimed variables \vec{x} given by:

$$I =_{\text{def}} \neg \vec{t} \wedge \neg \vec{h} \wedge c_1 \wedge \neg c_2 \wedge \cdots \wedge \neg c_N.$$

(Here \neg applied to a vector \vec{t} means the conjunction of \neg applied to each coordinate t_i .) The predicates describing Milner's Scheduler are summarized in figure 21.

Within this setup we could start asking a lot of questions. For example,

1. Can we find a predicate R over the unprimed variables characterizing exactly the states that can be reached from I ? R is called the set of *reachable states*.
2. How many reachable states are there?
3. Is it the case that in all reachable states only one token is present?
4. Is task t_i always only started after t_{i-1} ?

$$\begin{aligned}
\text{unchanged}_S &=_{\text{def}} \bigwedge_{x \in S} x = x' \\
\text{assigned}_{S'} &=_{\text{def}} \text{unchanged}_{\vec{x} \setminus S'} \\
P_i &=_{\text{def}} (c_i \wedge \neg t_i \wedge t'_i \wedge \neg c'_i \wedge h'_i \wedge \text{assigned}_{c_i, t_i, h_i}) \\
&\quad \vee (h_i \wedge c'_{i \bmod N+1} \wedge \neg h'_i \wedge \text{assigned}_{c_{i \bmod N+1}, h_i}) \\
E_i &=_{\text{def}} t_i \wedge \neg t'_i \wedge \text{assigned}_{t_i} \\
T &=_{\text{def}} \bigvee_{1 \leq i \leq N} P_i \vee E_i \\
I &=_{\text{def}} \neg \vec{t} \wedge \neg \vec{h} \wedge c_1 \wedge \neg c_2 \wedge \dots \wedge \neg c_N
\end{aligned}$$

Figure 21: Milner’s Scheduler as described by the transition predicate T and the initial-state predicate I .

5. Does Milner’s Scheduler possess a deadlock? I.e., is there a reachable state in which no transitions can be taken?

To answer these questions we first have to compute R . Intuitively, R must be the set of states that either satisfy I (are initial) or within a finite number of T transitions can be reached from I . This suggest an iterative algorithm for computing R as an increasing chain of approximations $R^0, R^1, \dots, R^k, \dots$ Step k of the algorithm find states that with less than k transitions can be reached from I . Hence, we take $R^0 = 0$ the constantly false predicate and compute R^{k+1} as the disjunction of I and the set of states which from one transition of T can be reached from R^k . Figure 22 illustrates the computation of R .

How do we compute this with ROBDDs? We start with the ROBDD $R = \boxed{0}$. At any point in the computation the next approximation is computed by the disjunction of I and T composed with the previous approximation R' . We are done when the current and the previous approximations coincide:

```

REACHABLE-STATES( $I, T, \vec{x}, \vec{x}'$ )
1:    $R \leftarrow \boxed{0}$ 
2:   repeat
3:      $R' \leftarrow R$ 
4:      $R \leftarrow I \vee (\exists \vec{x}. T \wedge R)[\vec{x}/\vec{x}']$ 
5:   until  $R' = R$ 
6:   return  $R$ 

```

7.1 Knights tour

Using the same encoding of a chess board as in section 6.1, letting $x_{ij} = 1$ denote the presence of a Knight at position (i, j) we can solve other problems. We can encode moves of a Knight as transitions. For each position, 8 moves are possible if they stay on the board. A Knight at (i, j) can be moved to any one of $(i \pm 1, j \pm 2), (i \pm 2, j \pm 1)$ assuming they are vacant and within the board boundary. For all i, j and k, l with $1 \leq k, l \leq N$ and $(k, l) \in \{(i \pm 1, j \pm 2), (i \pm 2, j \pm 1)\}$:

$$M_{ij,kl} =_{\text{def}} x_{ij} \wedge \neg x_{kl} \wedge \neg x'_{ij} \wedge x'_{kl} \wedge \bigwedge_{(i',j') \notin \{(i,j),(k,l)\}} x_{i'j'} = x'_{i'j'} .$$

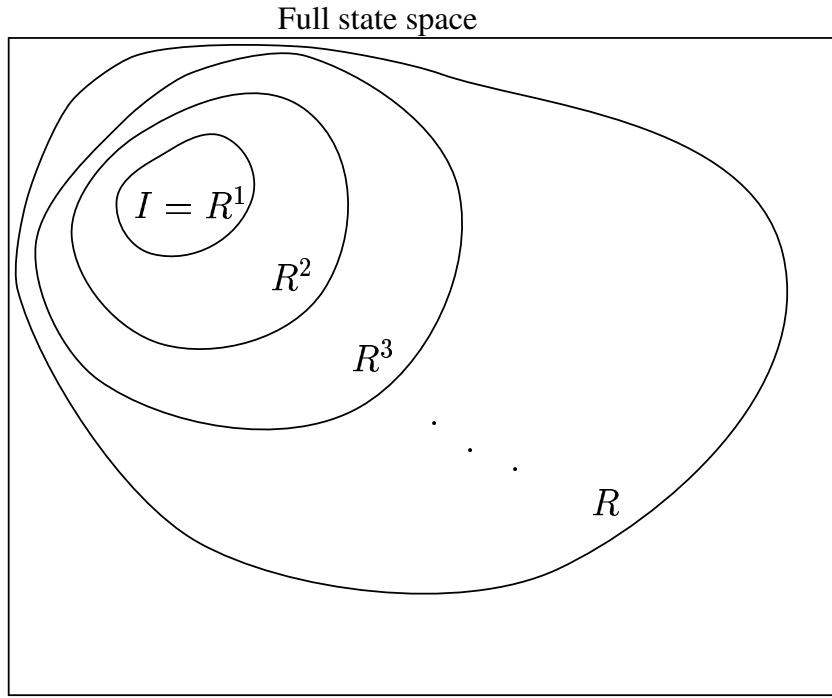


Figure 22: Sketch of computation of the reachable states

Hence, the transitions are given as the predicate $T(\vec{x}, \vec{x}')$:

$$T(\vec{x}, \vec{x}') =_{\text{def}} \bigvee_{\substack{1 \leq i, j, k, l \leq N, (k, l) \in \{(i \pm 1, j \pm 2), (i \pm 2, j \pm 1)\}}} M_{ij,kl}$$

Exercise 7.1 (Knight's tour) Write a program to solve the following problem using the ROBDD-package: Is it possible for a Knight, positioned at the lower left corner to visit all positions on an $N \times N$ board? (*Hint:* Compute iteratively all the positions that can be reached by the Knight.) Try it for various N .

Exercise 7.2 Why does the algorithm REACHABLE-STATES always terminate?

Exercise 7.3 In this exercise we shall work with Milner's Scheduler for $N = 4$. It is by far be the most convenient to solve the exercise by using an implementation of an ROBDD package.

- a) Find the reachable states as an ROBDD R .
- b) Find the number of reachable states.
- c) Show that in all reachable states at most one token is present on any of the placeholders c_1, \dots, c_N by formulating a suitable property P and prove that $R \Rightarrow P$.

- d) Show that in all reachable states Milner's Scheduler can always perform a transition, i.e., it does not possess a *deadlock*.

Exercise 7.4 Complete the above exercise by showing that the tasks are always started in sequence $1, 2, \dots, N, 1, 2 \dots$

Exercise 7.5 Write a program that given an N as input computes the reachable states of Milner's Scheduler with N cyclers. The program should write out the number of reachable states (using SATCOUNT). Run the program for $N = 2, 4, 6, 8, 10, \dots$. Measure the running times and draw a graph showing the measurements as a function of N . What is the asymptotic running time of your program?

8 Project: An ROBDD Package

This project implements a small package of ROBDD-operations. The full package should contain the following operations:

Init(n)

Initialize the package. Use n variables numbered 1 through n .

Print(u)

Print a representation of the ROBDD on the standard output. Useful for debugging.

Mk(i, l, h)

Return the number u of a node with $\text{var}(u) = i$, $\text{low}(u) = l$, $\text{high}(u) = h$. This could be an existing node, or a newly created node. The reducedness of the ROBDD should not be violated.

Build(t)

Construct an ROBDD from a Boolean expression. You could restrict yourself to the expressions x or $\neg x$ or finite conjunctions of these. (Why?)

Apply(op, u_1, u_2)

Construct the ROBDD resulting from applying op on u_1 and u_2 .

Restrict(u, j, b)

Restrict the ROBDD u according to the truth assignment $[b/x_j]$.

SatCount(u)

Return the number of elements in the set $\text{sat}(u)$. (Use a type that can contain very large numbers such as floating point numbers.)

AnySat(u)

Return a satisfying truth assignment for u

Sub-project 1

Implement the tables T and H with their operations listed in section 4. On top of these implement the operations INIT(n), PRINT(u), and MK(i, l, h).

Sub-project 2

Continue implementation of the package by adding the operations $\text{BUILD}(t)$ and $\text{APPLY}(op, u_1, u_2)$.

Sub-project 3

Finish your implementation of the package by adding $\text{RESTRICT}(u, j, b)$, $\text{SATCOUNT}(u)$, and $\text{ANYSAT}(u)$.

References

- [AH97] Henrik Reif Andersen and Henrik Hulgaard. Boolean expression diagrams. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 88–98, Warsaw, Poland, June 29–July 2 1997. IEEE Computer Society.
- [Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 8(C-35):677–691, 1986.
- [Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
- [CBM89] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems. Proceedings*, volume 407 of *LNCS*, pages 365–373. Springer-Verlag, 1989.
- [CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
- [Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.
- [Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

Exercises Lecture 3

Introduction to Artificial Intelligence (IAI)

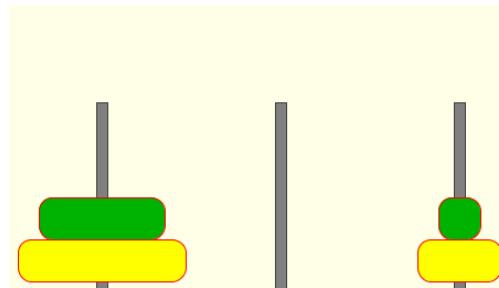
Exercise 1

The Towers of Hanoi is a mathematical game or puzzle. It consists of three pegs, and n disks of different sizes which can slide onto any peg. The puzzle starts with the disks neatly stacked in order of size on the left-most peg, smallest at the top, thus making a conical shape.

The objective of the game is to move the entire stack to the right-most peg, obeying the following rules:

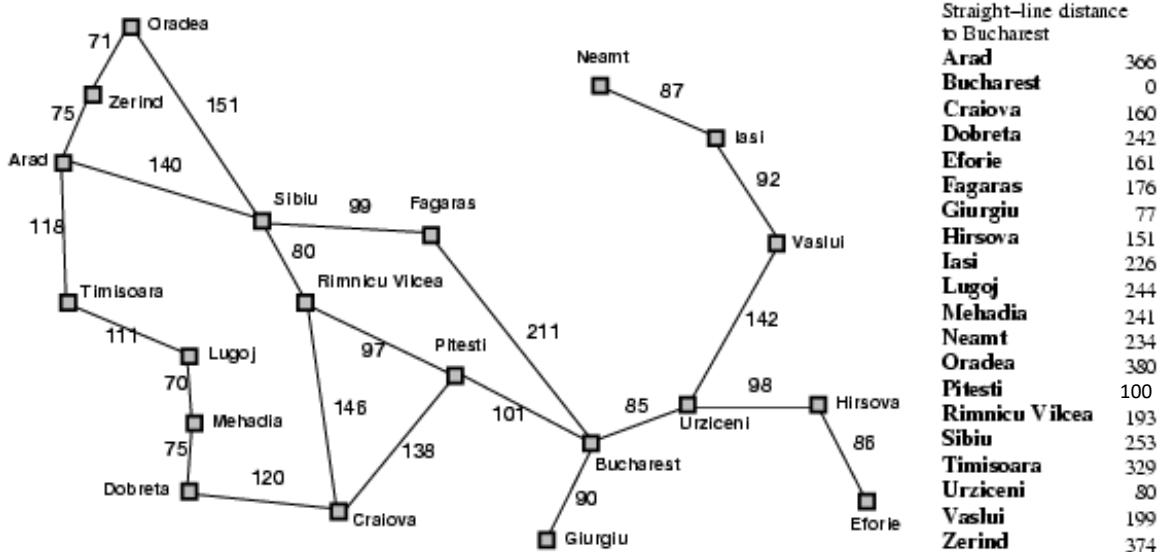
1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the pegs and sliding it onto another peg, on top of the other disks that may already be present on that peg.
3. No disk may be placed on top of a smaller disk.

As an example, the figure below shows a legal state of the Towers of Hanoi with 4 disks. From this state it is possible to move the smallest disk to the first or second peg.



- a) Define the Towers of Hanoi as a search problem. That is, define a representation of states and actions, and define
 - the initial state,
 - the actions function,
 - the result function
 - the is-goal function, and
 - the action cost function (assuming that solutions with minimum steps are optimal).
- b) Using your state-space definition from a), draw the state-space graph of Towers of Hanoi with $n=2$. Recall that vertices in this graph are legal states and edges are actions causing a change from source to target state.
- c) If we assume that all legal configurations of the discs are reachable, what is then the size of the state space as a function of n ? (This question may vary in difficulty depending on your representation. If you can't find an exact number then try to find a good upper bound)

Exercise 2



Consider A* using the straight-line distance heuristic $h_{SLD}(n)$ to find a shortest path from Lugoj (L) to Bucharest (B).

- Finish the table below showing a possible content of the frontier queue in each iteration (in iteration 0, no nodes have been expanded; in iteration 1, one node has been expanded, etc.). Represent a frontier node by a 4-tuple $\langle f, g, h, s \rangle$, where the state s is given by the first letter of the city it represents.

Iteration	Frontier
0	$\langle 244, 0, 244, L \rangle$
1	$\langle 311, 70, 241, M \rangle, \langle 440, 111, 329, T \rangle$
...	...

- What solution is returned by A* after finishing the last iteration in your table?
- What path does this solution correspond to?

Exercise 3 (challenge)

Let h be a consistent heuristic function.

- Assume that the goal is reachable from n and let $p^*(n)$ denote an optimal solution path from n to a goal state. Prove that $h(n) \leq h^*(n)$ by induction on the number of edges in $p^*(n)$.
- Use the result in a) to prove that h is admissible.

Mandatory Assignment

Let WEIGHTED A* denote A* using the evaluation function $f(n) = (1-w)g(n) + wh(n)$, where $0 \leq w \leq 1$ and $h(n)$ is admissible.

- a) Which algorithm does WEIGHTED A* correspond to with $w = 1$ and $w = 0.5$, respectively?
- b) For which values of w are WEIGHTED A* optimal (Assuming that A* only is optimal if it uses an admissible heuristic)?

Solutions to Exercises - Lecture 3

Introduction to Artificial Intelligence

Exercise 1

a)

We represent each of n disks with a number from $S = \{1, \dots, n\}$ assuming that smaller disks are identified with smaller numbers, i.e., the i -th disk is smaller than the j -th disk if $i < j$. A *state* is defined with a partition of S into three sets (S_1, S_2, S_3) (i.e., $\cup_{i=1}^3 S_i = S$, and $S_i \cap S_j = \emptyset$ for all $i \neq j$) where a set S_k represents disks that are on the k -th peg. Once a set S_k is given, we know exactly how the disks are put on the k -th peg, since there is only one way to order them (smaller on top). In particular, the topmost disk in set S_k is $\min\{a \mid a \in S_k\}$, which we denote simply as $\min S_k$.

We have six *actions* $M(1, 2), M(1, 3), M(2, 1), M(2, 3), M(3, 1), M(3, 2)$, where $M(i, j)$ denotes an action of moving a smallest disk from the i -th peg, and putting it on the j -th peg. For each state only some of the actions might be legal, i.e., the topmost disk on the i -th peg must be smaller than topmost disk on the j -th peg ($\min S_i < \min S_j$). The *Initial state* is given by $S_1 = \{1, \dots, n\}$, $S_2 = \emptyset$, $S_3 = \emptyset$.

The *actions function* for each state (S_1, S_2, S_3) returns all legal action-state pairs $(M(i, j), (S'_1, S'_2, S'_3))$, where the action $M(i, j)$ is legal. For each such action $M(i, j)$, in the *results function* gives the resulting state in which the two affected pegs are $S'_i = S_i \setminus \{\min S_i\}$, $S'_j = S_j \cup \{\min S_i\}$ while the remaining one stays the same.

Goal test is given by checking whether $S_3 = \{1, 2, \dots, n\}$. Path cost is a positive constant, for example $c(s, a, s') = 1$.

b)

The state space is shown in Figure 1.

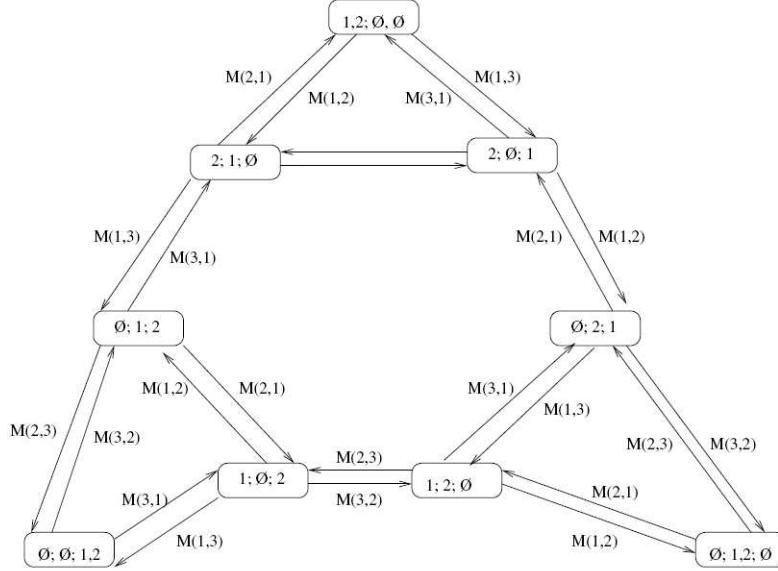


Figure 1: State space of Towers of Hanoi with $n = 2$. \emptyset denotes a peg without disks.

c)

Notice the difference between legal states (i.e., all states satisfying our definition of a state) and reachable states (i.e., states we can reach by executing some sequence of actions from the starting state). Reachable states are in general a subset of legal states. Since we are assuming all legal states are reachable, we need only to count the number of states (S_1, S_2, S_3) that satisfy our definition from a). The number of all legal configurations is 3^n since for every disk we can choose any of the three pegs to put it on.

Exercise 2

a)

The nodes in the frontier of the A*-algorithm when solving the specified problem are given in the table below. Notice that the frontier is a priority queue, and that the nodes in it are therefore ordered in accordance to their f value.

Iteration	Frontier
0	$\langle 244, 0, 244, L \rangle$
1	$\langle 311, 70, 241, M \rangle, \langle 440, 111, 329, T \rangle$
2	$\langle 387, 145, 242, D \rangle, \langle 440, 111, 329, T \rangle$
3	$\langle 425, 265, 160, C \rangle, \langle 440, 111, 329, T \rangle$
4	$\langle 440, 111, 329, T \rangle, \langle 503, 403, 100, P \rangle, \langle 604, 411, 193, RV \rangle$
5	$\langle 503, 403, 100, P \rangle, \langle 595, 229, 366, A \rangle, \langle 604, 411, 193, RV \rangle$
6	$\langle 504, 504, 0, B \rangle, \langle 595, 229, 366, A \rangle, \langle 604, 411, 193, RV \rangle$

b)

A^* returns an expanded goal node. In this case it is the node $\langle 504, 504, 0, B \rangle$.

c)

The solution path can be found by tracing the parent pointers of the goal node. In this case: $L \rightarrow M \rightarrow D \rightarrow C \rightarrow P \rightarrow B$.

Exercise 3 (Challenge)

a)

Let k denote the number of edges in $p^*(n)$, $k = |p^*(n)|$.

- If $k = 0$, a state n is a goal state s_G . Therefore $h(n) = 0$ according to the definition of heuristic function $h(n)$. But also, $h^*(n) = 0$ since obviously the cheapest path to goal has length 0. Therefore it holds $h(n) \leq h^*(n)$.
- Assume that the claim holds for paths with k edges. Let us show that the statement also holds for paths with $k+1$ edges. Let $p^*(n) = \{n, n_1, \dots, n_{k+1}\}$ be the optimal path with $k+1$ edges where n_{k+1} is the goal state. A cost of each edge $c(n_i, n_{i+1})$ is the standard cost of executing an action from a state n_i that leads to a state n_{i+1} .

Since the statement holds for paths with k edges, it also holds for state n_1 . Namely, the optimal path from n_1 to goal n_{k+1} , $p_1^*(n_1) = \{n_1, \dots, n_{k+1}\}$, has k edges. Therefore:

$$h(n_1) \leq h^*(n_1) \quad (1)$$

Also, since $p^*(n_1)$ is the optimal path from n_1 , its length $\sum_{i=1}^k c(n_i, n_{i+1})$ is equal to $h^*(n_1)$. Since the similar holds for $h^*(n)$ it follows:

$$h^*(n) = c(n, n_1) + h^*(n_1) \quad (2)$$

Now we have everything we need to prove $h(n) \leq h^*(n)$. Since, h is consistent heuristic it holds $h(n) \leq c(n, n_1) + h(n_1) \leq^{(1)} c(n, n_1) + h^*(n_1) =^{(2)} h^*(n)$. This proves the induction step.

- Since the statement holds for $k = 0$, and from the fact that if the statement holds for k we can show that it also holds for $k+1$, according to the principle of mathematical induction, the statement holds for all $k \in \mathbb{N}$.

b)

Assume that h is not admissible. Then for some state n_0 it holds $h(n_0) > h^*(n_0)$. However, relation $>$ makes sense only if both $h(n_0)$ and $h^*(n_0)$ are finite. But this means that a goal is reachable from n_0 in finite number of steps, i.e., there is an optimal path $p^*(n_0)$. Then according to a) it must hold $h(n_0) \leq h^*(n_0)$ which contradicts the initial assumption. Therefore h is admissible.

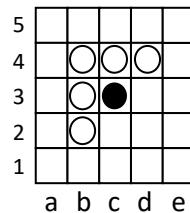
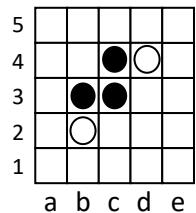
Exercises Lecture 4

Introduction to Artificial Intelligence (IAI)

Exercise 1

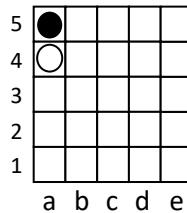
Consider a board game similar to Othello. There are two players, one with black pieces and one with white pieces. The players take turns and in each turn, a player places a single piece on a square of a quadratic board. A piece can only be placed next to a piece that is vertically or horizontally adjacent to a piece already on the board. If the placement of a piece causes some of the opponent's pieces to lie between opposite colored pieces either in the vertical or horizontal direction, they are “captured” and shift color. The game can be terminated at any time, and the utility of a game state for a player equals the number of pieces of the player's color.¹ Notice that this game is not zero-sum, but we ignore that.

As an example, in the game state shown below to the left, white can place a piece in b4, c5, d5, e4, d3, c2, b1, a2, and a3. If white places a piece in b4, the resulting game state is the one shown below to the right.



The utility for black is 3 for the left state and 1 for the right.

Assume that MAX plays black and consider the initial state shown below



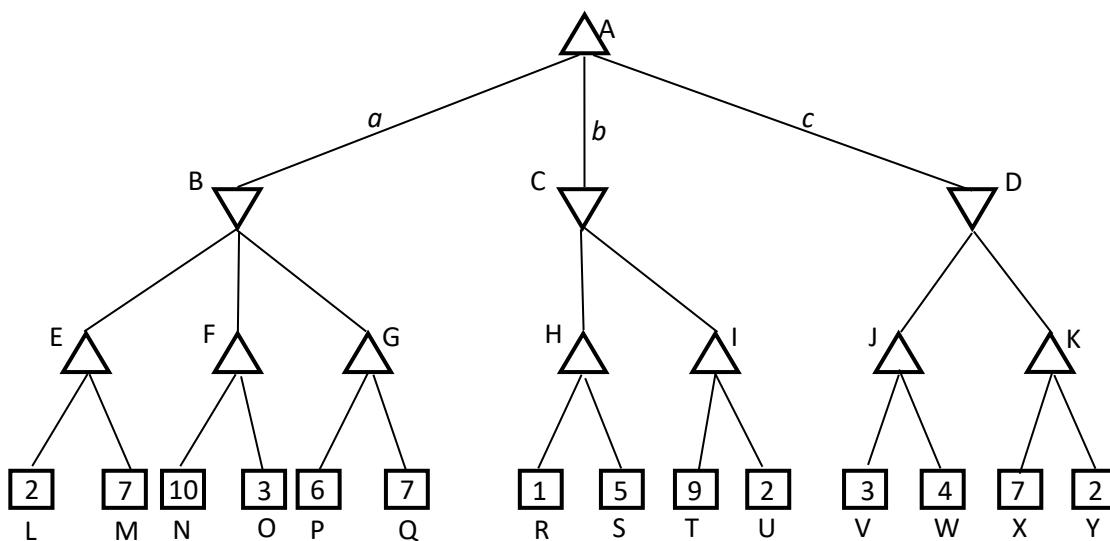
- Draw the game tree from the initial state to a depth of 2 ply (i.e., a single move of MAX and a single move of MIN).
- What is the minimax value of the initial state?
- What is the minimax decision of MAX?

¹ In real Othello, you can also place a new piece next to a piece already on the board on a diagonal. However, you can only place a piece if it captures one of the opponent's pieces (otherwise you lose your turn). A capture, though, can happen on the diagonal.

Exercise 2

(Adapted from exercise 12.2, Nils J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, 1998).

Consider the following game tree in which the static scores (numbers in leaf boxes) are all from the MAX player's point of view.



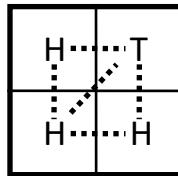
- What action should the MAX player choose?
- What nodes would not need to be examined using Alpha-Beta-Search – assuming that nodes are examined in the left-to-right order?

Exercise 3 (adapted from RN10 exercise 5.7)

Argue that the utility obtained by MAX using MINIMAX decisions against a suboptimal MIN will never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a *suboptimal* strategy against a suboptimal MIN?

Mandatory assignment

Consider a simplified tic-tac-toe game with four positions as shown below.



Assume that MAX and MIN play with circle and cross tokens, respectively. The game is played using a coin with 50% chance of landing on each side. The rules are as follows:

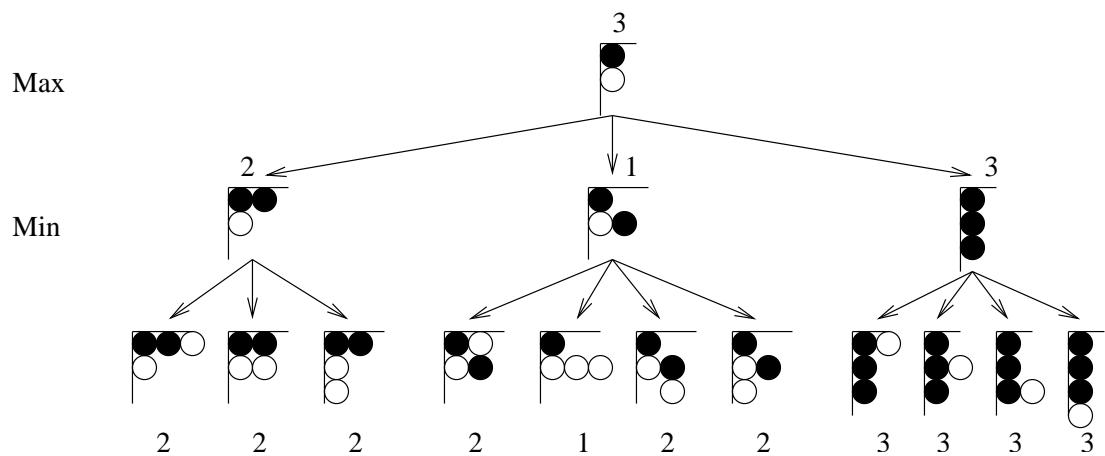
1. Initially the board is empty.
 2. A player starts his turn by flipping the coin.
 3. If the coin shows heads, the player can place a token on positions marked H, otherwise the player must place a token on the position marked T.
 4. The game terminates when
 - a. a player is unable to place a token, because there are no empty positions left with the right mark, or
 - b. a player wins the game.
 5. A player wins the game, if the player has two tokens on any of the 5 pair of positions connected by a dashed line.
 6. The utility of the game is 1 if MAX wins, -1 if Min wins, and 0 otherwise.
- 1) Modify the pseudocode of the MINIMAX algorithm (RN21 Fig. 6.3, also in slides) to the EXPECTIMINIMAX algorithm, and write its complete pseudo code.
[hint: you can add a CHANCE-VALUE function, but notice that its behavior depends on whether it is called from a MAX-VALUE or MIN-VALUE function]
- 2) Assume as usual that MAX starts the game and that MAX just flipped the coin and got heads.
- a. Carefully draw the complete game tree on an A3 paper.
 - b. Write the EXPECTIMINIMAX value for each node in the tree.
 - c. What is the EXPECTIMINIMAX decision for MAX?

Solution to Exercises - Week 4

Introduction to Artificial Intelligence

Exercise 1

a)



b)

3

c)

To place a black in a3

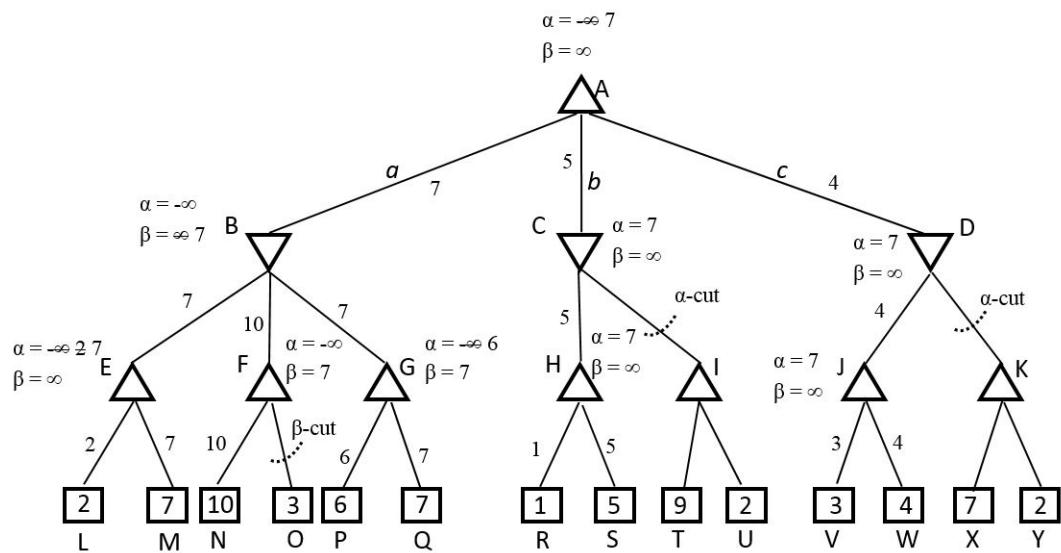
Exercise 2

a)

a

b)

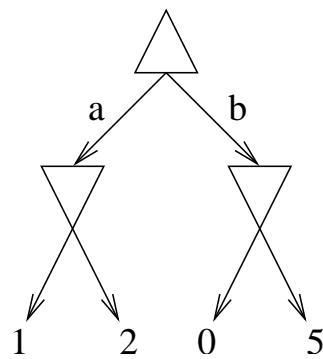
O, T, U, X, Y



Exercise 3

If MAX plays against a suboptimal MIN, the value returned by each MIN node will be larger than or equal to its MINVALUE. But then since MAX nodes maximizes all node values in the game tree will be larger than or equal to the MINIMAXVALUE. Thus, the utility obtained by MAX when playing against a suboptimal MIN can never be lower than the utility obtained playing against an optimal MIN, which is the MINIMAXVALUE.

Assume MIN is helping MAX and actually chooses children with maximum utility, then MAX in the game tree below should rather choose action *b* instead of the MINIMAXDECISION *a*.



Exercises Lecture 5

Introduction to Artificial Intelligence (IAI)

Exercise 1 (adapted from RN13 3.27)

n vehicles occupy squares $(1, 1)$ through $(n, 1)$ (i.e., the bottom row) of an $n \times n$ grid. The vehicles must be moved to the top row but in reverse order; so the vehicle i that starts in $(i, 1)$ must end up in $(n - i + 1, n)$. On each time step, every one of the n vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.

- a) Calculate the size of the state space as a function of n .
- b) Calculate the branching factor as a function of n .
- c) Suppose that vehicle i is at (x_i, y_i) ; write a nontrivial admissible heuristic h_i for the number of moves it will require to get to its goal location $(n - i + 1, n)$, assuming no other vehicles are on the grid.
- d) Which of the following heuristics are admissible for the problem of moving all n vehicles to their destination? Explain.
 - i) $\sum_{i=1}^n h_i$.
 - ii) $\max\{h_1, \dots, h_n\}$.
 - iii) $\min\{h_1, \dots, h_n\}$.

Exercise 2 (adapted from RN13 3.31)

We have defined relaxations of the 8-puzzle in which:

- a) A tile can move from square A to square B if A is adjacent to B.
- b) A tile can move from square A to square B if B is blank.
- c) A tile can move from square A to square B.

Relaxation a) corresponds to the sum of Manhattan distances heuristic (h_2) and relaxation c) corresponds to the number of misplaced tiles heuristic (h_1). Relaxation b) corresponds to a heuristic called Gaschnig's heuristic (h_G).

- a) Explain why h_G is at least as accurate as h_1 .
- b) Show cases where h_G is more accurate than both h_1 and h_2 .
- c) Explain how to calculate h_G efficiently.

Solutions Exercises Lecture 5

Introduction to Artificial Intelligence (IAI)

Exercise 1 (adapted from RN13 3.27)

n vehicles occupy squares $(1, 1)$ through $(n, 1)$ (i.e., the bottom row) of an $n \times n$ grid. The vehicles must be moved to the top row but in reverse order; so the vehicle i that starts in $(i, 1)$ must end up in $(n - i + 1, n)$. On each time step, every one of the n vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.

- a) Calculate the size of the state space as a function of n .

Since each vehicle is unique and since it is possible to move any vehicle to any position (i.e., all other vehicles can stay put), the reachable state space must equal all the different ways n different vehicles can be placed on an $n \times n$ grid. We have n^2 possible positions for the first vehicle, $n^2 - 1$ possible position for the second vehicle and so on. In total $\frac{(n^2)!}{(n^2-n)!}$.

- b) Calculate the branching factor as a function of n .

Since each vehicle in each step can do one out of five actions independently, the worst case branching factor is 5^n .

- c) Suppose that vehicle i is at (x_i, y_i) ; write a nontrivial admissible heuristic h_i for the number of moves it will require to get to its goal location $(n - i + 1, n)$, assuming no other vehicles are on the grid.

Since the vehicle cannot jump over other vehicles, it at least needs the Manhattan distance to reach its goal. That is $h_i = |(n - i + 1) - x_i| + |n - y_i|$.

- d) Which of the following heuristics are admissible for the problem of moving all n vehicles to their destination? Explain.

i) $\sum_{i=1}^n h_i$.

This heuristic is not admissible. It ignores that vehicles can move in parallel. E.g., for $n = 3$, consider the example shown below:

3		
	2	1

The heuristic value is 2, but it only requires one parallel move to get to the goal state.

- ii) $\max\{h_1, \dots, h_n\}$.

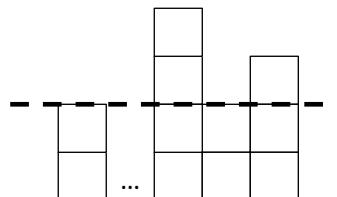
This heuristic is not admissible. It ignores that vehicles can jump over each other. E.g., for $n = 3$, consider the example shown below:

	2	3
		1

The heuristic value is 2, since 3 has a Manhattan distance of 2 to its goal position, but it only requires one parallel move to get to the goal state.

- iii) $\min\{h_1, \dots, h_n\}$.

This heuristic is admissible. Notice that in order for one vehicle to jump, another must stay put. In this way, a vehicle j that jumps over a vehicle i saves one move, while i must make an extra move. Thus, the moves can be transferred between vehicles but not eliminated. The least number of parallel moves is achieved by transferring moves from vehicles that must move far to vehicles that are close to their goal position. In some state, we get a distribution of Manhattan distances as shown below, where the dashed line indicates the minimum Manhattan distance of some vehicle.



Evening out the moves to achieve the minimal number of parallel steps can only increase this minimum Manhattan distance. This shows that the heuristic is admissible.

Exercise 2 (adapted from RN13 3.31)

We have defined relaxations of the 8-puzzle in which:

- a) A tile can move from square A to square B if A is adjacent to B.
- b) A tile can move from square A to square B if B is blank.
- c) A tile can move from square A to square B.

Relaxation a) corresponds to the sum of Manhattan distances heuristic (h_2) and relaxation c) corresponds to the number of misplaced tiles heuristic (h_1). Relaxation b) corresponds to a heuristic called Gaschnig's heuristic (h_G).

- a) Explain why h_G is at least as accurate as h_1 .

h_G is only equal to h_1 if there is a sequence, where the blank space exactly appear at the goal position of each misplaced tile. In all other cases, h_G is larger than h_1 .

- b) Show cases where h_G is more accurate than both h_1 and h_2 .
 $h_1 = h_2 = 2$, but $h_G = 3$ in the example below, since 5 or 6 must move to the blank space before the swap in the position of 5 and 6 can happen.

1	2	3
4	6	5
7	8	

- c) Explain how to calculate h_G efficiently.

Count the number of moves of the following algorithm:

Repeat until all tiles are at their goal position

- if the blank is at the bottom right position then move a misplaced tile to its position
- move the misplaced that has the blank as goal position, to the goal position

Exercises Lecture 6

Introduction to Artificial Intelligence (IAI)

Exercise 1

Consider the claims

1. If the sun shines then it is true that I get wet if it rains.
 2. If the sun shines then it is summer.
 3. It is not summer
 4. Therefore, I get wet if the sun shines.
- a) A proposition symbol represents something that can be either true or false. In claim 1., 2., 3. and 4. above, there are several statements like “the sun shines” that can be either true or false. Define a propositional symbol for each of these statements.
 - b) Translate 1., 2., 3. , and 4. to propositional logic sentences using your proposition symbols from a).
 - c) Does the conjunction of 1., 2. and 3. entail the conclusion in 4 (why / why not)?

Exercise 2 (adapted from RN03 7.8)

Decide whether each of the following sentences is valid, unsatisfiable, or neither. Verify your decisions using truth tables or equivalence rules.

- a) $\text{Smoke} \Rightarrow \text{Smoke}$
- b) $\text{Smoke} \Rightarrow \text{Fire}$
- c) $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow (\neg \text{Smoke} \Rightarrow \neg \text{Fire})$
- d) $\text{Smoke} \vee \text{Fire} \vee \neg \text{Fire}$
- e) $((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire}) \Leftrightarrow ((\text{Smoke} \Rightarrow \text{Fire}) \vee (\text{Heat} \Rightarrow \text{Fire}))$
- f) $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow ((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire})$
- g) $\text{Big} \vee \text{Dumb} \vee (\text{Big} \Rightarrow \text{Dumb})$
- h) $(\text{Big} \wedge \text{Dumb}) \vee \neg \text{Dumb}$

Exercise 3 (adapted from RN03 7.4)

Argue for the correctness of each of the following assertions:

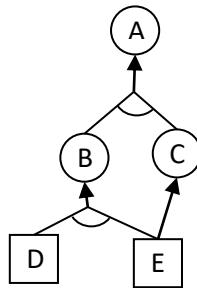
- a) α is valid if and only if $\text{True} \vDash \alpha$.
- b) For any α , $\text{False} \vDash \alpha$.
- c) $\alpha \vDash \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.
- d) $\alpha \equiv \beta$ if and only if the sentence $(\alpha \Leftrightarrow \beta)$ is valid.
- e) $\alpha \vDash \beta$ if and only if the sentence $(\alpha \wedge \neg \beta)$ is unsatisfiable.

Exercise 4

- a) Translate the sentence $\neg(a \Rightarrow b) \wedge (a \vee b \Leftrightarrow (c \Rightarrow b))$ to CNF
- b) Use resolution to prove $\neg(a \Rightarrow b) \wedge (a \vee b \Leftrightarrow (c \Rightarrow b)) \models \neg c$

Mandatory assignment

During the lecture 4 you learned that the forward-chaining algorithm can be used to efficiently entail propositions from a knowledge-base of definite clauses. In the textbook, a different approach called backward-chaining is introduced. This algorithm works backward from the query. If the query is known to be true, then no work is needed, otherwise it tries to prove the query by proving its premises true (by backward-chaining). Consider the following AND-OR graph of a knowledge-base KB :



If we were to query the KB for A using backward-chaining, we would first have to prove B and C . C can be proven because it is implied by E which is a known fact, and B can be proven because it is implied by D and E which are known facts. Since we have proven B and C we can conclude A .

- 1) In which situations can it be an advantage to do backward-chaining rather than forward-chaining?
- 2) Professor Smart has implemented the following backward-chaining algorithm:

```
function PL-BC-ENTAILS(KB, q) returns true or false
    inputs: KB, the knowledge base, a set of propositional definite clauses
            q, the query, a propositional symbol
    if q is known to be true in KB then return true
    for each clause c in KB where q is in c.CONCLUSION do
        if CHECK-ALL(KB, c.PREMISE) then return true
    return false
```

```
function CHECK-ALL(KB, premise) returns true or false
    inputs: KB, the knowledge base, a set of propositional definite clauses
            premise, a set of propositional symbols
    for each p in premise do
        if not PL-BC-Entails(KB, p) then return false
    return true
```

Given the following KB of definite clauses:

- $F \wedge E \Rightarrow D$
- $E \Rightarrow B$
- $B \wedge E \wedge G \Rightarrow C$
- $C \Rightarrow G$
- $D \wedge B \wedge C \Rightarrow A$
- $\Rightarrow F$
- $\Rightarrow E$

Can professor Smart's algorithm answer $KB \models A$? If yes, explain how the algorithm induces the answer. If no, explain in words where the problem lies and extend professor Smart's pseudocode with a solution.

- 3) The book mentions that an efficient implementation of a backward-chaining algorithm runs in linear time. Does professor Smart's algorithm run in linear time? If you extended professor Smart's algorithm as a part of the answer to 2), the question is if your extended algorithm runs in linear time. If yes, explain why. If no, explain why not and extend professor Smart's algorithm with improvements that reduces its runtime. It is not required that you come up with a linear time algorithm.

Solutions to Exercises - Week 6

Introduction to Artificial Intelligence

Exercise 1

- a) We define the proposition symbols $ss = \text{"The sun shines"}$, $r = \text{"It rains"}$, $w = \text{"I get wet"}$, $su = \text{"It is summer"}$.
- b)
1. $ss \Rightarrow (r \Rightarrow w)$.
 2. $ss \Rightarrow su$.
 3. $\neg su$.
 4. $ss \Rightarrow w$.
- c) We will show that 1., 2. and 3. entail 4 by showing that $1 \wedge 2 \wedge 3 \Rightarrow 4$ is valid:

$$\begin{aligned}
((ss \Rightarrow (r \Rightarrow w)) \wedge (ss \Rightarrow su) \wedge (\neg su)) &\Rightarrow (ss \Rightarrow w) \\
\equiv ((\neg ss \vee \neg r \vee w) \wedge (\neg ss \vee su) \wedge \neg su) &\Rightarrow (\neg ss \vee w) \\
\equiv \neg((\neg ss \vee \neg r \vee w) \wedge (\neg ss \vee su) \wedge \neg su) \vee (\neg ss \vee w) & \\
\equiv (ss \wedge r \wedge \neg w) \vee (ss \wedge \neg su) \vee su \vee (\neg ss \vee w) & \\
\equiv (ss \wedge r \wedge \neg w) \vee ((ss \vee su) \wedge (\neg su \vee su)) \vee (\neg ss \vee w) & \\
\equiv (ss \wedge r \wedge \neg w) \vee ((ss \vee su) \wedge \text{TRUE}) \vee (\neg ss \vee w) & \\
\equiv (ss \wedge r \wedge \neg w) \vee (ss \vee su) \vee (\neg ss \vee w) & \\
\equiv (ss \wedge r \wedge \neg w) \vee (ss \vee \neg ss) \vee su \vee w & \\
\equiv (ss \wedge r \wedge \neg w) \vee \text{TRUE} \vee su \vee w & \\
\equiv \text{TRUE}
\end{aligned}$$

Exercise 2

For space and time saving we use s for "Smoke", f for "Fire", h for "Heat" and so on.

a)

$$s \Rightarrow s \equiv \neg s \vee s \equiv \text{TRUE}.$$

Since TRUE is true in all models, $s \Rightarrow s$ is valid.

b)

$$s \Rightarrow f \equiv \neg s \vee f.$$

This expression evaluates to FALSE for $s = \text{TRUE}$ and $f = \text{FALSE}$, while it evaluates to TRUE for e.g. $s = f = \text{TRUE}$. I.e. $s \Rightarrow f$ is neither valid nor unsatisfiable.

c)

$$\begin{aligned}
 (s \Rightarrow f) \Rightarrow (\neg s \Rightarrow \neg f) &\equiv (\neg s \vee f) \Rightarrow (\neg f \vee s) \\
 &\equiv \neg(\neg s \vee f) \vee (\neg f \vee s) \\
 &\equiv (s \wedge \neg f) \vee (\neg f \vee s).
 \end{aligned}$$

For $s = \text{FALSE}$ and $f = \text{TRUE}$, the expression is false, while for $s = \text{TRUE}$ and $f = \text{FALSE}$ it is true. That is, $(s \Rightarrow f) \Rightarrow (\neg s \Rightarrow \neg f)$ is neither valid nor unsatisfiable.

d)

$$s \vee f \vee \neg f \equiv s \vee \text{TRUE} \equiv \text{TRUE},$$

i.e. the expression is valid.

e)

$$\begin{aligned}
 ((s \vee h) \Rightarrow f) \Leftrightarrow ((s \Rightarrow f) \vee (h \Rightarrow f)) &\equiv (\neg(s \wedge h) \vee f) \Leftrightarrow (\neg s \vee f \vee \neg h \vee f) \\
 &\equiv (\neg s \vee \neg h \vee f) \Leftrightarrow (\neg s \vee \neg h \vee f).
 \end{aligned}$$

Since the left hand side in the last expression is the exactly same as he right hand side of the last expression, the truth value of the left hand side will allways equal the truth value of the right hand side, so (compare with truth table of \Leftrightarrow) the expression is always true, i.e. it is valid.

f)

$$\begin{aligned}
 (s \Rightarrow f) \Rightarrow ((s \wedge h) \Rightarrow f) &\equiv (\neg s \vee f) \Rightarrow (\neg(s \wedge h) \vee f) \\
 &\equiv (\neg s \vee f) \Rightarrow (\neg s \vee \neg h \vee f) \\
 &\equiv \neg(\neg s \vee f) \vee (\neg s \vee \neg h \vee f) \\
 &\equiv \neg(\neg s \vee f) \vee (\neg s \vee f) \vee \neg h \\
 &\equiv \text{TRUE} \vee \neg h \\
 &\equiv \text{TRUE},
 \end{aligned}$$

i.e. the expression is valid.

g)

$$(b \wedge d) \vee \neg d.$$

For $b = \text{FALSE}$ and $d = \text{TRUE}$ the expression is false, while it is true for $b = d = \text{TRUE}$. Therefore it is neither valid nor unsatisfiable.

Exercise 3

- a) $\text{TRUE} \models \alpha$ is by definition the case if and only if α is true in every model where TRUE is true. Since TRUE is always true, we therefore have that $\text{TRUE} \models \alpha$ if and only if α is true in every model. This is by definition the case if and ony is α is valid.
- b) $\text{FALSE} \models \alpha$ is by definition the case if and only if α is true in every model where FALSE is true. Since FALSE never is true (i.e. it is true in no model), this gives us no constraints on the truth value on α , and therefore $\text{FALSE} \models \alpha$ is true for any α .

- c) $\alpha \models \beta$ if and only if in all models where α is true, β is also true (by definition).
 On the other hand $\alpha \Rightarrow \beta$ is valid if and only if $\alpha \Rightarrow \beta$ is true in all models, i.e. if and only if $\neg\alpha \vee \beta$ is true in all models. This is the case if and only if either α is false or β is true, so in order for $\alpha \Rightarrow \beta$ to be valid we see that in all model where α is true, β has to be true too.

From this we conclude, that $\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ is valid.

- d) $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$ (by definiton). So according to c), $\alpha \equiv \beta$ if and only if $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \alpha$. According to the definitin of " \Leftrightarrow ", we hereby get that $\alpha \equiv \beta$ if and ony if $\alpha \Leftrightarrow \beta$.
- e) Due to c) we have that $\alpha \models \beta$ if and only if $(\alpha \Rightarrow \beta)$ is valid. It is therefore enough to show that $\alpha \Rightarrow \beta$ is valid if and only if $(\alpha \wedge \neg\beta)$ is unsatisfiable. That $\alpha \wedge \neg\beta$ is unsatisfiable means that it is false in all models, i.e. that the negation is true in all models. I.e. we have to show that $\neg(\alpha \wedge \neg\beta)$ is valid if and only if $\alpha \Rightarrow \beta$ is valid. But this follows easily since

$$\neg(\alpha \wedge \neg\beta) \equiv (\neg\alpha \vee \beta) \equiv \alpha \Rightarrow \beta.$$

Exercise 4

a)

$$\begin{aligned} & \neg(a \Rightarrow b) \wedge (a \vee b \Leftrightarrow (c \Rightarrow b)) \\ & \equiv \neg(\neg a \vee b) \wedge ((a \vee b) \Rightarrow (\neg c \vee b)) \wedge ((\neg c \vee b) \Rightarrow (a \vee b)) \\ & \equiv (a \wedge \neg b) \wedge (\neg(a \vee b) \vee (\neg c \vee b)) \wedge (\neg(\neg c \vee b) \vee (a \vee b)) \\ & \equiv a \wedge \neg b \wedge ((\neg a \wedge \neg b) \vee (\neg c \vee b)) \wedge ((c \wedge \neg b) \vee (a \vee b)) \\ & \equiv a \wedge \neg b \wedge (\neg a \vee \neg c \vee b) \wedge (\neg b \vee \neg c \vee b) \wedge (c \vee a \vee b) \wedge (\neg b \vee a \vee b) \end{aligned}$$

b) By unit resolution on the literals a and $\neg b$ we get:

$$\frac{\begin{array}{c} a, (\neg a \vee \neg c \vee b) \\ \hline \neg b, (\neg c \vee b) \end{array}}{\neg c}$$

that is $\neg(a \Rightarrow b) \wedge (a \vee b \Leftrightarrow (c \Rightarrow b)) \models \neg c$.

Exercises Lecture 7

Introduction to Artificial Intelligence (IAI)

Exercise 1 (adapted from A97 2.1-2)

Describe a polynomial time algorithm for determining whether a DNF is satisfiable. Describe a polynomial time algorithm for determining whether a CNF is a tautology.

Exercise 2 (adapted from A97 2.5)

Explain how the question of tautology and satisfiability can be decided if we are given an algorithm for checking equivalence between two Boolean expressions.

Exercise 3 (adapted from A97 3.1)

Show how to write the Boolean expressions $\neg x$, $x \wedge y$, $x \vee y$, $x \Rightarrow y$, and $x \Leftrightarrow y$ using the if-then-else operator, tests on un-negated variables, and the constants 0 and 1. Draw the ROBDDs for the expressions.

Exercise 4

Draw the ROBDD for $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ using the ordering x_1, x_2, y_1, y_2 .

Exercise 5 (adapted from A97 3.2)

Draw the ROBDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3)$ using the ordering $x_1, y_1, x_2, y_2, x_3, y_3$ and $x_1, x_2, x_3, y_1, y_2, y_3$

Mandatory assignment

A threshold function $f^k(x_1, x_2, \dots, x_n)$ is a Boolean function on n Boolean variables that is true if at least k of the Boolean variables are true.

Thus,

$$f^k(x_1, x_2, \dots, x_n) \equiv (|\{x_i = 1\}| \geq k).$$

- 1) Draw the ROBDD of $f^3(x_1, x_2, x_3, x_4, x_5)$ using the variable order $x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$.
- 2) Label each internal node of your ROBDD with the number of true variables on the path leading to the node.
- 3) Argue that the size (number of nodes) of the ROBDD of $f^k(x_1, x_2, \dots, x_n)$ is $O(kn)$.
Hint: Use your solution from 2) to get an idea.

Solution to Exercises - Week 7

Introduction to Artificial Intelligence

Exercise 1

A polynomial time algorithm for checking if a CNF formula is a tautologi is given below. Here the formula F is $F = C_1 \wedge \dots \wedge C_m$, where $C_i = x_{i1} \vee x_{i2} \vee \dots \vee x_{in_i}$. For F to be a tautologi, each clause C_i must evaluate to TRUE no matter how the variables are assigned. In each clause, a variable together with its negation therefore must be present.

```
IsCNFTAUTLOGI( $F$ )
  for clause  $i \leftarrow 1$  to  $m$ 
     $trueFound \leftarrow \text{FALSE}$ 
    for  $j \leftarrow 0$  to  $n_i$ 
      for  $k \leftarrow 0$  to  $n_i$ 
        if  $x_{ij} = \neg x_{ik}$ 
           $trueFound \leftarrow \text{TRUE}$ 
        if  $trueFound = \text{FALSE}$ 
          return FALSE
    return TRUE
```

The running time of this algorithm is $O(m \cdot \max_{i \in \{1, \dots, m\}} n_i^2) = O(m) \cdot O(n^2)$.

A polynomial time algorithm for checking if a DNF formula is a satisfiable is given below. Here the formula F is $F = C_1 \vee \dots \vee C_m$, where $C_i = x_{i1} \wedge x_{i2} \wedge \dots \wedge x_{in_i}$. For F to be satisfiable, we must be able to find a clause C_i where we can choose an assignment of the variables in the clause, such that it will evaluate to TRUE. This can be done, as long as we can find a clause where none of the variables are present in both negated and unnegated form.

```

IsDNFSATISFIABLE( $F$ )
  for clause  $i \leftarrow 1$  to  $m$ 
    clauseGood  $\leftarrow$  TRUE
    for  $j \leftarrow 0$  to  $n_i$ 
      for  $k \leftarrow 0$  to  $n_i$ 
        if  $x_{ij} = \neg x_{ik}$ 
          clauseGood  $\leftarrow$  FALSE
        if clauseGood = TRUE
          return TRUE
    return FALSE
  
```

The running time of this algorithm is $O(m \cdot \max_{i \in \{1, \dots, m\}} n_i^2) = O(m) \cdot O(n^2)$.

Exercise 2

Given an algorithm EQUIVALENCE(F_1, F_2) that returns true iff F_1 and F_2 are logically equivalent, we can decide whether a formula F is a tautology by:

$$\text{TAUTOLOGI}(F) = \text{EQUIVALENCE}(F, \text{True})$$

Similarly we can decide if a formula F is satisfiable by:

$$\text{SATISFIABLE}(F) = \neg \text{EQUIVALENCE}(F, \text{False})$$

Exercise 3

Using the if-then-else operators, we get:

$$\neg x = x \rightarrow 0, 1.$$

$$x \wedge y = x \rightarrow (y \rightarrow 1, 0), 0.$$

The corresponding ROBDDs are shown in Figure 1.

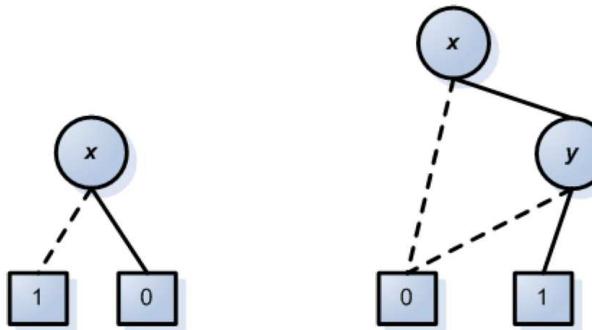


Figure 1: The ROBDDs for $\neg x$ and $x \wedge y$ respectively

$$x \vee y = x \rightarrow 1, (y \rightarrow 1, 0).$$

$$x \Rightarrow y = x \rightarrow (y \rightarrow 1, 0), 1.$$

The corresponding ROBBDs are shown in Figure 2.

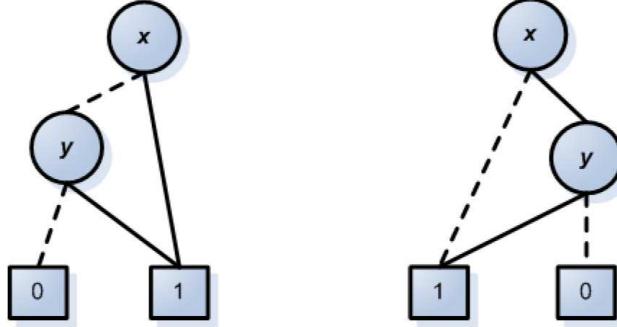


Figure 2: The ROBDDs for $x \vee y$ and $x \Rightarrow y$ respectively

$$x \Leftrightarrow y = x \rightarrow (y \rightarrow 1, 0), (y \rightarrow 0, 1).$$

The ROBBD are shown in Figure 3.

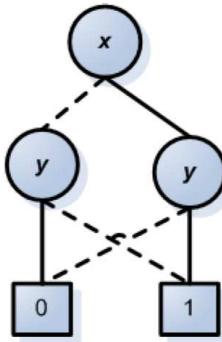


Figure 3: The ROBDD for $x \Leftrightarrow y$

Exercise 4

The ROBBD for $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ using the ordering x_1, x_2, y_1, y_2 is shown in Figure 4.

Exercise 5

The ROBBD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2) \wedge (x_3 \Leftrightarrow y_3)$ using the ordering $x_1, y_1, x_2, y_2, x_3, y_3$ and $x_1, x_2, x_3, y_1, y_2, y_3$ are shown in Figures 5 and 6 respectively.

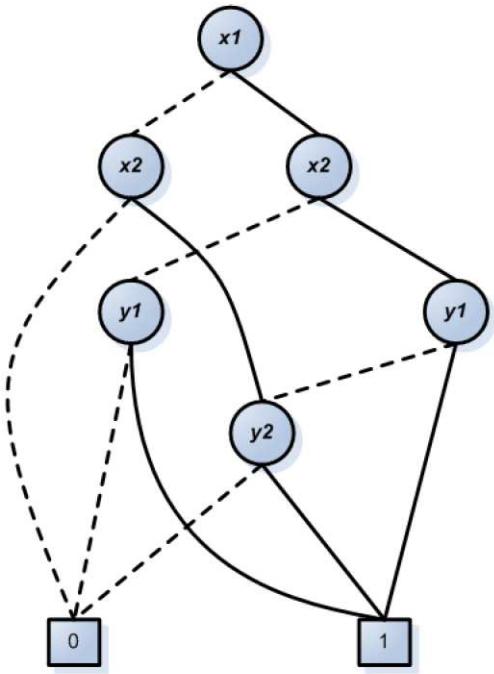


Figure 4: The ROBDD for $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

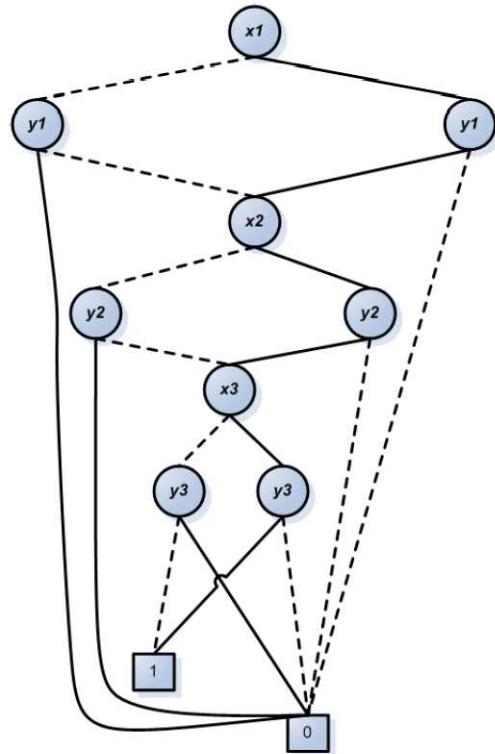


Figure 5: Using the ordering $x_1, y_1, x_2, y_2, x_3, y_3$

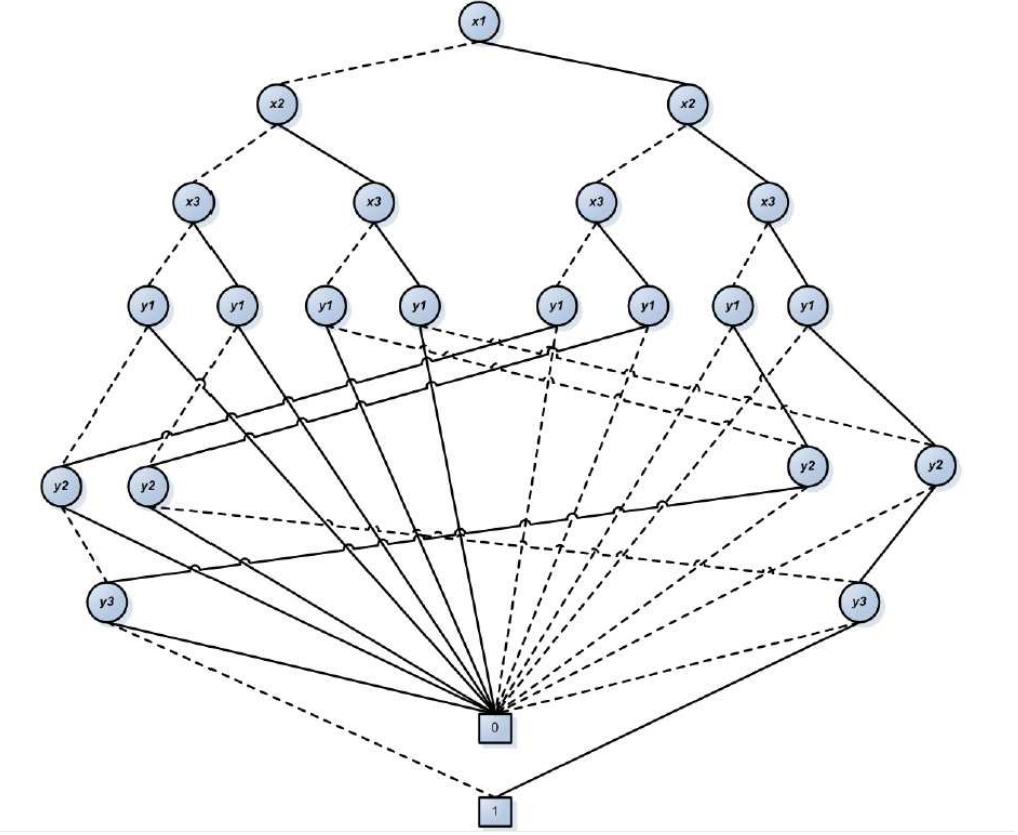


Figure 6: Using the ordering $x_1, x_2, x_3, y_1, y_2, y_3$

Exercises Lecture 8

Introduction to Artificial Intelligence (IAI)

Exercise 1 (adapted from A97 4.1)

Construct the ROBDD for $\neg x_1 \wedge (x_2 \Leftrightarrow \neg x_3)$ with ordering $x_1 < x_2 < x_3$ using the algorithm BUILD in figure 9 of A97. Show the recursive call structure and the final content of the unique table.

Exercise 2 (adapted from A97 4.3)

Suggest an improvement $\text{BUILDCONJ}(t)$ of Build which generates only a linear number of calls for Boolean expressions t that are conjunctions of variables and negations of variables.

Exercise 3 (adapted from A97 4.4)

Construct the ROBDDs for x and $x \Rightarrow y$ for any variable ordering you want. Compute the disjunction of the two ROBDDs using APPLY.

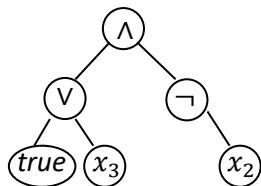
Exercise 4 (adapted from A97 4.5)

Draw the ROBDD for $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$ using BUILD with the ordering $x_1 < x_2 < x_3$. Use APPLY to find the ROBDD for $\neg(x_1 \wedge x_3) \vee (x_2 \wedge x_3)$.

Mandatory assignment

During lecture you have learned how to create an ROBDD using BUILD in $O(2^n)$ time. As stated during lecture, a more efficient way is to use APPLY (and MK) and construct the ROBDD bottom-up from the expression tree. The goal of this assignment is to write the pseudo-code for this algorithm.

Consider the expression tree of $(\text{true} \vee x_3) \wedge \neg x_2$ shown below



This expression tree and general Boolean expression trees can be represented by the data structure:

Expr	
type() {VAR, NOT, AND, OR, TRUE, FALSE}	<i>Return the type of expression</i>
left() Expr	<i>Return the left Expr node of an operation</i>
right() Expr	<i>Return the right Expr node of an operation</i>
idx() integer	<i>Return the ID of Expr (only for type VAR)</i>

For negated expression (type = NOT), you can assume that right() holds the expression under negation.

1. How can you use APPLY to negate a ROBDD u ?
(hint: use one of the 16 Boolean operators and a terminal ROBDD (0 or 1) as your two other arguments to APPLY)
2. How can you use Mk to construct the ROBDD for a variable x_i ?
3. Use your result in 1. and 2. to write the pseudo-code of an algorithm called EXPR2ROBDD that only uses APPLY and Mk to create an ROBDD from a Boolean expression of type Expr.

Solution to Exercises - Week 8

Introduction to Artificial Intelligence

Exercise 1

The recursive call structure of constructing the ROBBD of $\neg x_1 \wedge (x_2 \Leftrightarrow \neg x_3)$ using the ordering $x_1 < x_2 < x_3$ is shown in Figure 1 and the resulting ROBBD is shown in Figure 2.

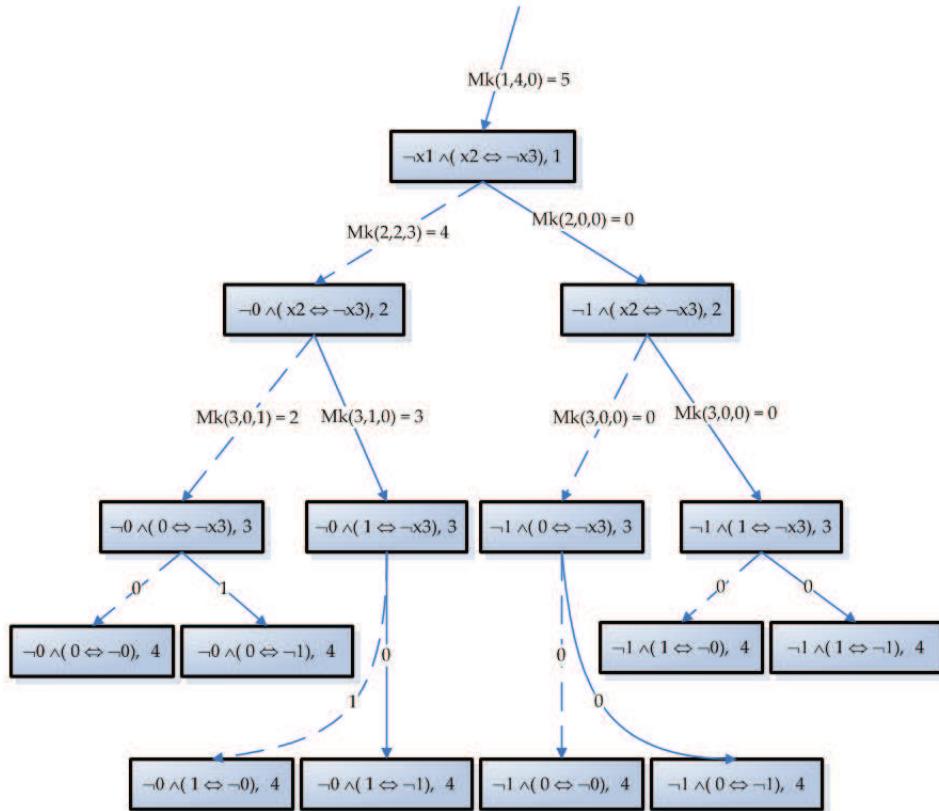


Figure 1: The recursive call structure of BUILD for $\neg x_1 \wedge (x_2 \Leftrightarrow \neg x_3)$

The final content of the unique table is given in Figure 3.

Exercise 2

When we are looking at an expression t which is a conjunctions of variables and negated variables, we know that if the evaluation of the variable we are branching

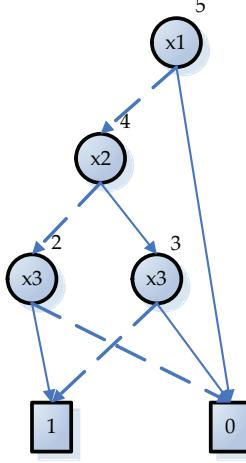


Figure 2: The ROBDD for $\neg x_1 \wedge (x_2 \Leftrightarrow x_3)$

ID	variable	low	high
0	4	-	-
1	4	-	-
2	3	0	1
3	3	1	0
4	2	2	3
5	1	4	0

Figure 3: The table for $\neg x_1 \wedge (x_2 \Leftrightarrow x_3)$

on results in false, then the whole expression will evaluate to false. To improve the BUILD we can therefore do the following:

```
BUILDCONJ(t)
    return BUILD'CONJ(t, 1)
```

where $\text{BUILD}'\text{CONJ}(t, i)$ is described in the following pseudo code:

```
BUILD'CONJ(t, i)
    if  $i > n$ 
        if  $t$  is false
            return 0
        else
            return 1
    else
        if  $x_i$  is unnegated in  $t$ 
             $v_0 \leftarrow 0$ 
             $v_1 \leftarrow \text{BUILD}'\text{CONJ}(t[1/x_i], i + 1)$ 
        else
             $v_0 \leftarrow \text{BUILD}'\text{CONJ}(t[0/x_i], i + 1)$ 
             $v_1 \leftarrow 0$ 
        return MK(i,  $v_0, v_1$ )
```

Exercise 3

The ROBBDs of x and $x \Rightarrow y$ is shown in Figure 4 and the table of nodes after these have been build is shown in Figure 5.



Figure 4: The ROBBDs for x and $x \Rightarrow y$

ID	variable	low	high
0	3	-	-
1	3	-	-
2	1	0	1
3	2	0	1
4	1	1	3

Figure 5: The table after building ROBBDs for x and $x \Rightarrow y$

Running the APPLY results in the structure in Figure 6. The table after running

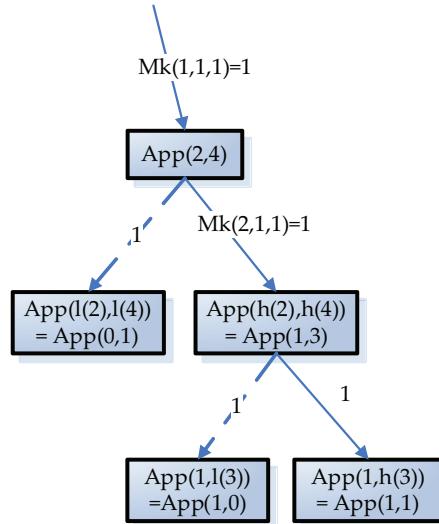


Figure 6: The recursive calls of Apply

the algorithm is still the one shown in Figure 5. The resulting ROBBD is shown in Figure 7; it is easily seen that

$$x \vee (x \Rightarrow y) \equiv x \vee \neg x \vee y \equiv \top.$$



Figure 7: Resulting ROBBD from $x \vee (x \Rightarrow y)$

Exercise 4

The recursive call structure of constructing the ROBBDs of $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$ using the ordering $x_1 < x_2 < x_3$ is shown in Figure 10 and the resulting ROBBDs is shown in Figure 8.

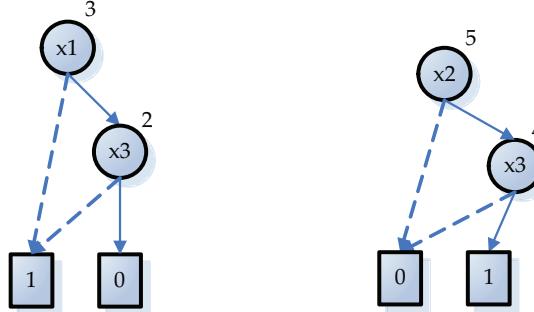


Figure 8: The ROBDDs for $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$

ID	variable	low	high
0	4	-	-
1	4	-	-
2	3	1	0
3	1	1	2
4	3	0	1
5	2	0	4

Figure 9: The table after construction of ROBBDs for $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$

The content of the unique table after constructing these ROBBDs is given in Figure 9.

Using APPLY to construct the ROBBD for $\neg(x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ results in the recursive call structure given in Figure 11. The resulting ROBBD is shown in Figure 12.

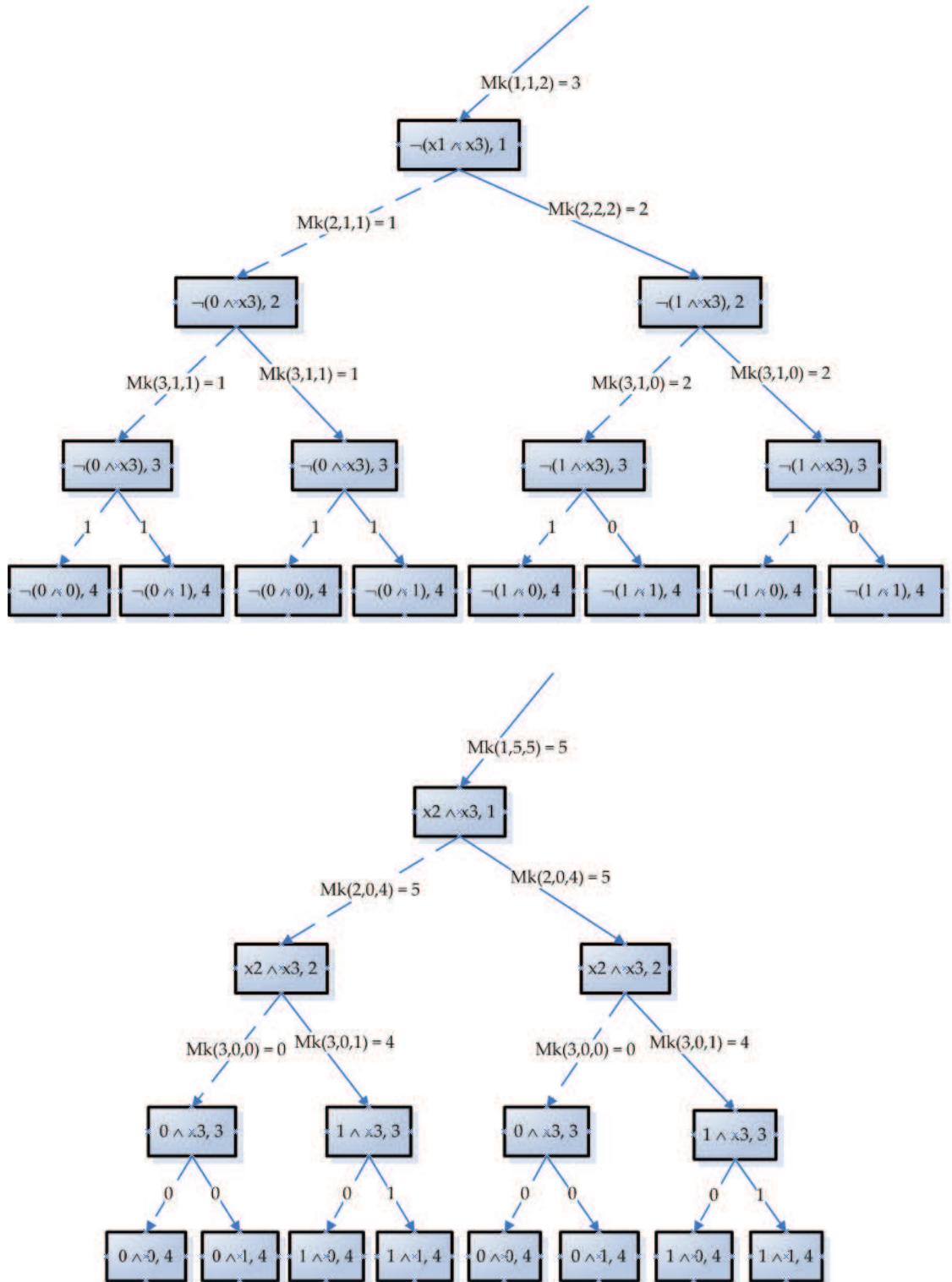


Figure 10: The recursive call structure of `BUILD` for $\neg(x_1 \wedge x_3)$ and $x_2 \wedge x_3$

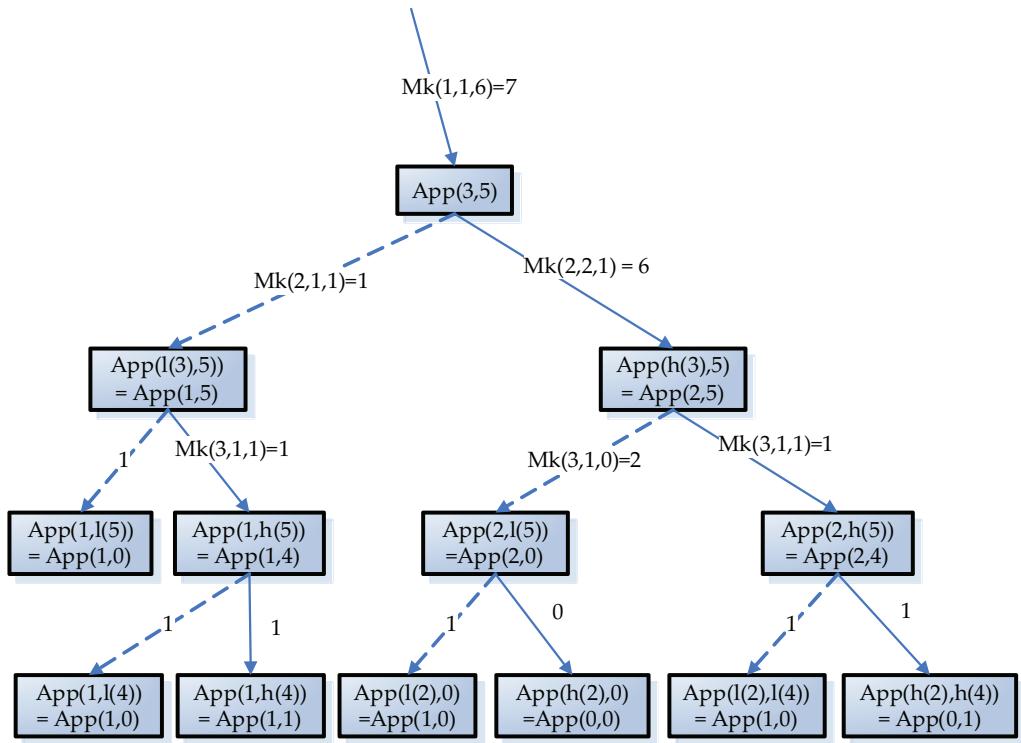


Figure 11: The recursive call structure of APPLY for $\neg(x_1 \wedge x_3) \vee (x_2 \wedge x_3)$

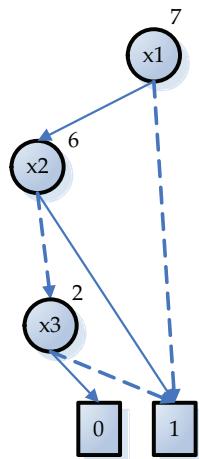


Figure 12: The ROBDD for $\neg(x_1 \wedge x_3) \vee (x_2 \wedge x_3)$

Exercises Lecture 9

Introduction to Artificial Intelligence (ISP)

Exercise 1 (adapted from Mozart-OZ online tutorial)

A CSP model consists of a set of variables, a set of domains (possible values) for each variable and a set of constraints between the variables that limit the values that variables can take, all describing a specific problem.

The code of Professor Smart's safe is a sequence of 4 distinct (i.e., all different) nonzero digits, C_1, \dots, C_4 such that the following equations and inequalities are satisfied:

$$C_1 \neq 1, C_2 \neq 2, C_3 \neq 3, C_4 \neq 4$$

$$C_4 - C_3 > 2$$

$$C_3 + C_1 \geq 5$$

$$C_1 * C_2 \geq 6$$

$$C_1 + C_2 \leq 7$$

$$C_3 \leq 5$$

$$C_4 > 4$$

A CSP model for this problem is:

Variables: $\{C_1, C_2, C_3, C_4\}$

Domains: $\{[1\dots6], [1\dots6], [1\dots6], [1\dots6]\}$

Constraints: $\{C_1 \neq C_2, C_1 \neq C_3, C_1 \neq C_4, C_2 \neq C_3, C_2 \neq C_4, C_3 \neq C_4, C_1 \neq 1, C_2 \neq 2, C_3 \neq 3, C_4 \neq 4, C_4 - C_3 > 2, C_3 + C_1 \geq 5, C_1 * C_2 \geq 6, C_1 + C_2 \leq 7, C_3 \leq 5, C_4 > 4\}$

Now what you have to do:

- a) Use backtracking to find a solution, assume the order $\{C_1, C_2, C_3, C_4\}$ for the selection of the variables, draw the search tree and show in the failed leaves which constraint is the one that fails.
- b) Use FC and MRV to find a solution, draw the search tree and show the new domains of the variables for each leave after the execution of FC. Make the CSP node consistent before you start the search.

Exercise 2 (adapted from Mozart-OZ online tutorial)

Betty, Chris, Donald, Fred, Gary, Mary, and Paul want to align in one row for taking a photo. Some of them have preferences next to whom they want to stand:

1. Betty wants to stand next to Gary and Mary.
2. Chris wants to stand next to Gary.
3. Fred wants to stand next to Donald.
4. Paul wants to stand next to Donald.

Be aware that 2 people can't stand in the same position at the same time, meaning that there is a different constraint between each pair of variables representing the position where two people stand.

We can define such a constraint using mathematical operators: $\text{diff}(X, Y) : X \neq Y$, where X and Y are variables.

A partial CSP model for this problem is:

Variables: $\{B, C, D, F, G, M, P\}$

Domains: $\{[1\dots 7], [1\dots 7], [1\dots 7], [1\dots 7], [1\dots 7], [1\dots 7], [1\dots 7]\}$

Constraints: $\{B \neq C, B \neq D, B \neq F, B \neq G, B \neq M, B \neq P, C \neq D, C \neq F, C \neq G, C \neq M, C \neq P, D \neq F, D \neq G, D \neq M, D \neq P, F \neq G, F \neq M, F \neq P, M \neq G, M \neq P, G \neq P\}$

- a) Define a constraint $\text{adj}(X, Y)$ representing the fact that for a possible value of X , the value of Y has to be next to it. Use logical and mathematical operators to do so.
- b) As you may remember from the lecture, a constraint graph is a graph where each node represents a variable and each edge represents one of the binary constraints defined in the set of constraints. Draw a constraint graph for a CSP' just considering the adjacency constraints of this problem. Variables and domains are the same as defined above.
- c) Add the adjacency constraints together with constraint $B=1$ to the CSP and run the **AC-3** algorithm in order to make it arc-consistent. Show the domains of the variables after the execution of the algorithm.

Exercise 3

Johnny Seafood works at the aquarium in his town. He is attending some courses at the local university and now he has a problem at work where he might be able to apply what he has learned in school. The aquarium where Johnny works has recently got 5 new fish species that need to be allocated in the three available tanks. There are some constraints concerning what species can be together in a tank and also some specific requirements.

- Sharks and dolphins can't be together in the same tank.
- Sharks and remoras must be in the same tank.
- Dolphins can't be in tank 3. It is not possible for them to perform their show in this tank.

- There are too many dolphins so they cannot share the tank with the whale.
 - The tank closest to the food deposit is tank 2. Johnny wants to allocate the whale there so he doesn't have to carry its food too far.
 - Remoras are so small that the whale could eat them. Thus, they can't be in the same tank as the whale.
 - Lobsters can be anywhere.
- a) Define the problem as a CSP. That is, define the variables, domains, and constraints of the problem. All your constraints should be binary or unary.
 - b) Draw the constraint graph of the CSP.
 - c) Use the Forward Checking algorithm to solve the problem, make the CSP node consistent before starting the algorithm. Draw the search tree and the domain of the variables at each node. Assume the fixed variable order: sharks, lobsters, remoras, dolphins, and whale.
 - d) Use the MAC algorithm to solve the problem. Draw the search tree and the domain of the variables at each node. Assume the same order for selecting the variables as in c).

Mandatory assignment

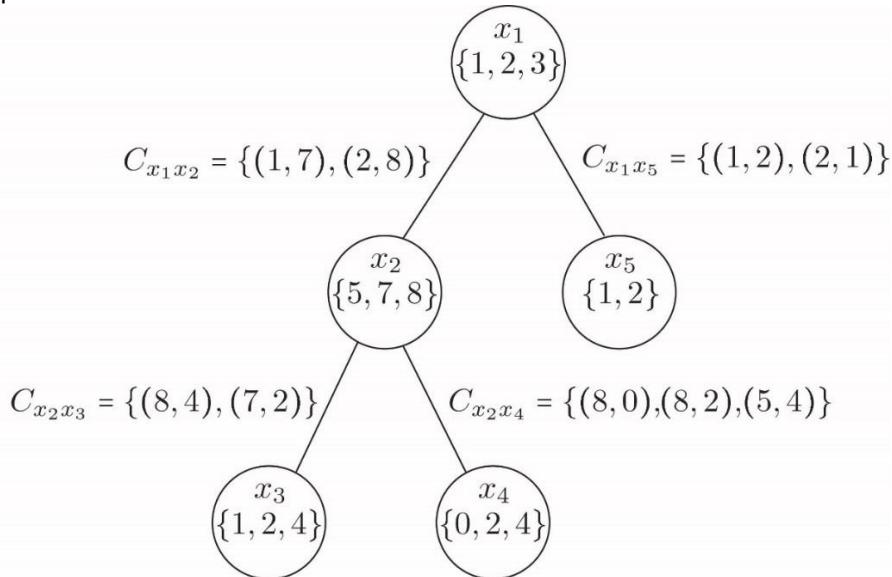
In this assignment, we consider binary CSPs where the constraint graph forms a tree. Recall that the nodes and edges of the constraint graph represent variables and constraints, respectively. As an example consider the binary CSP $W = (X, D, C)$ where:

$$X = \{x_1, x_2, x_3, x_4, x_5\};$$

$$D_1 = \{1, 2, 3\}, D_2 = \{5, 7, 8\}, D_3 = \{1, 2, 4\}, D_4 = \{0, 2, 4\}, D_5 = \{1, 2\};$$

$$\begin{aligned} C = \{ & \quad C_{x_1 x_2} = \{(1, 7), (2, 8)\}, \\ & \quad C_{x_1 x_5} = \{(1, 2), (2, 1)\}, \\ & \quad C_{x_2 x_3} = \{(8, 4), (7, 2)\}, \\ & \quad C_{x_2 x_4} = \{(8, 0), (8, 2), (5, 4)\}. \end{aligned}$$

The constraint graph of W is:



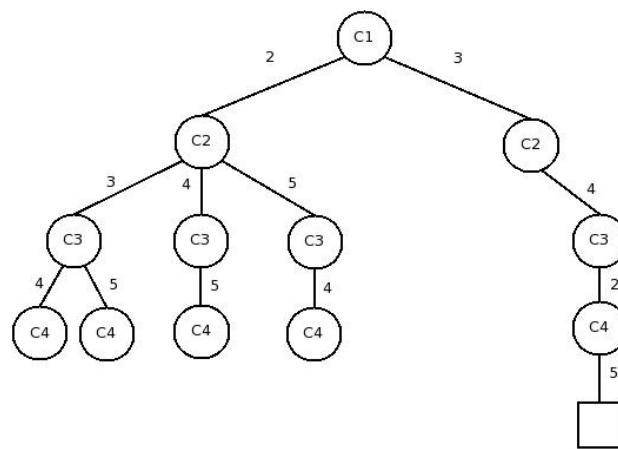
- 1) Reduce the domains of the variables of W such that W becomes arc consistent.
 - 2) Write a solution to W .
 - 3) Describe in words a polynomial time algorithm that can find a solution to an arbitrary binary CSP where the constraint graph forms a tree.
- (Hint: use REVISE on the arcs in the tree, eg. from parent to child, and/or from child to parent.)

Solutions to Exercises Week 9

Introduction to Artificial Intelligence (ISP)

Exercise 1

a) Backtracking:

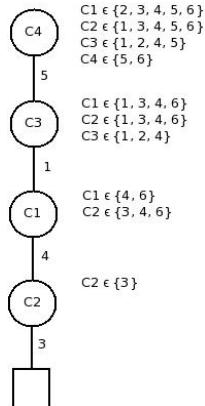


Each **leaf** on the tree with a circular shape represents a failed node. The variable inside the circle is the variable that doesn't have a possible value in its domain to fulfill the constraints. The square represents a solution.

The solution found is: $C_1 = 3, C_2 = 4, C_3 = 2, C_4 = 5$.

All failed leaves are failed for the same reason, it is not possible to assign a value to C_4 that satisfies the constraint $C_4 - C_3 > 2$.

b) Now the FC + MRV tree:



Here the combination of forward checking and the MRV heuristic finds a solution without any failed leaves. However, another solution, $C_1 = 4$, $C_2 = 3$, $C_3 = 1$, $C_4 = 5$ was found by the MRV.

Exercise 2

CSP Model:

Variables: $\{B, C, D, F, G, M, P\}$

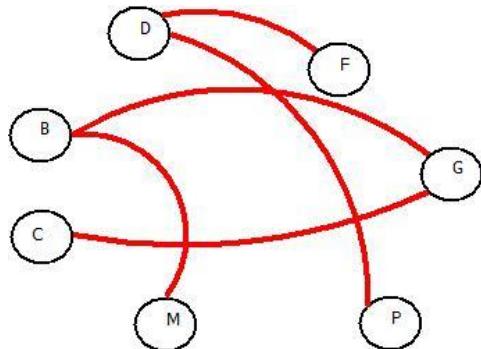
Domains: $\{[1 \dots 7], [1 \dots 7]\}$

Constraints: $\{B \neq C, B \neq D, B \neq F, B \neq G, B \neq M, B \neq P, C \neq D, C \neq F, C \neq G, C \neq M, C \neq P, D \neq F, D \neq G, D \neq M, D \neq P, F \neq G, F \neq M, F \neq P, M \neq G, M \neq P, G \neq P\}$

a) $adj(X, Y) : X = Y + 1 \quad | \quad X = Y - 1$.

The main idea of the adj constraint is to represent the fact that for a pair of variables the values assigned to them have to be next to each other, no matter which of those values come first. In the case of the constraint above you can observe that for any possible value assigned to X, the value assigned to Y has to be either the next or the one before to the X value.

b)



This is the graph representation of the cps model considering just the adjacency constraints.

c) The new CSP is defined as:

Variables: $\{B, C, D, F, G, M, P\}$

Domains: $\{[1 \dots 7], [1 \dots 7]\}$

Constraints: $\{B \neq C, B \neq D, B \neq F, B \neq G, B \neq M, B \neq P, C \neq D, C \neq F, C \neq G, C \neq M, C \neq P, D \neq F, D \neq G, D \neq M, D \neq P, F \neq G, F \neq M, F \neq P, M \neq G, M \neq P, G \neq P, B=1, adj(B,G), adj(B,M), adj(C,G), adj(F,D), adj(P,D)\}$

The first constraint to be considered is $B=1$, since it is the only unary constraint. After B has been assigned to 1, G gets assigned to 2 due to $\text{adj}(B,G)$, and so does M due to $\text{adj}(B,M)$, (neither of the two variables have in their domains the value 0, therefore the only adjacent position to B is 2). The assignments $M=2$ and $G=2$ break the constraint $M \neq G$, making the CSP inconsistent.

Exercise 3

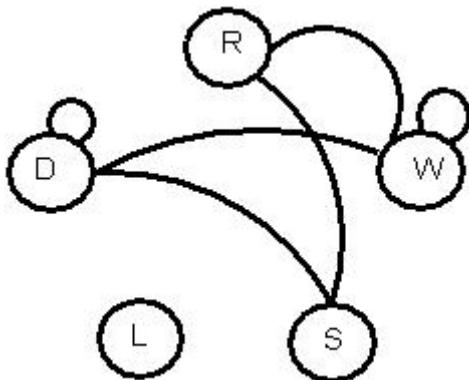
- a) CSP model:

Variables: $\{R, D, L, S, W\}$

Domains: $\{1, 2, 3\}$ for each variable

Constraints: $\{S \neq D, S = R, D \neq W, W = 2, W \neq R\}$

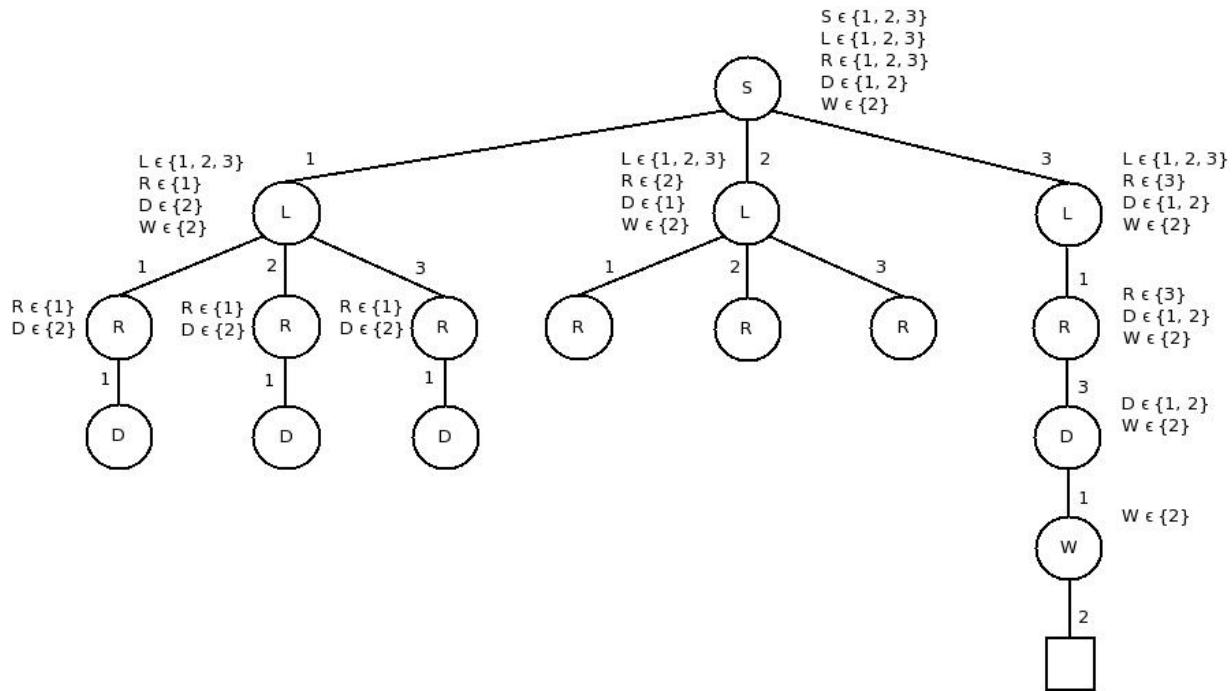
- b) CSP graph:



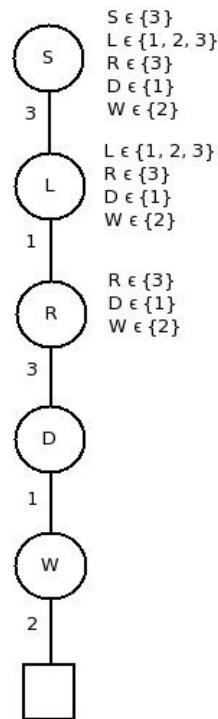
- c) Each node has right next to it the domain of the variables after executing FC with the previous variable-value assignment. In the case of the first node the domains shown are the result of executing node consistency on the CSP. The values on the edges of the search tree represent the variable-value assignment represented by the branch. Each **leaf** with a circular shape represents a failed node (not possible variable-value assignment). The **leaf** with a square shape represents a solution.

The solution found by FC was $\{S = 3, L = 1, R = 3, D = 1, W = 2\}$

FC search tree:



d) MAC search tree:



The solution found by the MAC algorithm was $\{S = 3, L = 1, R = 3, D = 1, W = 2\}$, the same found by the FC algorithm.

Introduction to Artificial Intelligence

Lecture 1 WE START 10:10!

Introduction to IAI

Today's Program

- 10:00-?: Introduction to IAI
 - My background
 - The history of AI
 - Path to Artificial General Intelligence (AGI)
 - The IAI Course
 - Intended learning objectives
 - A word about prerequisites
 - Formalities
 - Schedule

My Background

Research

PhD AI (SetA*)
Carnegie Mellon

Vessel phase in/out optimization

Hierarchically decomposed
stowage algorithms

Stowage textbook

Nondeterministic / Symbolic / Temporal planning algorithms

Capacity models

RL / LNS – based
stowage algorithms

2000 2005 2010 2015 2020 2025

Maersk consulting stowage / cargomix



ML Lashing

StowmanS

Stowman/SAM

Breakbulk Optimizer

SLX | STOW™

Industry R&D

2022 Rune Møller Jensen

SLX | YIELD™

SLX | STOW

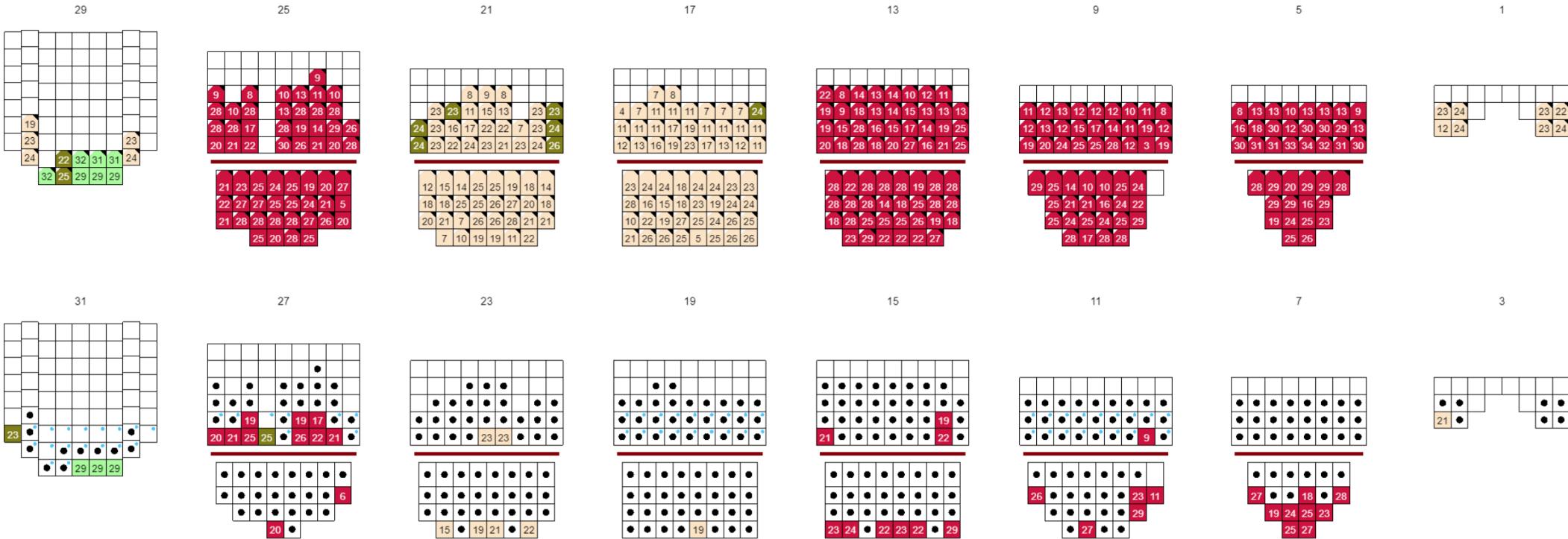
SLX | STOW™

SCENARIO | STOWAGE | LOG | TANKS | TERMINALS | SETTINGS | INPUT | RESULTS | CRANES | FORECAST | USER STATISTICS | INTERNAL

rune.moller.jensen@sealytix.com | UNIFEEDER

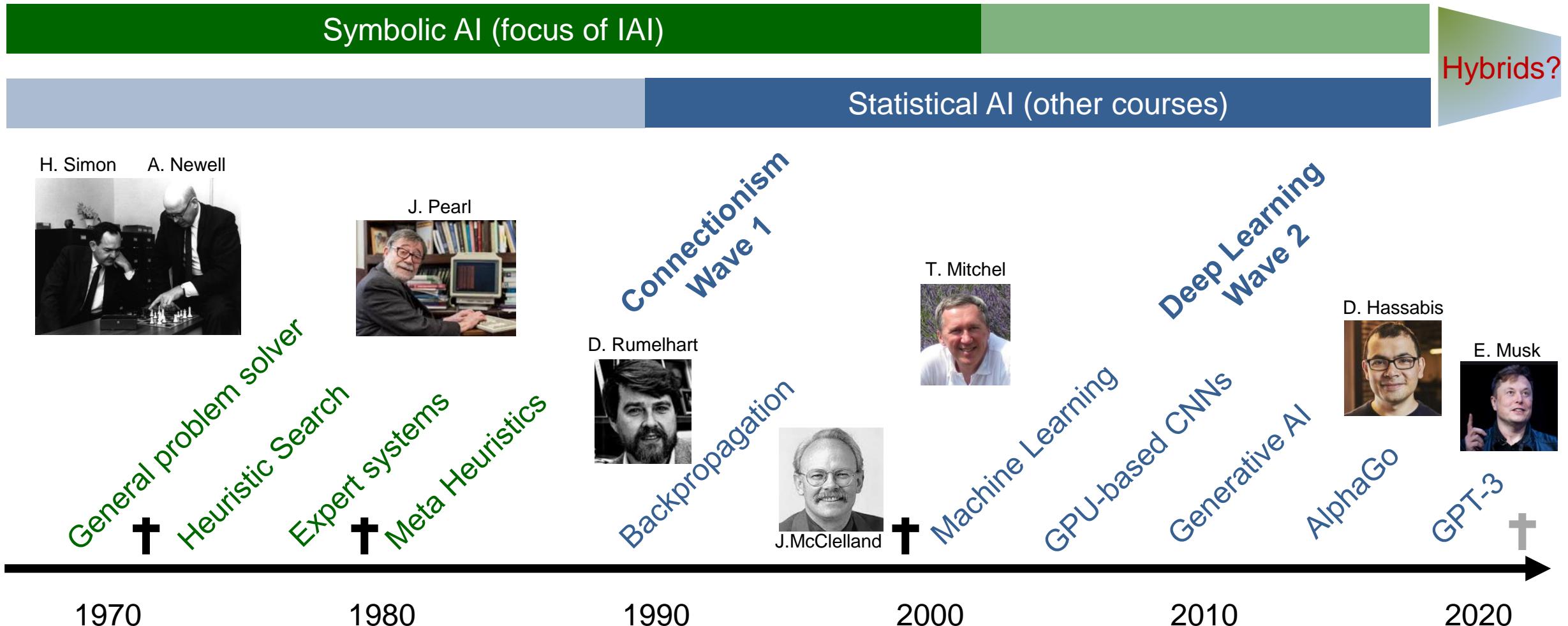
ELBLUE (DG) Voyage: 14 POD 20 ▾ Rob Restows Zoom: ● Reset Save Edit ↻ ↻ ↻ ↻

DKODE NLEMT NLEDE NLRWG NLEMT_1 BEG17 RUBRN RUA4S RUPLP RUFCT NLEDE_1 NLEMT_2 NLEDE_2 NLRWG_1

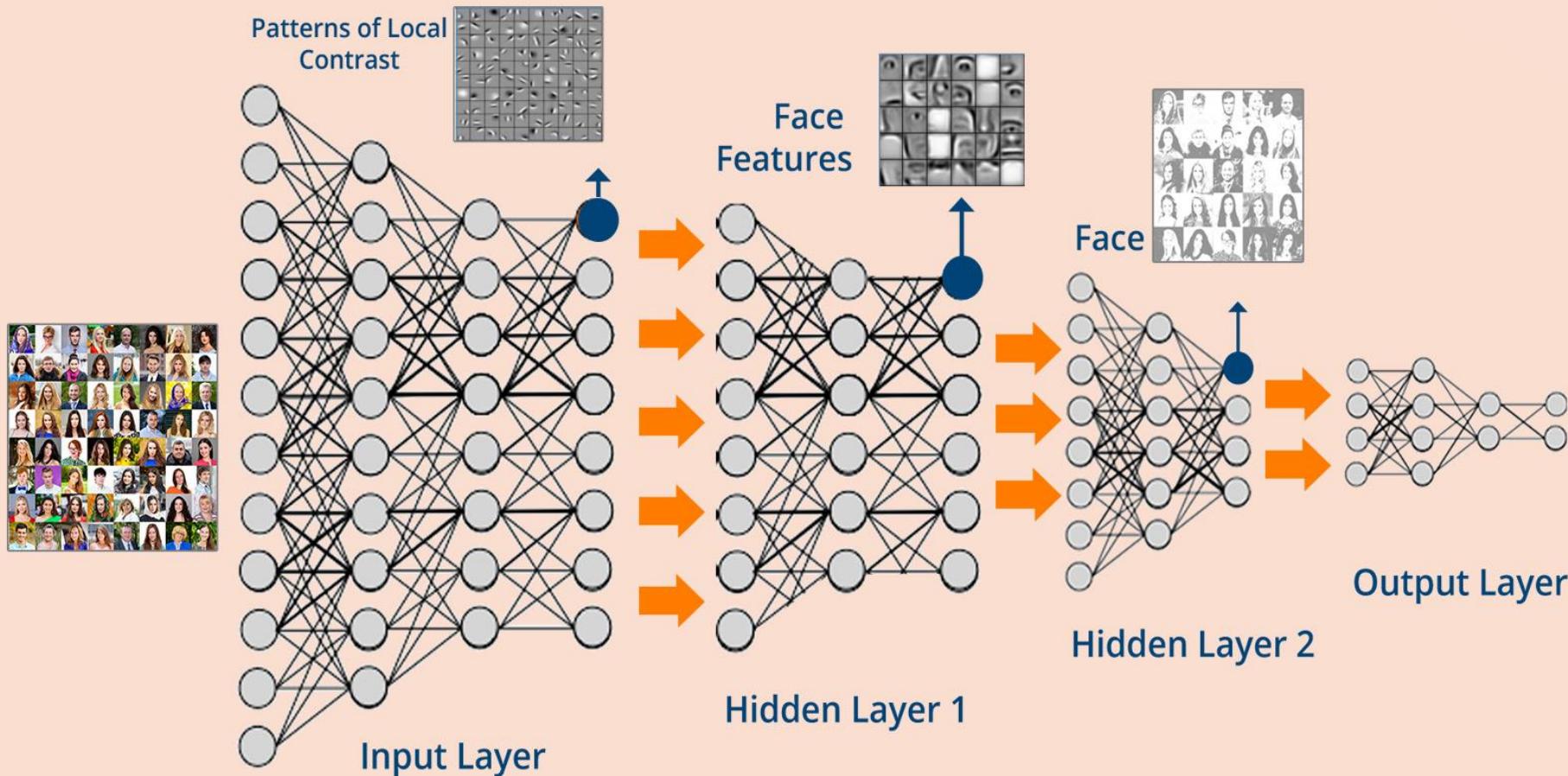


Units 402 TEU 712 (96%) Reefers 21 (8%) Weight 8241t (80%) | Tanks BW 1885.17t Other 568.73t Constants 0t | Draft Fore 6.63m Aft 7.79m Displacement 15799.65t | GM Corr. 1.26m Limit 0.95m | Trim 1.16m | List 0.7° | SF 20% BM 63% TM 0%

The history of AI

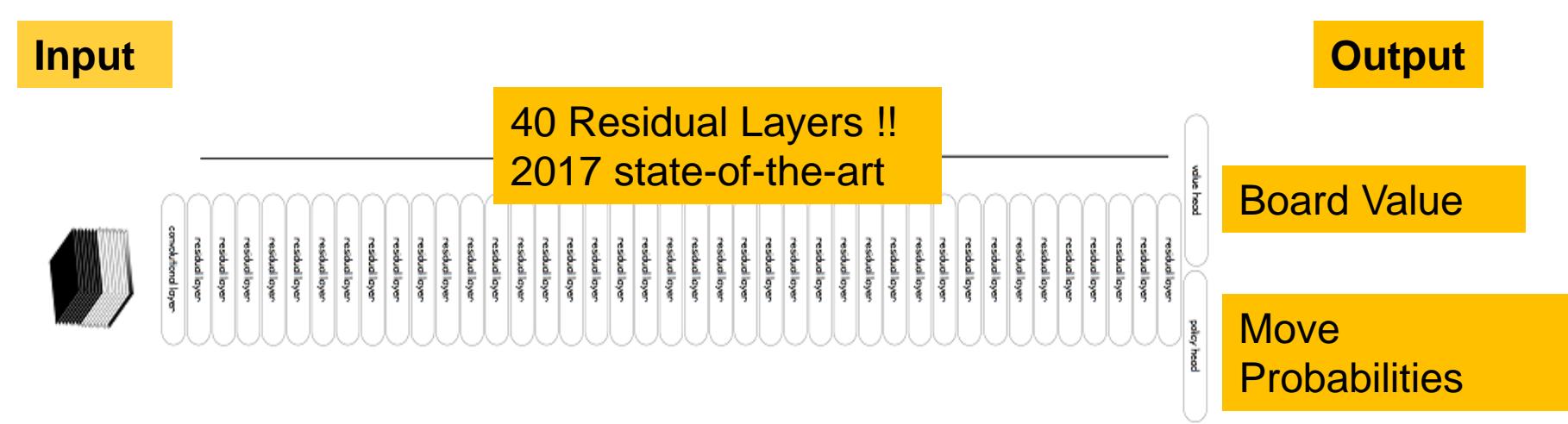


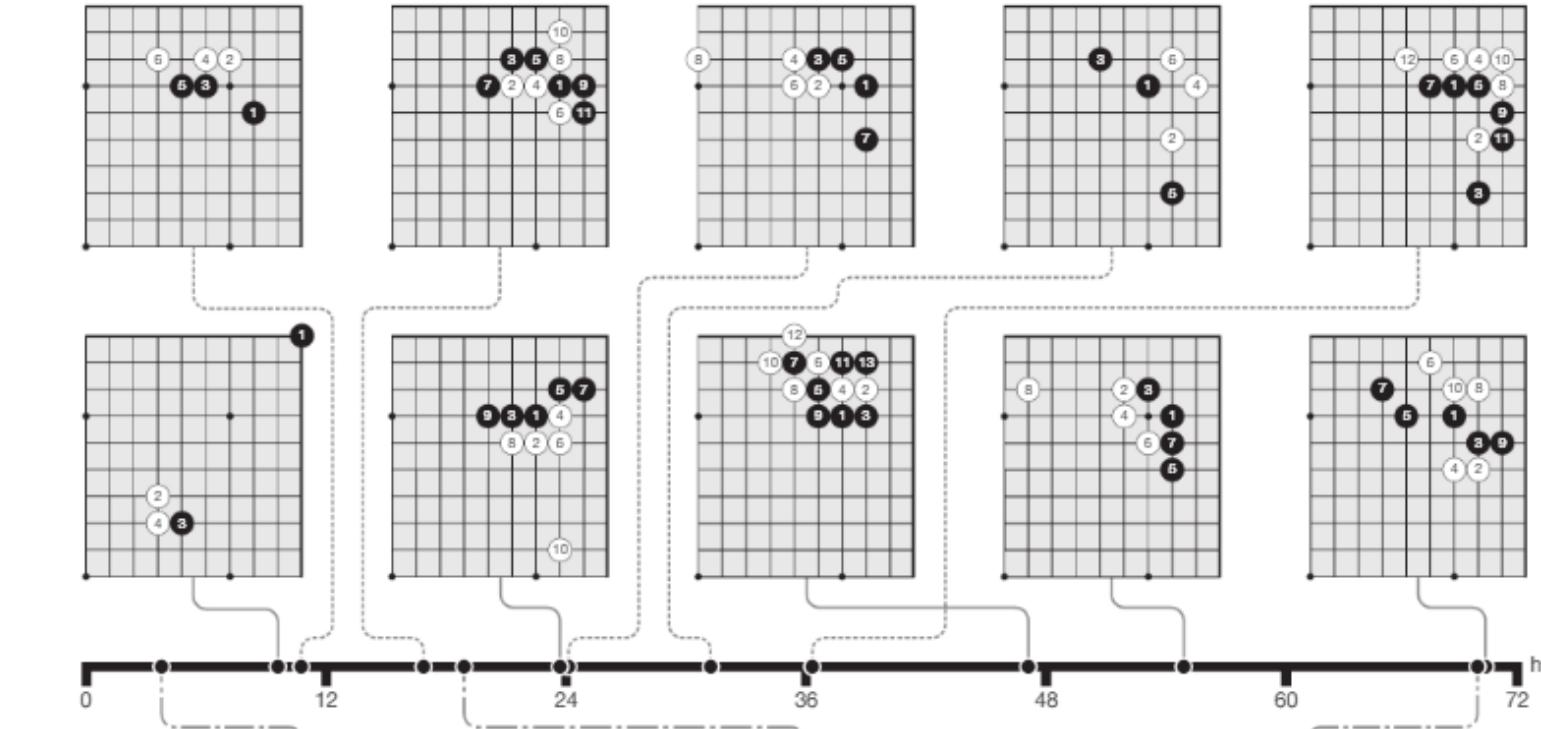
2006 Deep Learning Classic: Object Recognition



2018 DeepMind - AlphaGo (watch on Netflix)

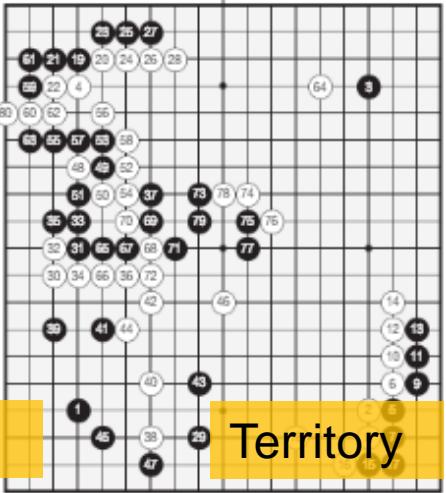
- https://www.youtube.com/watch?v=WXuK6gekU1Y&ab_channel=DeepMind





5 known
Joseki
discovered

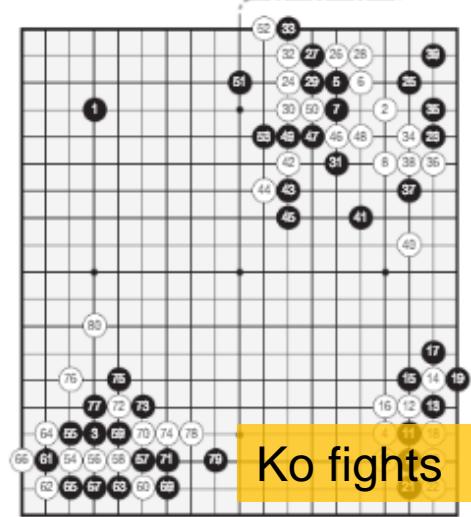
Most
frequent
Joseki



Capturing

Territory

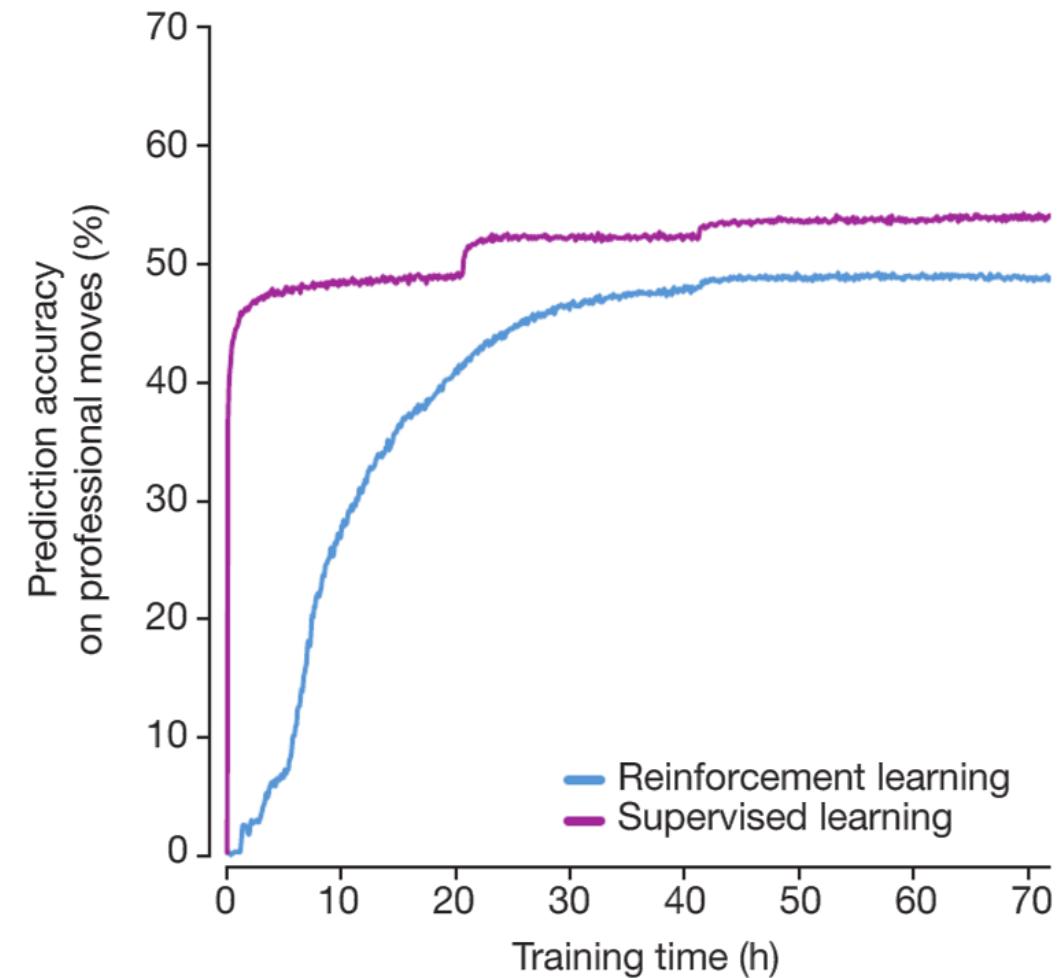
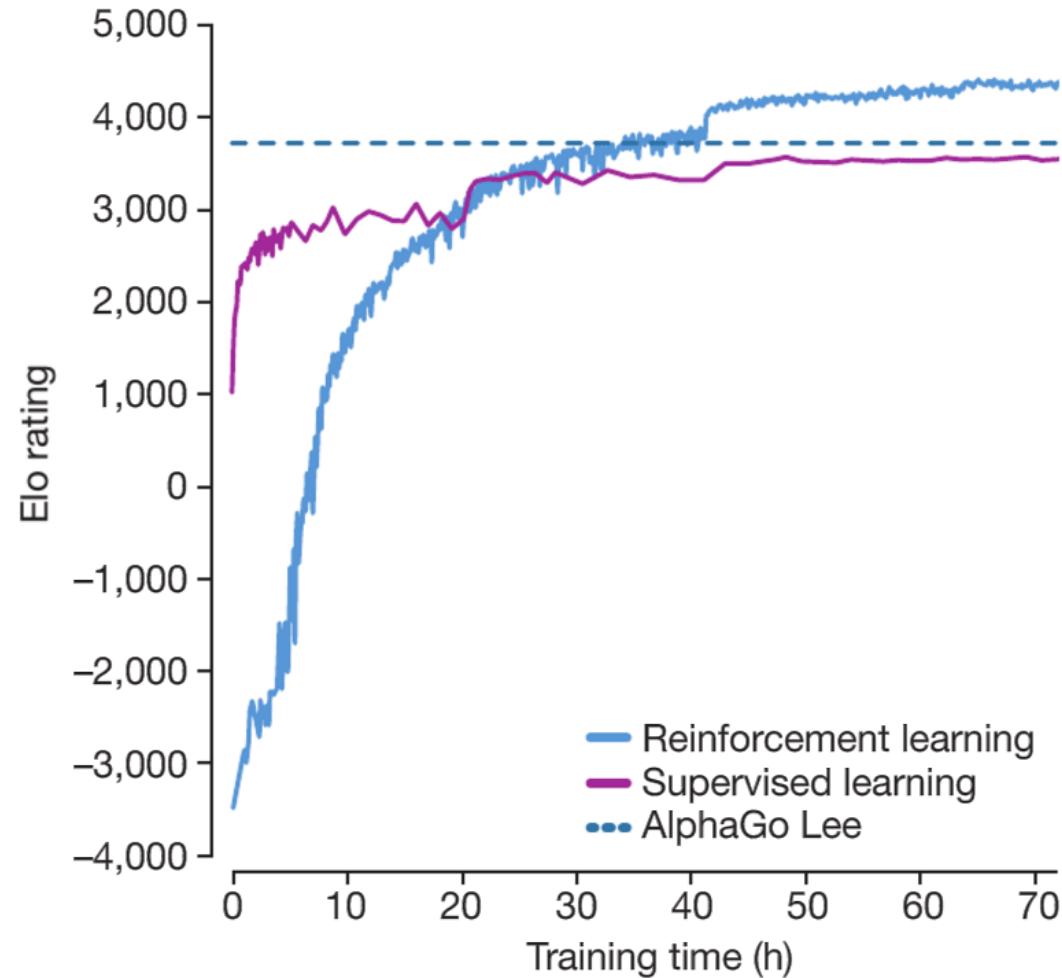
27 at 17 30 at 29 37 at 21 42 at 34 55 at 44 61 at 38
 64 at 40 67 at 39 70 at 40 71 at 25 73 at 21 74 at 60
 75 at 38 76 at 34



Ko fights

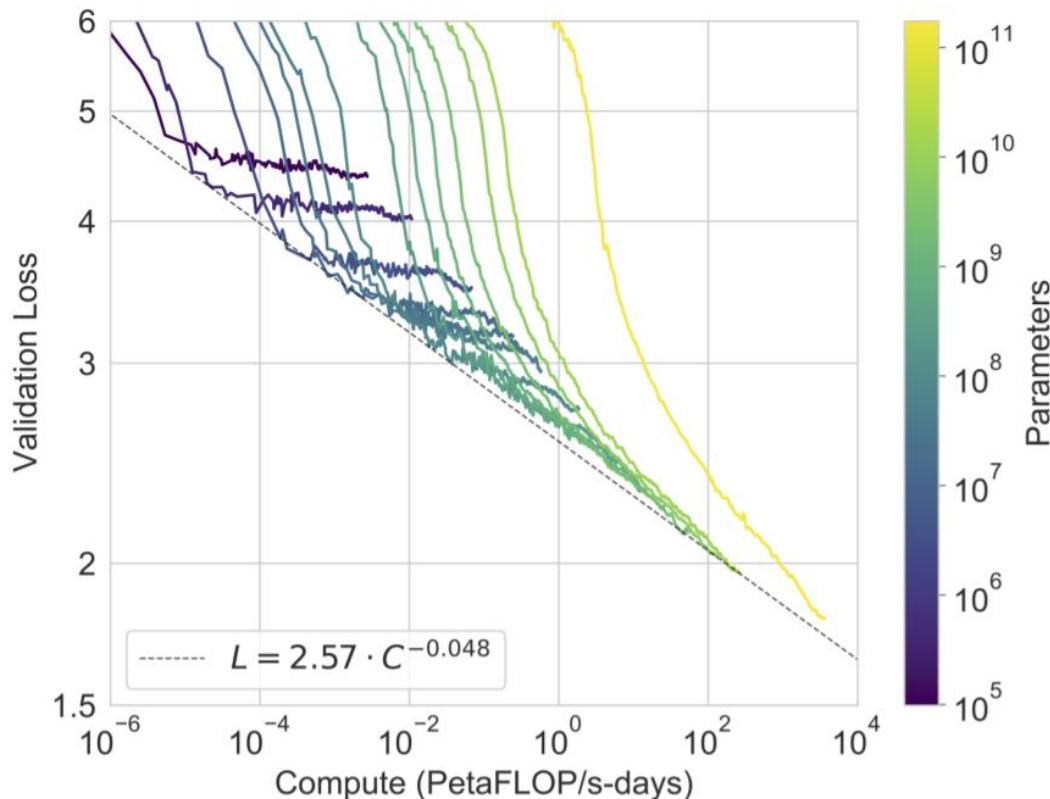
Lee Sedol
“Divine
moves”
played by
AlphaGo

Sup-Human Level Play (Same in Chess)

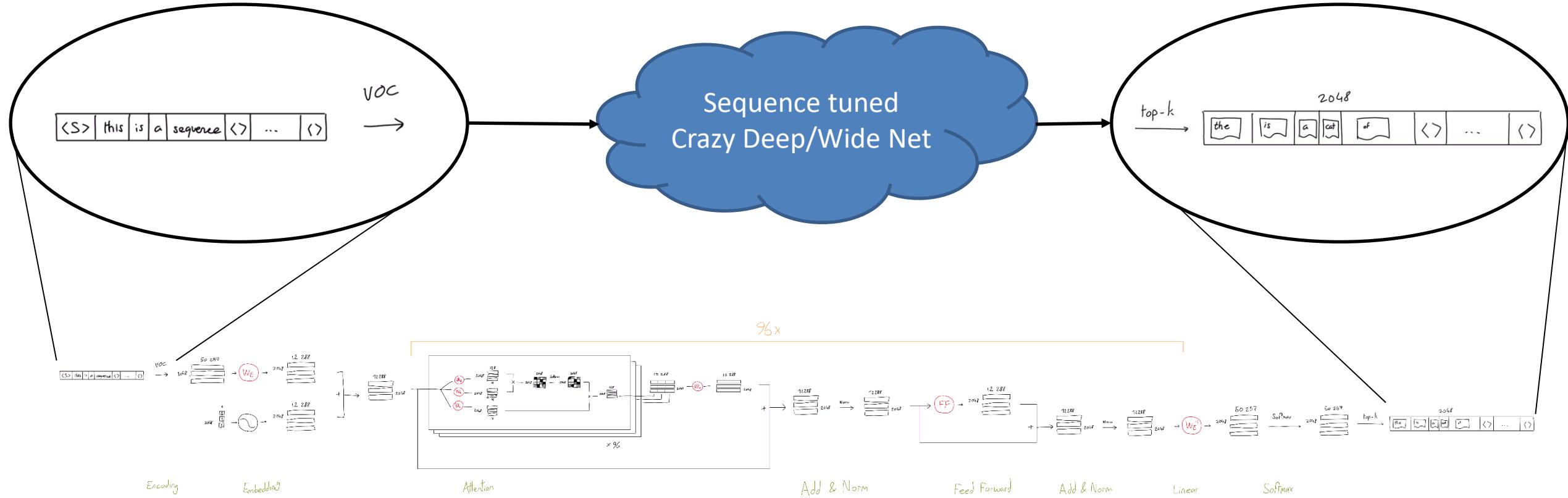


2020 OpenAI - GPT-3

- https://www.youtube.com/watch?v=Te5rOTcE4J4&t=105s&ab_channel=ColdFusion
- 175B parameters
- GPT-3 Paper on Learnit
Brown et al.

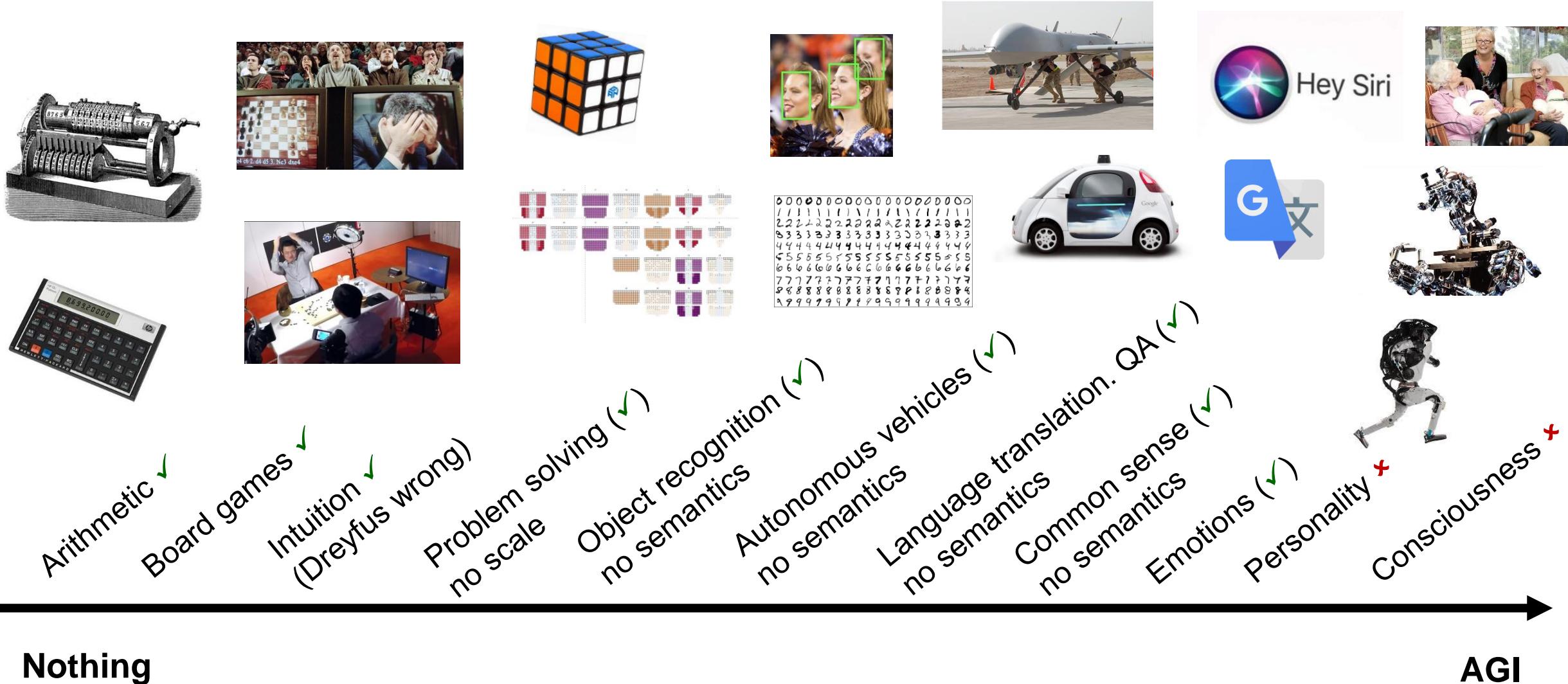


GPT-3 Architecture

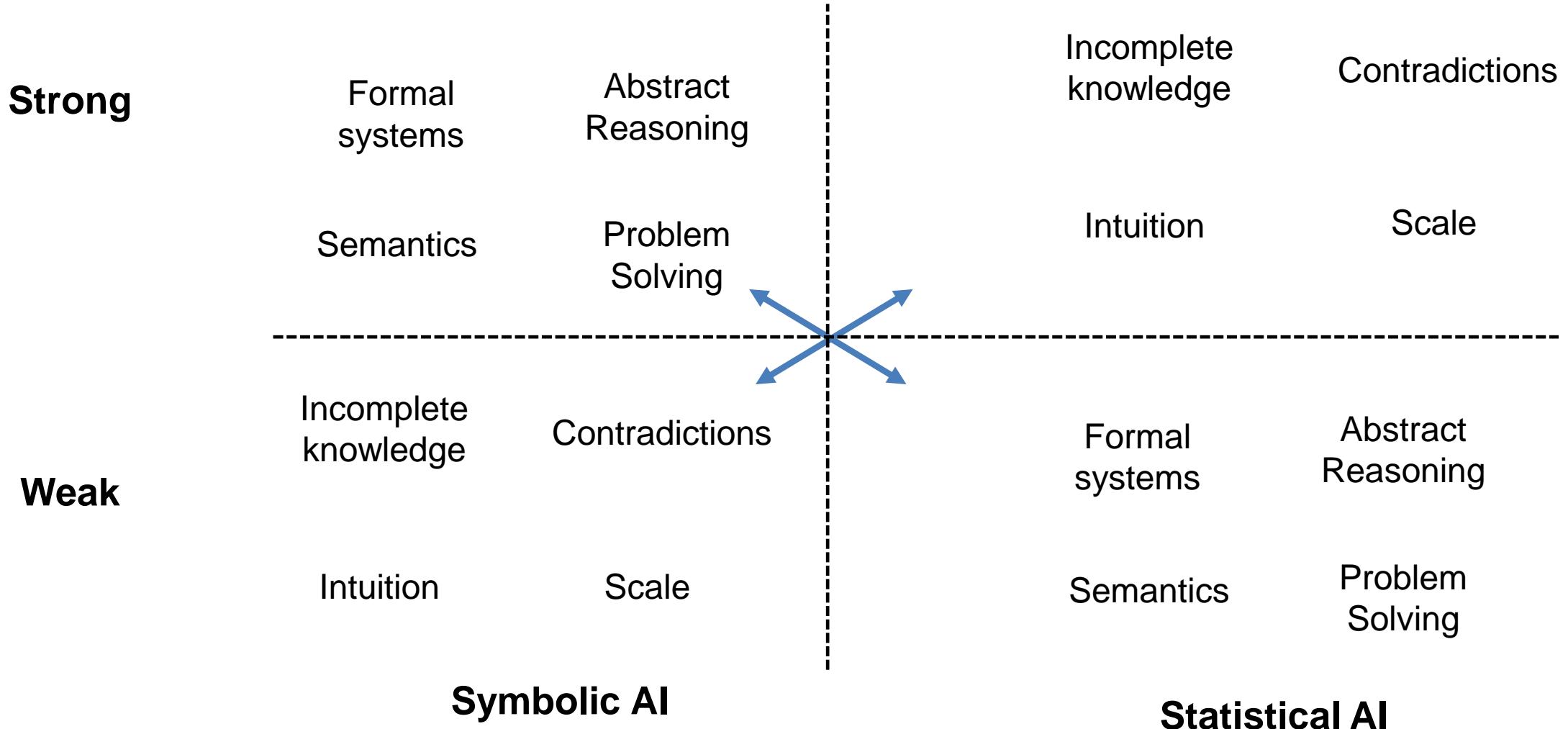


http://dugas.ch/artificial_curiosity/GPT_architecture.html

Path to Artificial General Intelligence (AGI)

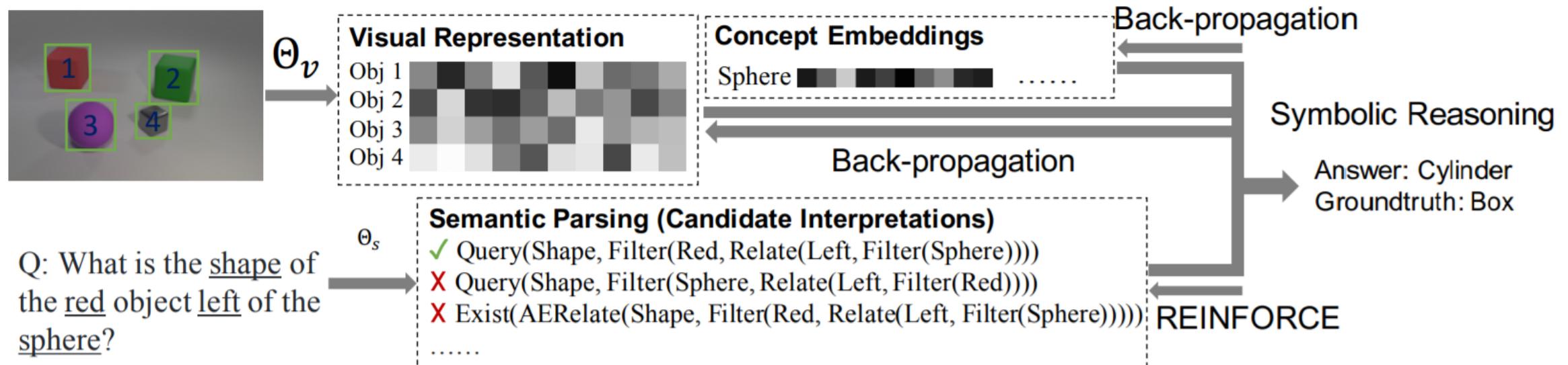


Symbolic AI versus Statistical AI



Symbolic Statistic AI Hybrids

- The Neuro-Symbolic Concept Learner
Paper on Learnit: Mao et al. 2019





2022 Rune Møller
Jensen

Intended Learning Outcomes

After the course, the student should be able to:

- **Identify** decision problems that can be solved or supported by symbolic AI and optimization algorithms.
- **Apply** advanced symbolic AI and optimization modeling techniques to describe these problems formally.
- **Implement** symbolic AI and optimization software components to solve these problems efficiently.
- **Apply standard** symbolic AI and optimization models and solvers.
- **Participate in concept development** of advanced decision support systems.

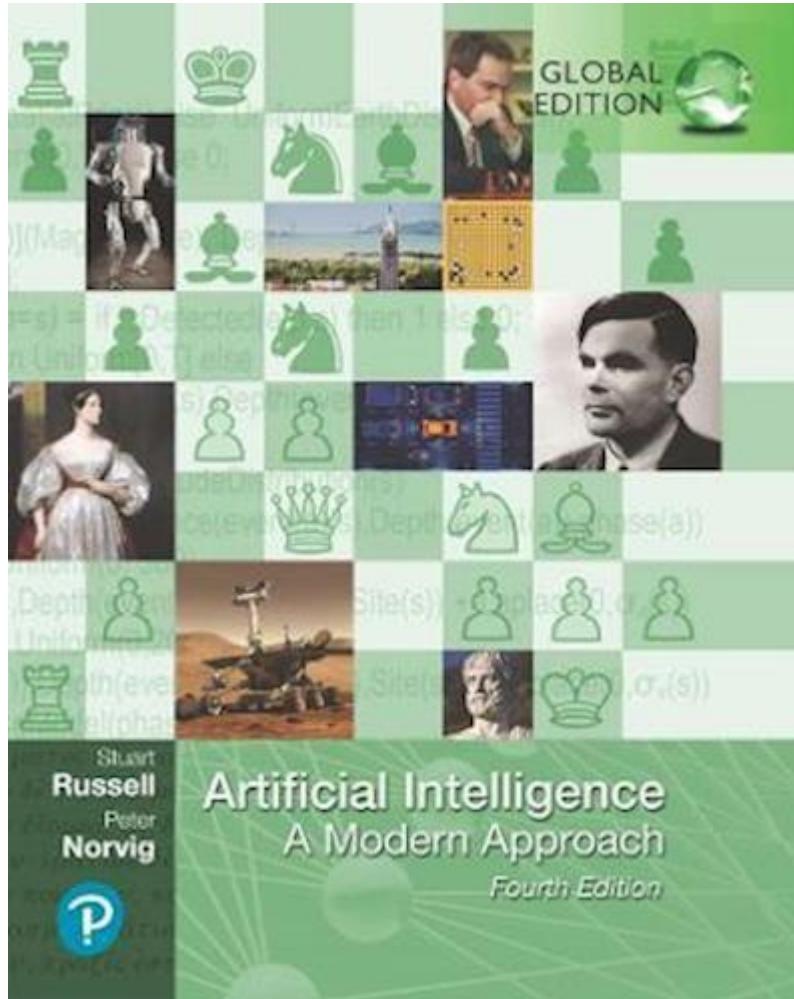
Prerequisites

	Softwaredesign			
1.semester	Software Engineering and Software Qualities 7,5 ECTS	Discrete Mathematics 7,5 ECTS	Introductory Programming 15 ECTS	
2.semester	Introduction to Database Design 7,5 ECTS	Algorithms and Data Structures 7,5 ECTS	Valgfag 7,5 ECTS	IAI
3.semester	Research Project 7,5 ECTS	Valgfag 7,5 ECTS	Specialisering 7,5 ECTS	Specialisering 7,5 ECTS
4.semester	Speciale 30 ECTS			

Overview of IAI 2022

- Students 139 (previous years: 136, 96, 151, 130, 95, 72)
- Course Manager: **Rune Møller Jensen** (rmj)
- Teaching Assistants:
Laurenz Aisenpreis (laai)
Alexandru Andrei Ardelean (aard)
Jonathan Mogensen (jmog)
Agita Solzemniece (agso)
Gustav Gyrst (gugy)
- Format: 12 (9) lectures, 10 (7) recitations
2 mandatory programming projects (out of 3 options)
3 mandatory homework problems (out of 9-10 options)
- 4 Hour written exam

Artificial Intelligence: A Modern Approach, Global Edition



https://www.saxo.com/dk/artificial-intelligence-a-modern-approach-global-edition_stuart-russell_paperback_9781292401133

saxo

IAI Schedule

Introduction to Artificial Intelligence

Lecture 2: Heuristic Search



Today's Program

- 10:00-10:55: Search Algorithms
 - Search problem definition
 - Search Tree Exploration
 - Best-First Search
- 11:05-12:00: Heuristic Search
 - Greedy best-first search
 - A* (including weighted A* and Uniform-cost search)
 - Heuristic Theory
- No exercises



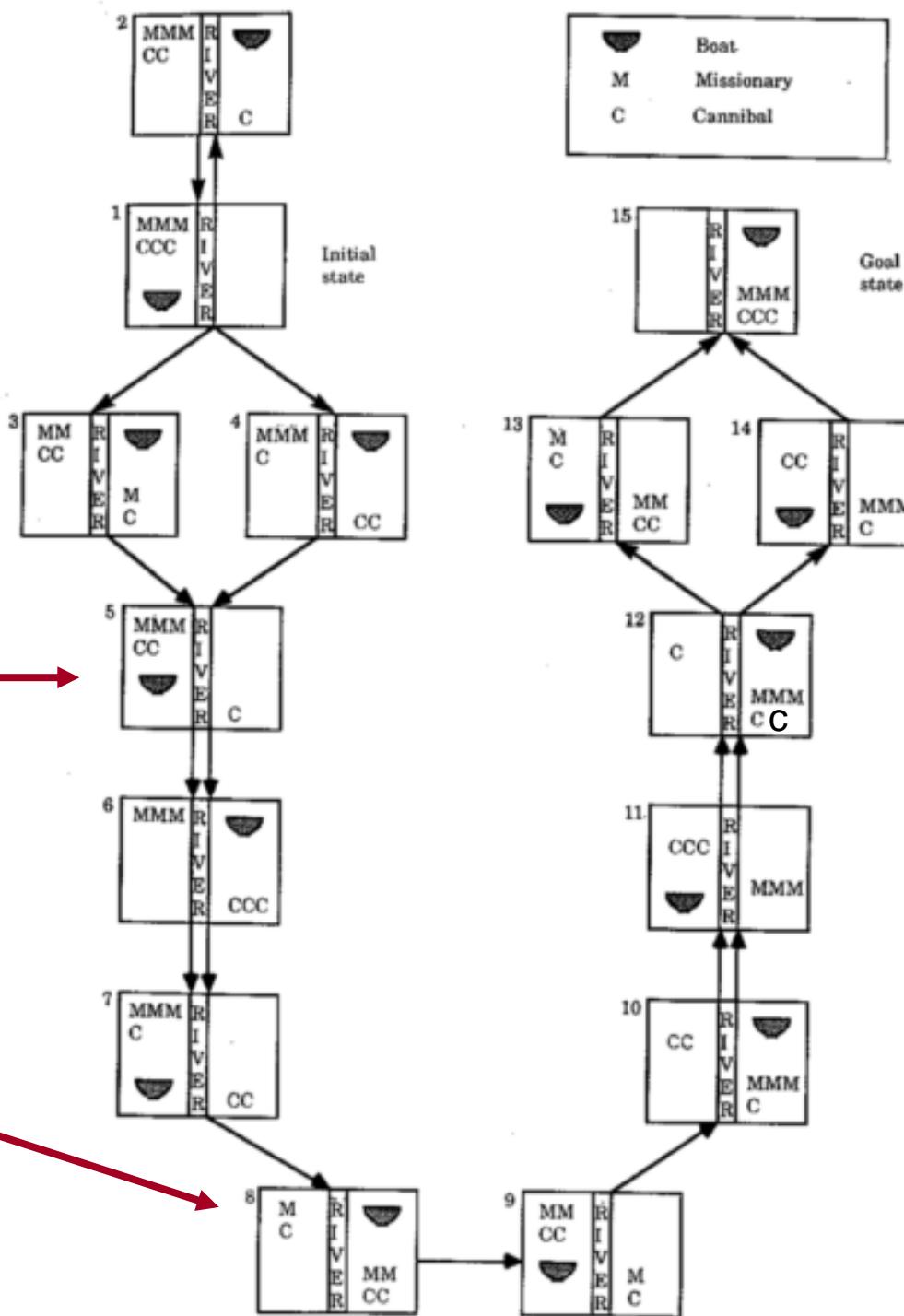
AI Search Problems



Human Problem Solving, 1971

Problem Space

Herbert Simon &
Allen Newell



What Characterizes AI Search?

- The search space is **implicitly** defined
- The search is **goal directed**
- The search space often grows **exponentially** with problem size
- Search actions are **abstract** representations of **real-world** actions
 - **Valid**: can be decomposed to atomic actions
 - **Useful**: “atomic” plans are easy such that states are decision points

How can we abstract sailing actions?

Assumptions about the search domain

Assumptions

- The environment is **static**
- Actions are **deterministic**
- States are **observable**
- States and actions are **discrete**

Hold

- Toy puzzles, board games, computers

Don't hold at all

- Anything real unless highly abstracted



Search Problem Definition

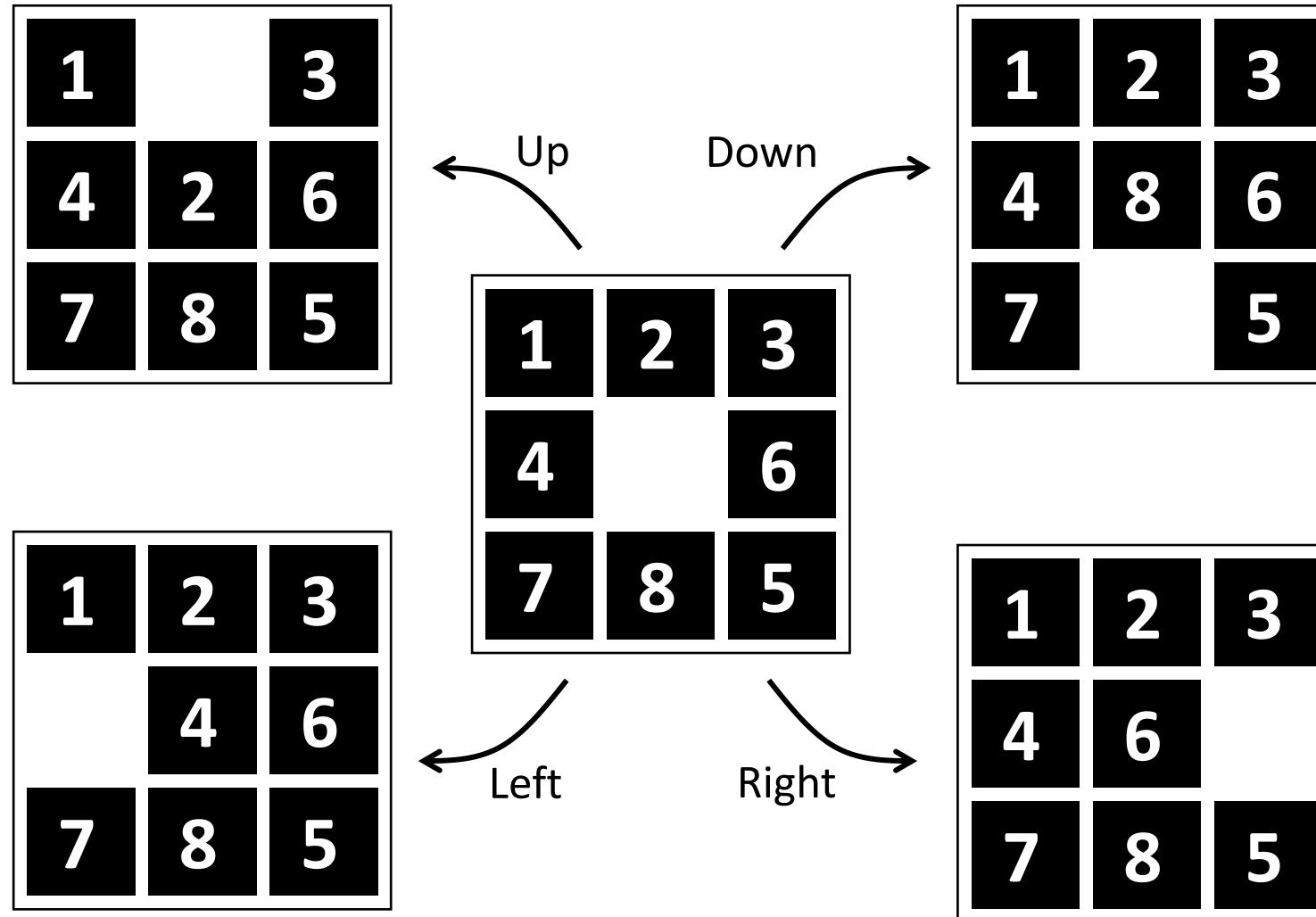
A **search problem** has 5 components:

1. An initial state s_0
 2. A set of actions. $\text{ACTIONS}(s)$ returns the set of actions applicable in state s
 3. A transition model. $\text{RESULT}(s,a)$ returns the state s' reached from s by applying a

1., 2., and 3. form a graph called the **state space**
 4. A set of goal states. $\text{Is-GOAL}(s)$ returns true iff s is a goal state
 5. An action cost function. $\text{ACTION-COST}(s,a,s') > 0$ returns the cost of taking action a to go from state s to state s'
- A solution is a path from the initial state s_0 to a goal state g
 - An optimal solution is a path with minimum sum of action costs

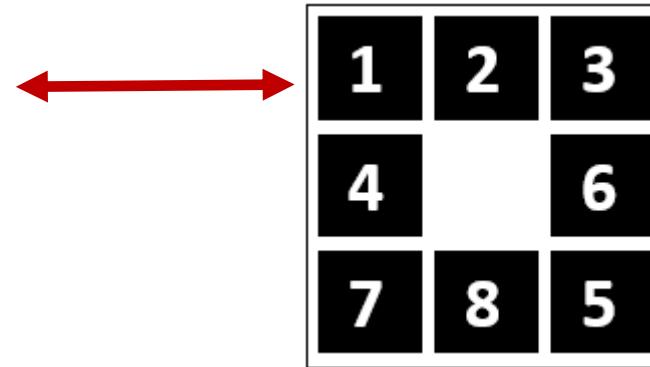


Example: 8-Puzzle

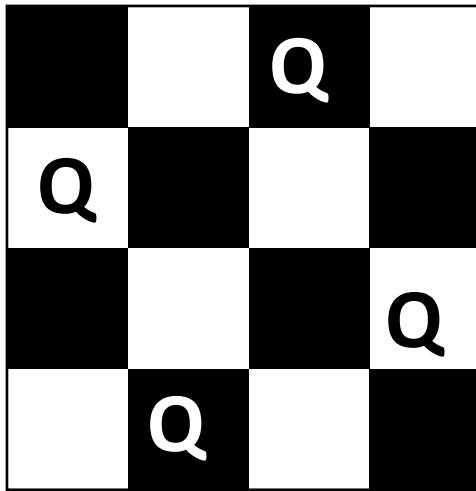


Example: 8-Puzzle

- Initial state: $s_0 = \langle 1, 2, 3, 4, *, 6, 7, 8, 5 \rangle$
- Goal test: $s = \langle 1, 2, 3, 4, 5, 6, 7, 8, * \rangle$
- Actions: {Up, Down, Left, Right}
- Transition model: Defined by the rules:
 - 1: Up (Down): applicable if some tile t above (below) *
 - 2: Left (Right): applicable if some tile t left (right) side of *
 - 3: The effect of actions is to swap t and *
- Size of state space: $9!/2$
- Action cost: 1 for all actions



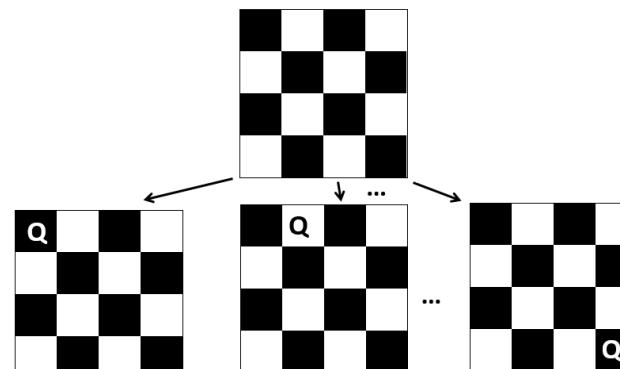
Example: 4-Queens



Example: 4-Queens

- Initial state: no queens on the board
- Goal test: 4 queens on the board, none can attack each other
- Actions: add a queen to an empty square
- Transition model: Returns board with queen added
- Size of state space: $16*15*14*13 / 4! = 1820$
- Action cost: irrelevant!

Can we do better?



Real Example: Ticket Search

The screenshot shows a flight search interface for a return trip from Copenhagen (CPH) to Marseille (MRS). The search parameters are: Rejsetype: Retur, Billettype: Economy, Date: 11 feb, 11 feb, 1 passager.

Søgning afsluttet 640 resultater

Stop:
○ Direkte
○ Maks. 1 stop kr. 2.500
● Alle kr. 2.500

Maksimal rejsetid: 20h11m

Tid på dagen (Udrejse): Afrejse / Ankomst: 00:00 - 14:00

Tid på dagen (Hjemrejse): Afrejse / Ankomst: 10:00 - 23:59

Billetter **Ticket Builder**

Billigste 2.882 DKK 9t 12m (Gnm.snit)
Hurtigste 4.637 DKK 4t 07m (Gnm.snit)
Bedste 3.655 DKK 4t 45m (Gnm.snit)

3 filtre sat 247 resultater passer på dine filtre. **FJERN ALLE FILTRE**

Brussels Airlines
CPH 06:50 København 5t 05m 1 STOP 11:55 MRS Marseille Economy 6.9

Air France
MRS 20:00 Marseille 13t 20m SKIFT LUFTHAVN 09:20 +1 CPH København Vælg 2.882 DKK

Brussels Airlines
CPH 06:50 København 5t 05m 1 STOP 11:55 MRS Marseille Economy 6.7

Air France
MRS 19:25 Marseille 13t 55m 1 STOP 09:20 CPH København 2.882 DKK

Search Algorithms

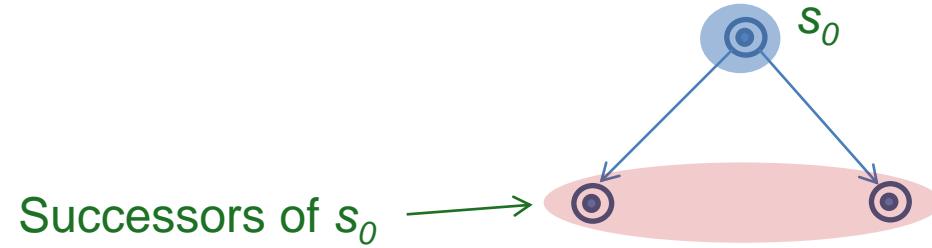


Search Tree Exploration

Start from search
node with s_0

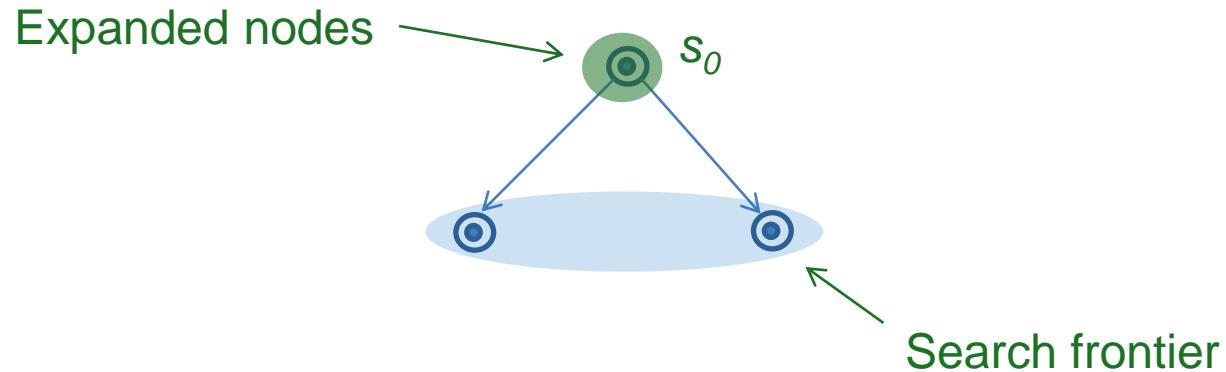


Search Tree Exploration

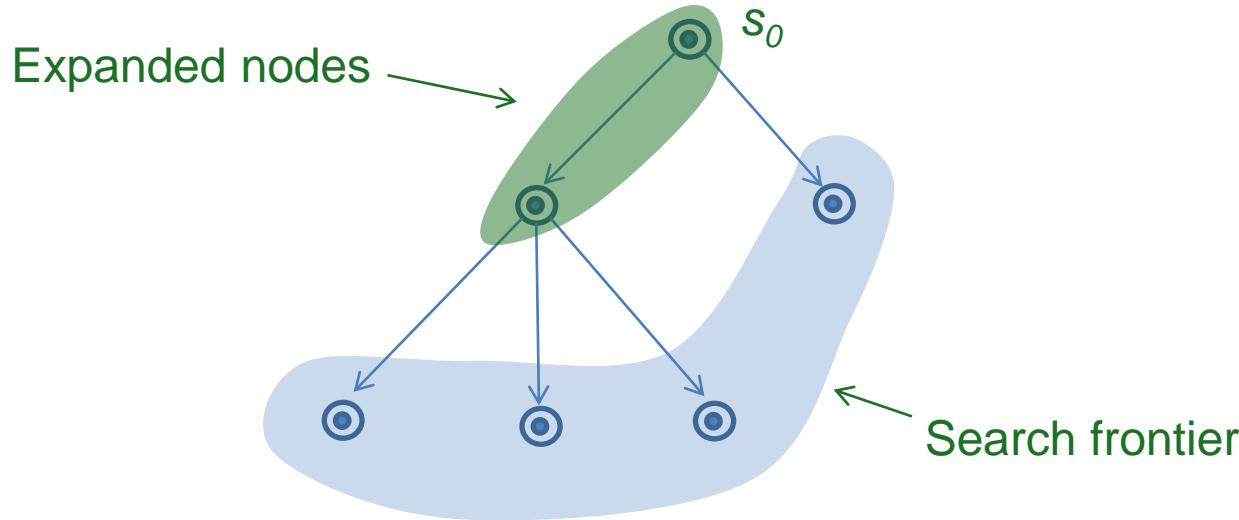


Search Tree Exploration

Expand s_0



Search Tree Exploration



Algorithms differ by which frontier node they expand (search strategy)!

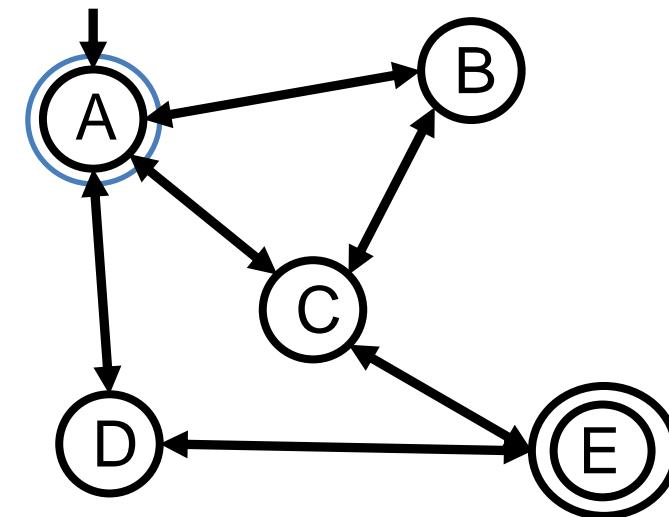
What is the depth-first and breadth-first search strategy?

Example

Search Tree

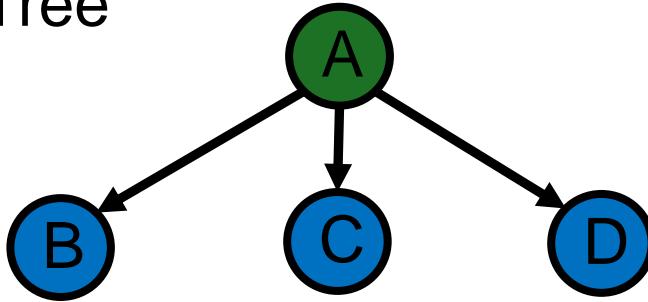


State Space

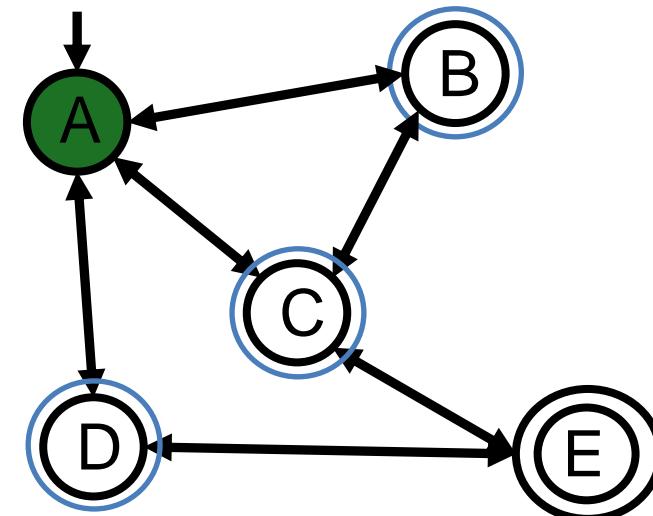


Example

Search Tree

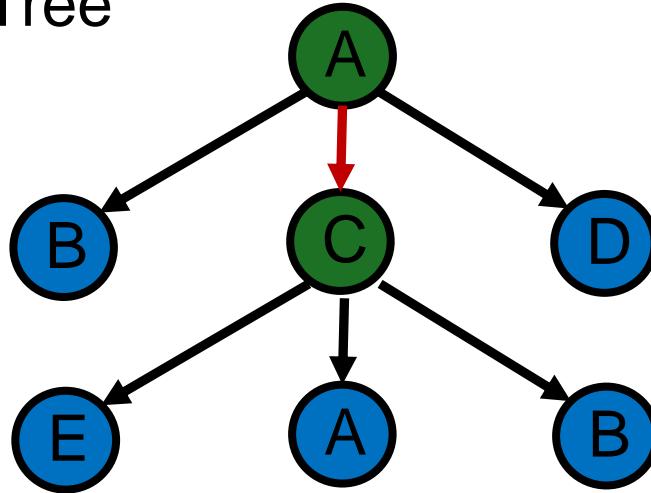


State Space

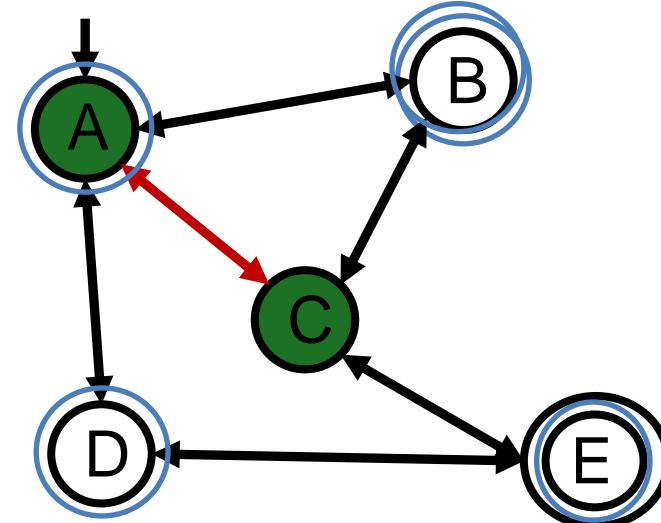


Example

Search Tree

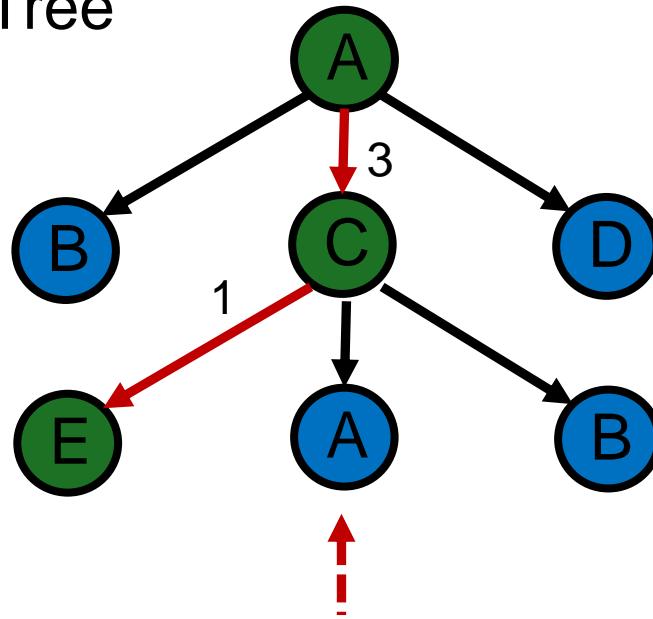


State Space



Example

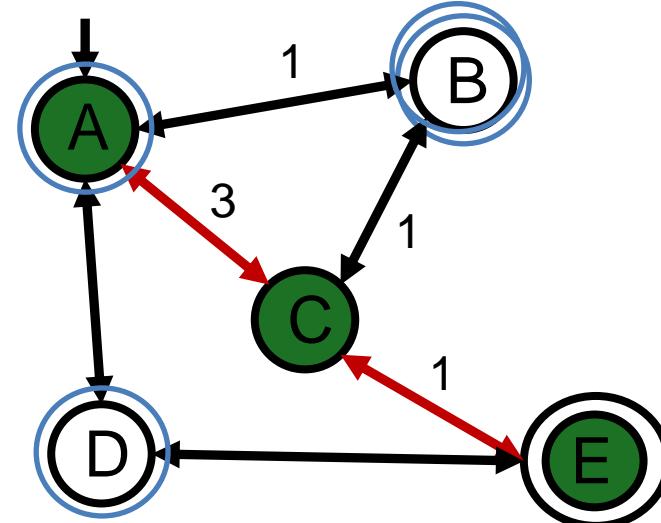
Search Tree



Looping back to the same state
is always bad (why?)

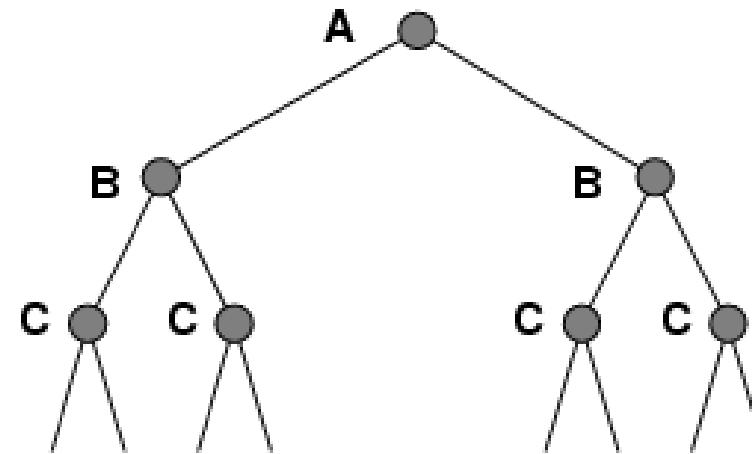
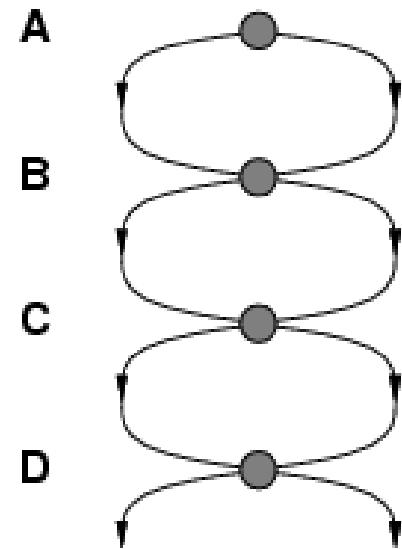
Wrt. Path cost, is it also always
bad to explore alternative paths
to the same state?

State Space



Redundant Paths

- Failure to detect repeated states can turn a linear problem into an exponential one!



Performance of Search Algorithms



Performance characteristic

- **time complexity:** number of nodes generated
- **space complexity:** maximum number of nodes in memory
- **(soundness:** is the produced solution correct)
- **completeness:** does it always find a solution if one exists
- **optimality:** does it always find a least-cost solution?

Time and space complexity

Implicit problem representation:

Number of bits in the input not a good measure of problem size.

Input size is measured in terms of

- b : maximum **branching factor** of the search tree
- d : **depth** of the shallowest goal node
- m : maximum **depth** of the state space
(may be infinite)



Big-O Notation (RN21 Appendix A)

$T(n)$ is $O(f(n))$ if $\frac{T(n)}{f(n)} \rightarrow k$ for $n \rightarrow \infty$

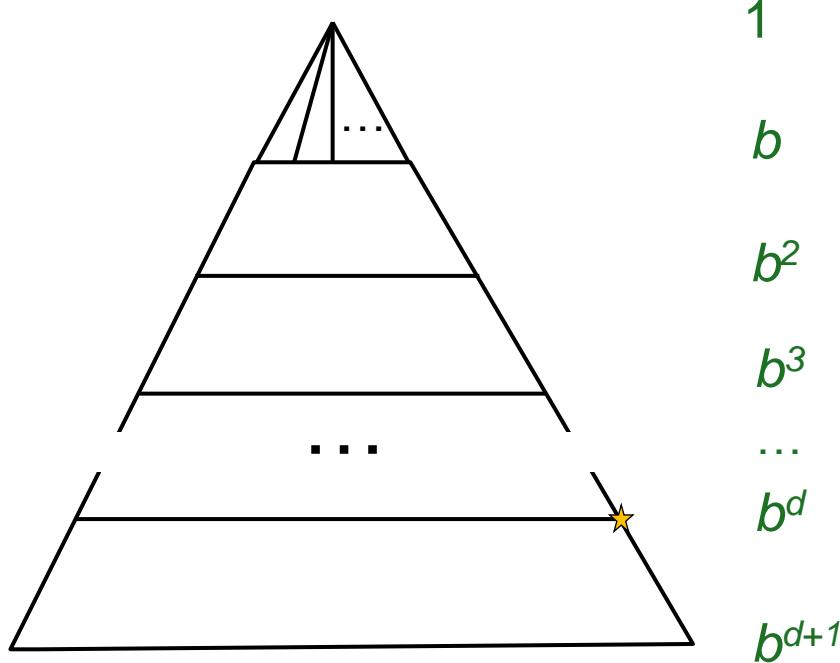
Ex.: $3n^2 + 4n^4$ is $O(n^4)$

Ex.: $3n^2 + 4n^4$ is $O(2^n)$

```
q ← 0  
for i = 1 to n  
  for j = 1 to i  
    q ← q + 1
```



Example: Breath-First Search

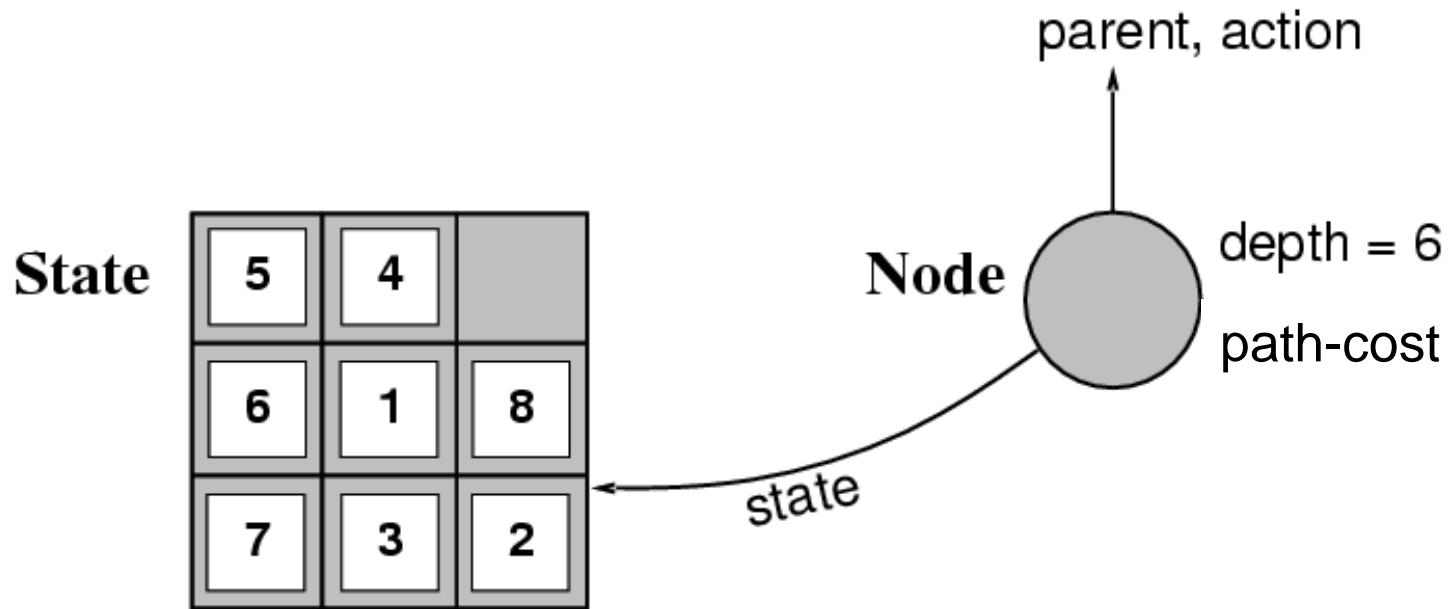


$$O\left(\sum_{i=0}^{d+1} b^i\right) = O(b^{d+1})$$

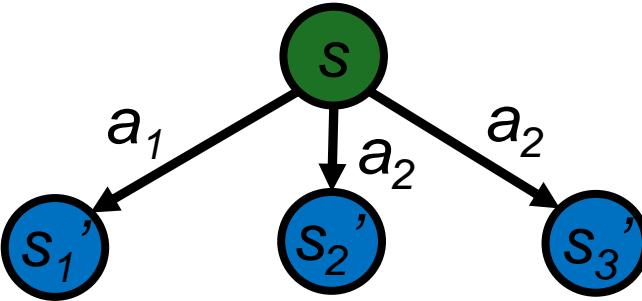
Best-First Search



Search Node Data Structure



Expand Function



Pseudocode definition in
RN21 Appendix B

Function EXPAND (*problem, node*) **yields** *nodes*

s \leftarrow *node.STATE*

for each *action* **in** *problem.ACTIONS(s)* **do**

s' \leftarrow *problem.RESULT(s,action)*

cost \leftarrow *node.PATH-COST + problem.ACTION-COST(s,action,s')*

yield NODE(STATE=*s'*, PARENT= *node*, ACTION= *action*, PATH-COST = *cost*)

Queue or Priority Queue

IS-EMPTY(*frontier*)

POP(*frontier*)

ADD(*node, frontier*)

Best-First Search

Expands frontier nodes with lowest cost according to an **evaluation function $f(n)$**

(i.e., frontier is a prioritized queue)



Best-First Search

Function BEST-FIRST-SEARCH(*problem,f*) **returns** a solution node or *failure*

node \leftarrow NODE(STATE = *problem.INITIAL*, PATH-COST = 0)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry with key *problem.INITIAL* and value *node*

while not Is-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.Is-GOAL(node.STATE)* **then return** *node*

for each *child* in Expand(*problem,node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* or *child.PATH-COST < reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

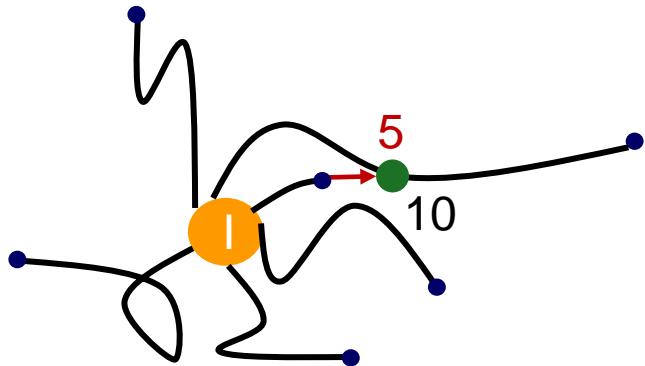
add *child* **to** *frontier*

return *failure*

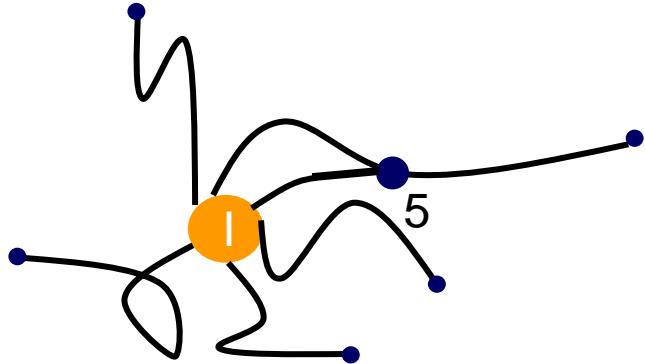


Properties of Best-First Search

- Complete in the sense that all states that can be reached are traced by some path
- Keeps track of a lowest cost path found so far to each reached state
(why is an update reached state added to the frontier?)



Properties of Best-First Search



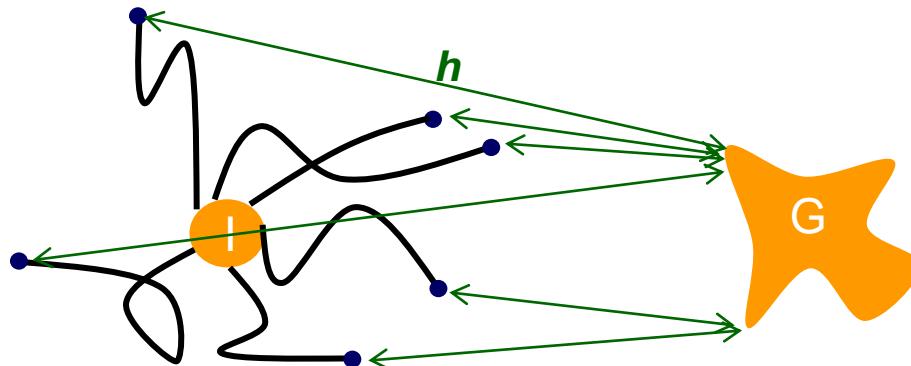
- Complete in the sense that all states that can be reached are traced by some path
- Keeps track of a lowest cost path found so far to each reached state
(why is an update reached state added to the frontier?)
- It removes loops and redundant paths
(how?)
- But best-first search is not guaranteed to find optimal solutions *(why?)*

Heuristic Search



Informed Search

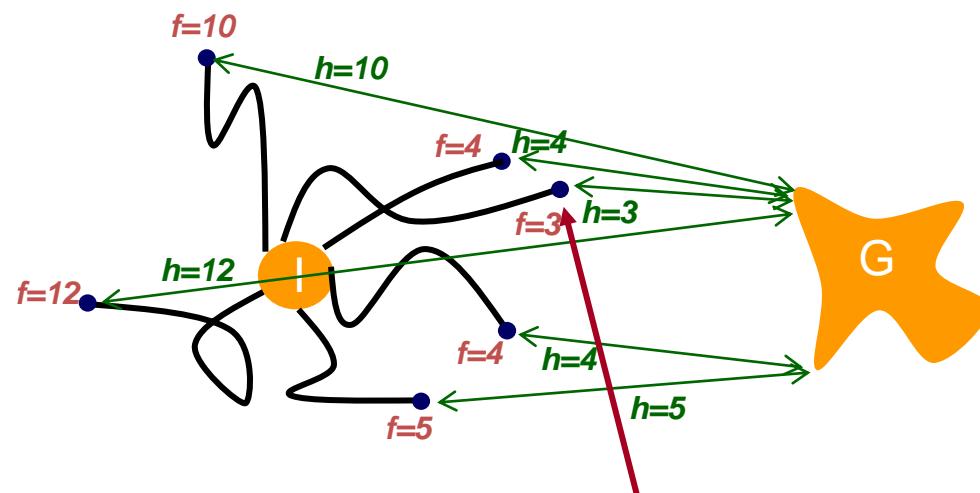
- Idea: use problem specific knowledge to pick which node to expand
- Typically involves a **heuristic function** $h(n)$ estimating the cheapest path from $n.\text{STATE}$ to a goal state



Requirements
 $h(s) \geq 0$
 $h(goal) = 0$

Greedy Best-First Search

- $f(n) = h(n)$: Expand a node that appears to be closest to the goal



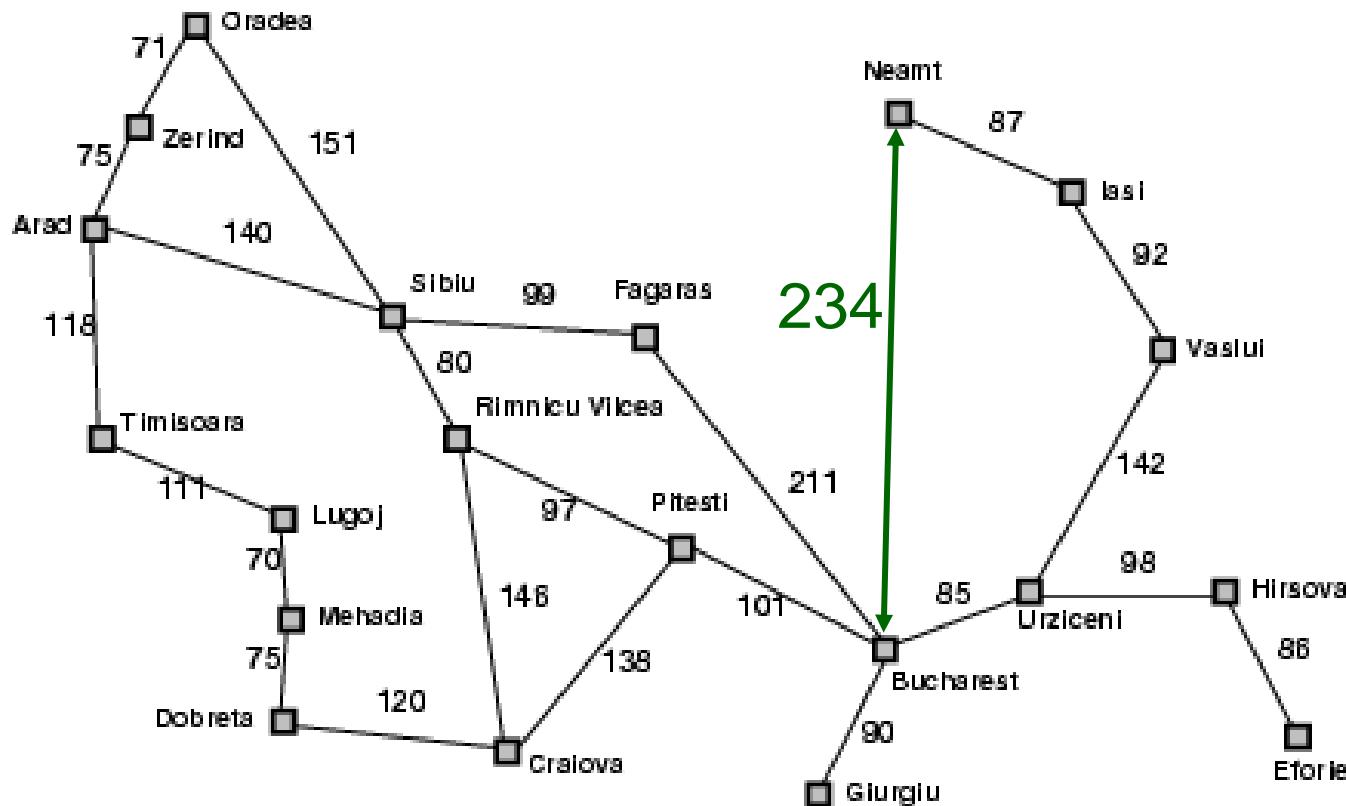
This node would be expanded next

Greedy Best-First Search Example

- Go from Arad to Bucharest
- $f(n) = h_{SLD}(n) =$
straight-line distance from $n.\text{STATE}$ to Bucharest



Romania with Action Costs in km

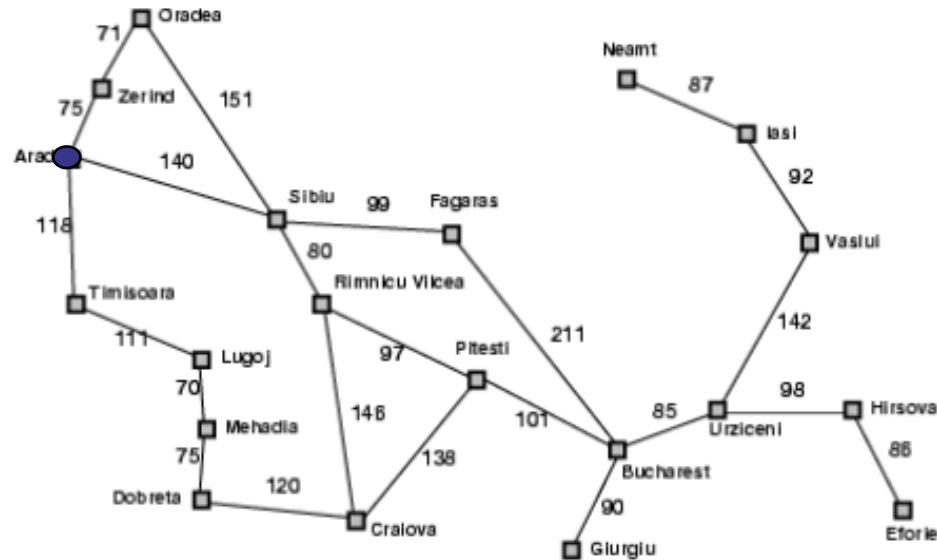


$h(n)$

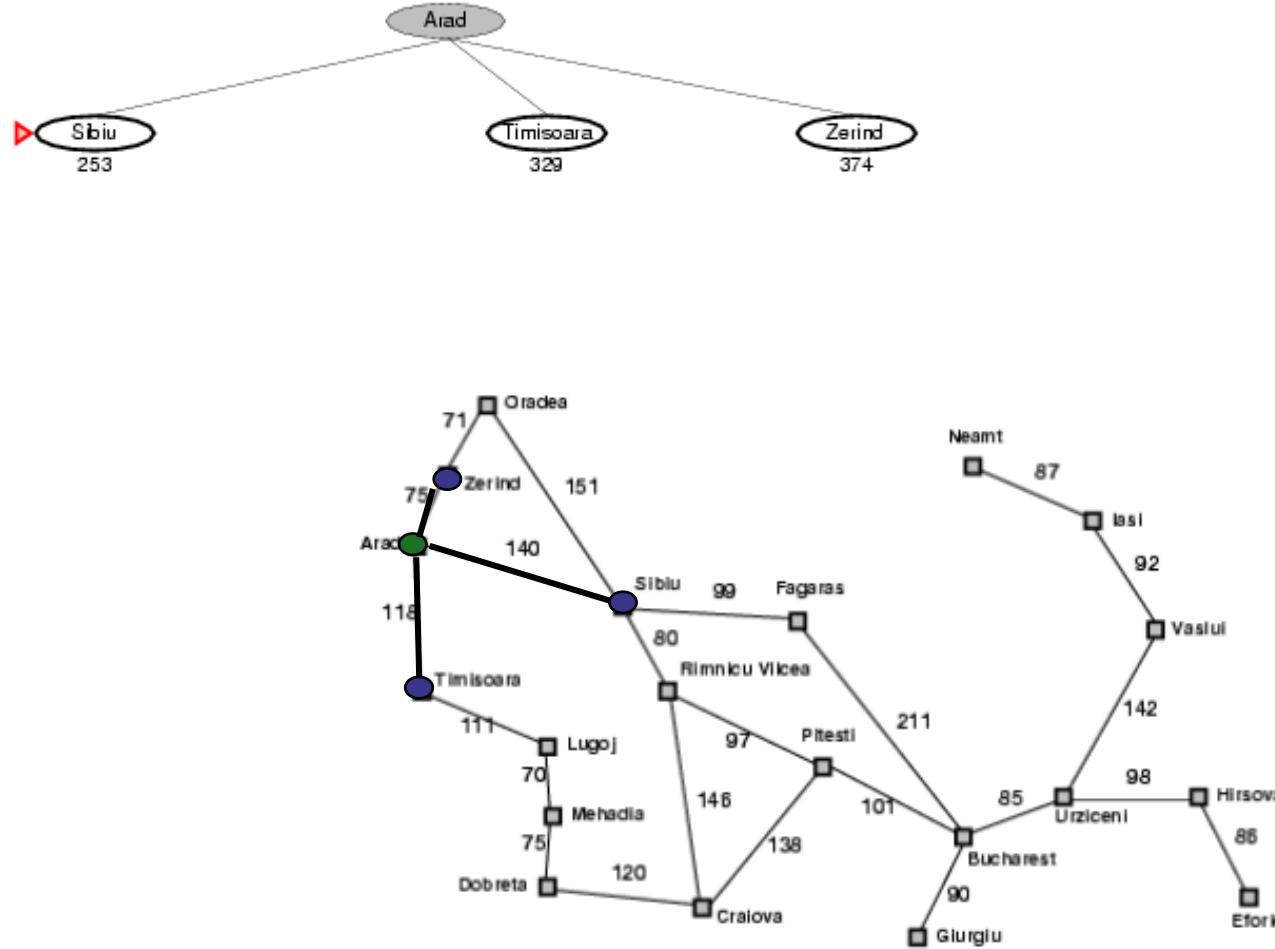
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search Example

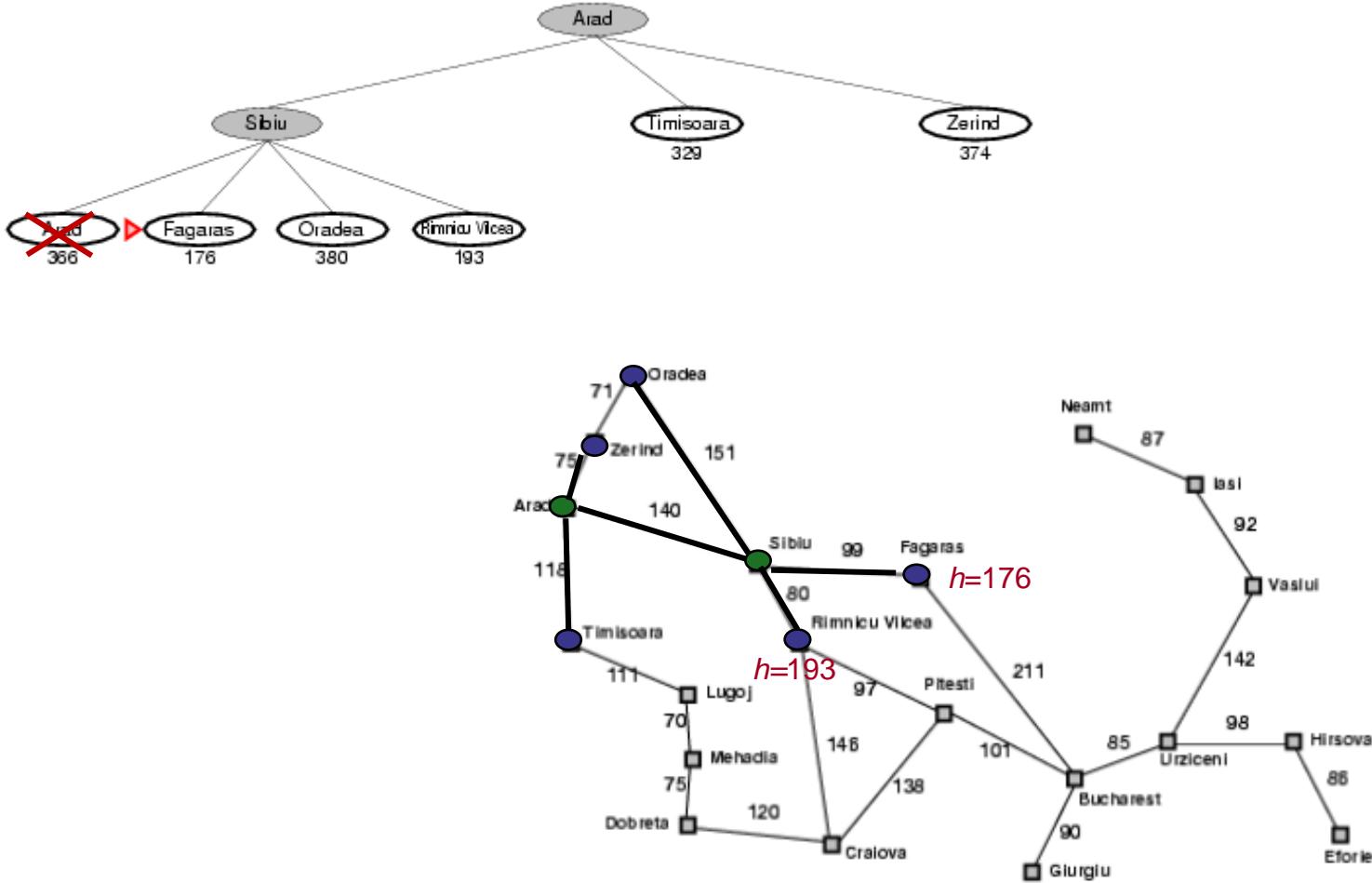
► Arad
366



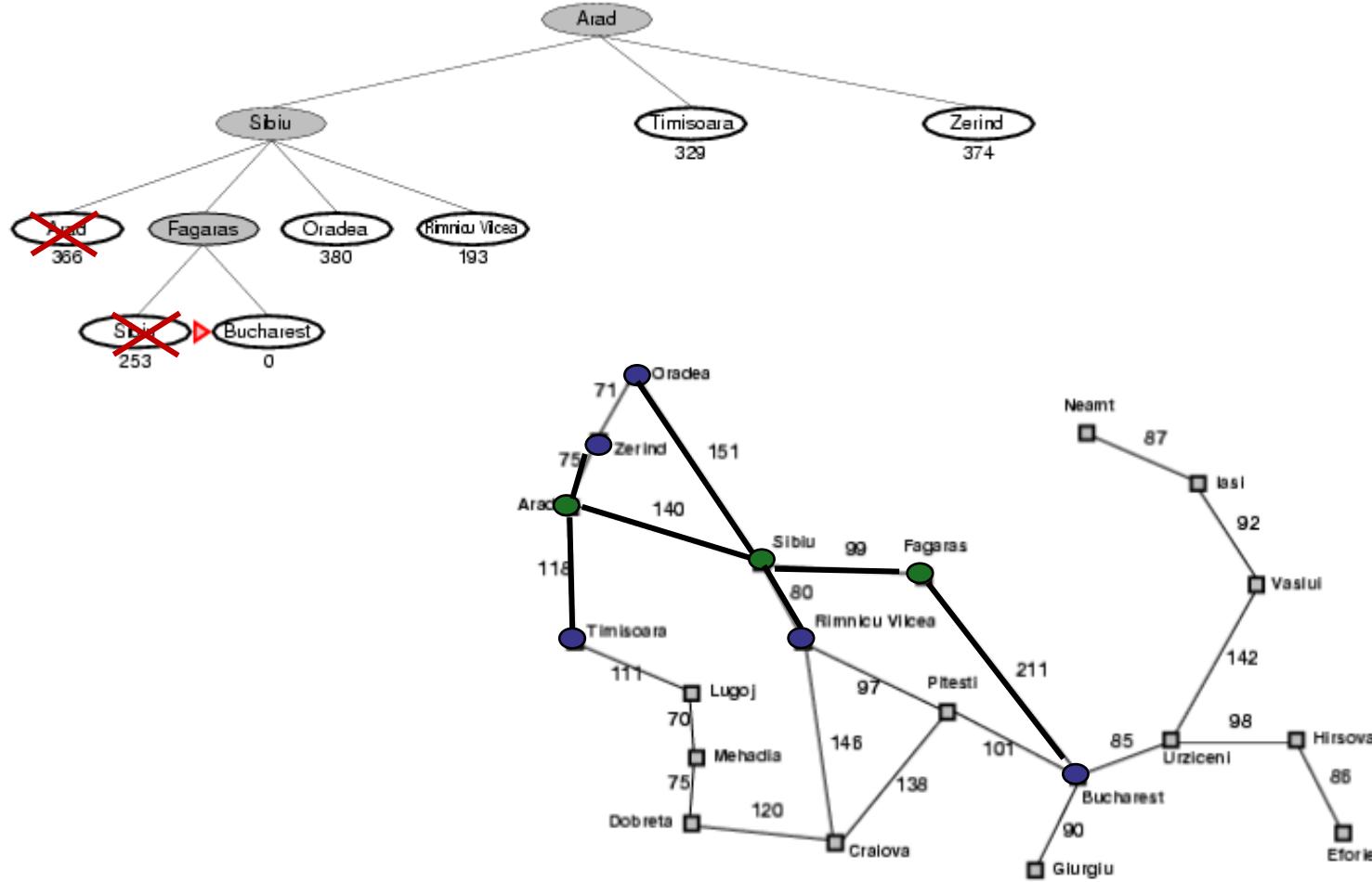
Greedy Best-First Search Example



Greedy Best-First Search Example

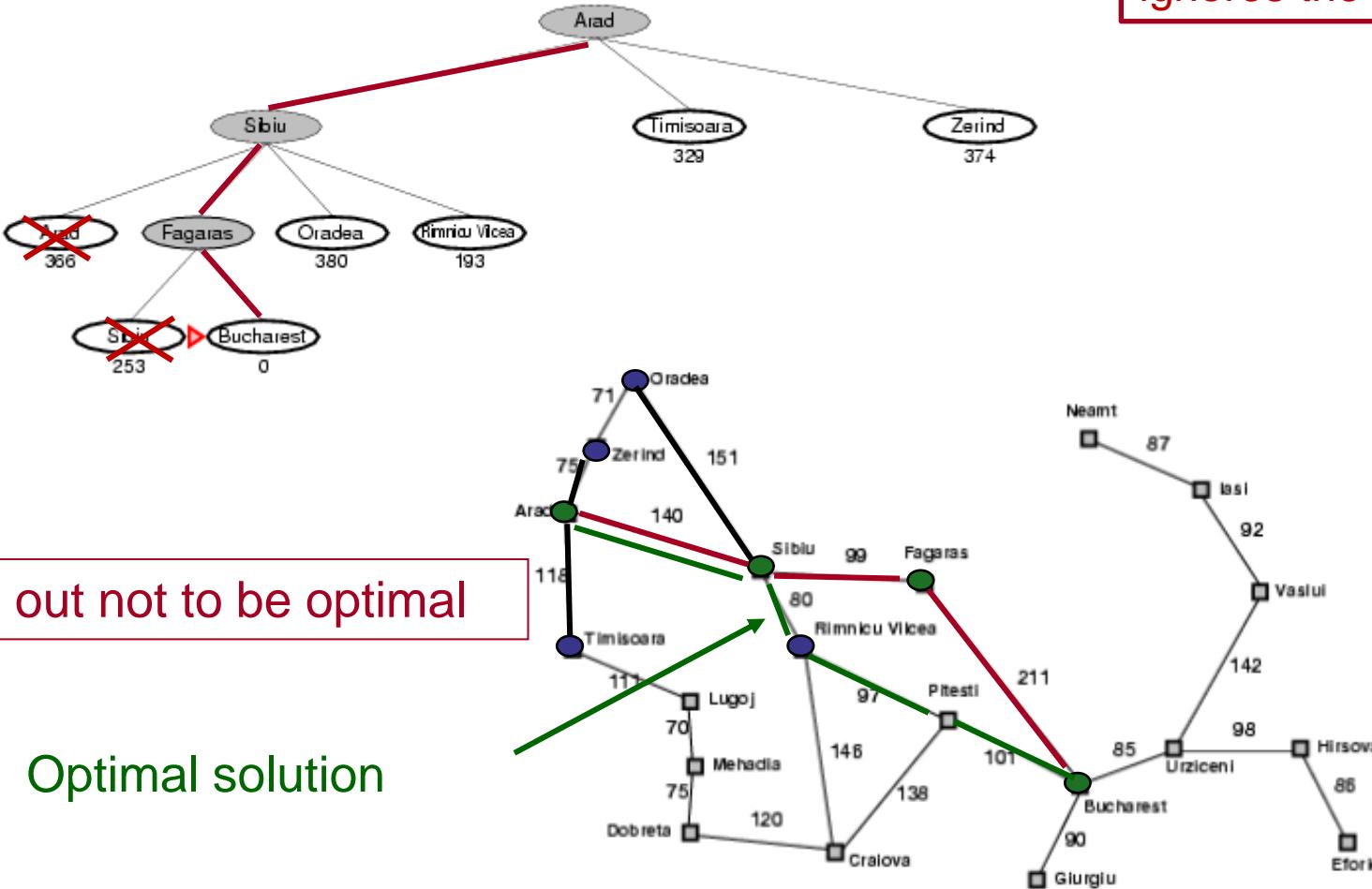


Greedy Best-First Search Example



Greedy Best-First Search Example

Issue: Greedy Best First ignores the path-cost.



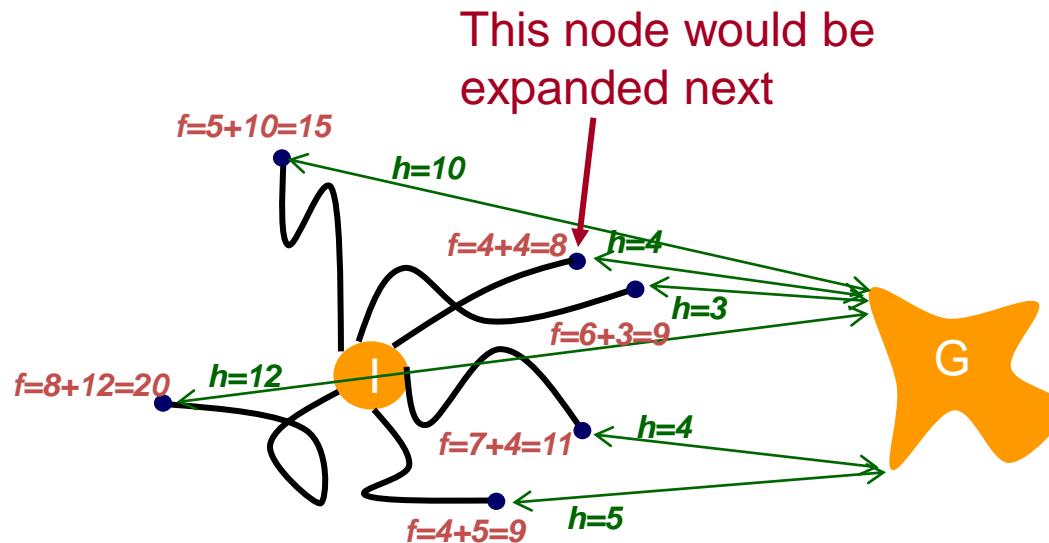
A* Search

- Idea: include cost of reaching node
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost of reaching n
- $h(n)$ = estimated cost of reaching goal from state of n
- $f(n)$ = estimated cost of the cheapest path to a goal state that goes through path of n



A* Search

- $f(n) = g(n) + h(n)$: Expand node that appears to be on cheapest paths to the goal



Admissible Heuristics

- A heuristic $h(n)$ is **admissible** if for every node n ,
 $h(n) \leq h^*(n)$, where $h^*(n)$ is the **minimum** cost to reach the goal state from n
- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Can you think of a heuristic function that is trivially admissible?

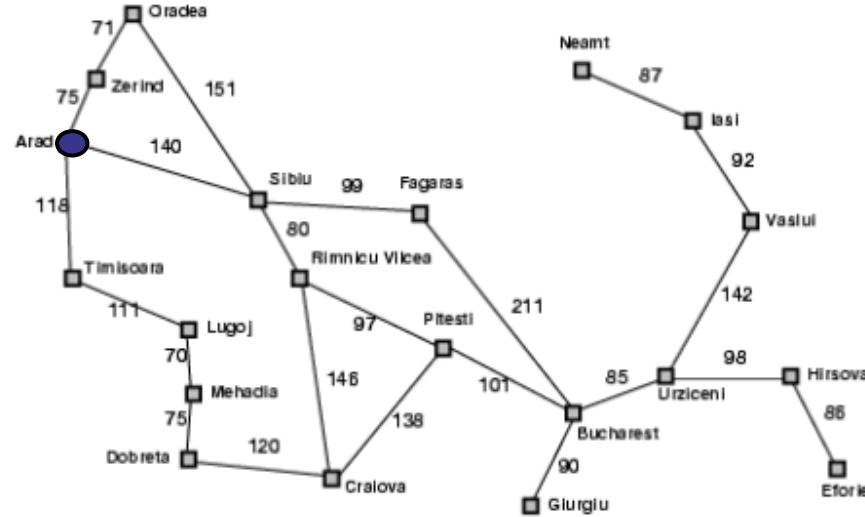
A* Search Example

- Go from Arad to Bucharest
- $f(n) = g(n) + h_{SLD}(n)$

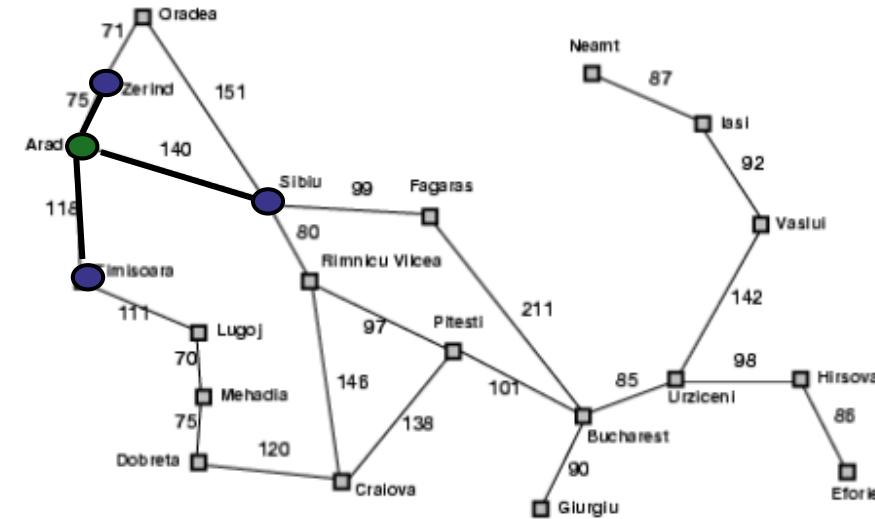


A* Search Example

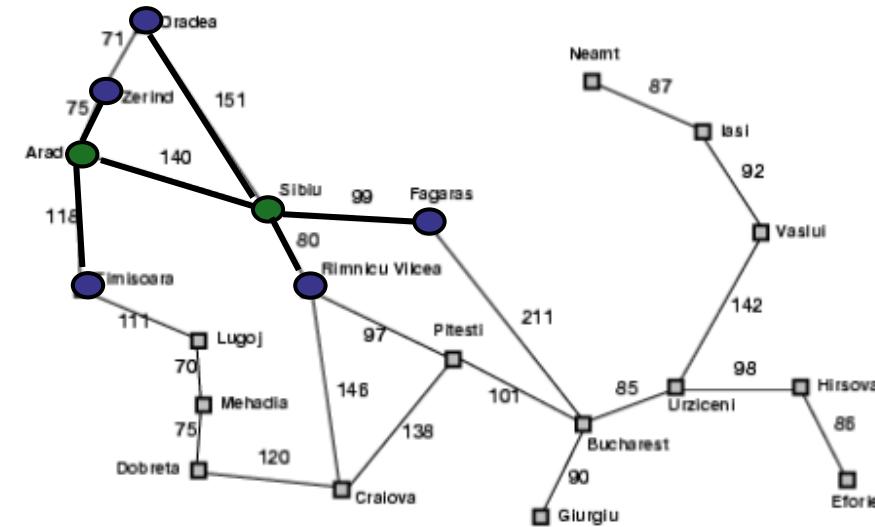
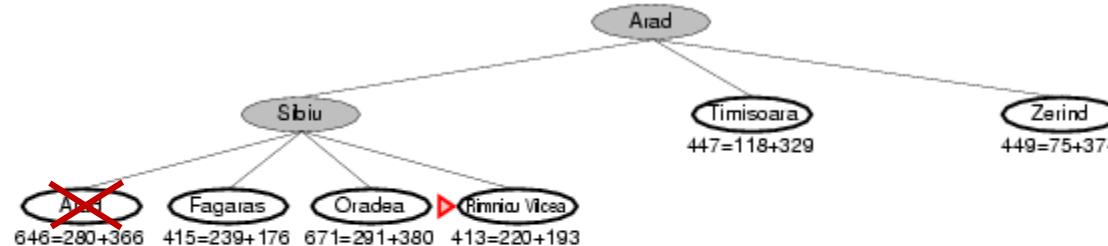
► Arad
366=0+366



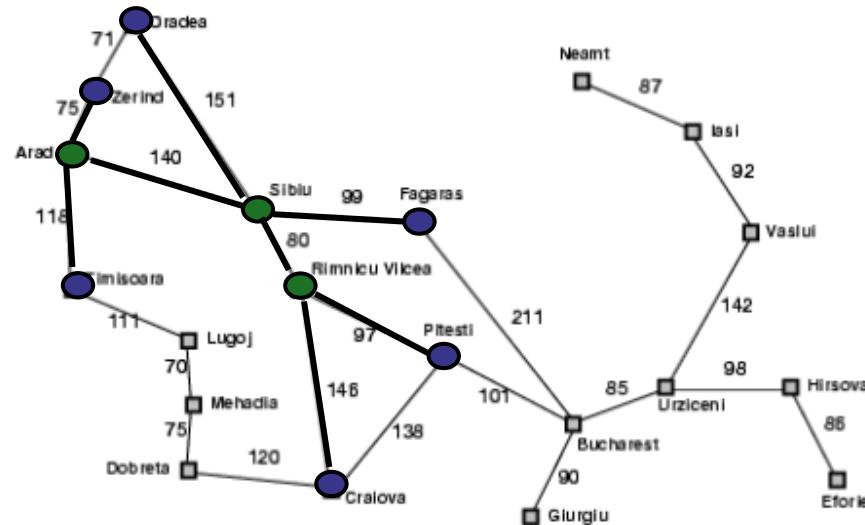
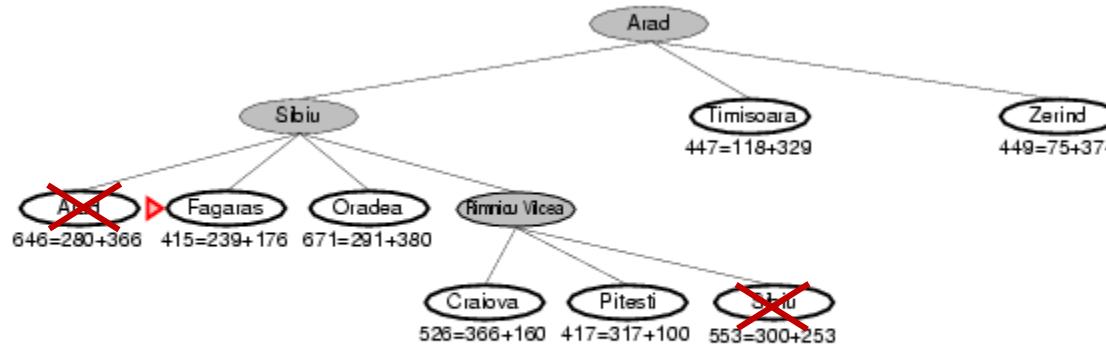
A* Search Example



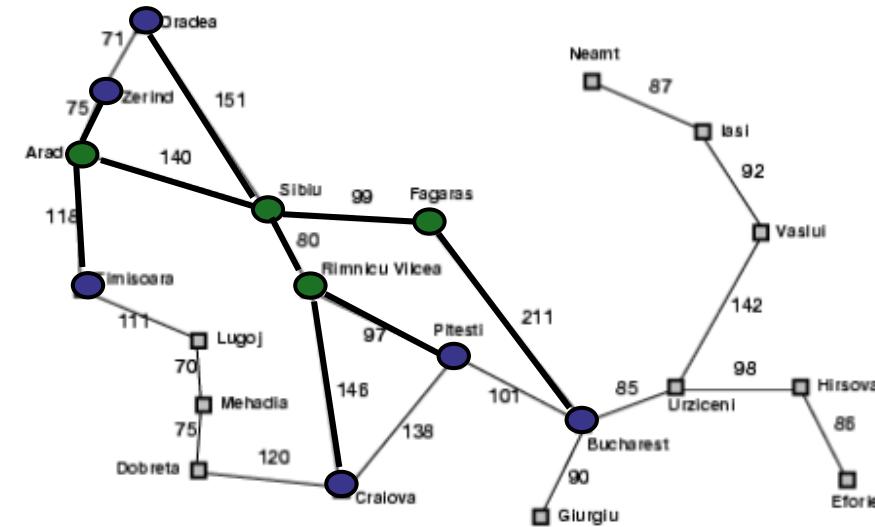
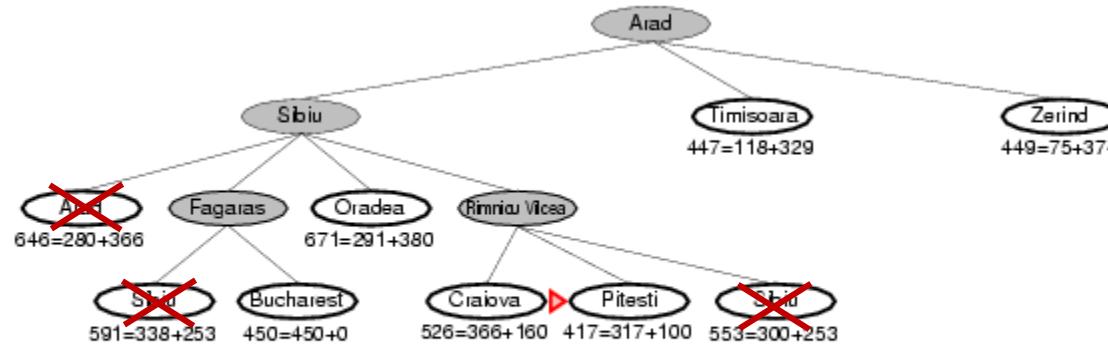
A* Search Example



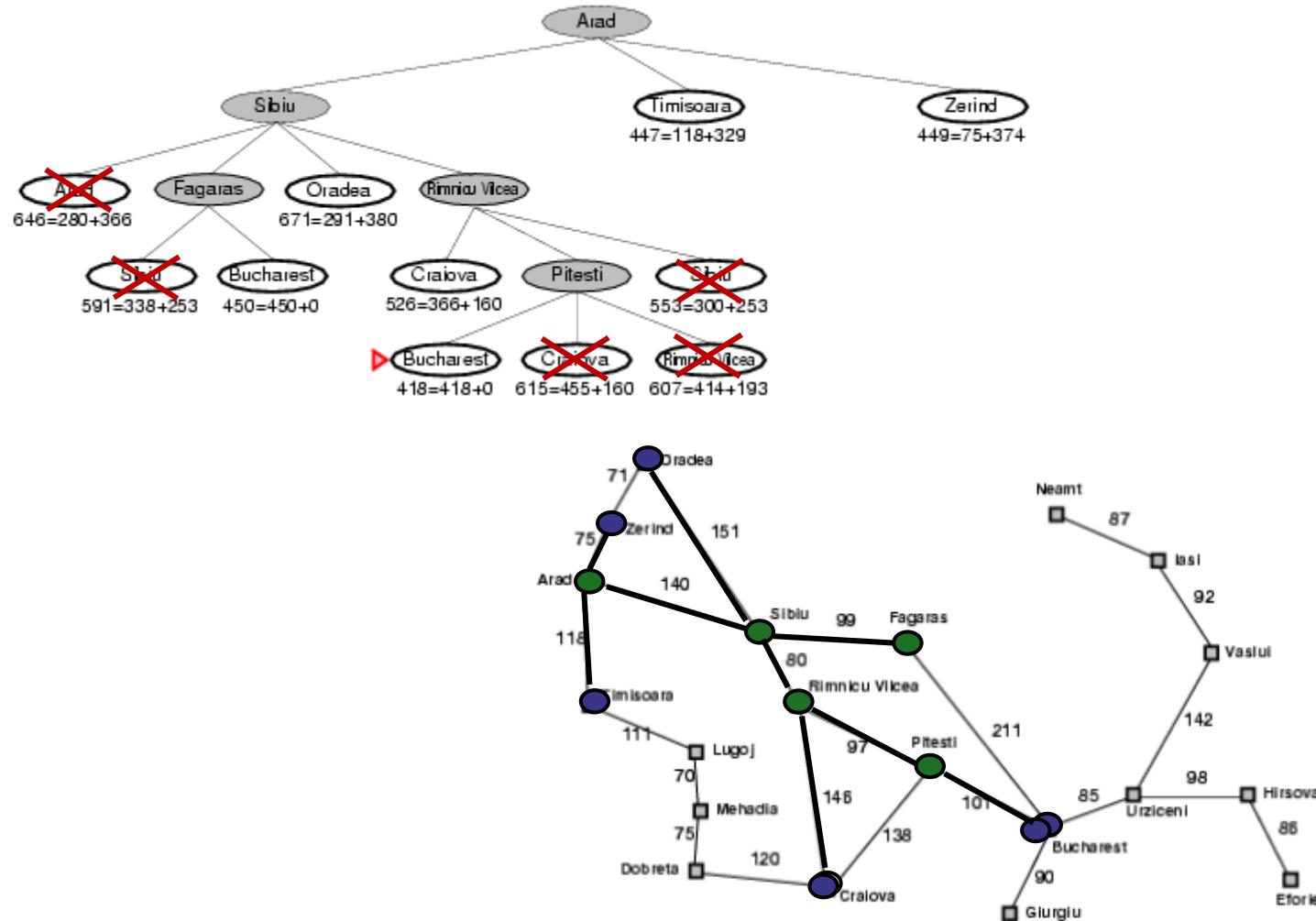
A* Search Example



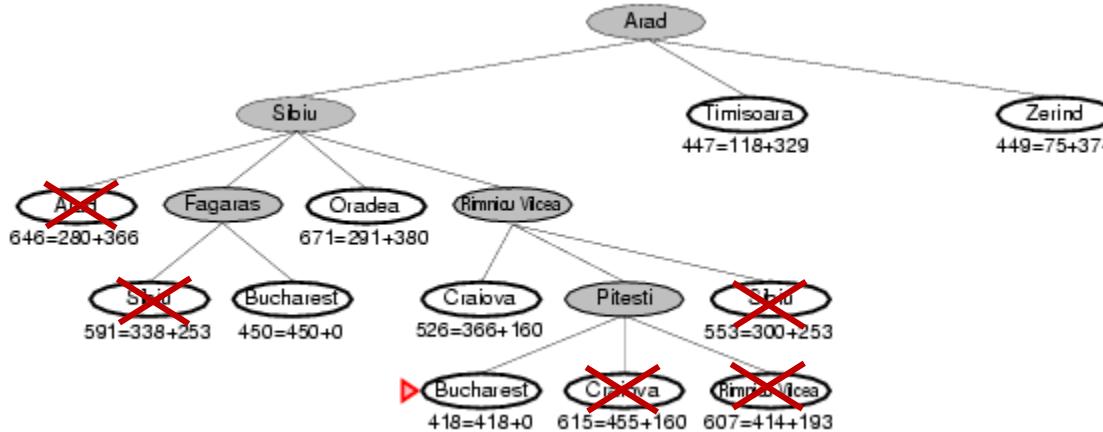
A* Search Example



A* Search Example

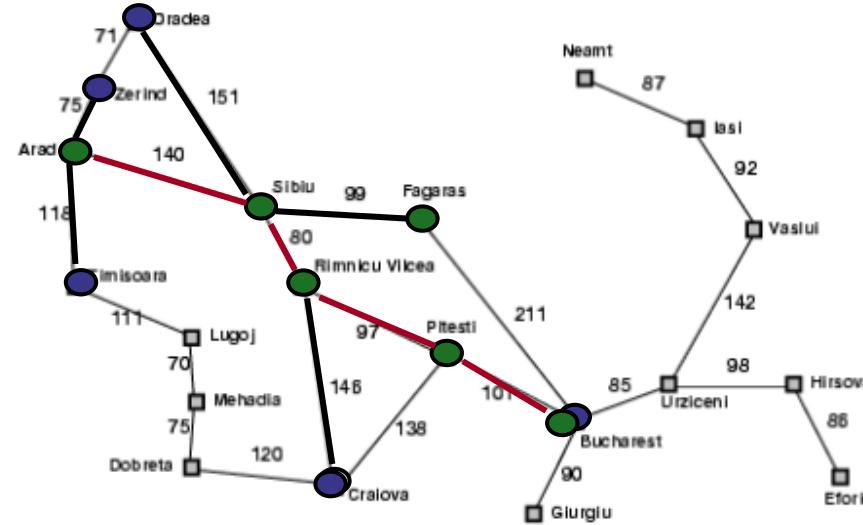


A* Search Example



$f(n)$ on an optimal path will be smaller than $f(goal)$ of a suboptimal path.

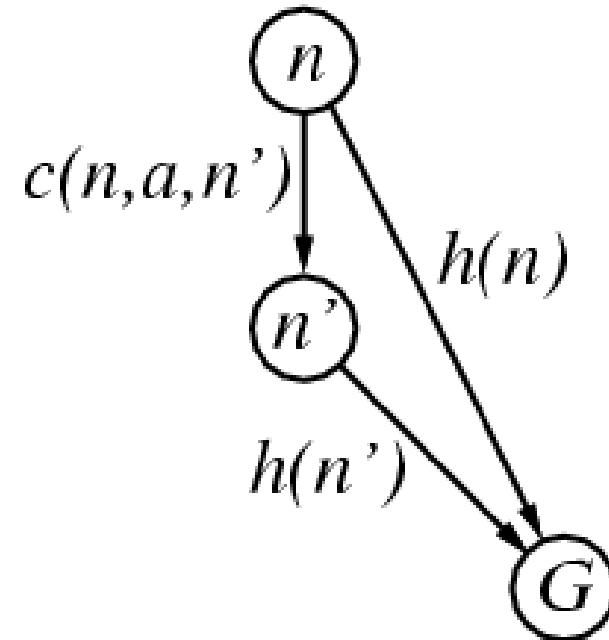
When $h(n)$ underestimates the cost of reaching a goal (h is admissible), A* is optimal



Consistent Heuristics

- A heuristic is **consistent** if for every node n , every successor n' of n generated by any action a :

$$\begin{aligned} c(n, a, n') + h(n') &\geq h(n) \\ \Leftrightarrow c(n, a, n') &\geq h(n) - h(n') \end{aligned}$$



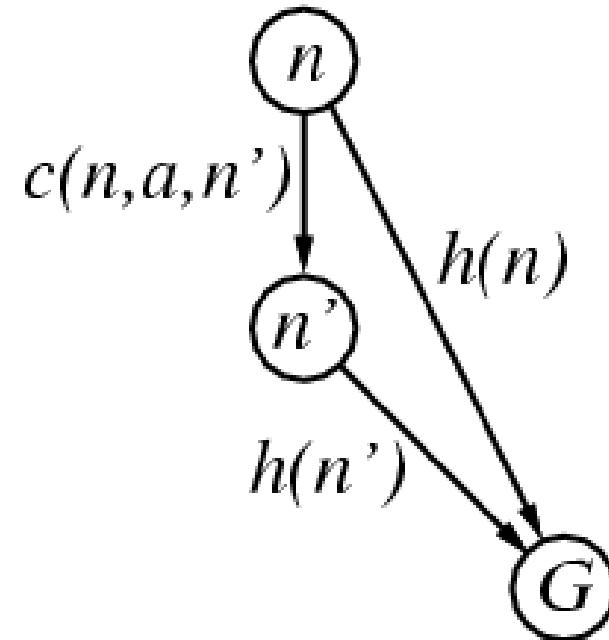
Consistent Heuristics

- A heuristic is **consistent** if for every node n , every successor n' of n generated by any action a :

$$c(n, a, n') + h(n') \geq h(n)$$

- If h is consistent, we have

$$\begin{aligned}f(n') &= g(n') + h(n') \\&= g(n) + c(n, a, n') + h(n') \\&\geq g(n) + h(n) = f(n)\end{aligned}$$



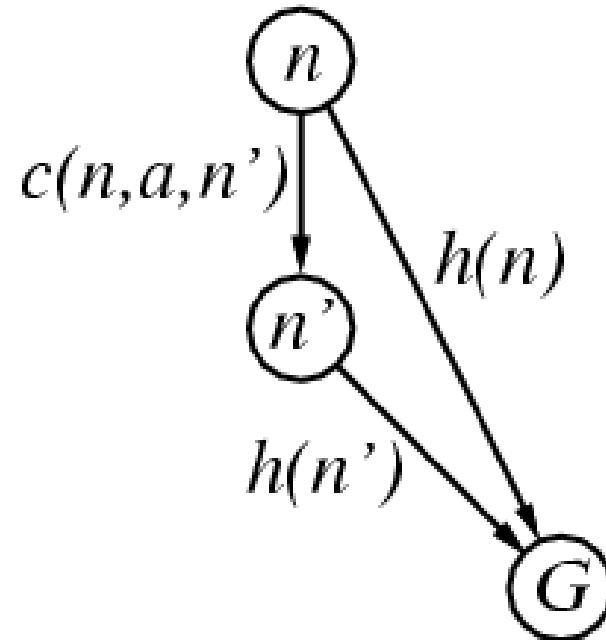
Consistent Heuristics

- A heuristic is **consistent** if for every node n , every successor n' of n generated by any action a :

$$c(n, a, n') + h(n') \geq h(n)$$

- If h is consistent, we have

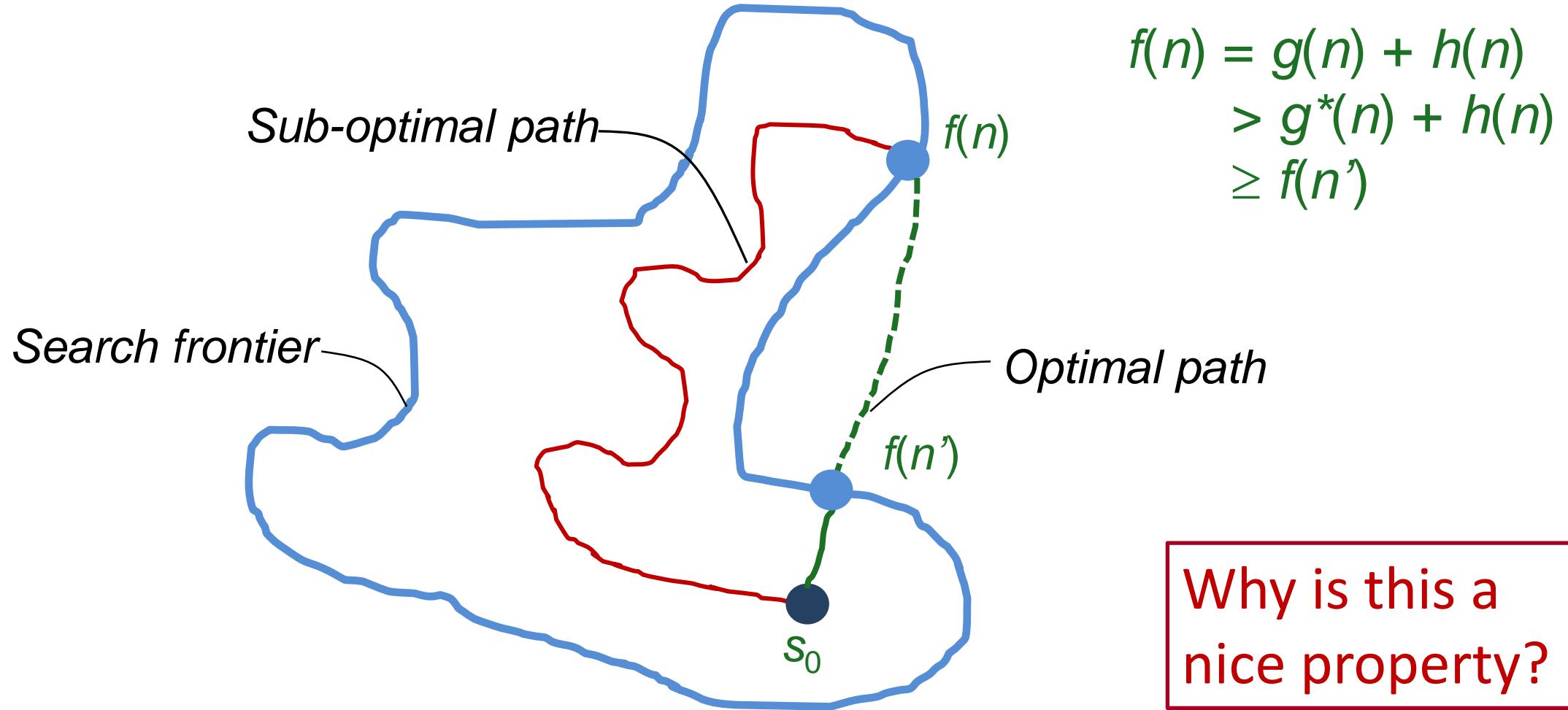
$$\begin{aligned}f(n') &= g(n') + h(n') \\&= g(n) + c(n, a, n') + h(n') \\&\geq g(n) + h(n) = f(n)\end{aligned}$$



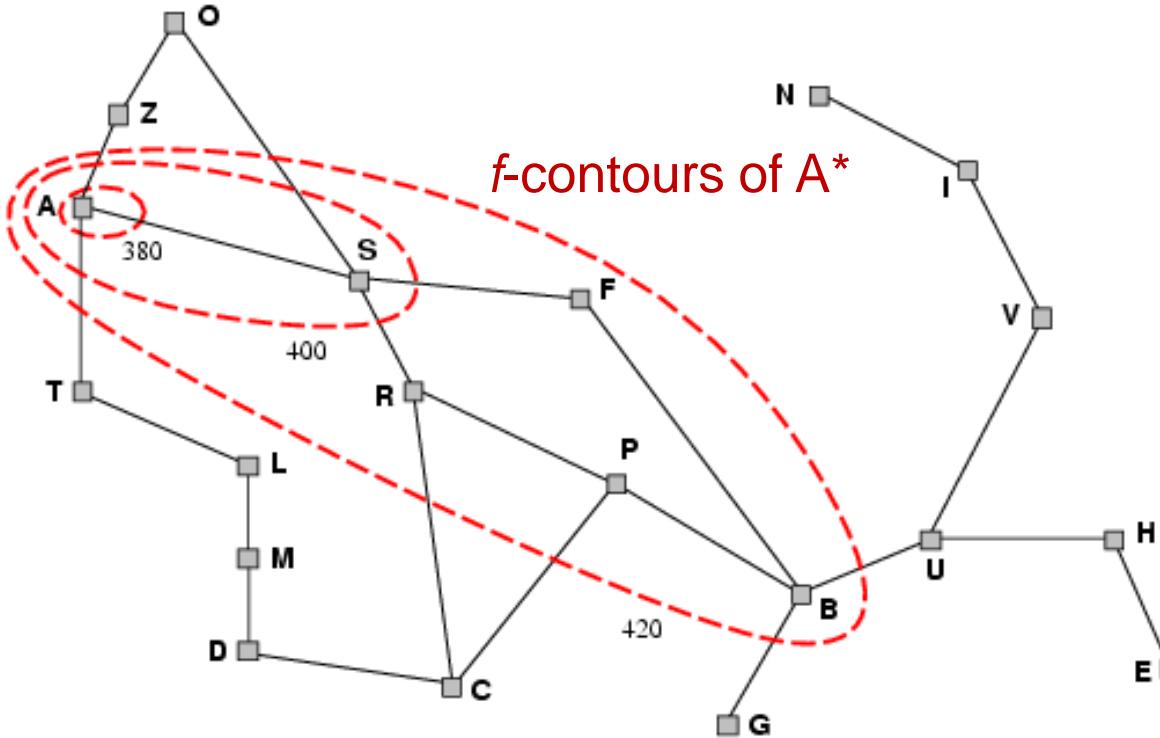
- Thus, $f(n)$ is non-decreasing along any search path
- **A consistent heuristic is also admissible (challenge exercise)**

Claim: When A* expands a node n , an optimal path to its state is found

Proof:



A* Search with Consistent h



- A* graph search expands nodes of optimal paths in order of increasing f -value
- It expands all nodes with $f(n) < C^*$

Properties of A*

Consistent heuristic, graph-search version

- Complete? Yes, unless there are infinitely many nodes with $f(n) \leq C^*$
- Time? Exponential in solution length, unless h is very accurate
 $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- Space? Keeps all nodes in memory
- Optimal? Yes



Weighted A* Search and Uniform Cost Search

- Weighted A* search: $(1-W) \times g(n) + W \times h(n)$
- Uniform-cost search: $g(n)$ ($W = 0$)
- Greedy best-first search: $h(n)$ ($W = 1$)
- A* Search: $g(n) + h(n)$ ($W = 0.5$)

Ok, still A* search just think of it as halving the action costs

Introduction to Artificial Intelligence

Lecture 3: Adversarial Search

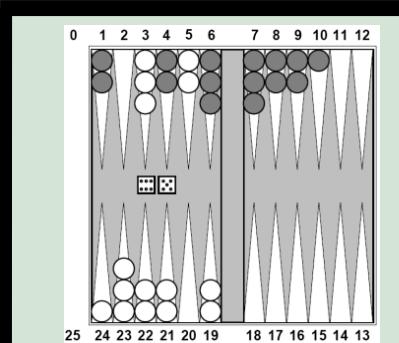
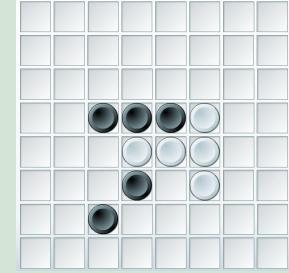
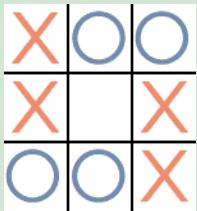
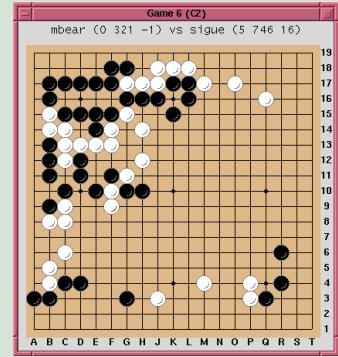
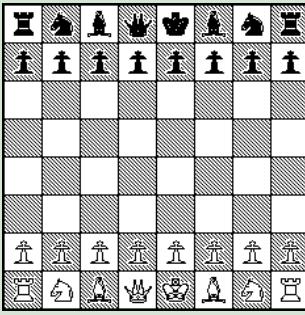
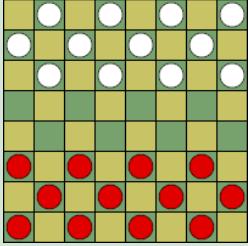


Why Learn About Game Algorithms?

- Large game industry
- Policy based methods are smoking HOT, not just for games but any problem
 - Sequential decision problems
 - Optimization
 - Control systems
- Game trees are fundamental to understand adversarial situations

Types of Games

Board Games



“Physical” Games



Our Assumptions

- Discrete states and actions
- Turn-taking
- Observable state
- Zero-Sum utility
- Mainly 2-Player
- Mainly deterministic

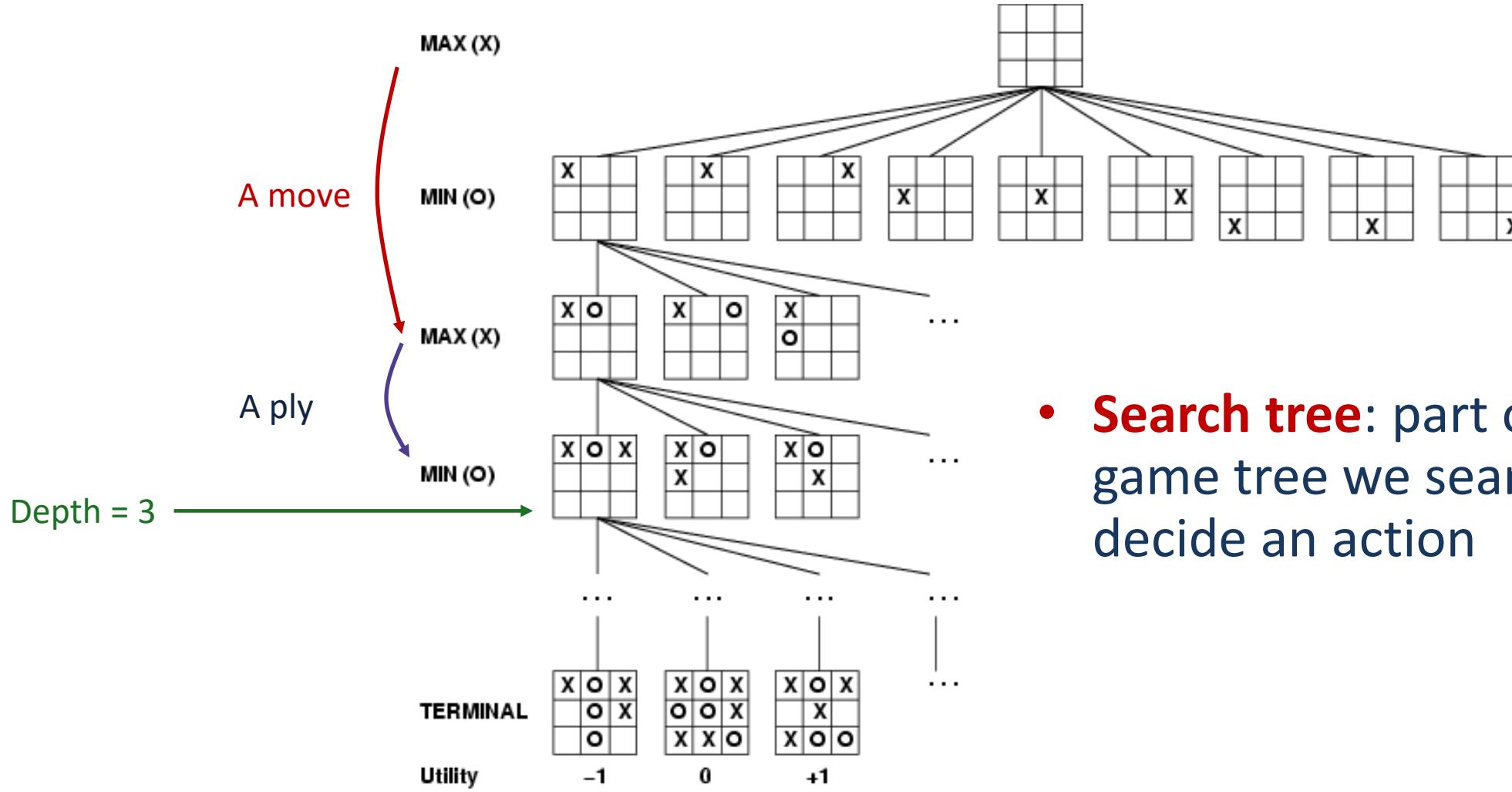


Today's Program

- [10:00-11:00]
 - Game trees
 - Minimax optimal strategy
 - MINIMAX
 - ALPHA BETA SEARCH
- [11:10-12:00]
 - Evaluation and Cutoff functions
 - H-MINIMAX
 - HEURISTIC ALPHA BETA SEARCH
 - MONTE CARLO TREE SEARCH (MCTS)
 - EXPECTIMINIMAX
- [12:00-14:00]
 - Exercises



Tic-Tac-Toe Game Tree



- **Search tree**: part of game tree we search to decide an action

Game Tree

- Initial state S_0
- To-Move(s)
 - Max node  (us)
 - Min node  (opponent)
- ACTIONS(s)
- RESULT (s, a)
- Is-TERMINAL(s)
- UTILITY(s, p) (e.g., win +1, draw 0, loss -1)

Optimal Strategy in a Game

Among all strategies, one that gives us the best outcome playing against an opponent that makes no mistakes and even may know our strategy



Minimax Decision

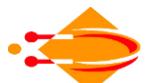
Maximize worst-case utility for Max!

$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if } \text{Is-TERMINAL}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{To-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{To-MOVE}(s) = \text{MIN} \end{cases}$$

Minimax decision at root =

Action to a with highest MINIMAX VALUE



Minimax Algorithm

```
function MINIMAX-SEARCH(game,state) returns an action value, move
    value, move  $\leftarrow$  MAX-VALUE(game,state)
    return move
```

```
function MAX-VALUE(game,state) returns (utility,move)
    if game.Is-Terminal(state) then
        return game.Utility(state,MAX), null
    v  $\leftarrow$   $-\infty$ 
    for each a in game.Actions(state) do
        v2,a2  $\leftarrow$  MIN-VALUE(game,game.Result(state,a))
        if v2 > v then
            v,move  $\leftarrow$  v2,a
    return v,move
```

```
function MIN-VALUE(game,state) returns (utility,move)
    if game.Is-Terminal(state) then
        return game.Utility(state,MIN), null
    v  $\leftarrow$   $+\infty$ 
    for each a in game.Actions(state) do
        v2,a2  $\leftarrow$  MAX-VALUE(game,game.Result(state,a))
        if v2 < v then
            v,move  $\leftarrow$  v2,a
    return v,move
```

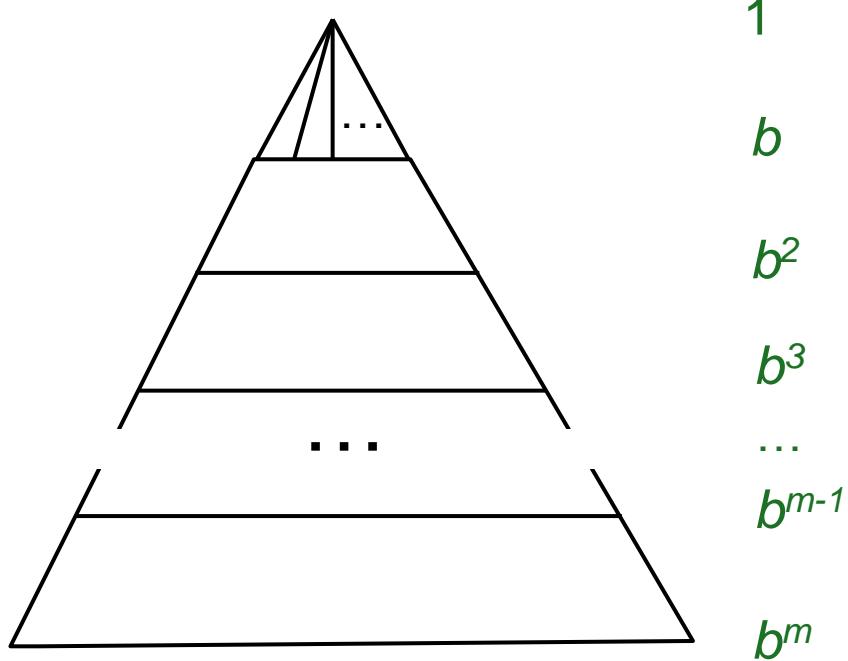


Complexity

- b : max number of successors of a node
- m : max depth of tree
- Time: ?
- Space: ?



Time complexity of Minimax



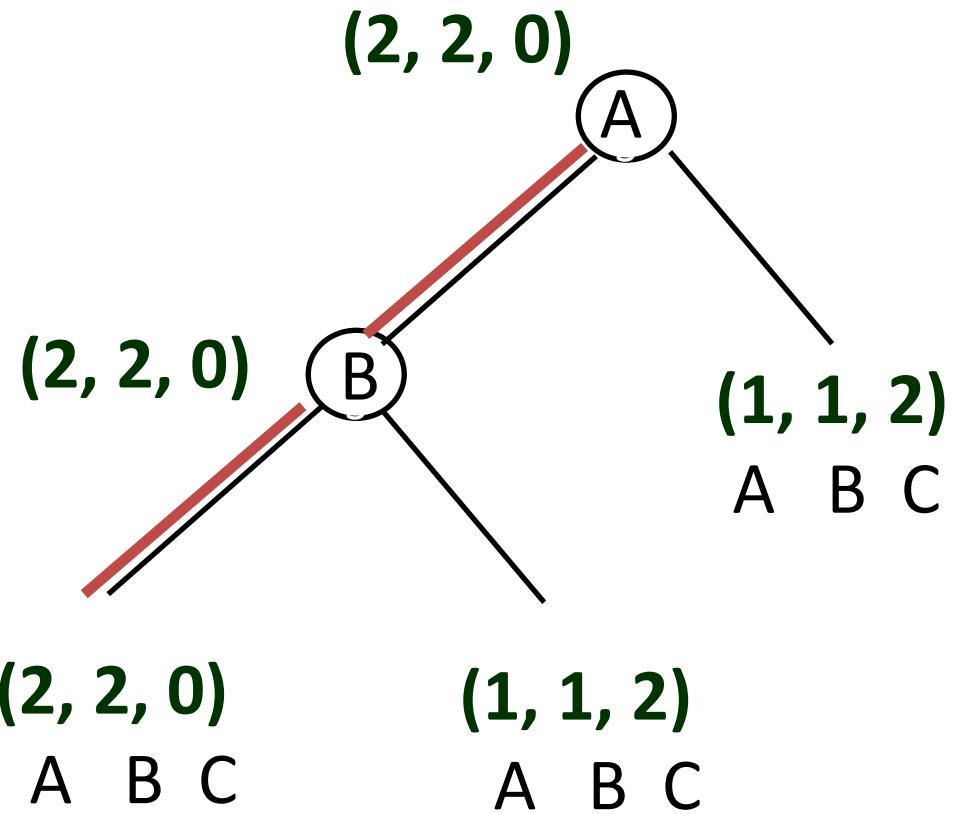
$$O\left(\sum_{i=0}^m b^i\right) = O(b^m)$$

Complexity

- b : max number of successors of a node
- m : max depth of tree
- Time: $O(b^m)$
- Space: $O(m)$



Multi-Player Minimax

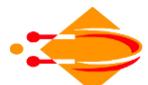


α - β Pruning

Idea: keep track of MAX and MIN's best choice found so far on the explored path

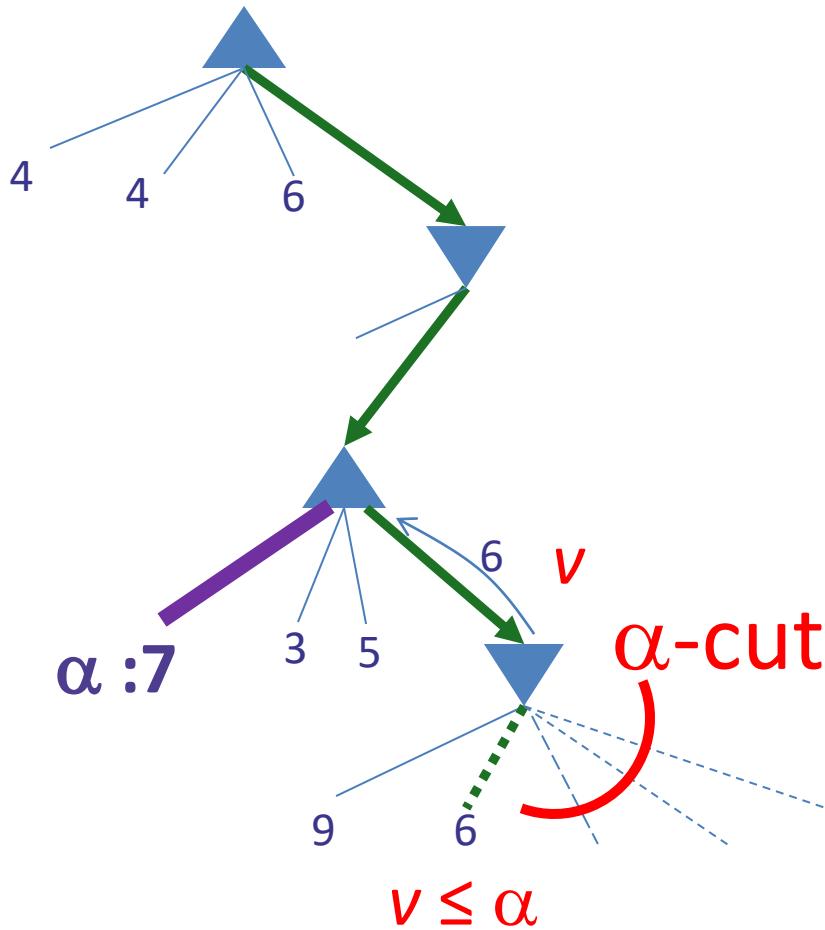
- α = the value (i.e., highest value) of the best choice for MAX
- β = the value (i.e., lowest value) of the best choice for MIN

Use this to prune the tree

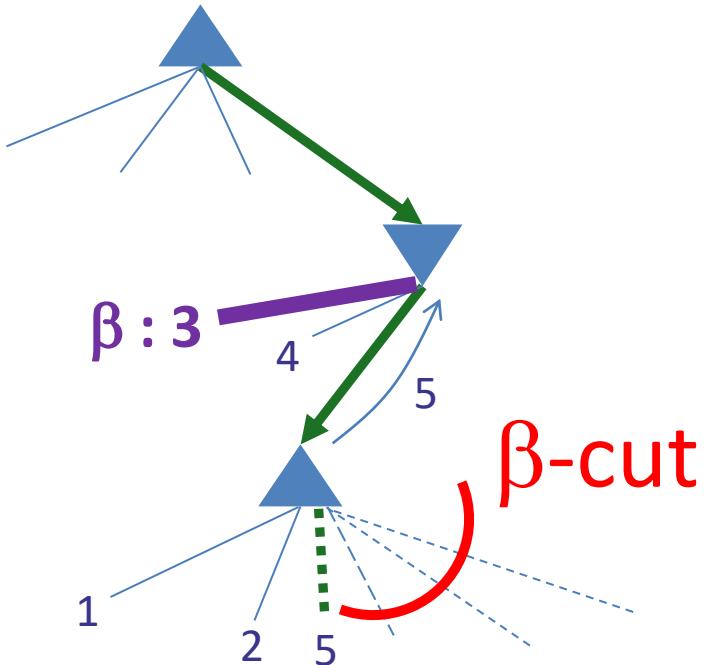


α -cut

$$\begin{aligned}\alpha &= \max\{4, 4, 6, 7, 3, 5\} \\ &= 7\end{aligned}$$



β -cut



Alpha-Beta-Search

```
function ALPHA-BETA-SEARCH(game,state) returns an action
    value, move  $\leftarrow$  MAX-VALUE(game,state, $-\infty$ , $+\infty$ )
    return move
```

```
function MAX-VALUE(game,state, $\alpha$ , $\beta$ ) returns (utility,move)
    if game.Is-TERMINAL(state) then
        return game.UTILITY(state,MAX), null
    v  $\leftarrow$   $-\infty$ 
    for each a in game.ACTIONS(state) do
        v2,a2  $\leftarrow$  MIN-VALUE(game,game.RESULT(state,a), $\alpha$ , $\beta$ )
        if v2 > v then
            v,move  $\leftarrow$  v2,a
             $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
            if v  $\geq \beta$  then return v,move
    return v,move
```

```
function MIN-VALUE(game,state, $\alpha$ , $\beta$ ) returns (utility,move)
    if game.Is-TERMINAL(state) then
        return game.UTILITY(state,MAX), null
    v  $\leftarrow$   $+\infty$ 
    for each a in game.ACTIONS(state) do
        v2,a2  $\leftarrow$  MAX-VALUE(game,game.RESULT(state,a), $\alpha$ , $\beta$ )
        if v2 < v then
            v,move  $\leftarrow$  v2,a
             $\beta \leftarrow \text{MIN}(\beta, v)$ 
            if v  $\leq \alpha$  then return v,move
    return v,move
```



Properties of α - β Pruning

- Does not affect the final result
- Exploring “best” nodes first improve pruning (**killer heuristic**)
- $O(b^{m/2})$ with perfect ordering
 - doubles reachable search depth!



Break



Cut-off and evaluation functions

Standard Approach for MINIMAX

- Use **Is-CUTOFF** instead of **Is-TERMINAL**
 - e.g., depth-limit
- Use **EVAL** instead of **UTILITY**
 - i.e., **evaluation function** that estimates **desirability** of position
 - 1) Order terminal states in the same way as **UTILITY**
 - 2) Value strongly correlated with chance of winning
 - 3) Efficiently computable

H-Minimax

$H\text{-MINIMAX}(s, d) =$

$$\begin{cases} \text{EVAL}(s, \text{MAX}) & \text{if } \text{Is-CUTOFF}(s, d) \\ \max_{a \in \text{ACTIONS}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if } \text{To-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if } \text{To-MOVE}(s) = \text{MIN} \end{cases}$$



Evaluation Functions

Idea to reduce complexity

1. Simplify states by extracting **features**
2. Use features as input to evaluation functions

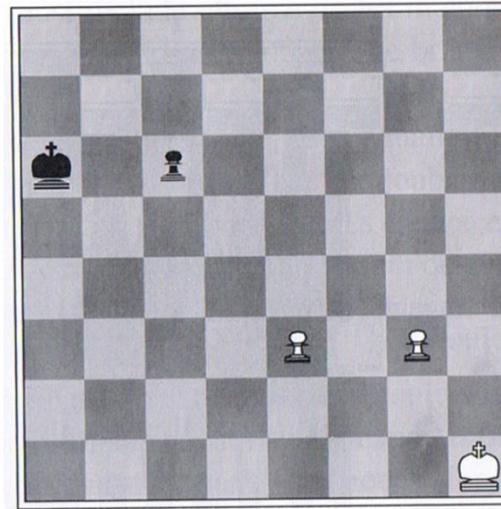
Examples

- Chess: # of pawns, rooks, queens,... of each side
- Tic-tac-toe: # adjacent tokens of each side
- Etc...



Expected Utility

- Eval = expected utility of a category (=all states with same state features)
- Category example:
two pawns vs. one

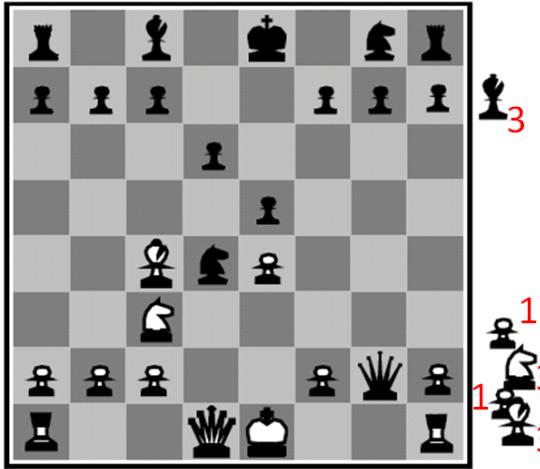


From experience : 72% win, 20% loss 8% draw

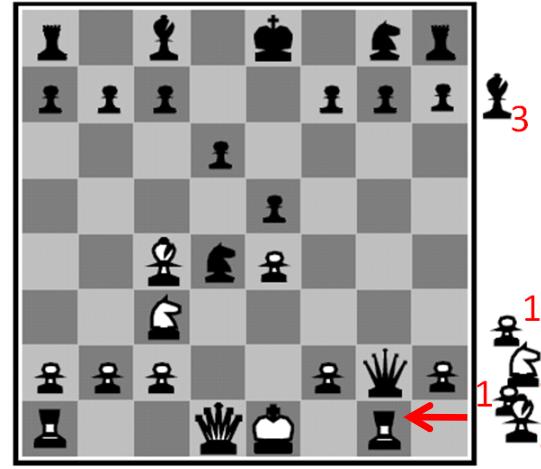
$$\text{Expected Value: } (0.72 * 1) + (0.20 * 0) + (0.08 * \frac{1}{2}) = 0.76$$

But: too many states of each category in practice!

Material Value



White to move

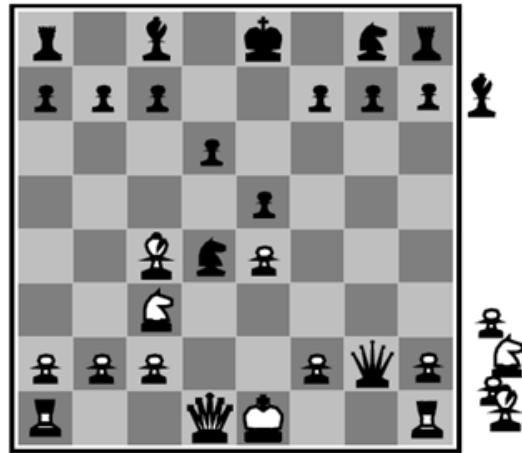


White to move

- Eval = sum of feature values
- For chess typically linear weighted sum of features
 - $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
- E.g. $w_1 = 9$ with
 - $f_1(s) = (\# \text{ of white queens}) - (\# \text{ of black queens})$, etc.

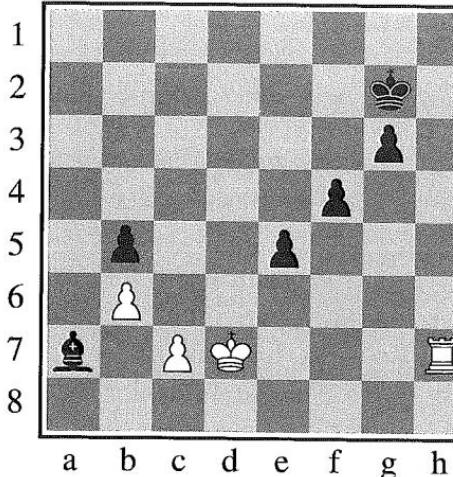
CUT-OFF Function

White to move



Quiescence State

Black to move



Horizon Effect

Iterative Deepening, possibly extended with

- Quiescence search: only make cut-off in quiescence states
- Singular extensions: apply singular extension to avoid horizon effect
- Transposition table: memorize previously evaluated states

Heuristic Alpha-Beta-Search

Replace

```
if game.Is-TERMINAL(state) then  
    return game.UTILITY(state,player), null
```

with

```
if game.Is-CUTOFF(state,depth) then  
    return game.EVAL(state,player), null
```

and keep track of the search depth.



Monte Carlo Tree Search (MCTS)



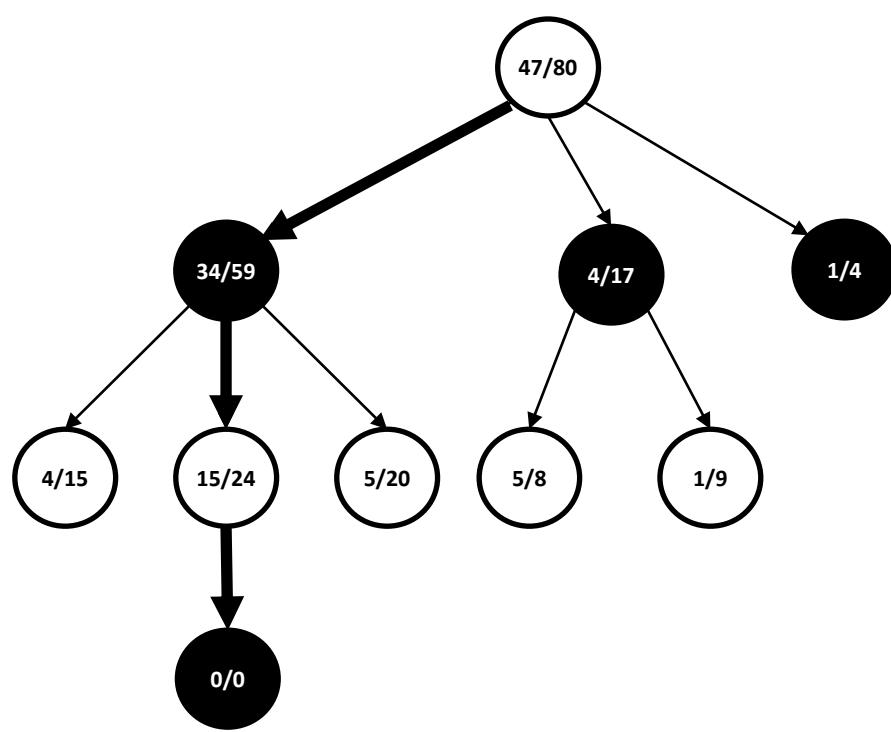
MCTS Motivation

- Alpha-Beta Search is inefficient if
 - Branching factor is high
(does not search deep enough)
 - Evaluation functions are hard to define
(bad guidance from evaluation function)
- Go has both characteristics
- The idea of MCTS is simulate the game and find “hot paths” in the game tree



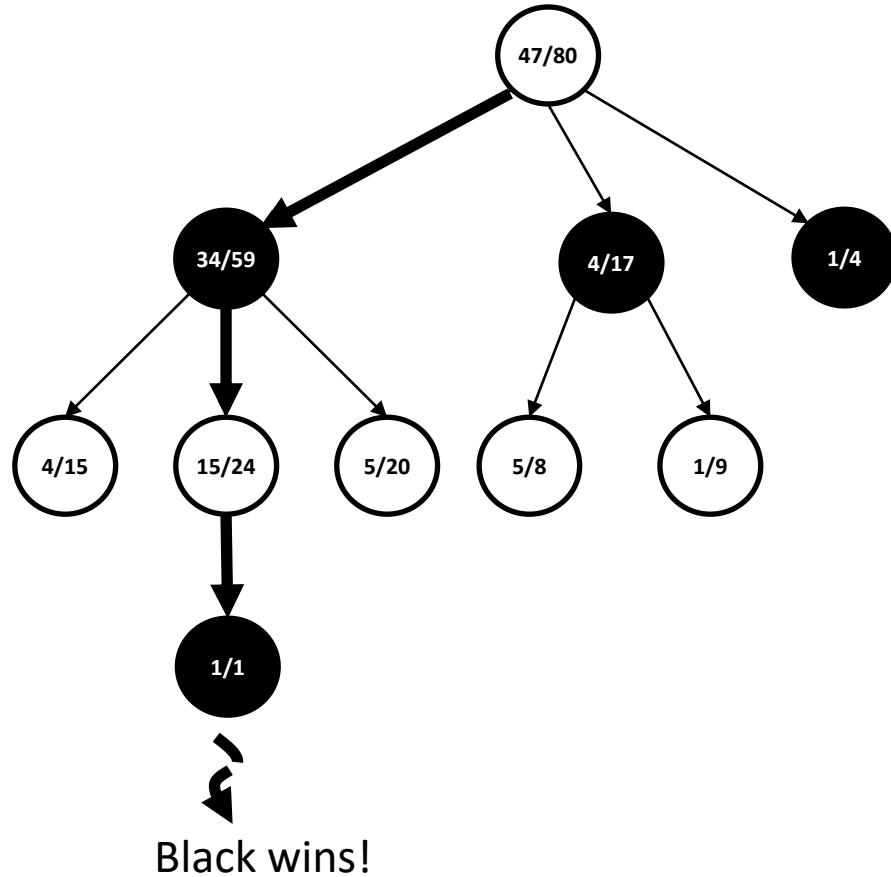
Unclear whether partial game is won or lost

MCTS Algorithm



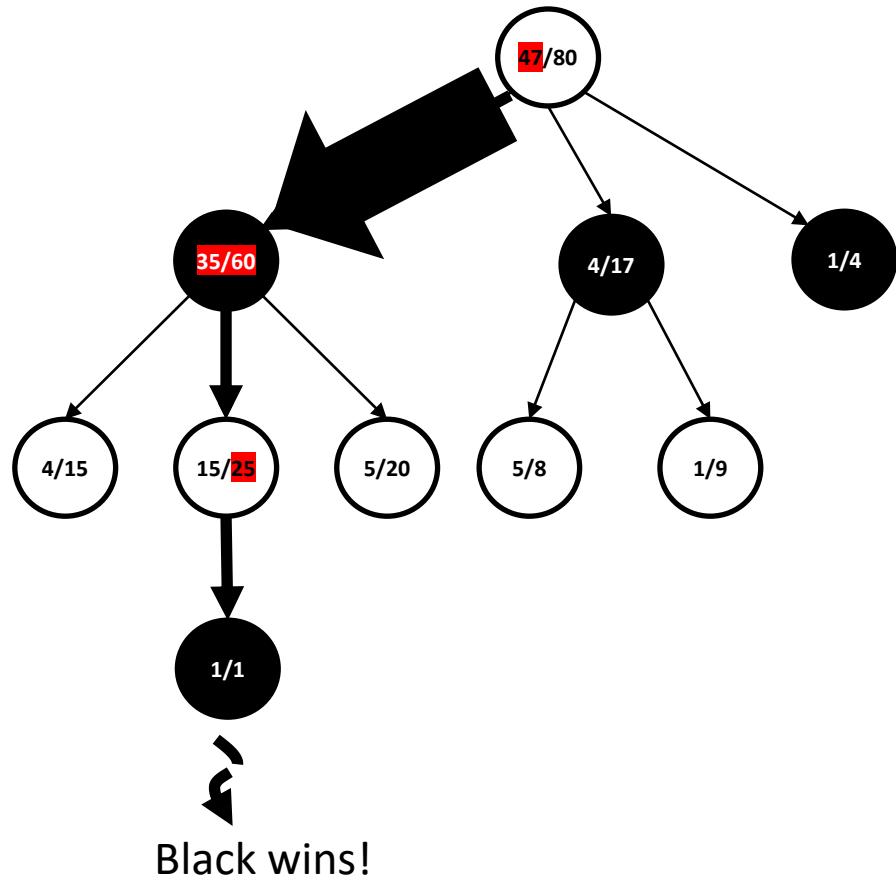
- 1) Select leaf node
- 2) Expand leaf node

MCTS Algorithm



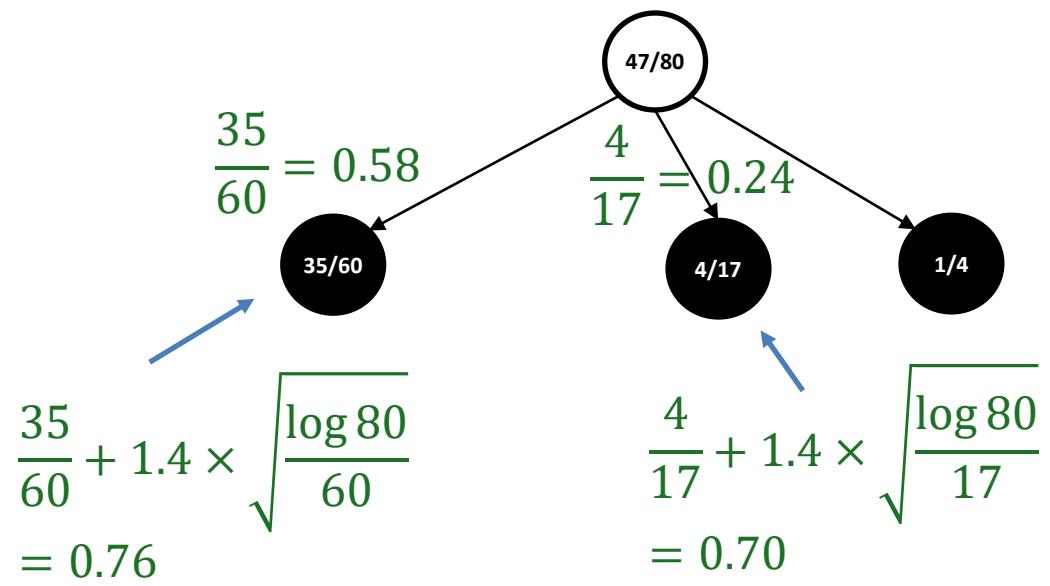
- 1) Select leaf node
- 2) Expand leaf node
- 3) Simulate game according to playout policy

MCTS Algorithm



- 1) Select leaf node
- 2) Expand leaf node
- 3) Simulate game according to playout policy
- 4) Backpropagate simulation result
- 5) Repeat 1-4 until out of time
- 6) Return move with highest number of playouts

MCTS Selection Heuristic



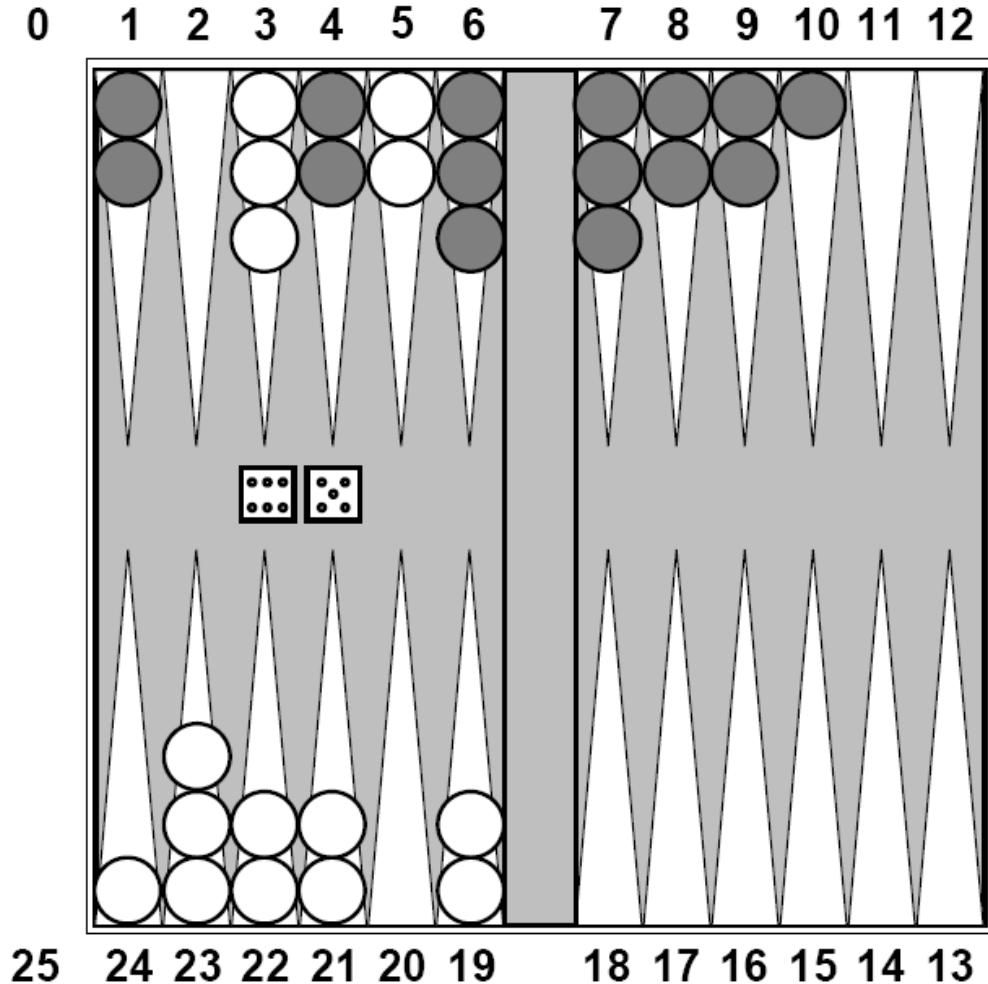
Select node with highest
upper confidence bound (UCT)

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(Parent(n))}{N(n)}}$$

Stochastic Games



Nondeterministic Games



Backgammon

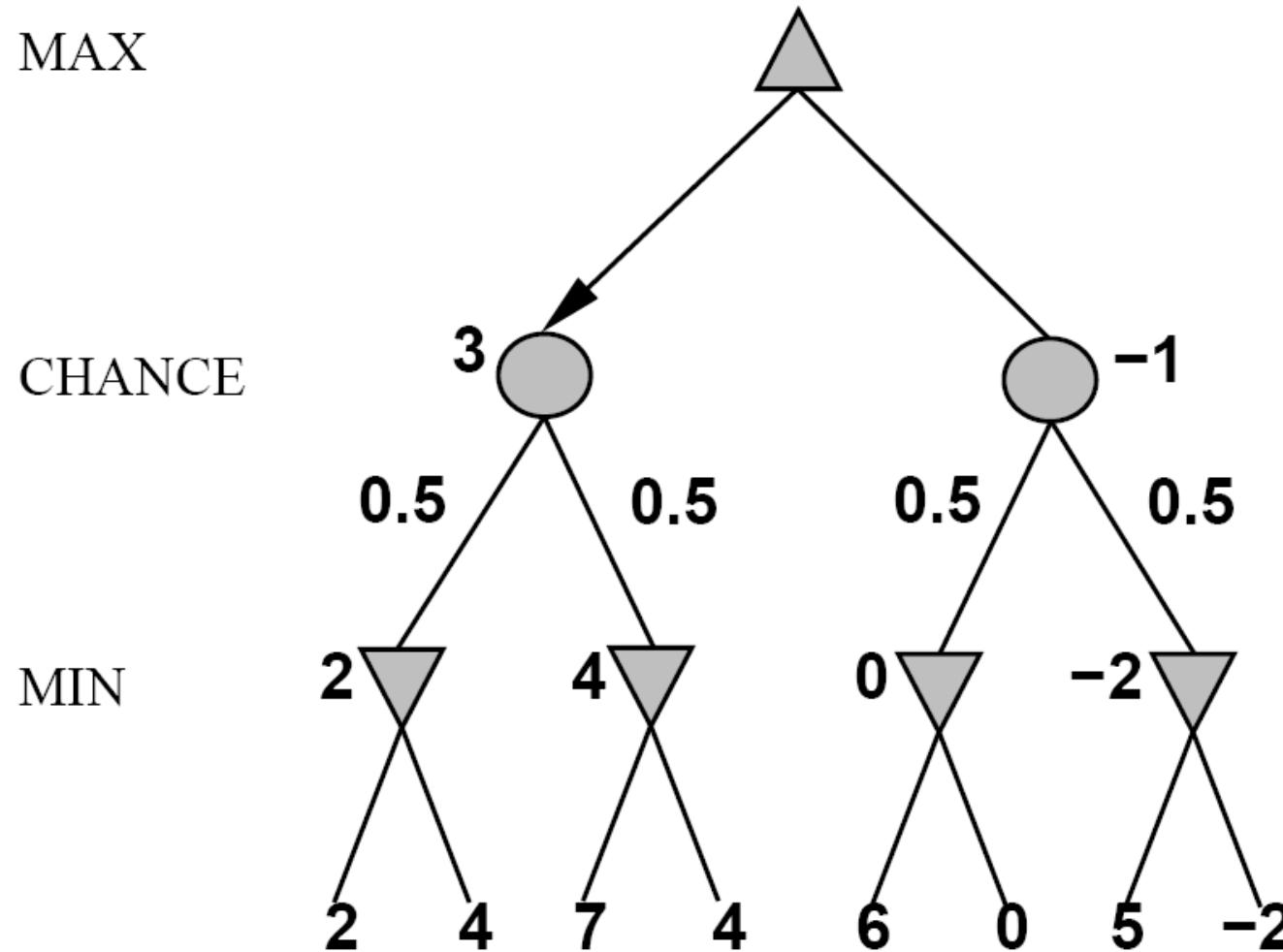
Branching factor ≈ 20

Distinct dice rolls = 21

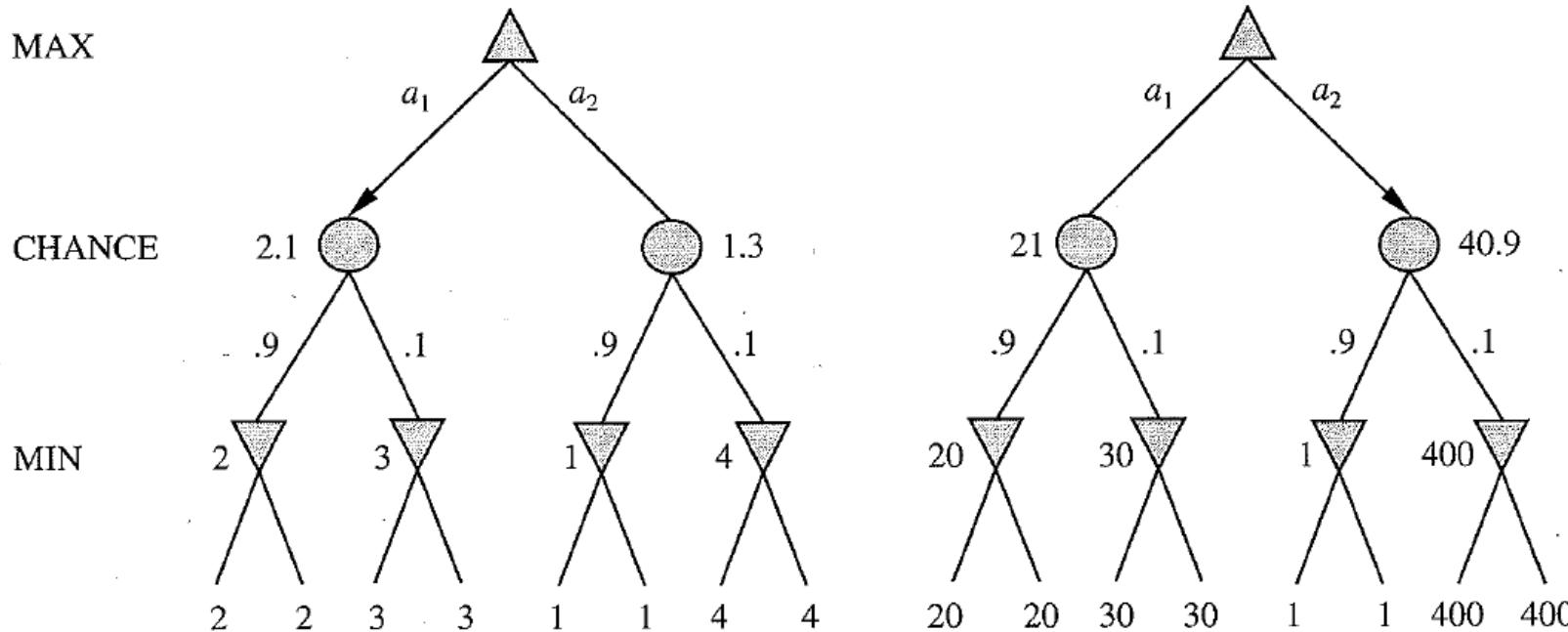
Total outcomes 420

Too high branching
factor!

Expectiminimax



Evaluation Function



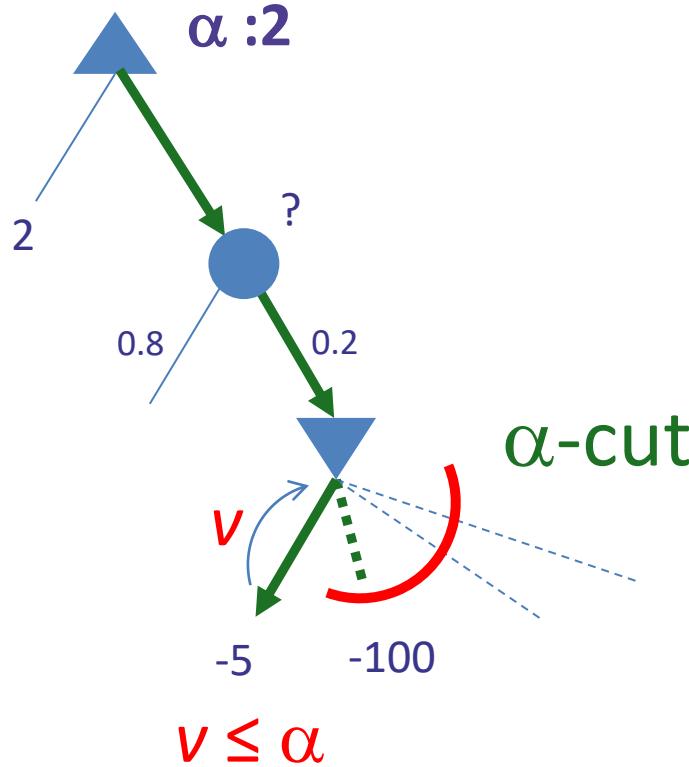
- Only **linear transformations** of utility function allowed!

Expectiminimax

EXPECTIMINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s) & \text{if Is-Terminal}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if To-Move}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if To-Move}(s) = \text{MIN} \\ \sum_r P(r) \cdot \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if To-Move}(s) = \text{CHANCE} \end{cases}$$

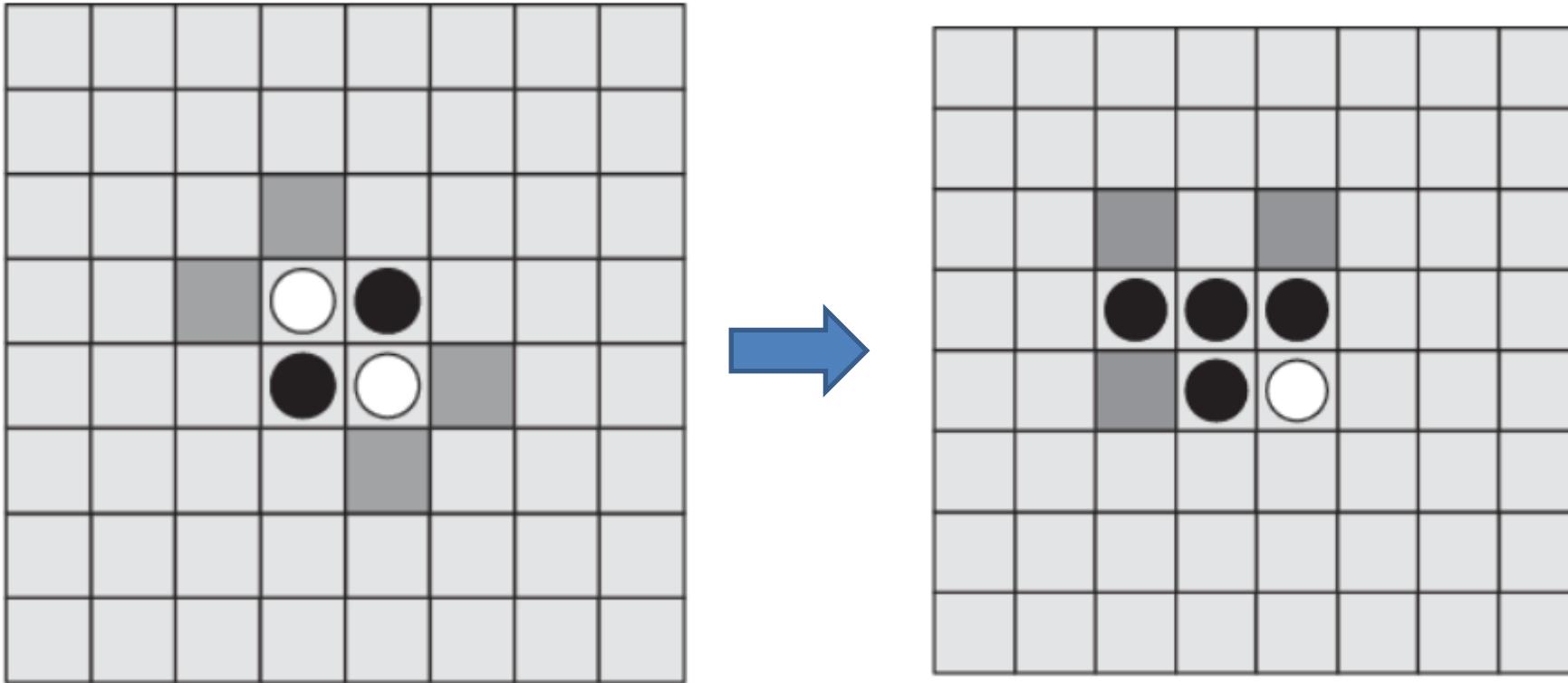
Does simple α - β pruning still work?



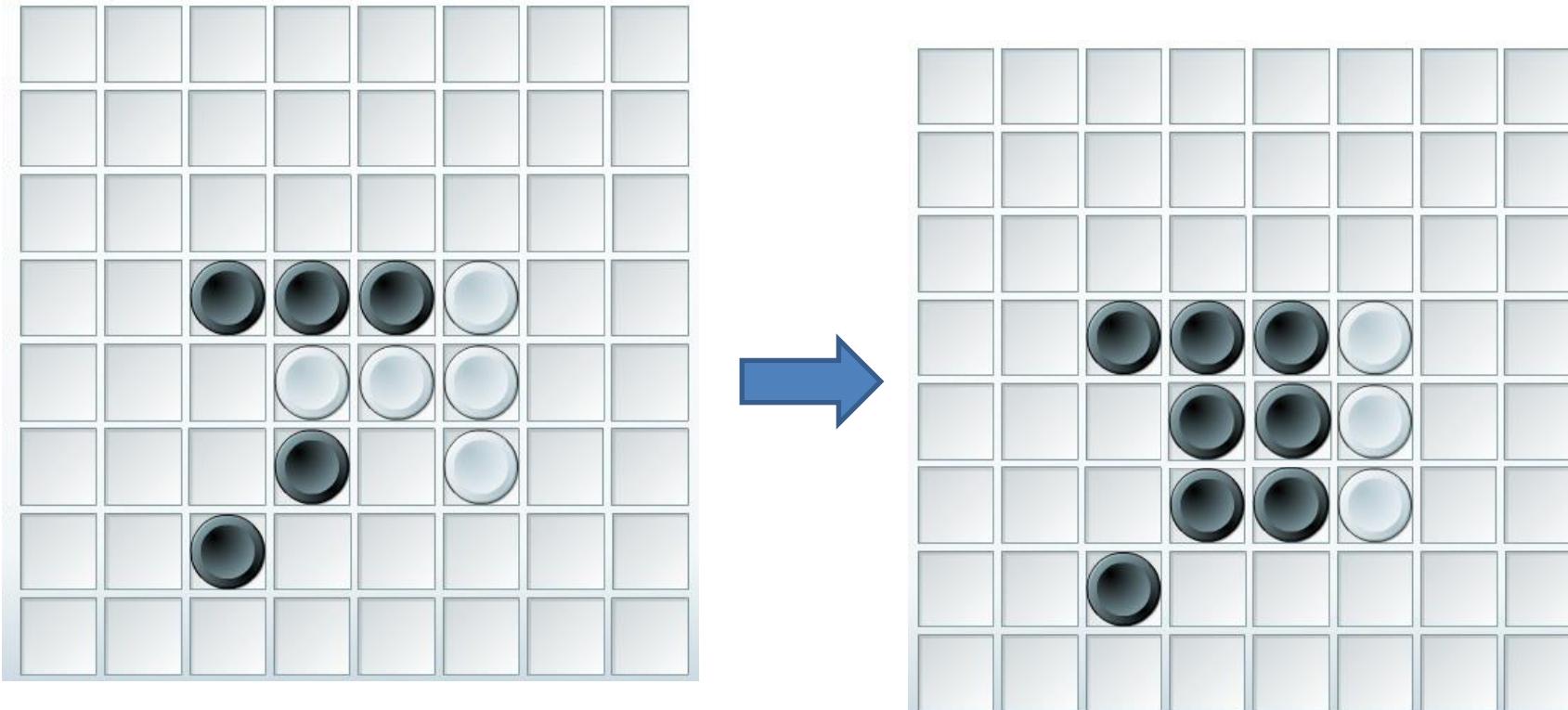
Problem: due to uncertainty the value of the chance node may be better than -5.

So MIN cannot conclude that the cut can be made!

Othello Initial State



Othello Another Example



Introduction to Artificial Intelligence

Lecture 4: Heuristic Theory
DeepMind Game Algorithms



Today's Program

[10-10:40] Heuristic Theory

- Domination
- Relaxations

[10:50-12:00] DeepMind Game Algorithms

- Methods
 - Reinforcement learning
 - Convolutional neural networks
- AlphaGo Zero
 - AlphaGo Zero's neural network
 - Self-play reinforcement learning

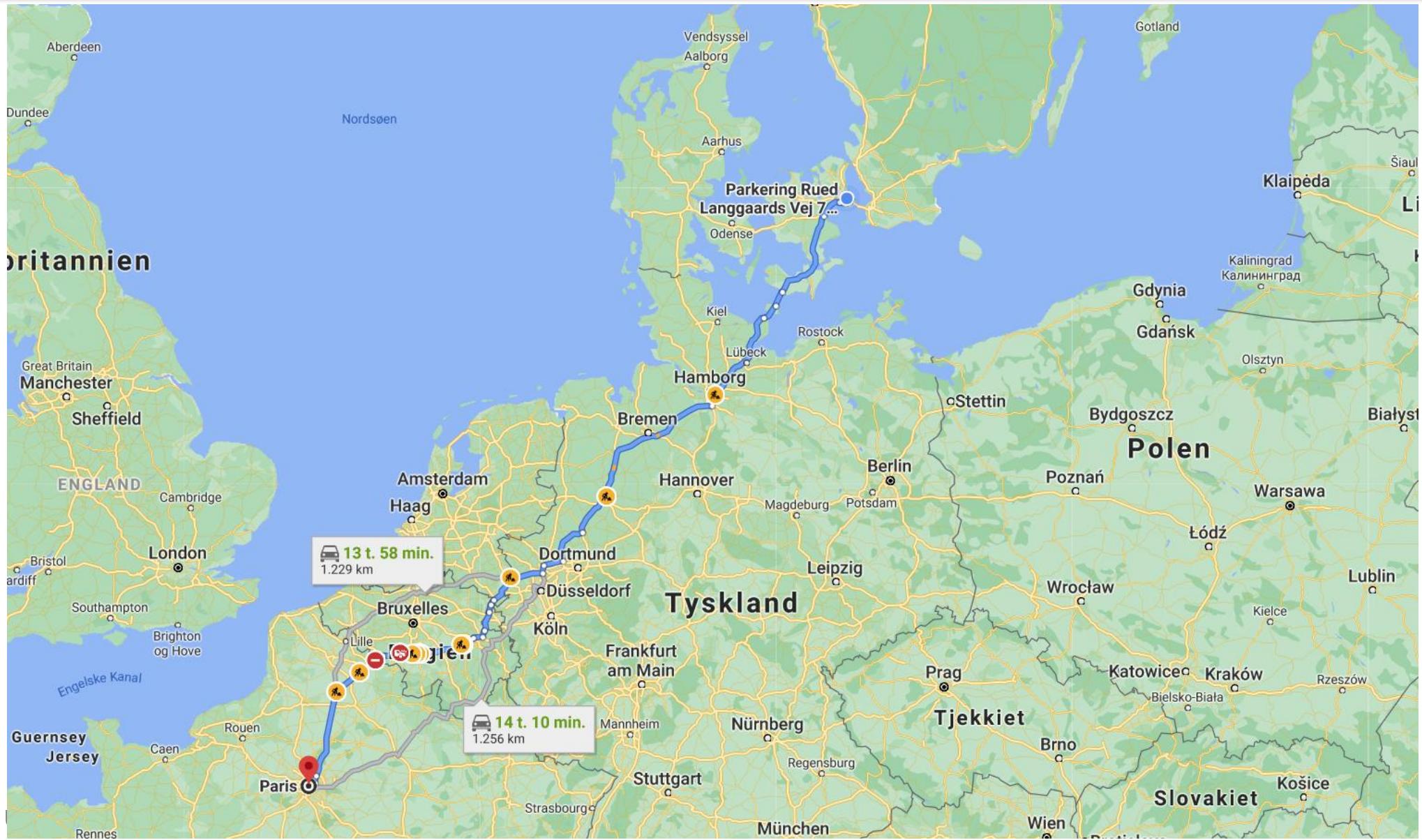
[12-14] Exercises



Heuristic Theory



Routing problems are “easy” (polynomial fringe)



Puzzles are hard (exponential frontier)



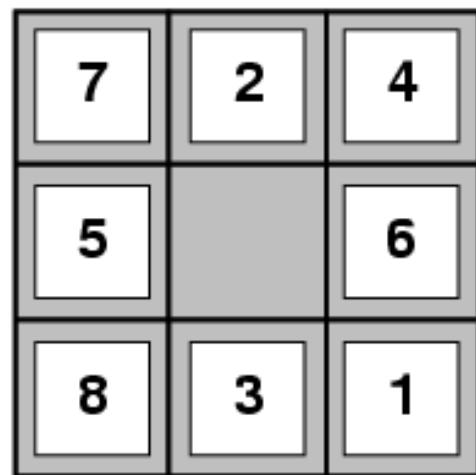
Admissible Heuristics

E.g., for the 8-puzzle:

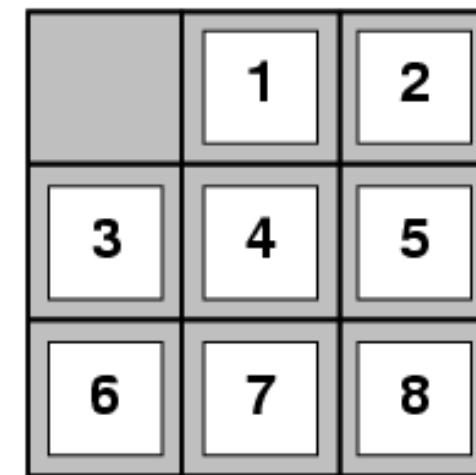
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = sum of Manhattan distances

$$h_1(\text{Start}) = ?$$

$$h_2(\text{Start}) = ?$$



Start State



Goal State

26 steps

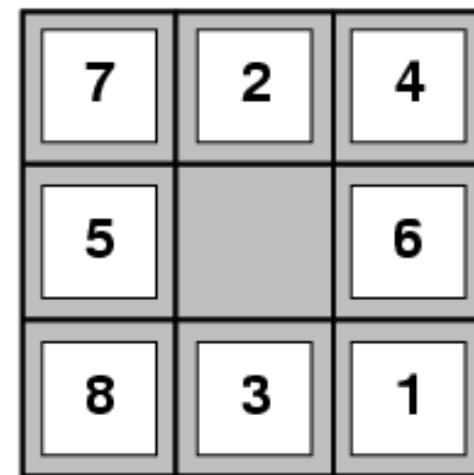
Admissible Heuristics

E.g., for the 8-puzzle:

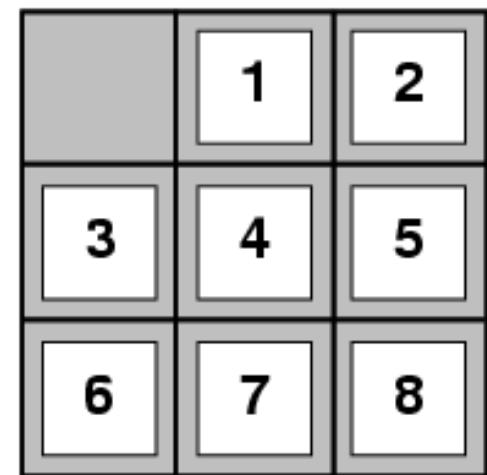
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = sum of Manhattan distances

$$h_1(\text{Start}) = 1+1+1+1+1+1+1 = 8$$

$$h_2(\text{Start}) = 3+1+2+2+2+3+3+2=18$$



Start State



Goal State

26 steps

Measuring Efficiency of Heuristics

- Efficiency measure: **effective branching factor**

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

d	Search Cost			Effective Branching Factor		
	IDS	A*(h ₁)	A*(h ₂)	IDS	A*(h ₁)	A*(h ₂)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Dominance of Admissible Heuristics

- If $h_2(n) \geq h_1(n)$ for all n then h_2 **dominates** h_1
- For **A* search with a consistent heuristic**,
a reachable state s will be expanded if:

$$\begin{aligned} f(s) &< C^* \\ \Leftrightarrow g^*(s) + h(s) &< C^* \\ \Leftrightarrow h(s) &< C^* - g^*(s) \end{aligned}$$

- Thus, If h_2 dominates h_1 then $A^*(h_1)$ expands at least as many reachable states as $A^*(h_2)$

Combining Heuristics

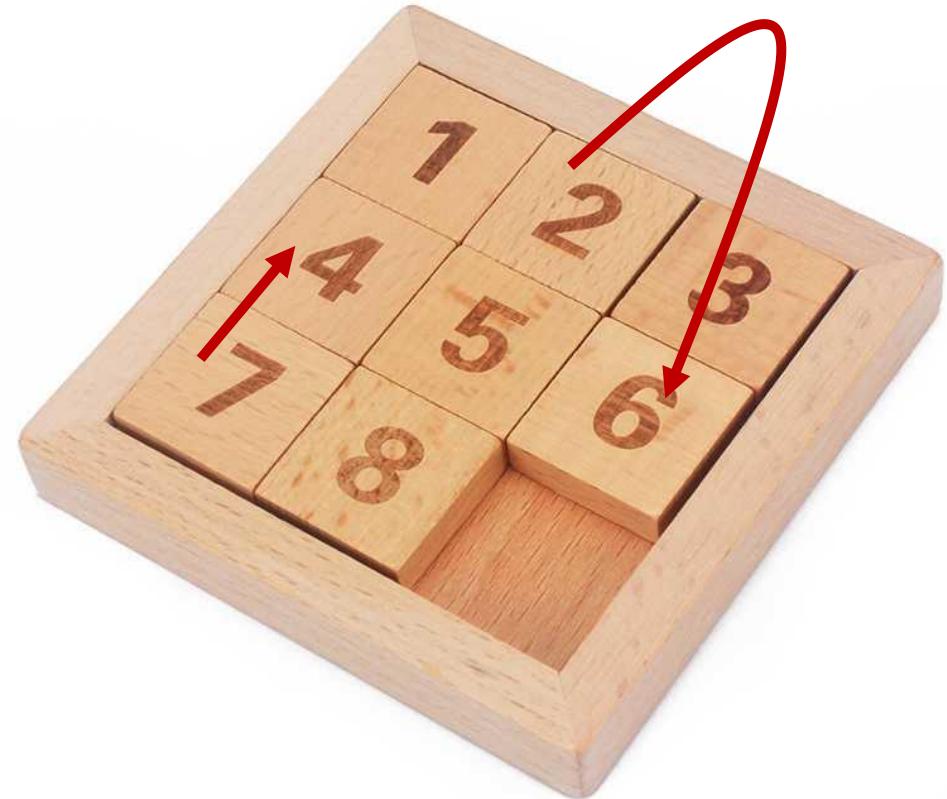
- Given k admissible heuristics $h_1(n), \dots, h_k(n)$
- An admissible and dominating heuristic is

$$h_{\max}(n) = \max[h_1(n), \dots, h_k(n)]$$

Relaxed Problems

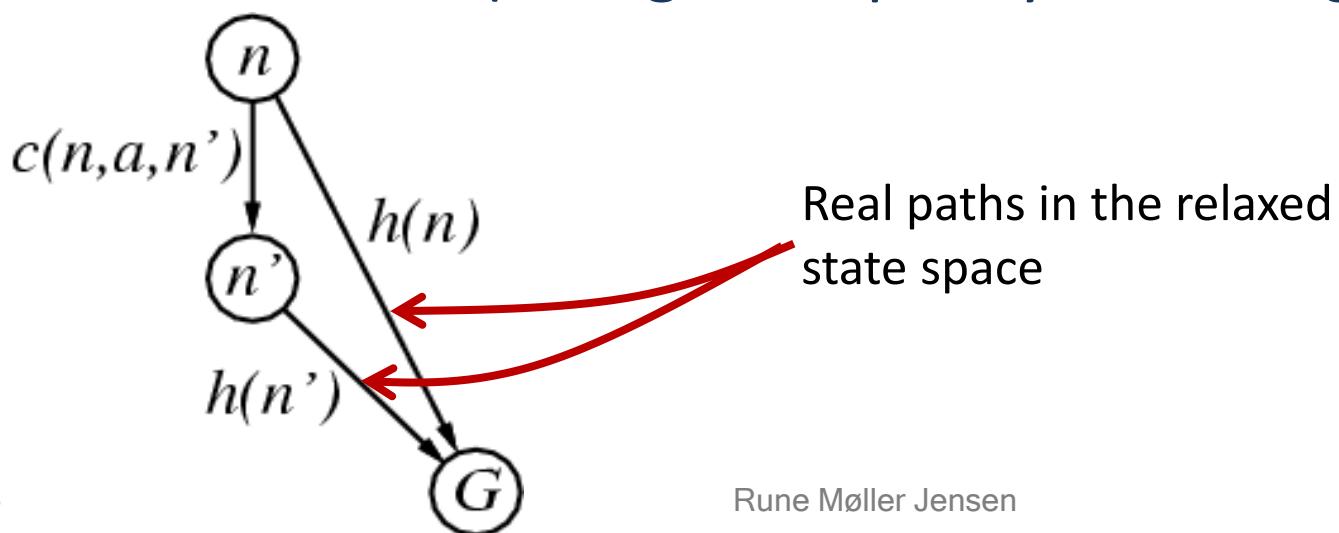
Examples

- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ gives an optimal solution
- If the rules are relaxed so that a tile can move to **any adjacent square**, then $h_2(n)$ gives an optimal solution



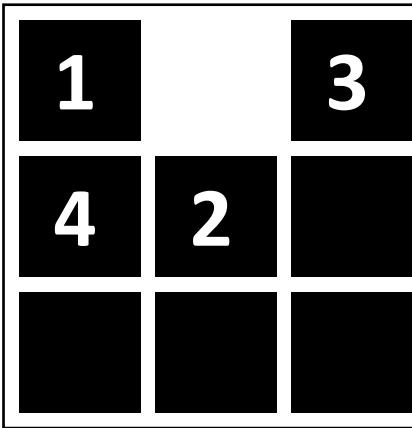
Relaxed Problems

- A problem with fewer restrictions on the actions is called a **relaxed problem** (it adds edges to the state space graph)
- The cost of an optimal solution to a relaxed problem is:
 - an **admissible heuristic** of the original problem
(more paths in state space \Rightarrow optimal paths not longer)
 - a **consistent heuristic** of the original problem
(triangle inequality holds in graphs)



Pattern Databases

- Compute a database storing the **solution cost** of each instance of a subproblem
- E.g.: every possible configuration of the tiles 1-2-3-4 in the 8-puzzle



- For each state in the original problem **the cost of the associated pattern is an admissible heuristic**
- Can we combine a 1-2-3-4 pattern with a 5-6-7-8 pattern to get a better heuristic?

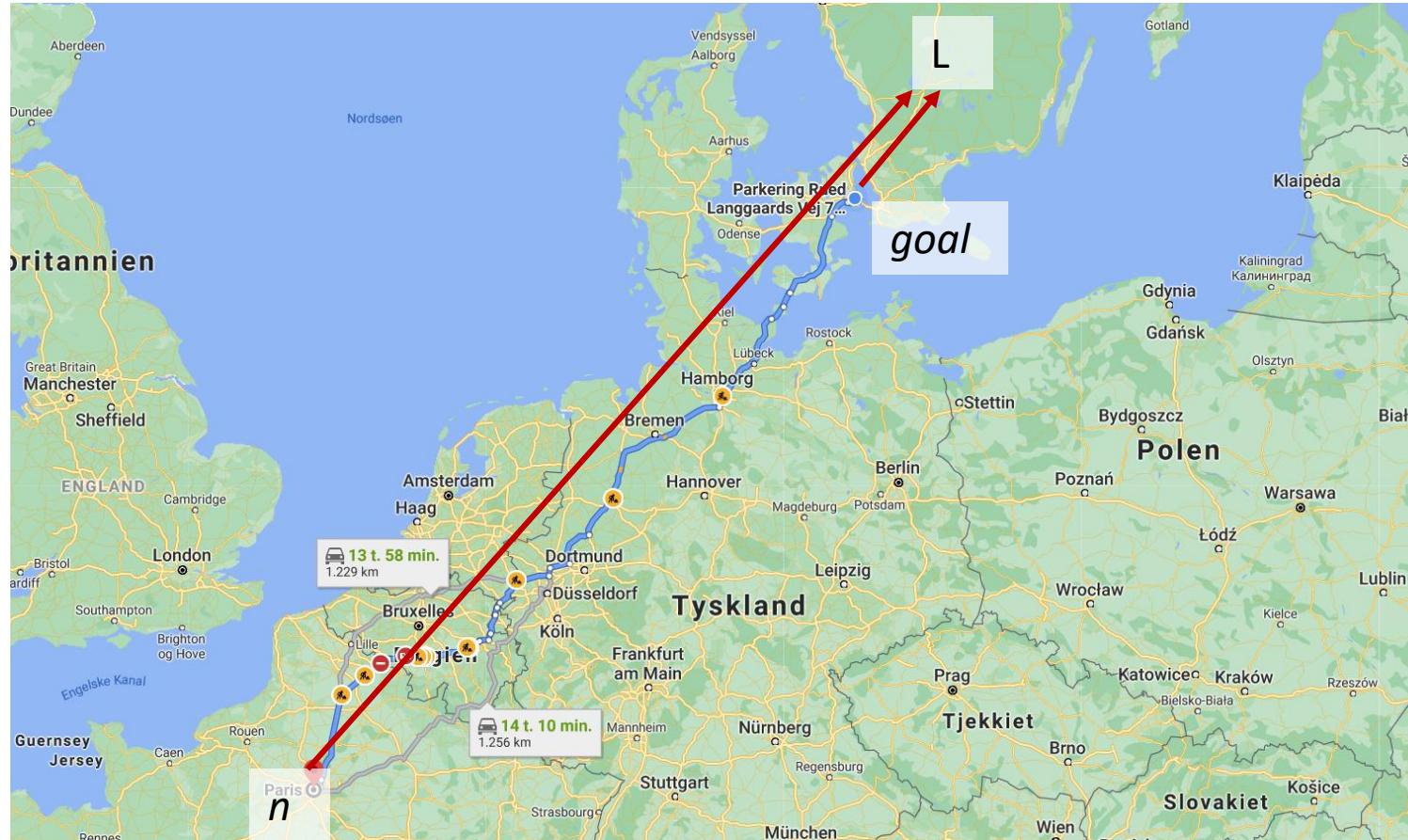
Landmark Heuristic

$$h_L(n) = \min_{L \in \text{Landmarks}} C^*(n, L) + C^*(L, goal)$$



Differential Heuristic

$$h_{DH}(n) = \max_{L \in Landmarks} |C^*(n, L) - C^*(goal, L)|$$



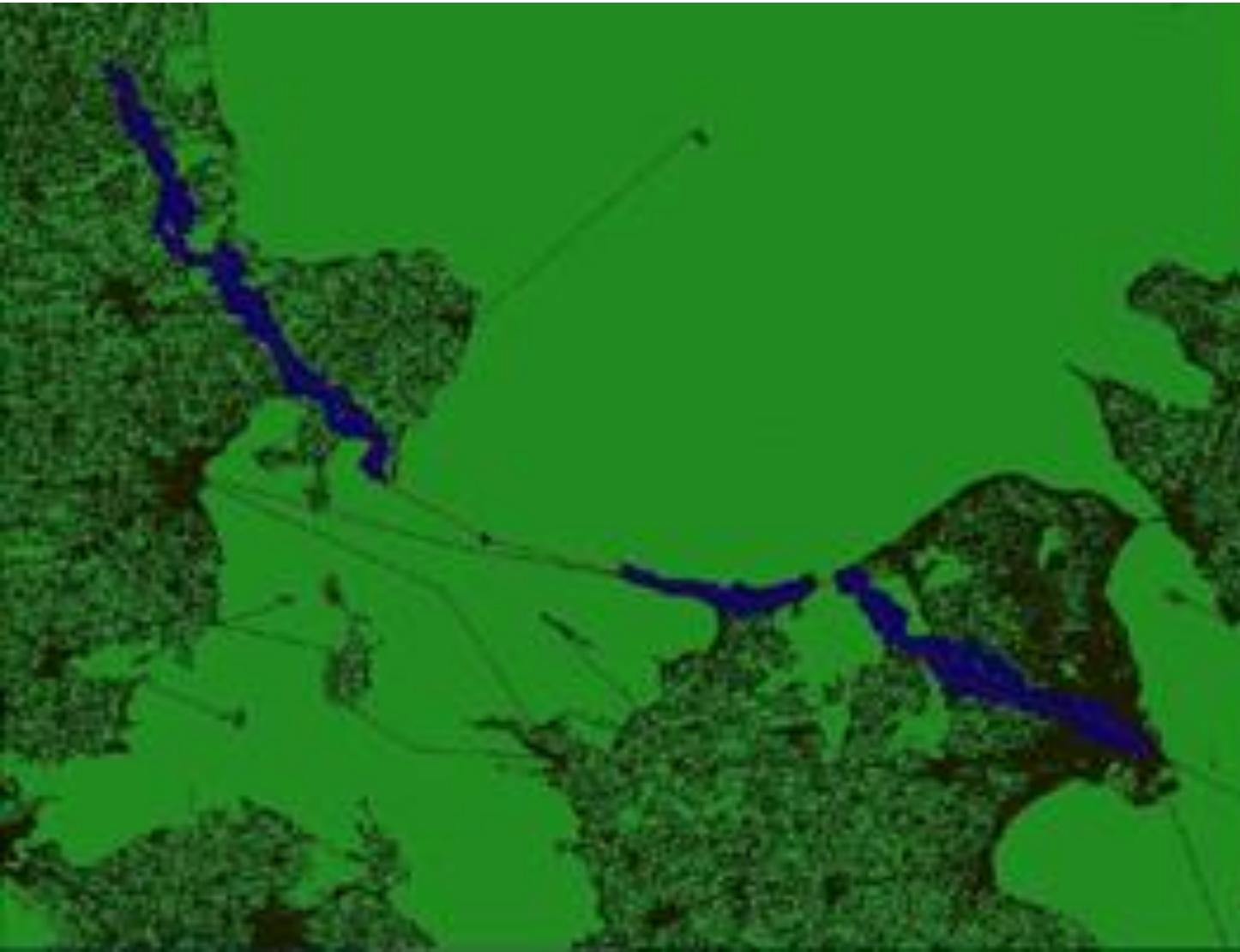
By triangle inequality

$$C(n, goal) + C^*(goal, L) \geq C^*(n, L) \Rightarrow$$

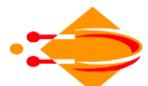
$$C(n, goal) \geq C^*(n, L) - C^*(goal, L)$$

admissible !

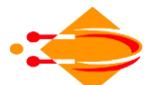
Landmark Heuristics Work!



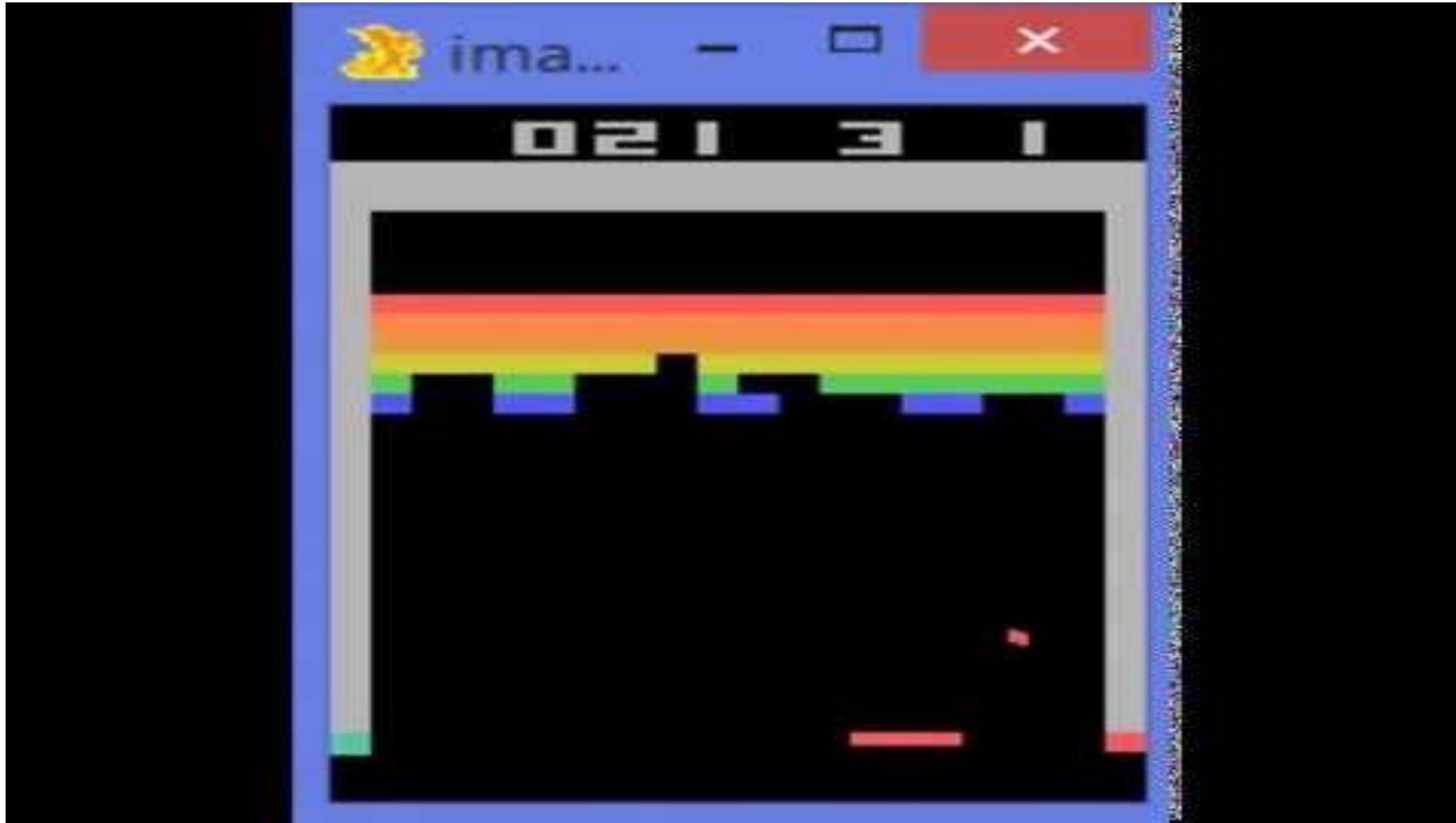
Break



DeepMind Game Algorithms



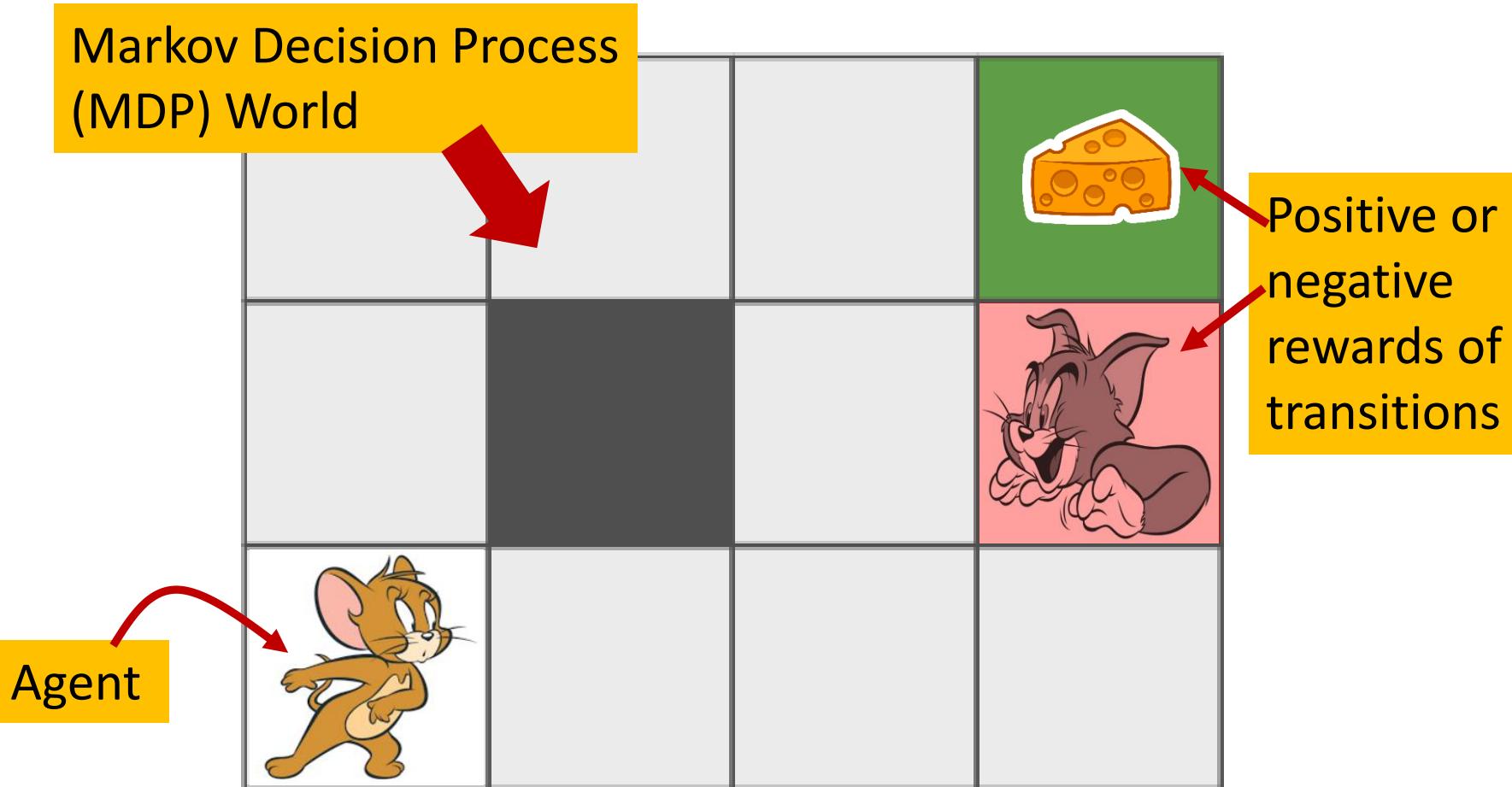
What's the Fuss About DeepMind?



How to Learn a Game from Zero Knowledge



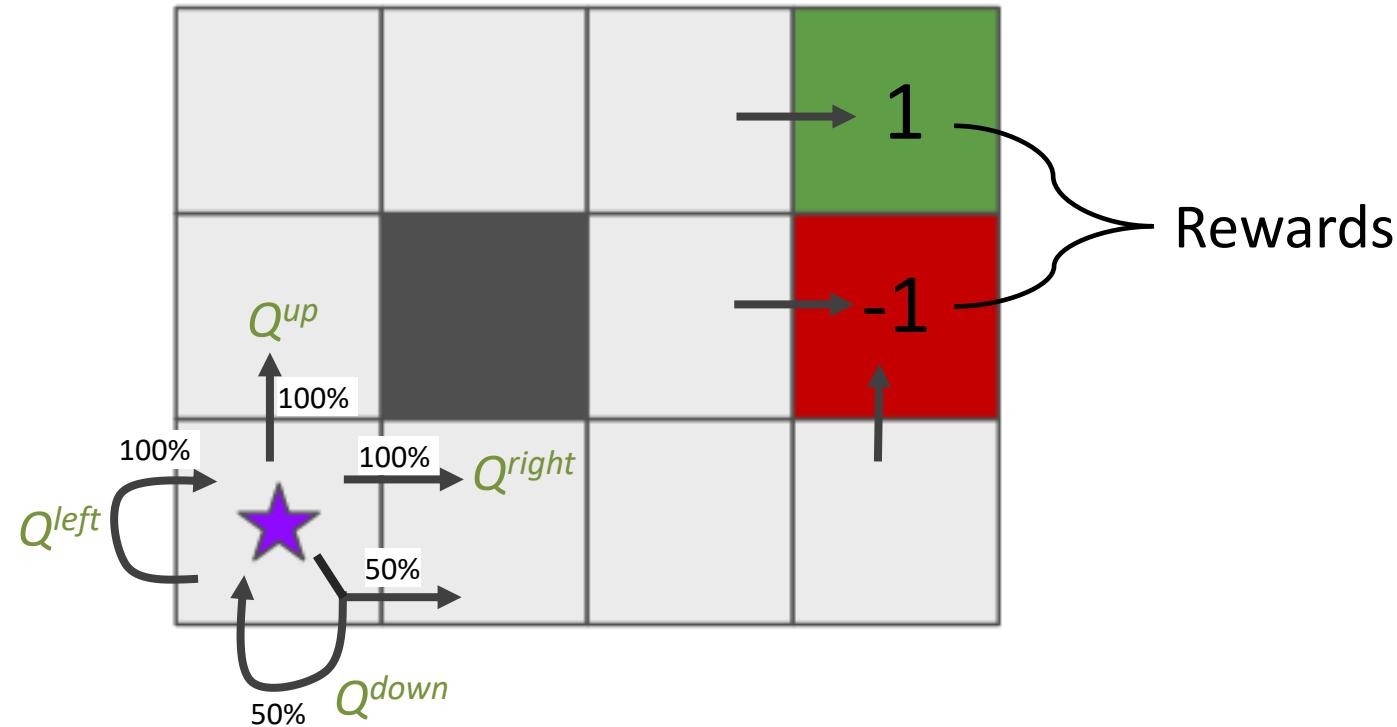
Reinforcement Learning (RL)



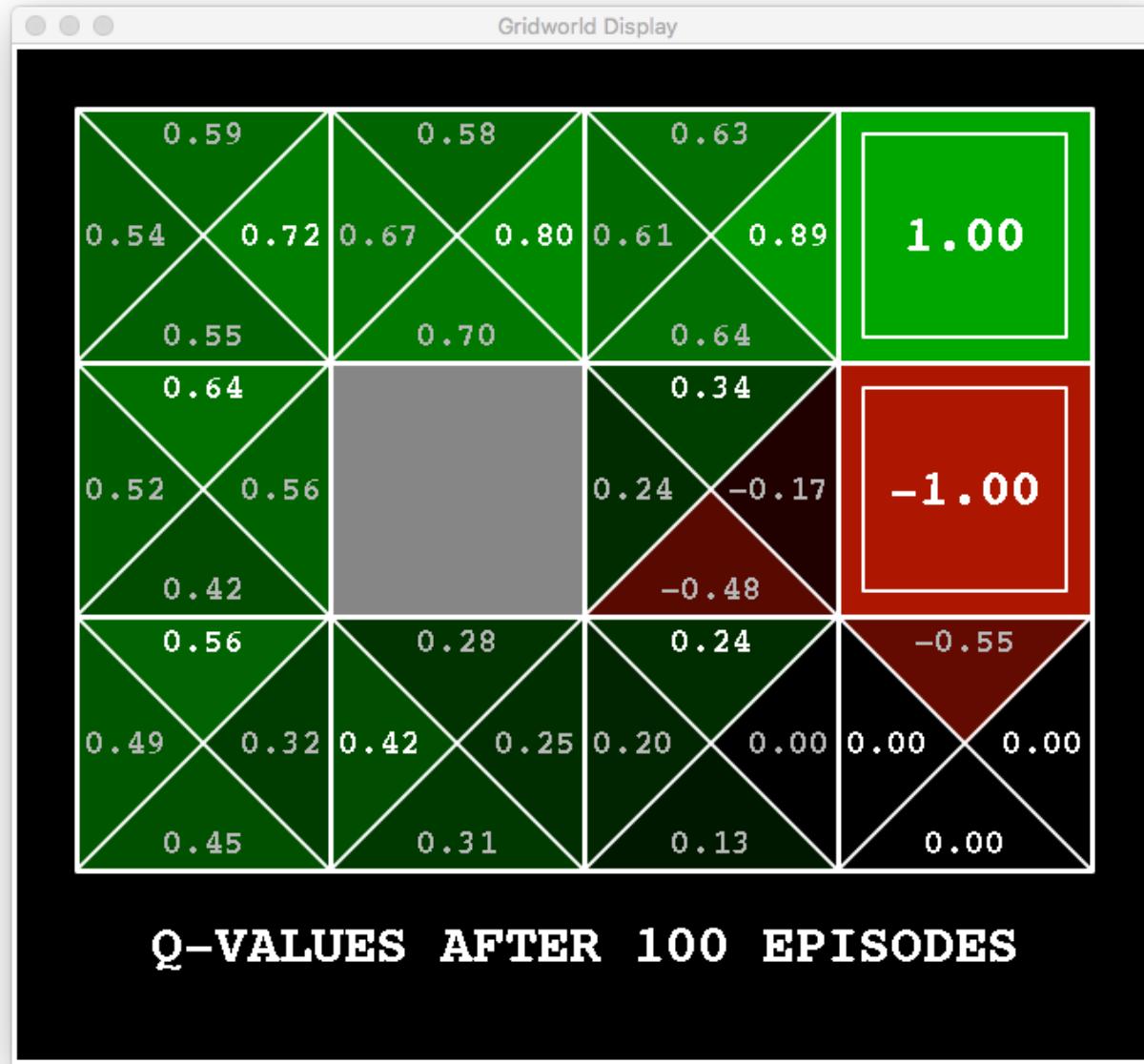
Q-Learning

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left(r_t + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}_{\text{learned value}}$$

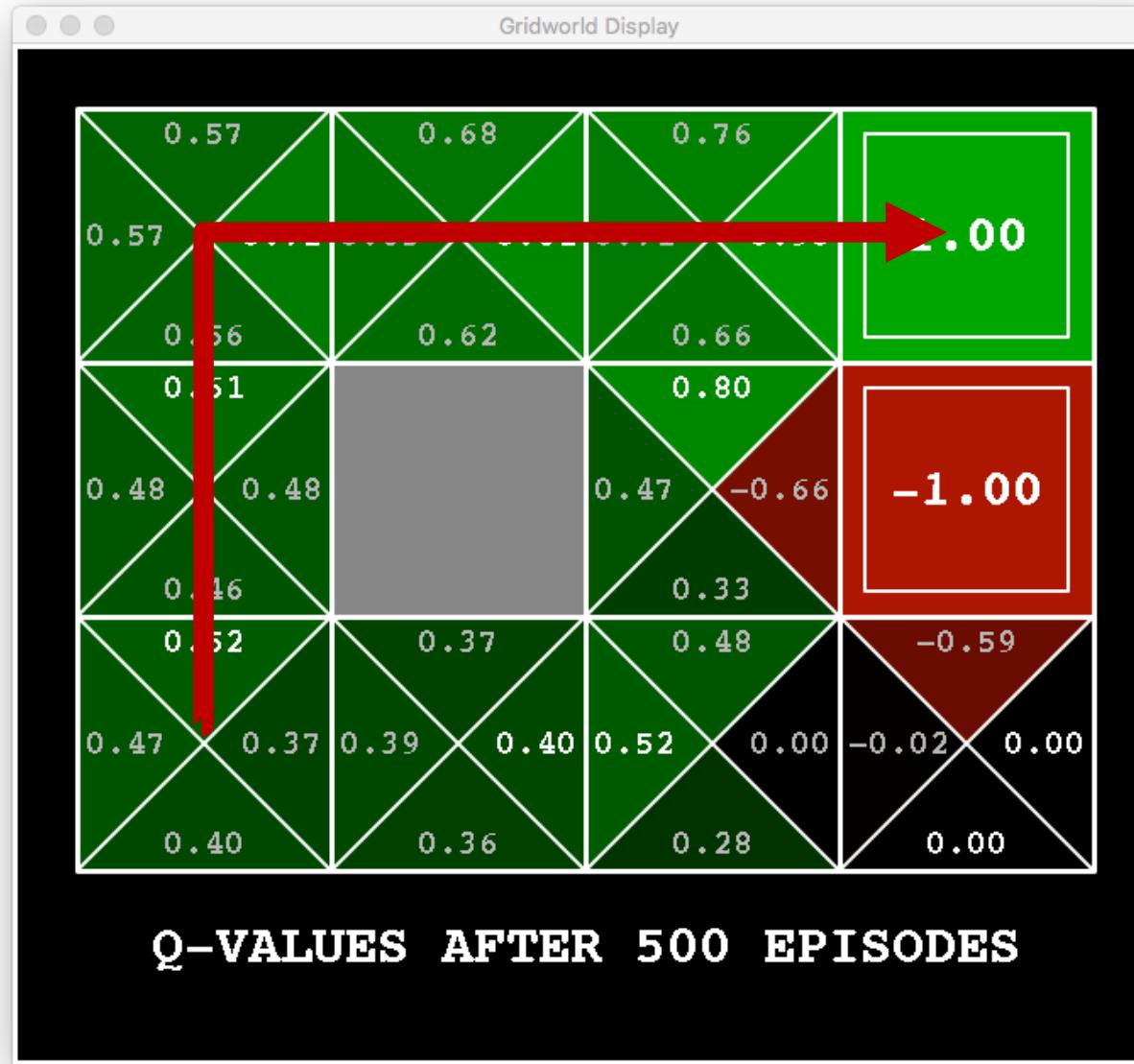
Policy (π):
Take action
with max
Q-value



Q-Learning Example (1/2)



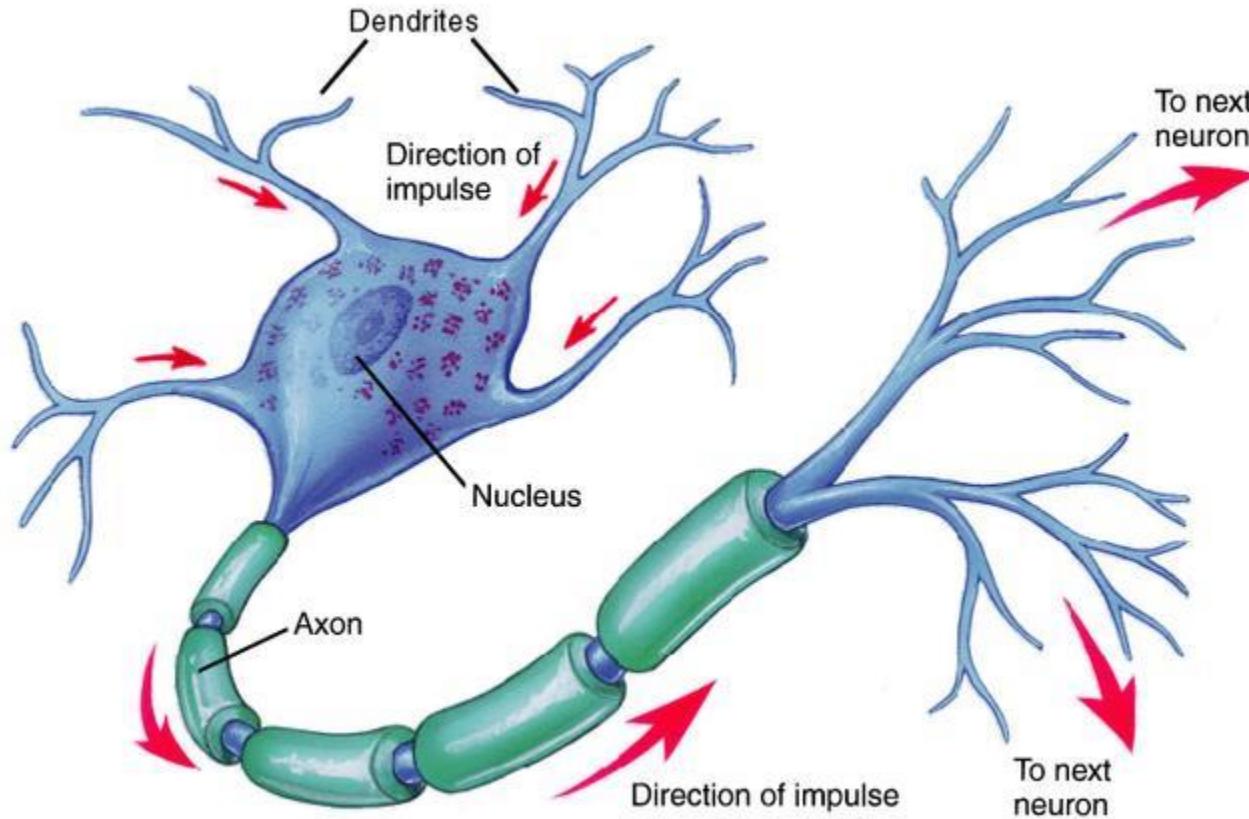
Q-Learning Example (2/2)



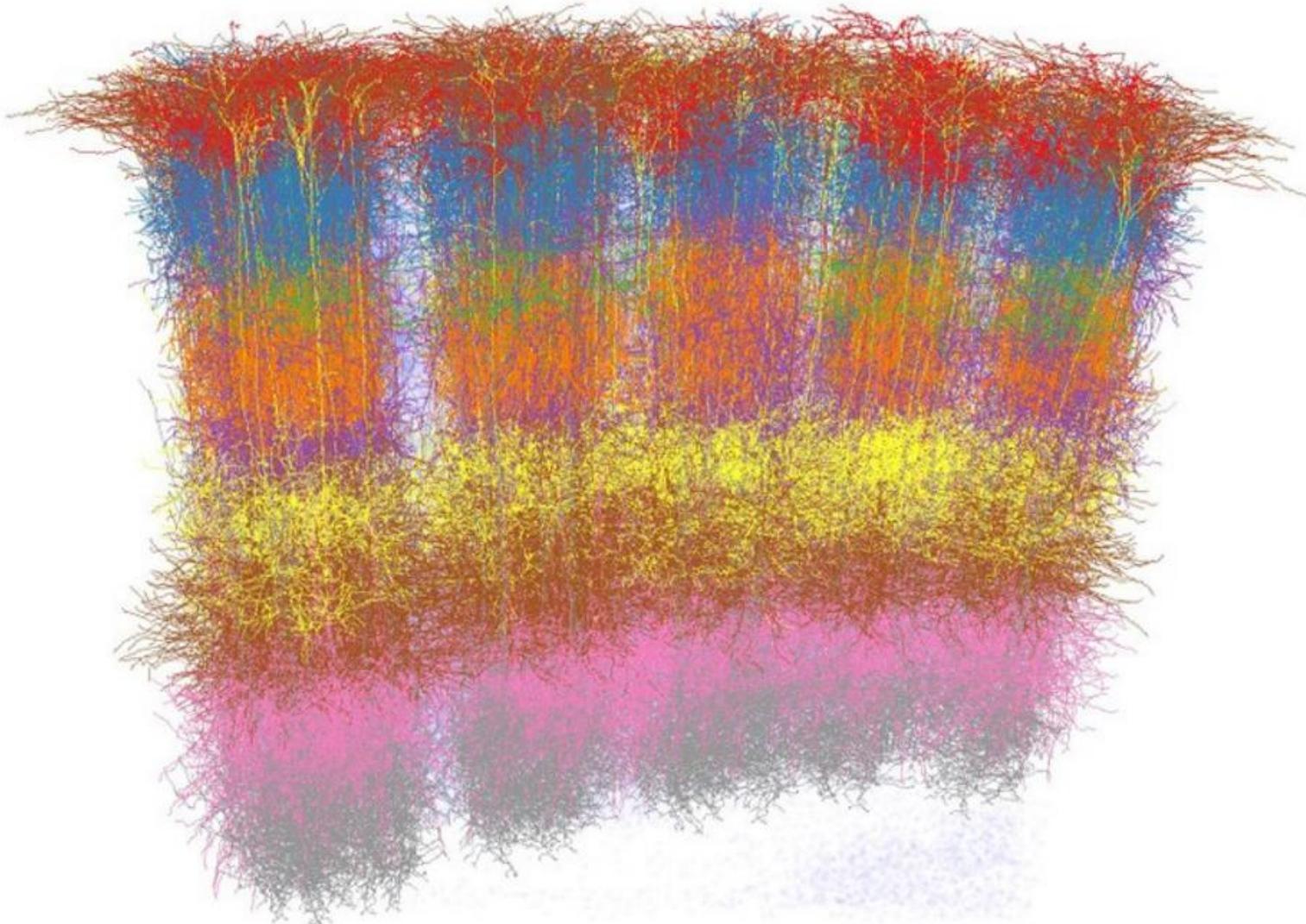
Trick: Use a **Neural Network** to Represent
the Q-Table



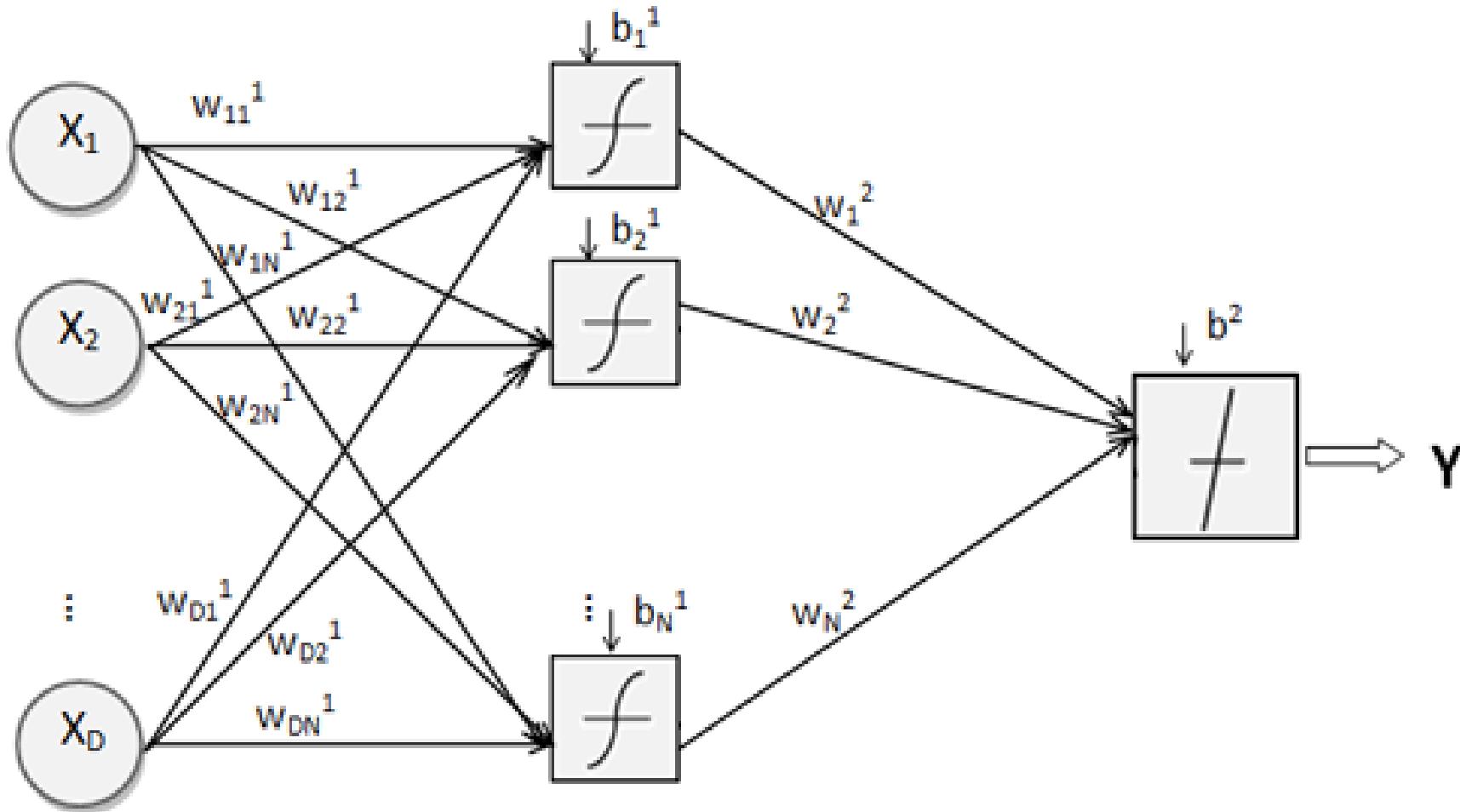
Neurons



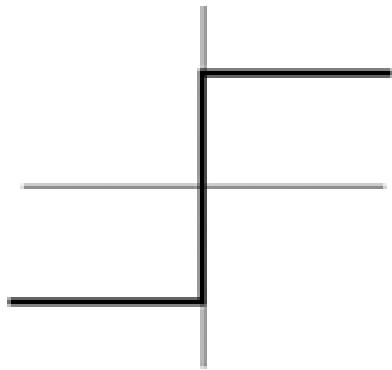
Neuron Network in Cortex



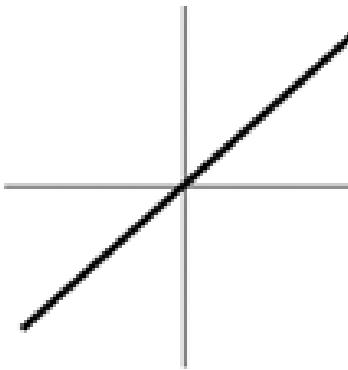
Artificial Neural Networks (ANNs)



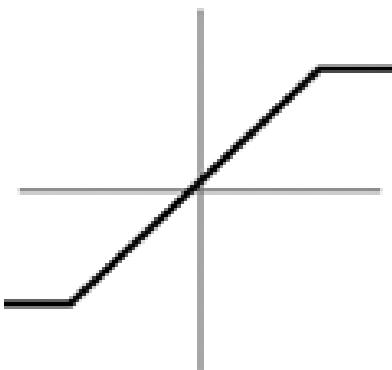
Popular Threshold Units



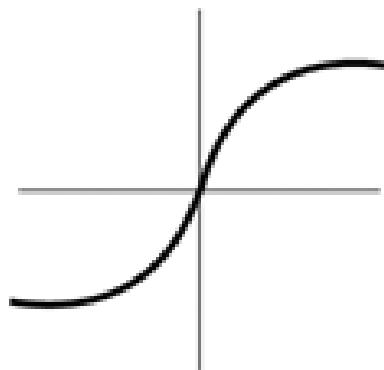
Step Function



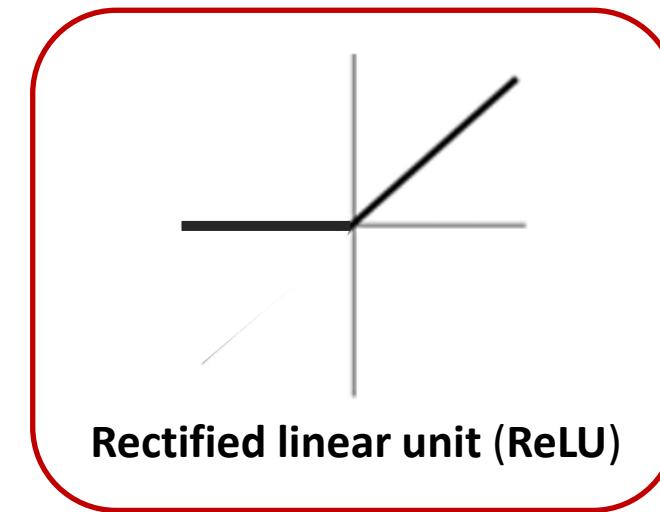
Linear Function



Threshold Logic



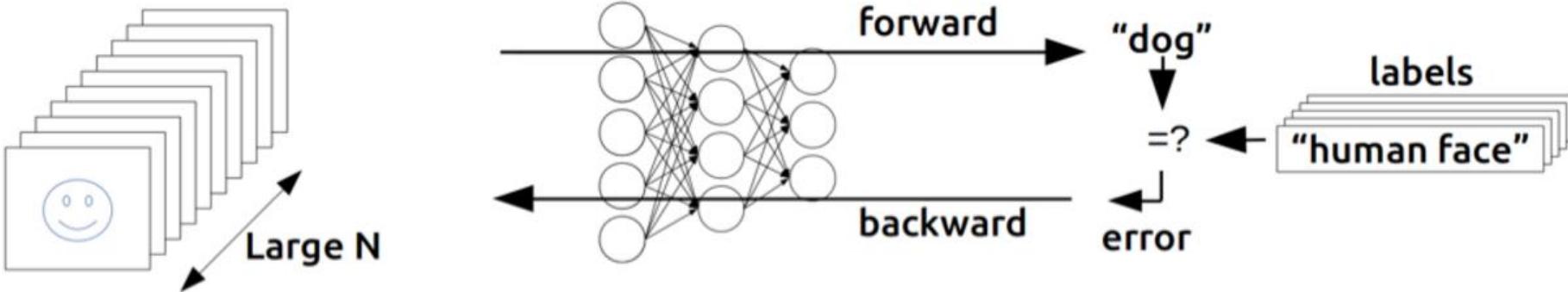
Sigmoid Function



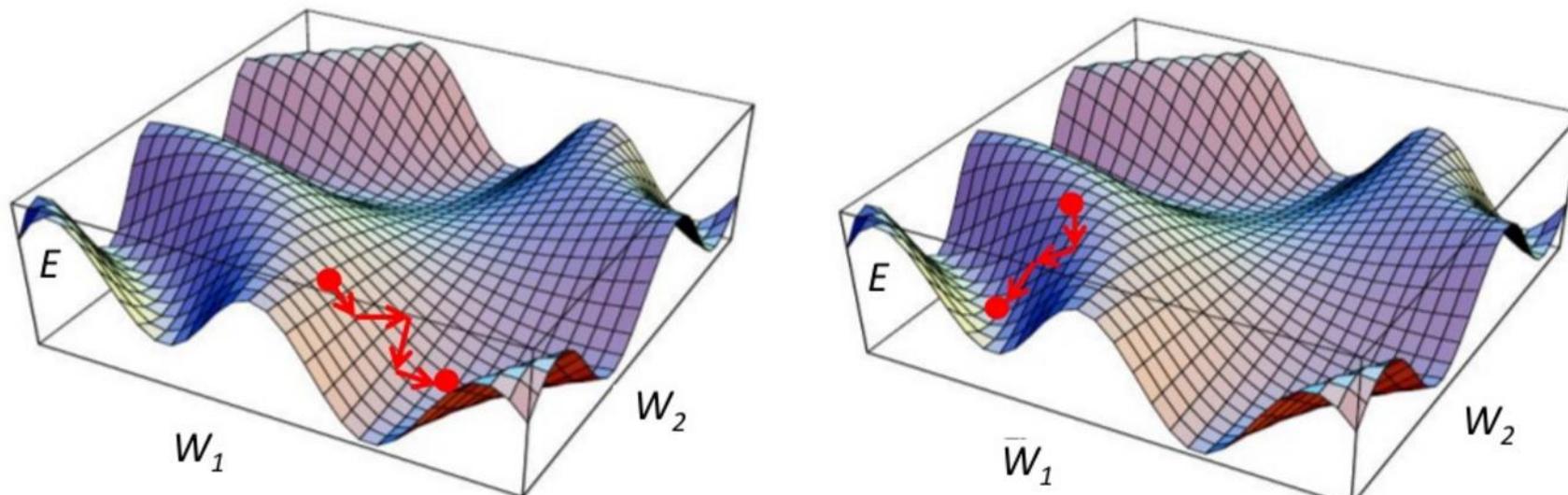
Rectified linear unit (ReLU)

<http://www.yaldex.com>

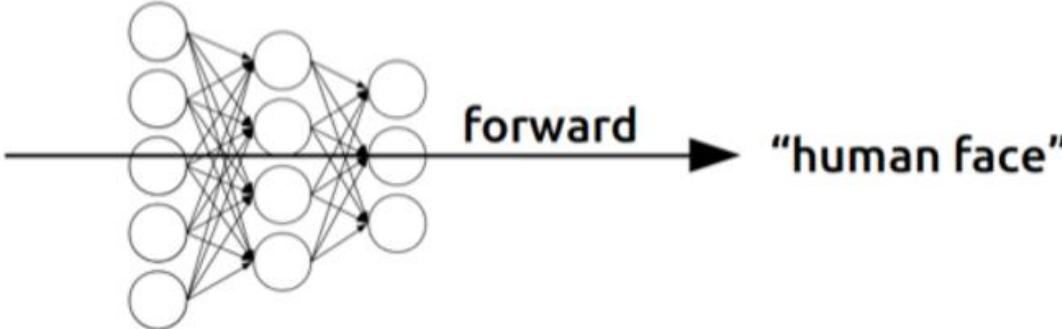
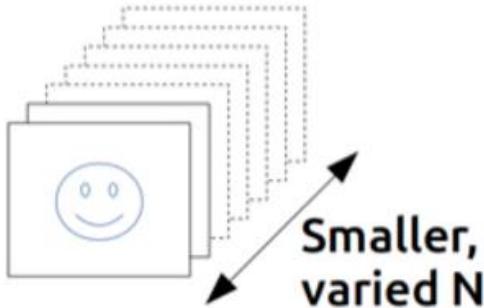
Training an ANN



Gradient Descent



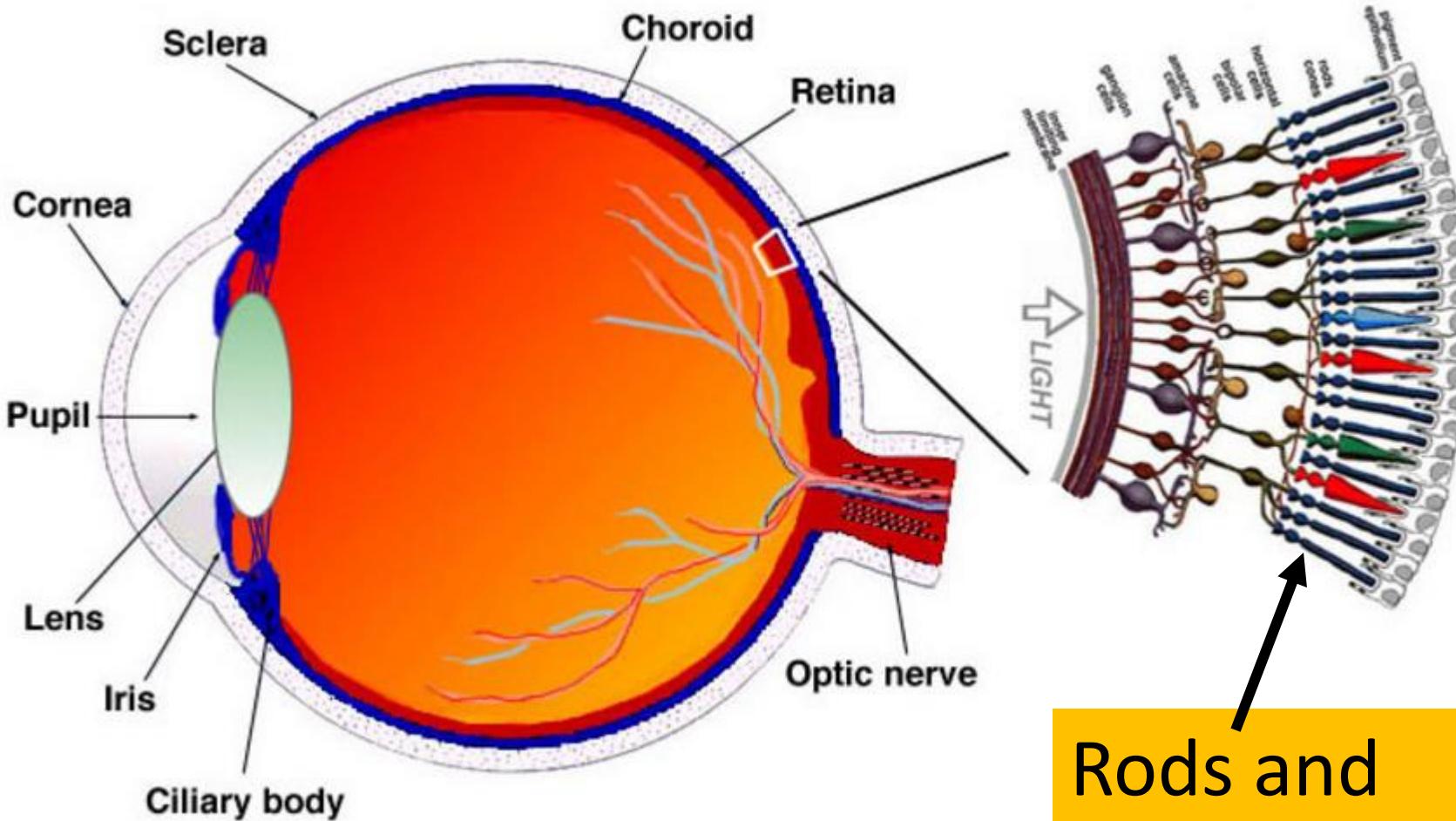
Classifying with an ANN



Practical Challenges

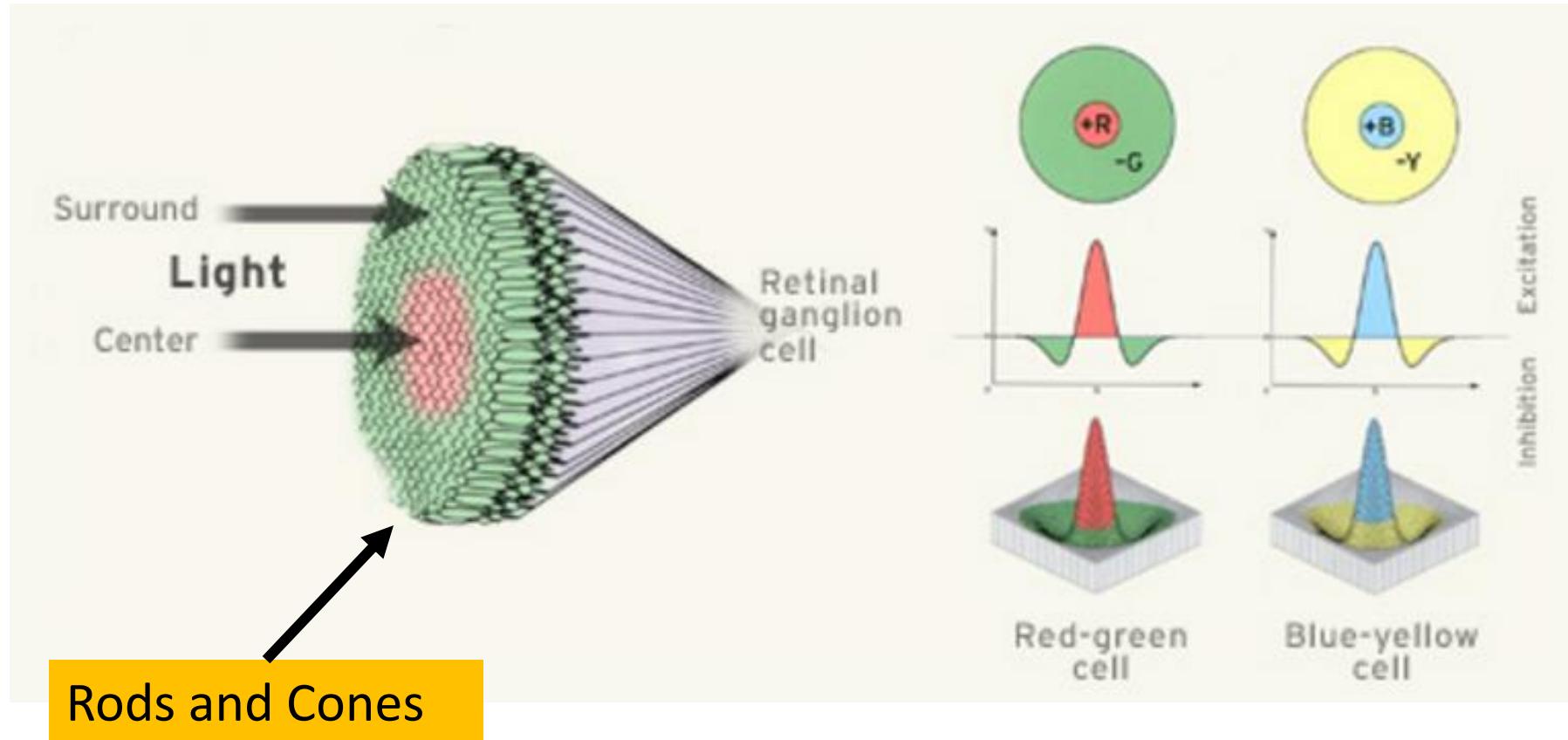
- **Curse of dimensionality:** too many weights to learn
- In image classification, full connectivity overkill
- Idea: reduce number of weights by imitating **visual perception**

Visual Processing Starts in Retina

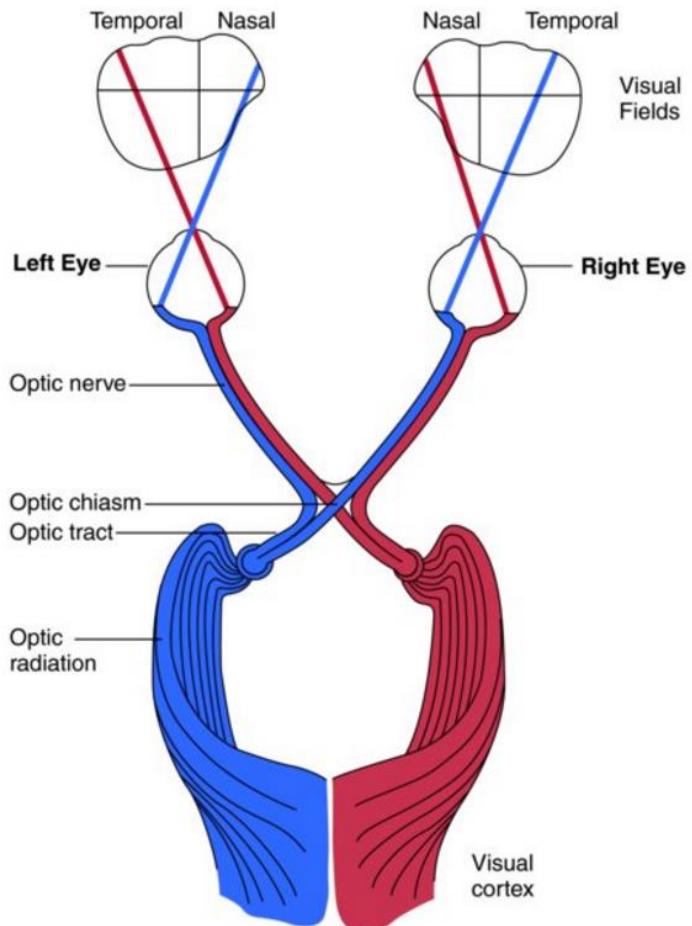


Rods and
Cones

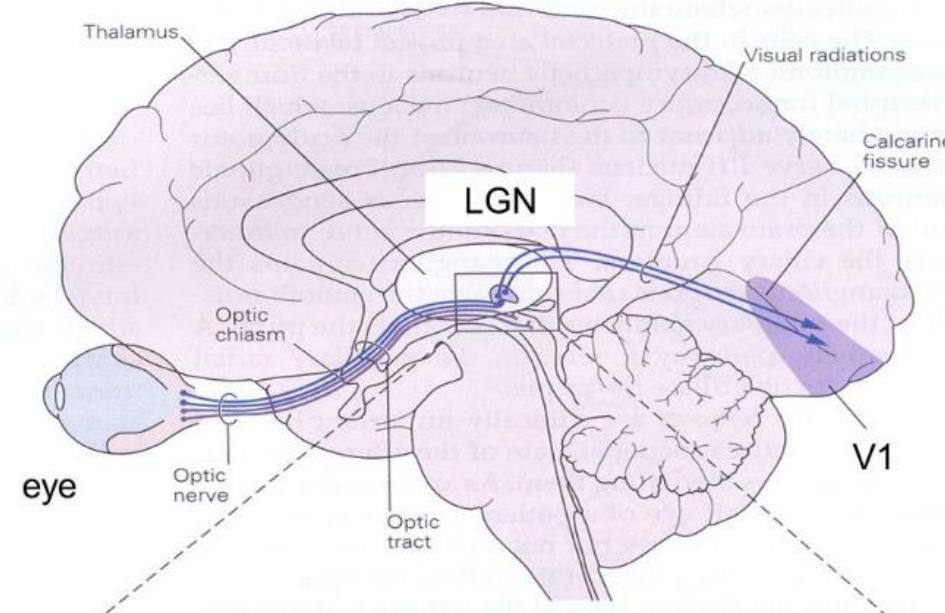
Receptive Field of Ganglion Cells



Central Visual Pathways

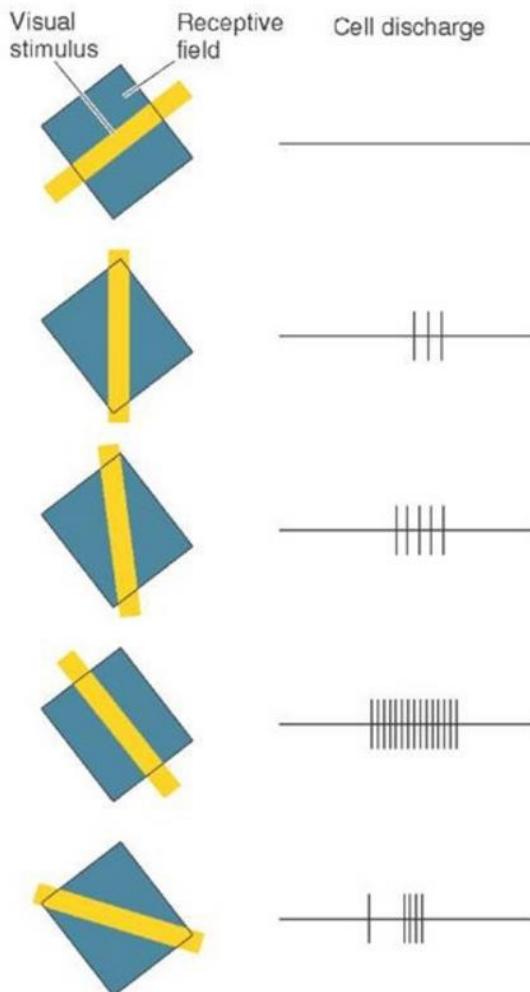


Top View



Side View

Receptive Field of Simple Cells in V1

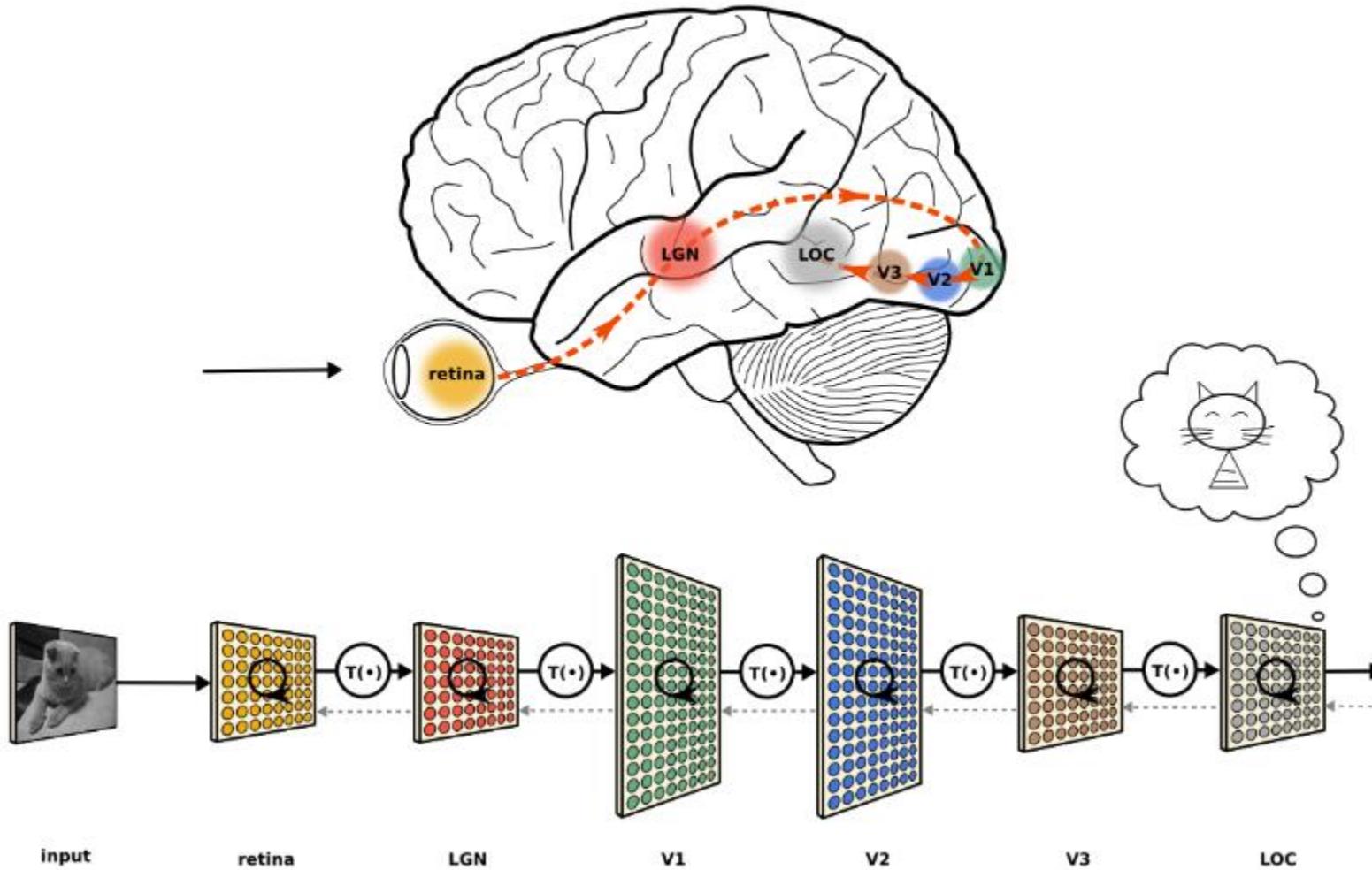


Conclusions

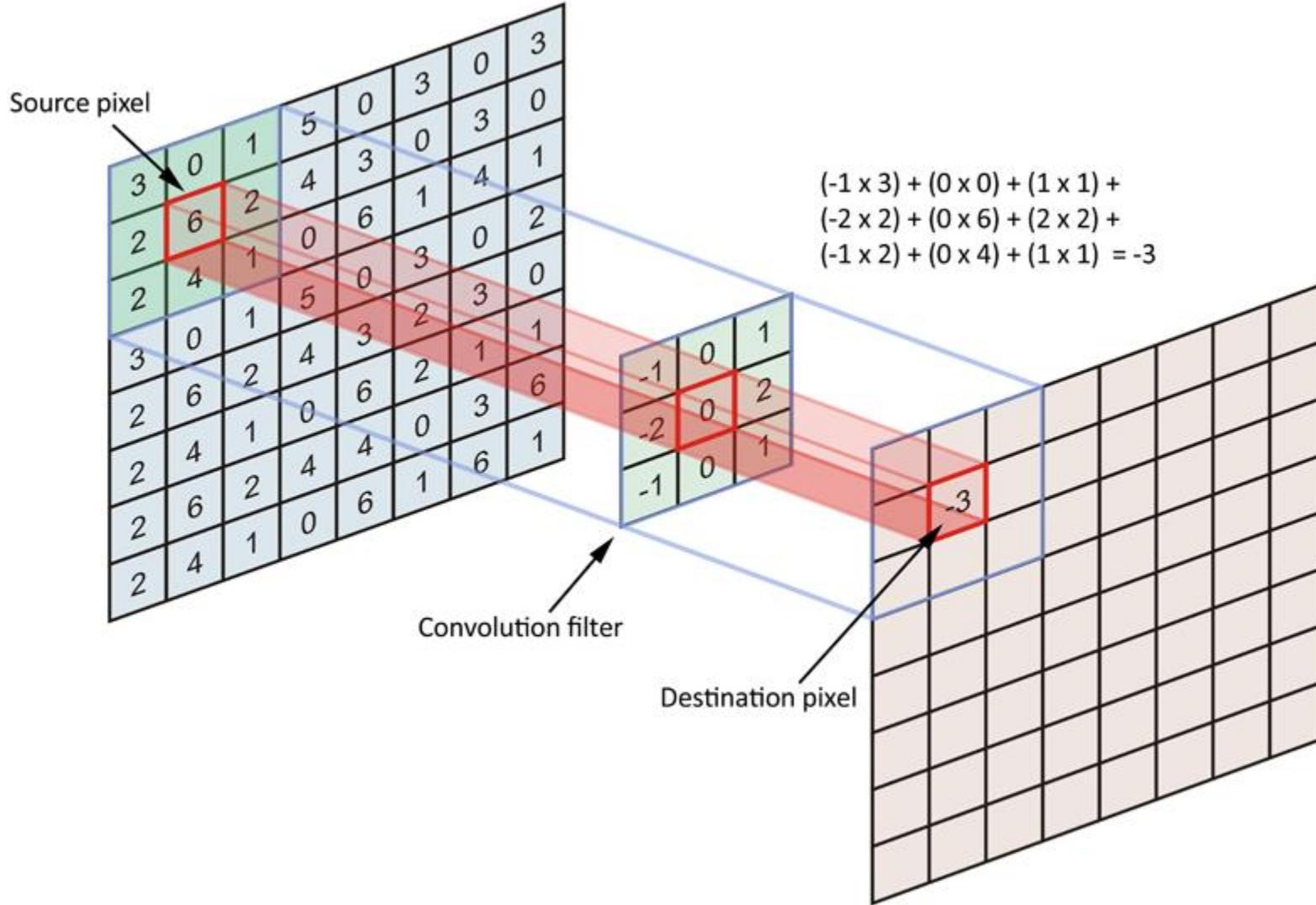
- Retina and V1 are topologically organized
- Ganglion and simple cells have receptive fields equal to **convolution filters**

Modelling the Visual Pathway

neuwritesd.org

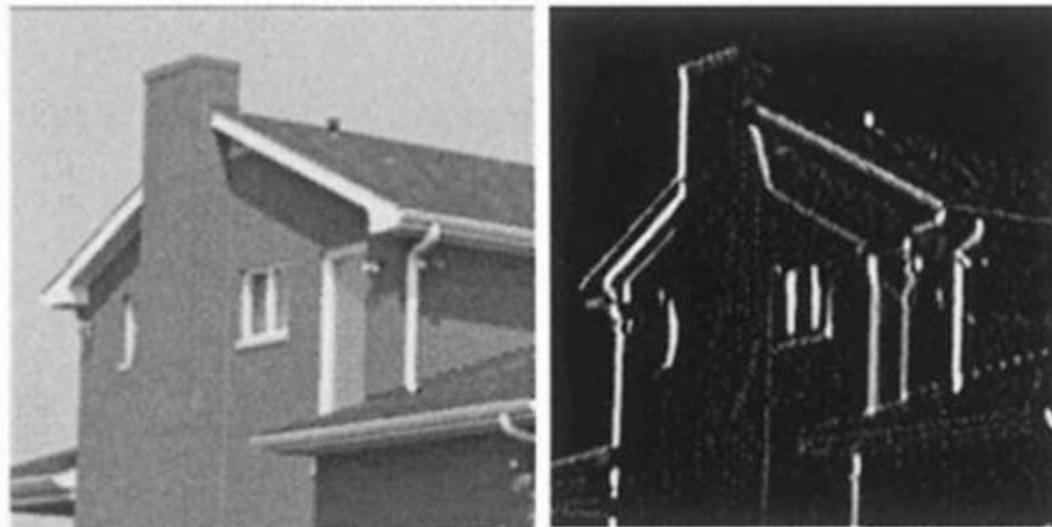


Convolution Filter



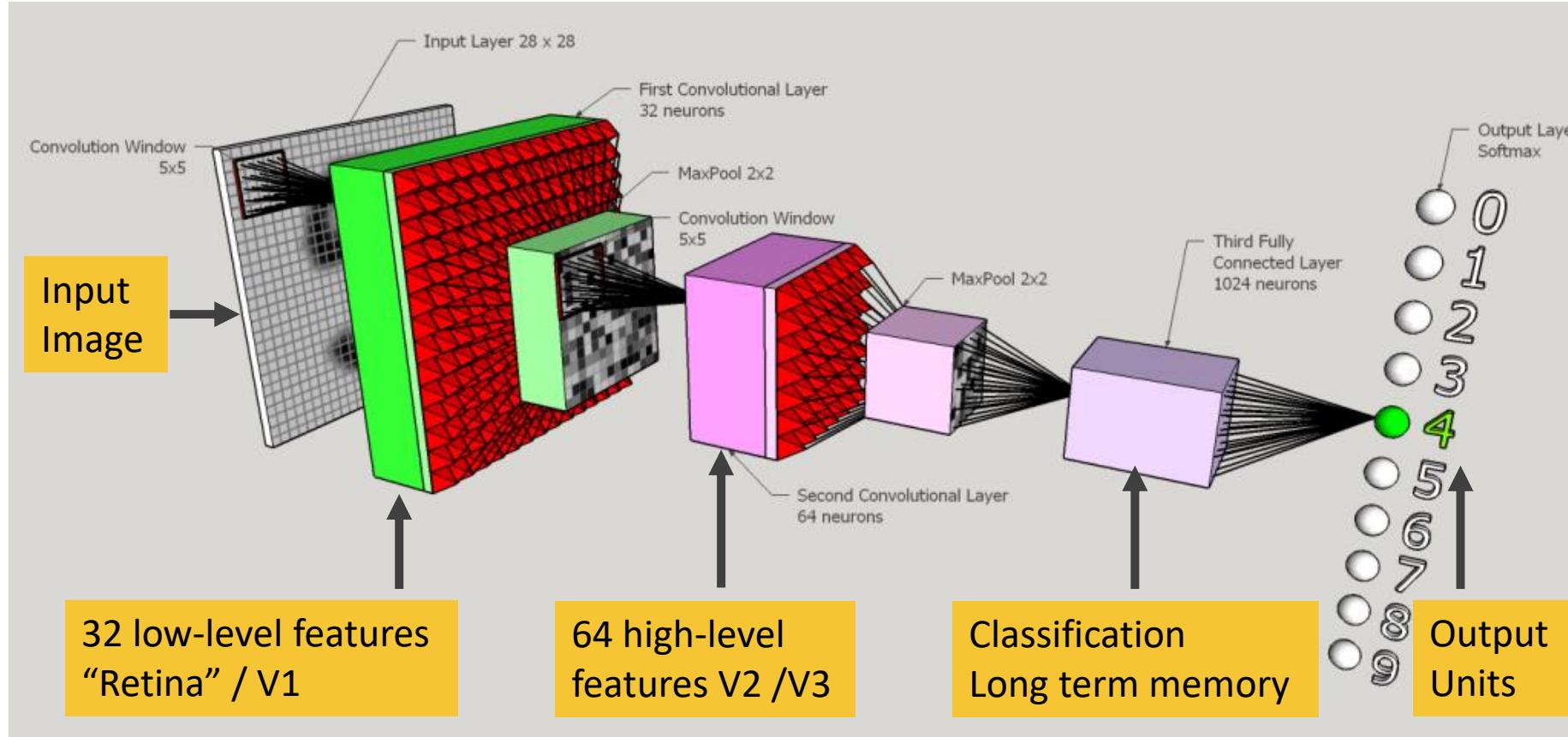
Example: Vertical Edge Detection

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$



what-when-how.com

Convolutional Neural Networks (CNNs)



zderadicka.eu

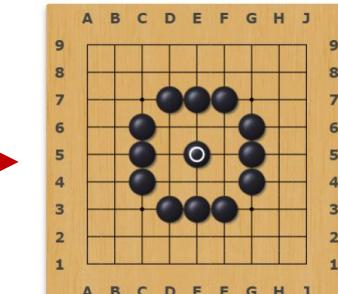
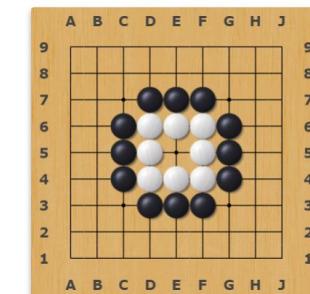
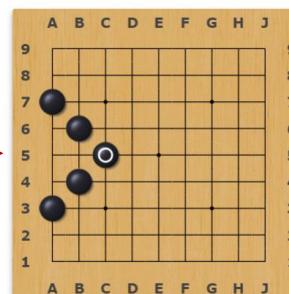
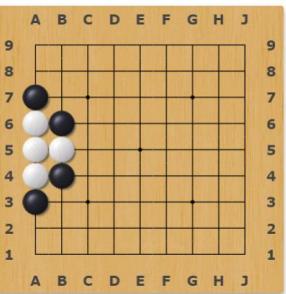
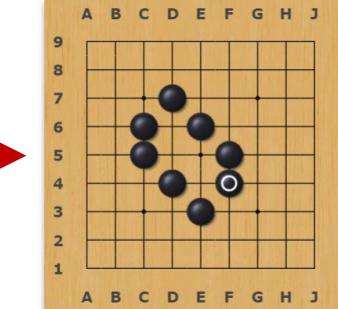
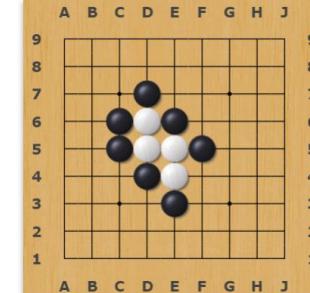
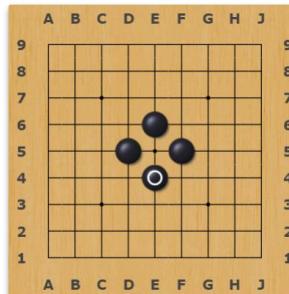
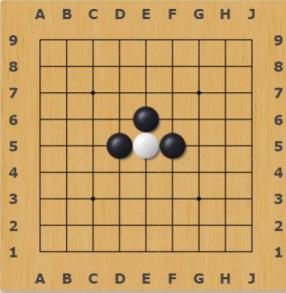
AlphaGo Zero



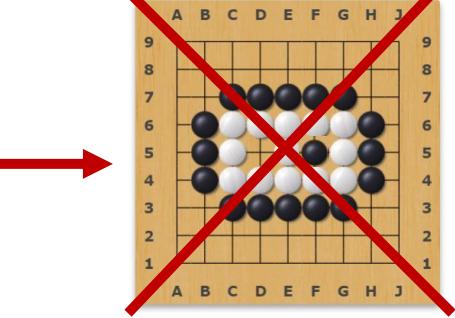
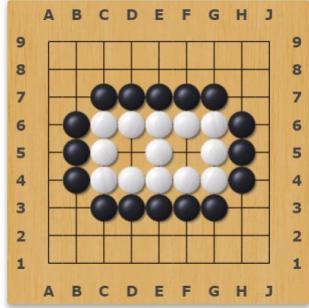
Go: A Zero-Sum Turn-Taking Game



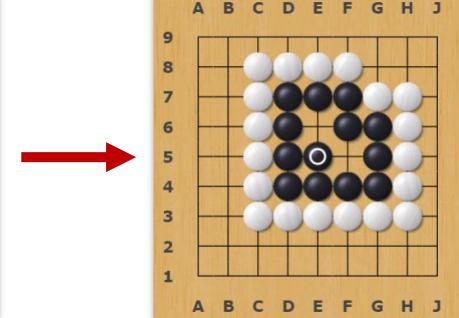
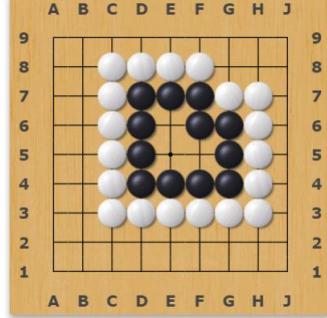
Captures



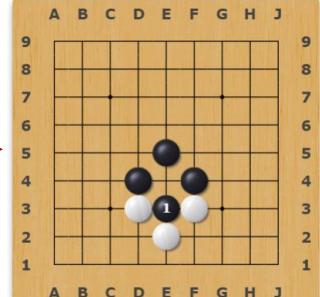
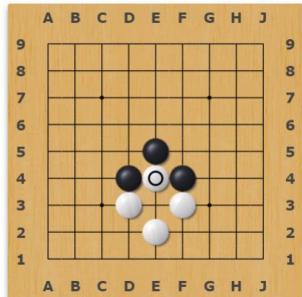
Capture Protection and Ko Rule



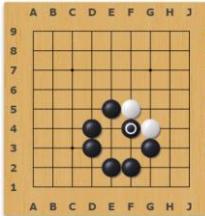
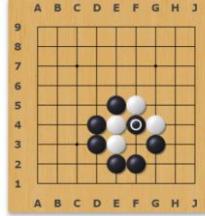
No capture with two holes!
Black F5 will suicide off board...



Capture protection

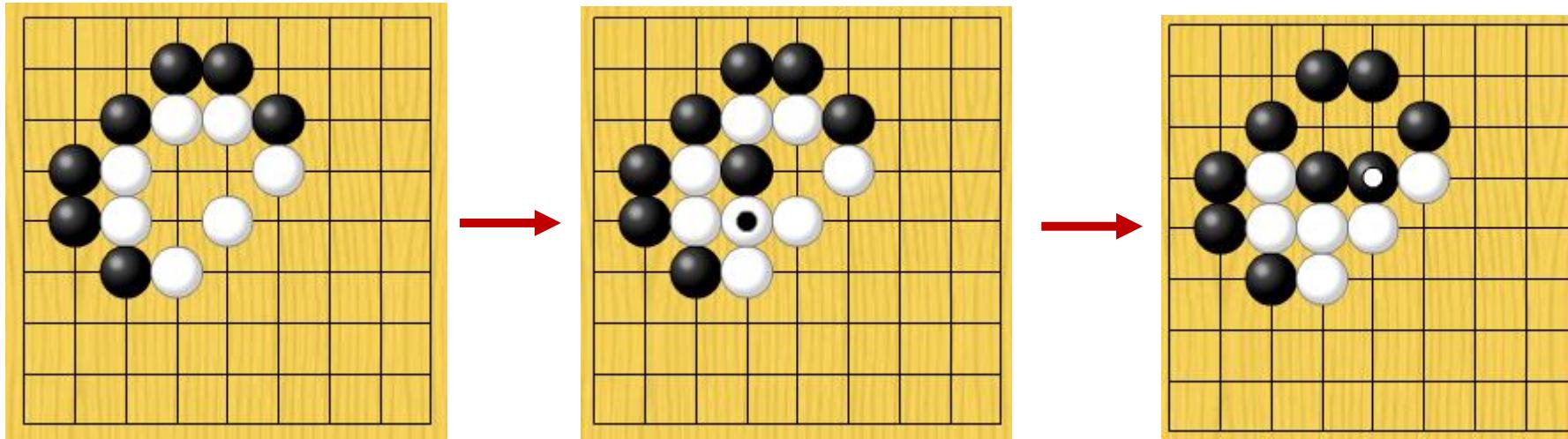


Ko rule: White cannot do E4 in next move, but ok later

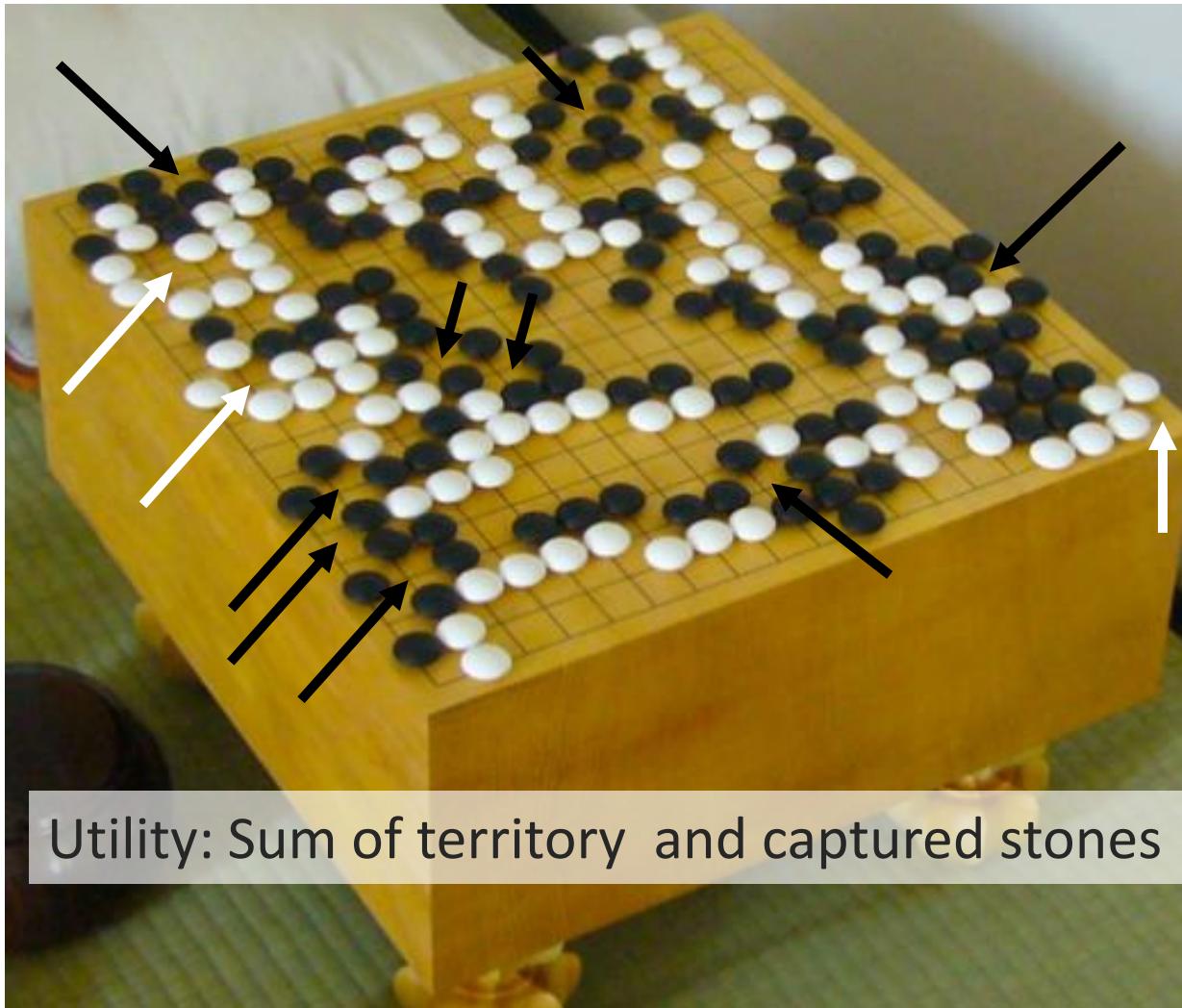


No Ko rule applies, since no direct reverse to a previous state

Double Atari Example



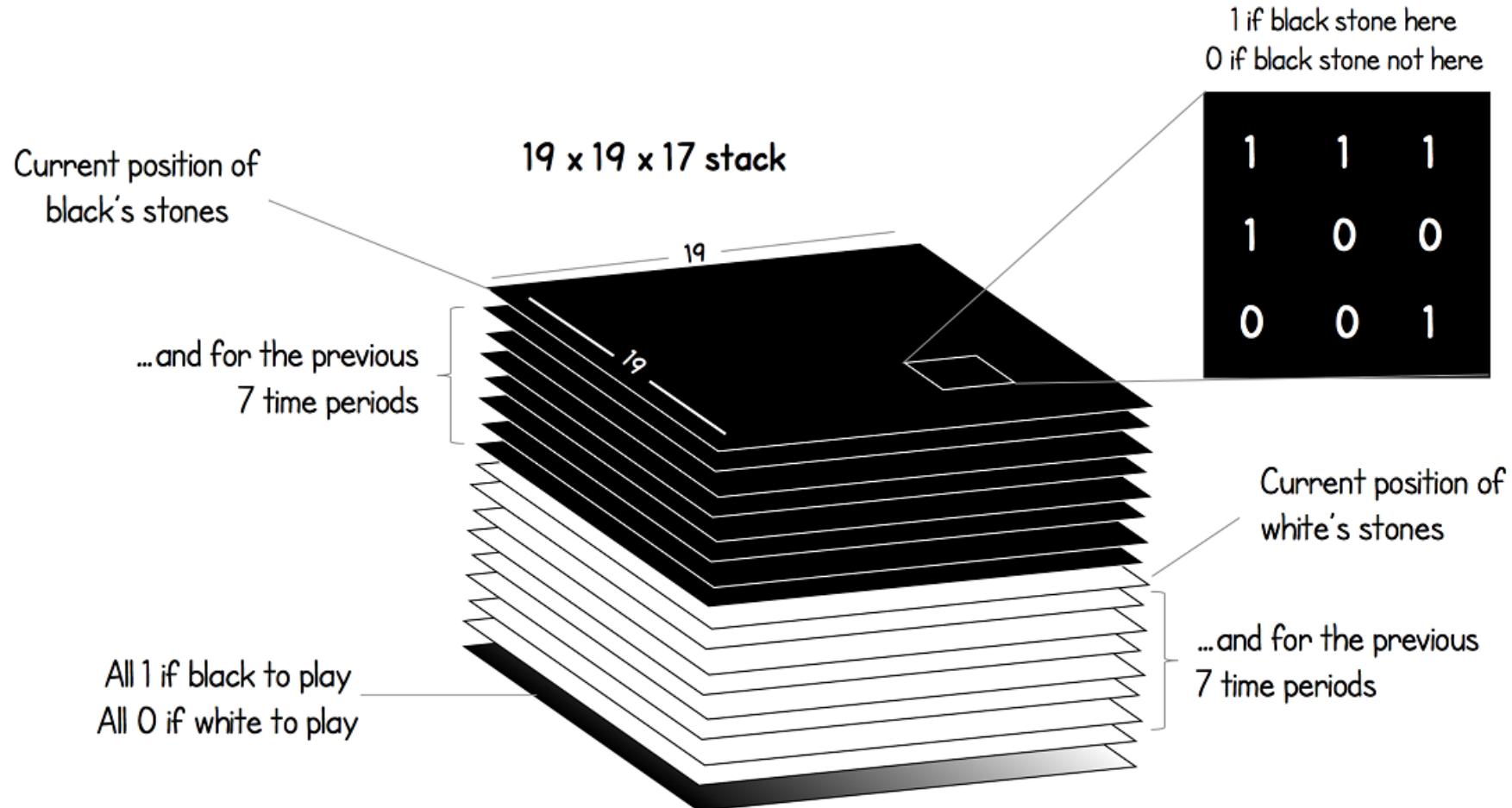
Final State Example



AlphaGo Zero's Neural Network

- Input / Output: $f_{\theta}(s) \rightarrow (p, v)$
 - s Board state including history (last 7 moves)
 - P Best move probabilities for s
 - v Board value of s
 - θ Arc weights of neural network

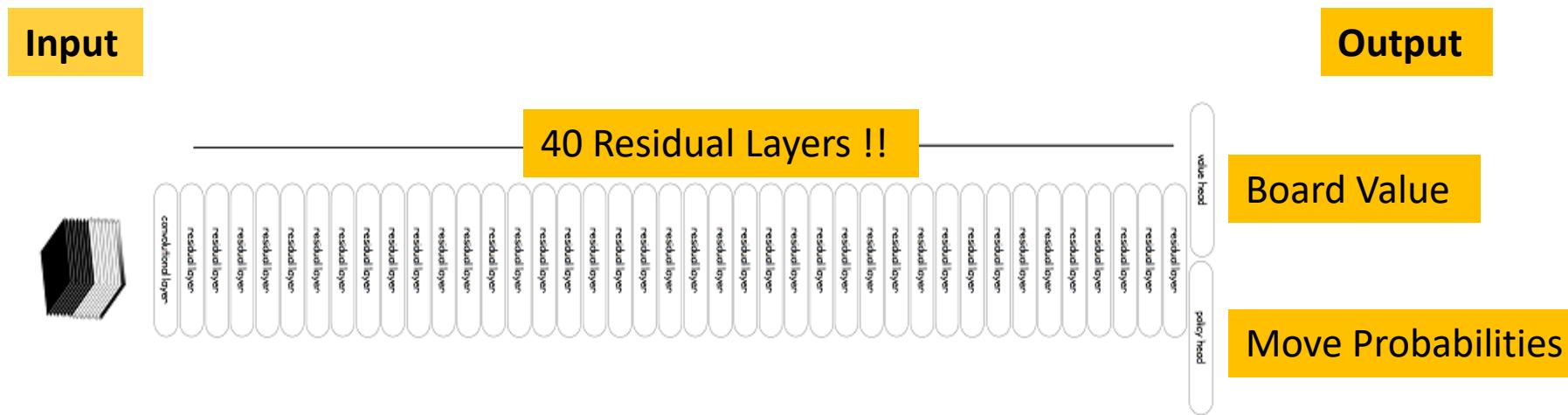
NN Architecture: Input



medium.com



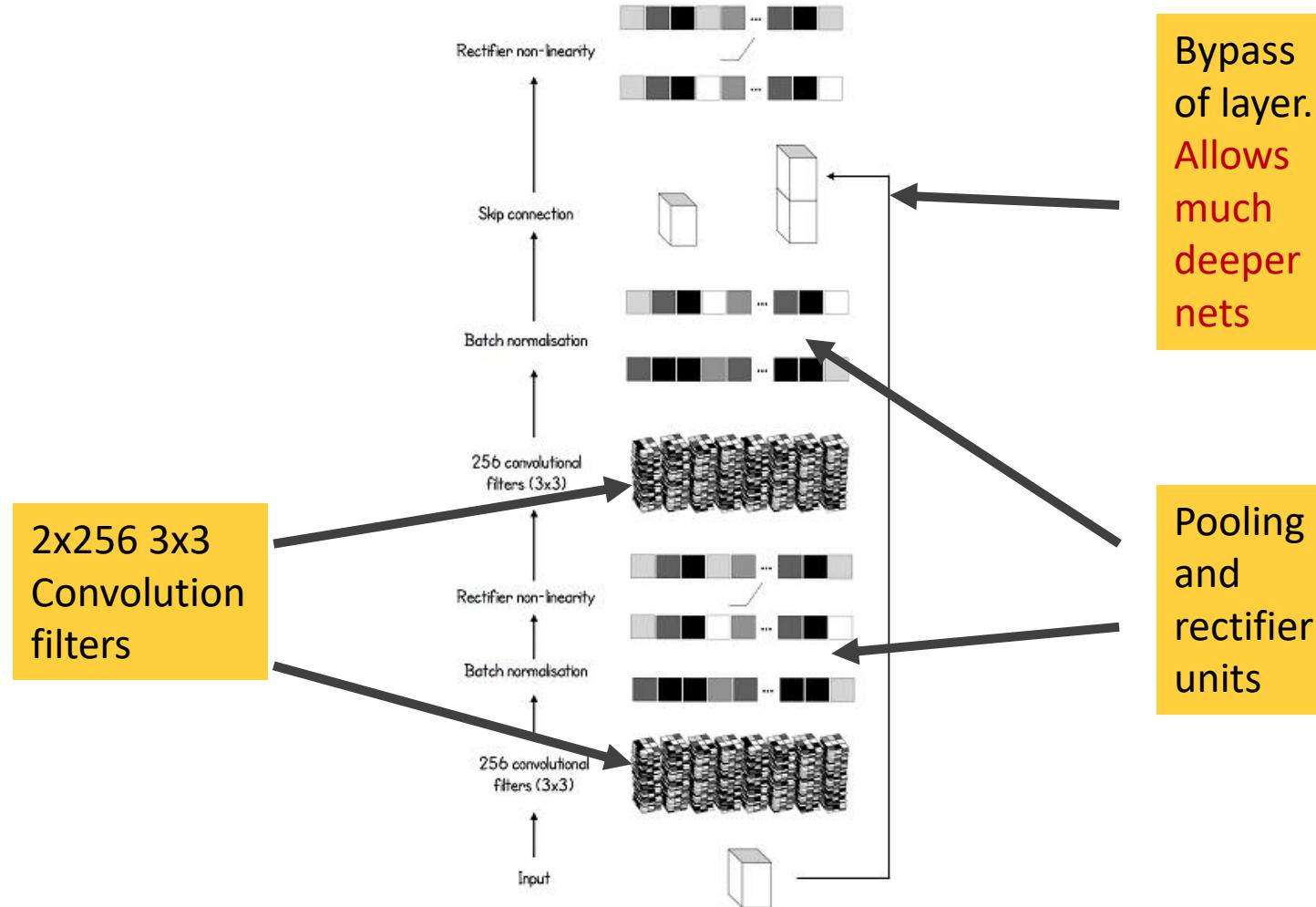
Deep Network



medium.com



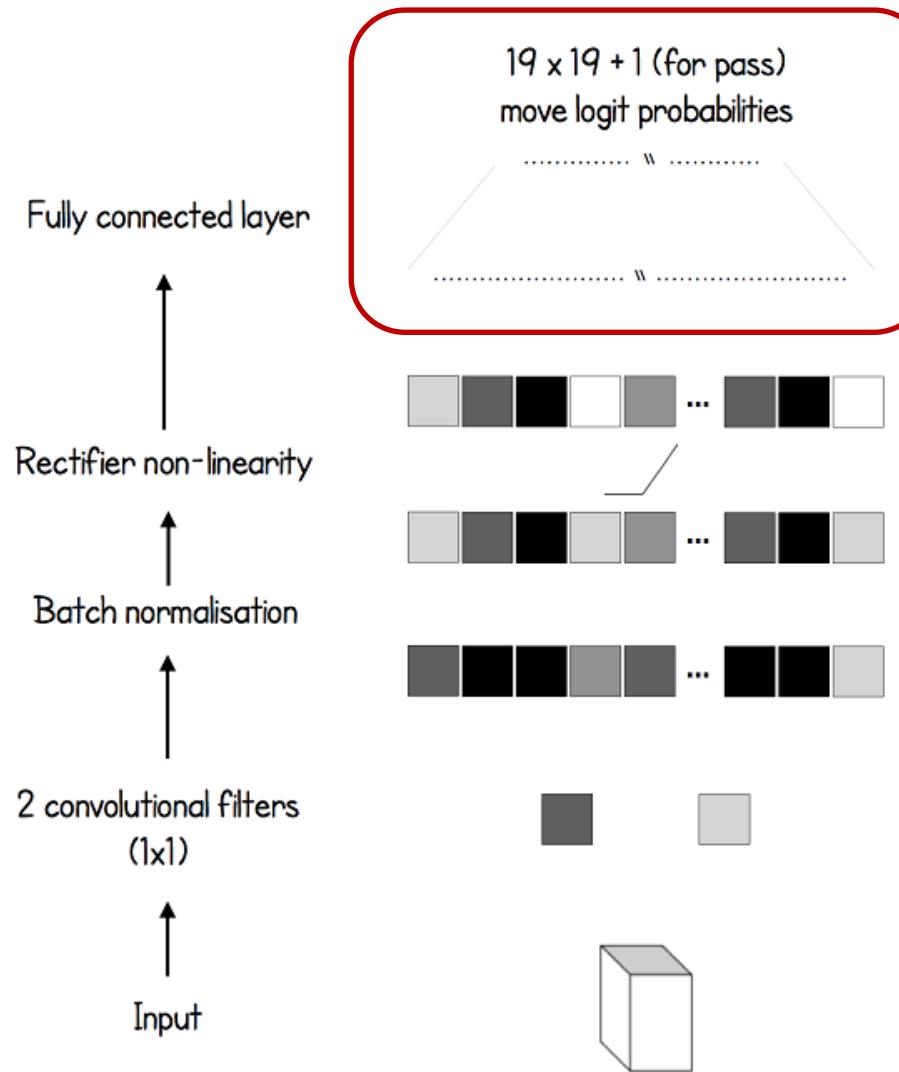
A Residual Layer



medium.com



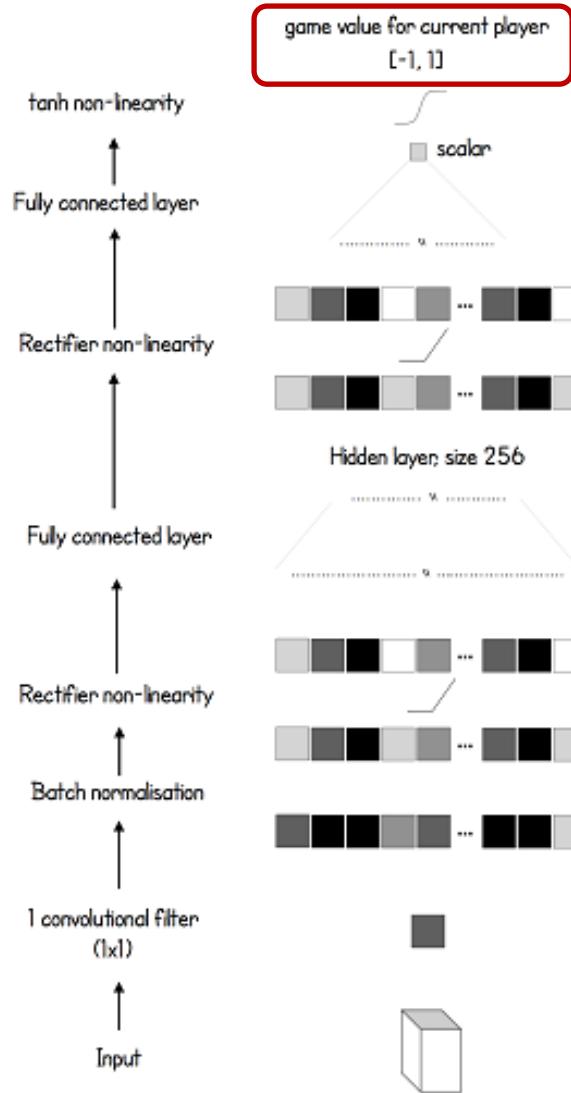
Move Probability Output



medium.com



Board Value Output

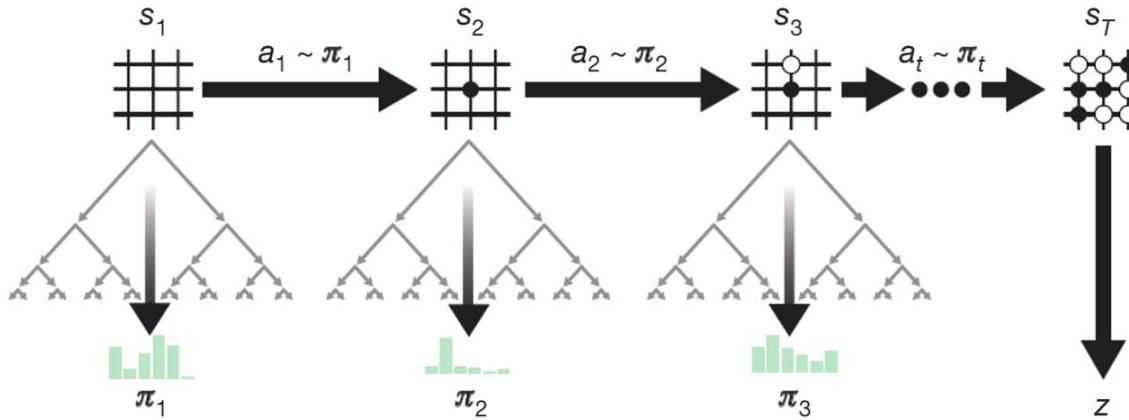


medium.com

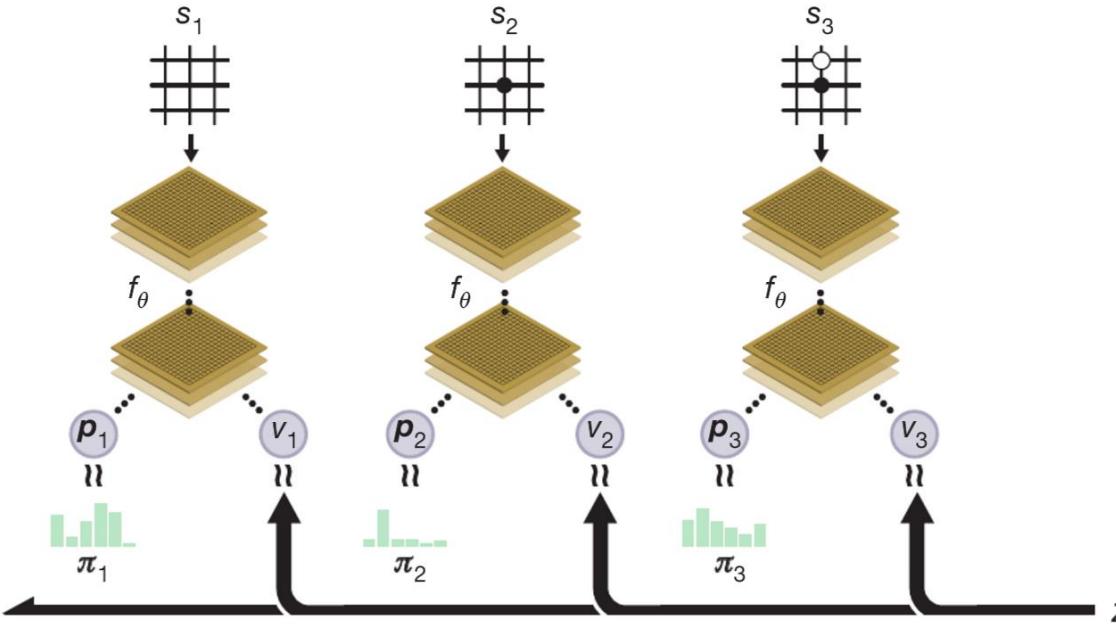


Self-Play Reinforcement Learning

Self-play



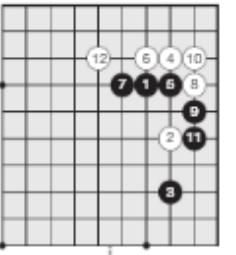
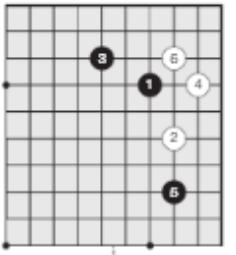
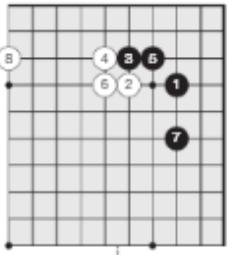
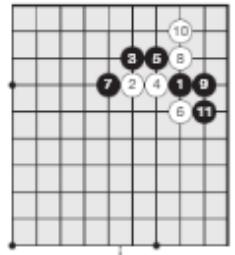
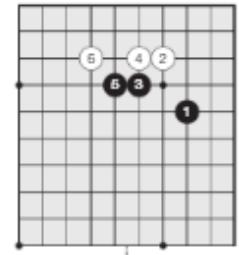
Neural
network
training



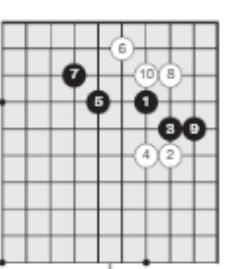
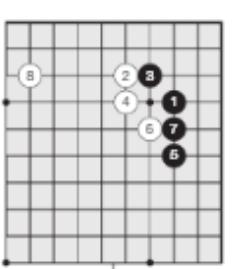
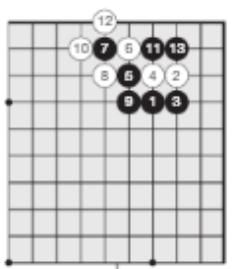
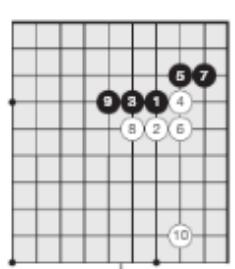
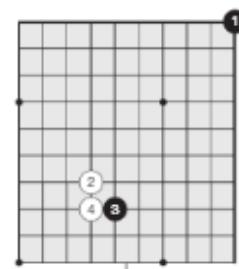
Empirical Analysis

- NN initiated with random weights (tabula rasa)
- 4.9M games of self-play generated
- 1600 simulations of each MCTS (~ 0.4 sec)
- 72h on single machine with 4 TPUs

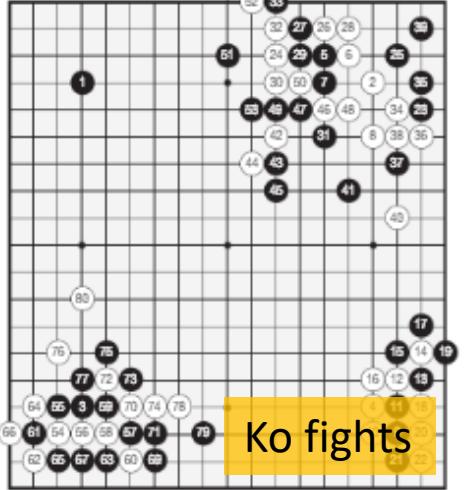
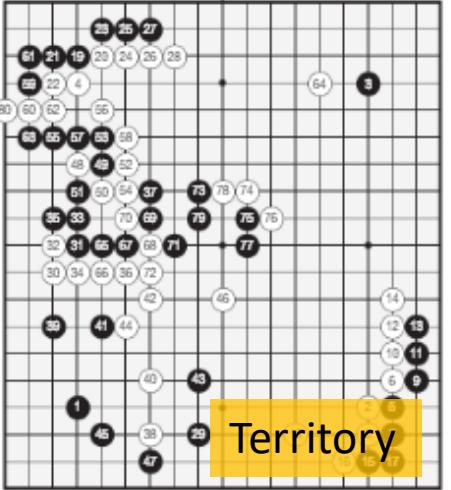




5 known
Joseki
discovered



Most
frequent
Joseki



Lee Sedol
“Divine
moves”
played by
AlphaGo

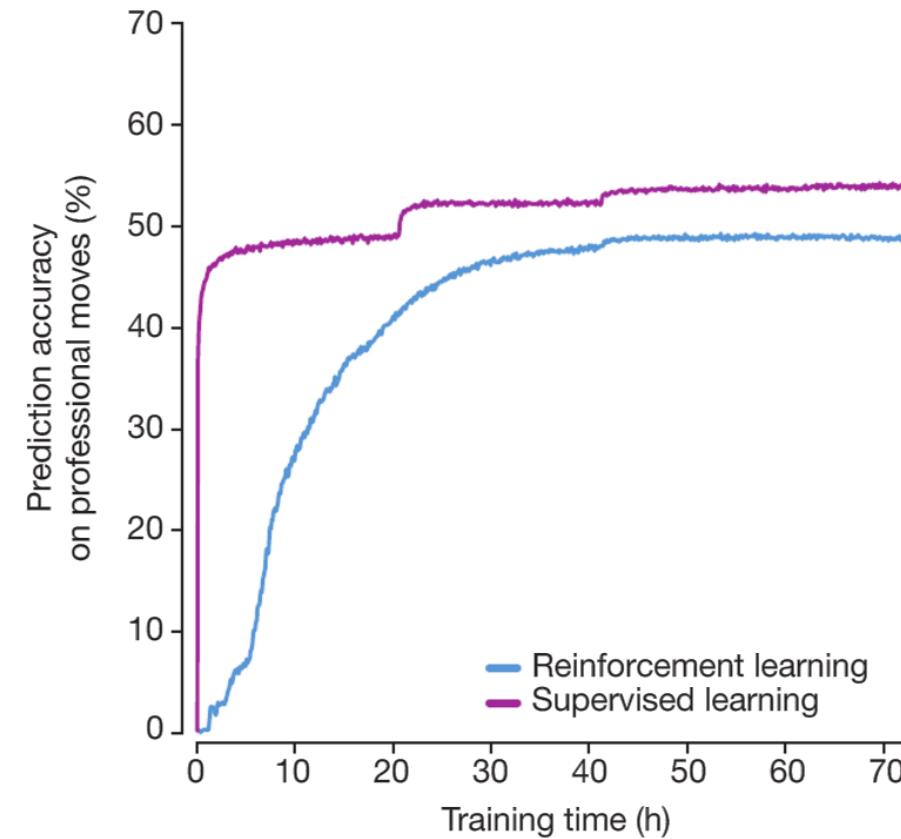
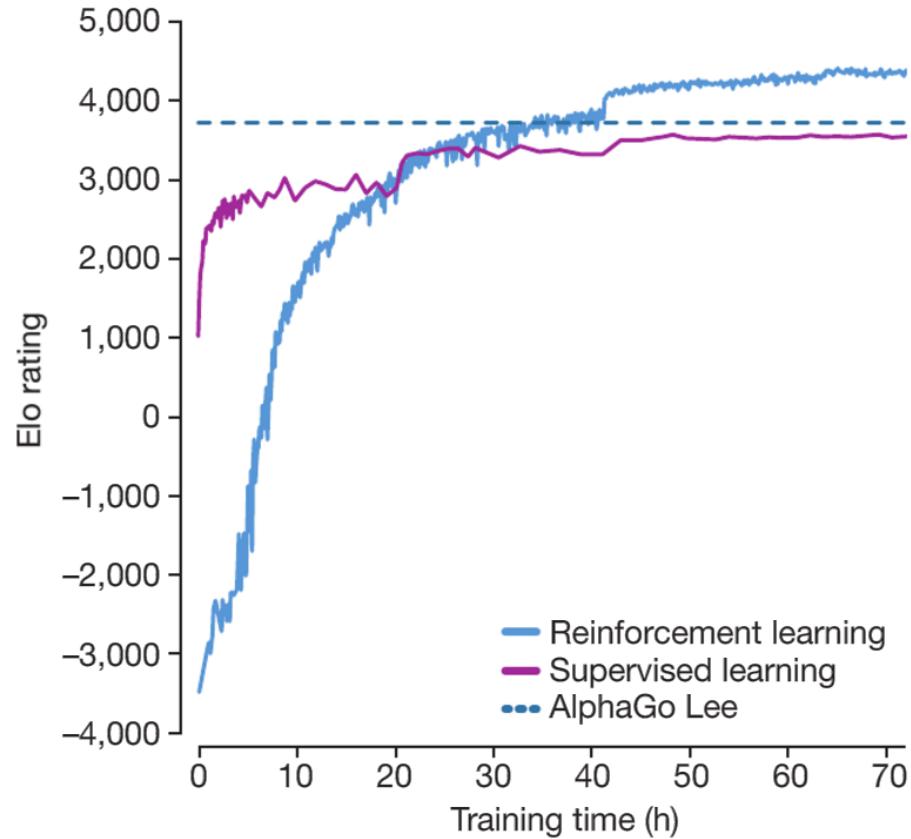
Capturing

Territory

Ko fights

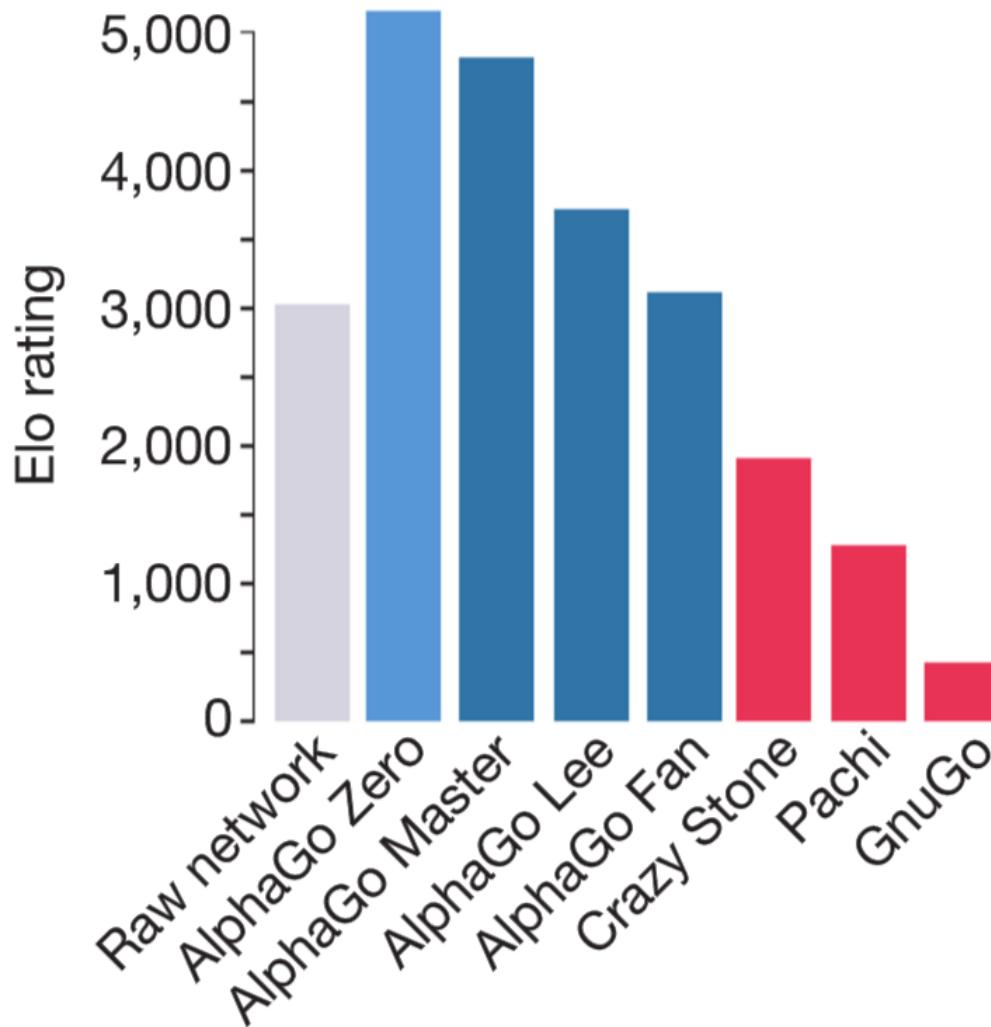
27 at 17 30 at 29 37 at 21 42 at 34 55 at 44 61 at 39
64 at 40 67 at 39 70 at 40 71 at 25 73 at 21 74 at 60
75 at 39 76 at 34

Sup-Human Level Achieved



www.nature.com

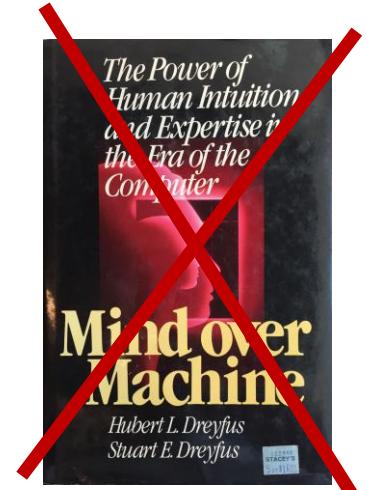
Related Work



www.nature.com

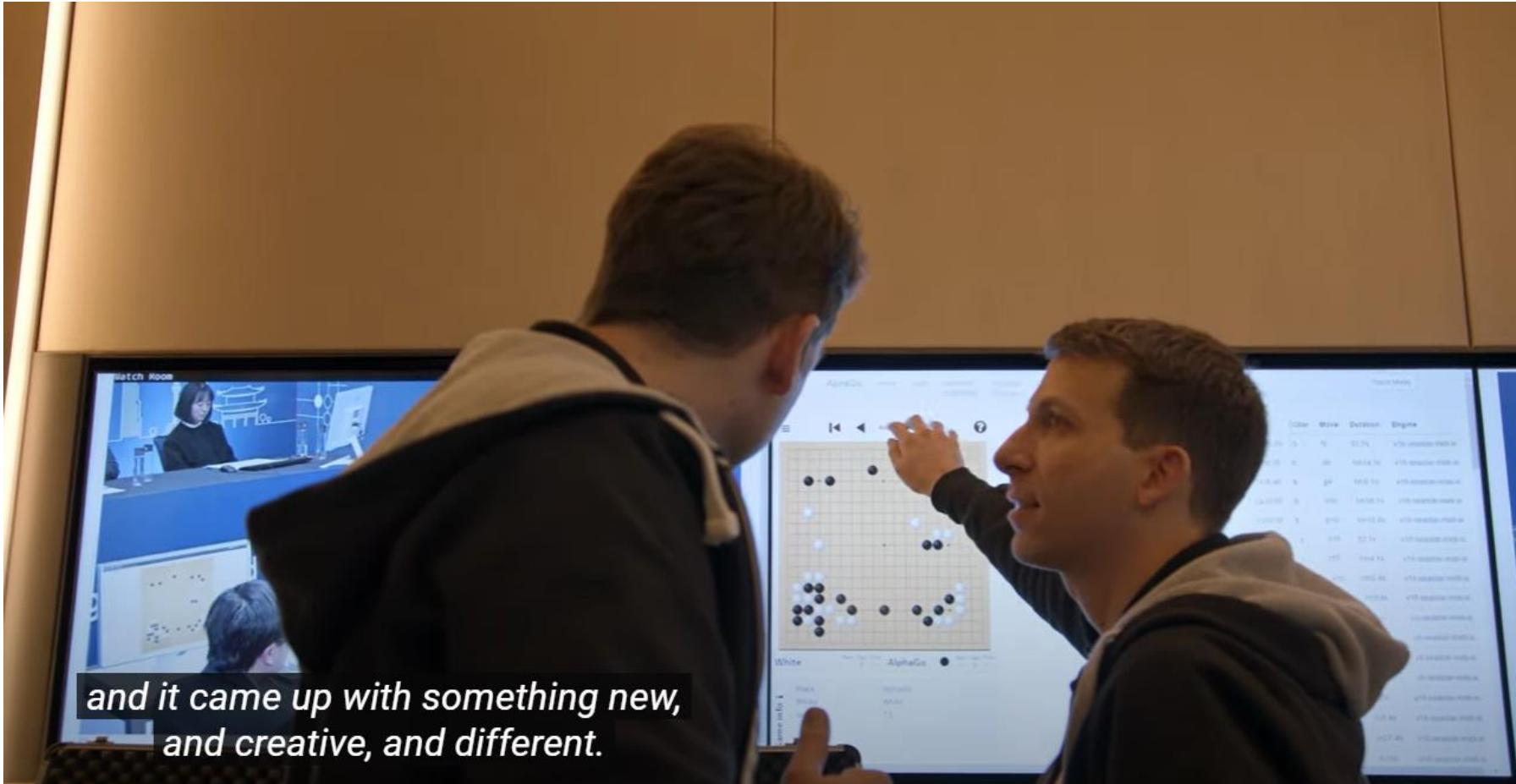
Conclusions

- For zero-sum turn-taking board games:
Alpha Zero > Minimax with Alpha Beta pruning
- Opportunities in
 - Autonomous control
- Limitations
 - ML cannot solve combinatorial problems well
 - “Self-play” not always possible to get training data
 - Inaccuracy can be a problem



AlphaGo Documentary

- https://www.youtube.com/watch?v=WXuK6gekU1Y&ab_channel=DeepMind



Readings and Project

SSS17 (URL link on Learnit)

ARTICLE

Mastering the game of Go without human knowledge

David Silver*, Julian Schrittwieser*, Karen Simonyan*, Ioannis Antonoglou*, Aja Huang*, Arthur Guez*, Thomas Hubert*, Lucas Baker*, Matthew Lai*, Adrian Bolton*, Yutian Chen*, Timothy Lillicrap*, Fan Hui*, Laurent Sifre*, George van den Driessche*, Thore Graepel* & Demis Hassabis*

A long-standing goal of artificial intelligence is an algorithm that learns, *talibus rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The trained by self-play AlphaGo used a neural network trained on millions of games from its own play. These neural networks were trained by self-play learning from human expert moves and by reinforcement learning from Go rules. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *talibus rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts^{1–4}. However, expert data sets are often expensive, trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features.

SHS17 (PDF Link on Learnit)

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,^{1*} Thomas Hubert,^{1*} Julian Schrittwieser,^{1*} Ioannis Antonoglou,¹ Matthew Lai,¹ Arthur Guez,¹ Marc Lanctot,¹ Laurent Sifre,¹ Dharshan Kumaran,¹ Thore Graepel,¹ Timothy Lillicrap,¹ Karen Simonyan,¹ Demis Hassabis¹

¹DeepMind, 6 Pancras Square, London N1C 4AG.

*These authors contributed equally to this work.

Abstract

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been

Othello Proj. (ZIP on Learnit)

ISP Project 1- Othello

In project 3-4 you are going to make your own implementation of a universal search algorithm for the general problem of Othello. The project will be implemented in Java and submitted in ZIP. If you have other language-preferences, you should talk to the TA to have it approved. It is required that you implement the project with alpha-beta pruning and that a user's response time should be less than 10 seconds although this may take up to 30 seconds to be acceptable. Moreover, your implementation should be able to handle a board size of 8x8. You will need to apply several of the topics discussed during lecture for designing the evaluation and cut-off function.

What is provided?

You are provided with five classes and an interface. You do not have to change anything in these classes. In fact, we recommend that you do not change anything in these classes.

- **OthelloGUI**: A GUI class that shows the game board and takes for input from the human player in graphical form.
- **GameState**: A class to represent the state of the game, i.e., the board and whose turn it is. It has methods e.g., for finding the possible legal moves and inserting a token at a specified position.
- **Player**: An abstract class that defines the interface for a player.
- **OthelloAI**: The interface that you need to implement. It only has one method, namely `evaluate`, which takes a `GameState` and returns a float value between -1.0 and 1.0. This value indicates what the player in turn should make, i.e., at which position it should place a token.
- **Othello**: This is the class with the main-method. It reads in the provided parameters, and starts the game loop. Note that the `Player` class does not provide it with the required parameters. These are described a little further below.
- **Double**: A simple implementation of the required interface. It returns the first possible move if it finds.

Read the code and documentation of class `GameState` carefully as you may want to use the provided methods for your implementation of `OthelloAI`. If you implement in Java, you do not need to implement the `Player` class as well. If you implement in C/C++, you should implement the `Player` class and how to do so. Otherwise, i.e. if you implement in CA, you have to implement similar classes and document how to use them.

For convenience, players are represented with integers, specifically:

- + 1 = player 1 (the human/your AI's opponent + black).
 - 1 = player 2 (the computer/your AI's opponent + white).
- Similarly, the game board is represented in `GameState` by a 2-dimensional integer array, where the values range from -1 to 1. The board is indexed like this: `GameState[0][0]` is the top-left square. When starting the game (see instructions further below), the game board should look like the one in Figure 1. The columns on the board are numbered from left to right with the numbers from 0 to 7. On an 8x8 board, the rows are numbered from bottom to top with the numbers from 0 to 7. The top-left square is the top-most white token in Figure 1 on position (0,0).

* Of course, there are various implementations to be found on the internet, but do make your own implementation. If found out, citing is reason for exclusion from the course.

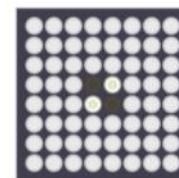


Figure 1 - The initial Othello game board

Running the game

Setup
Download the provided ZIP. Set up a project folder as you would usually do. In other words, let's assume you call "OthelloProject". In order to train the image classifier it is important that "OthelloProject" has a subdirectory called "image" which contains the images (one as it is in the folder you downloaded).

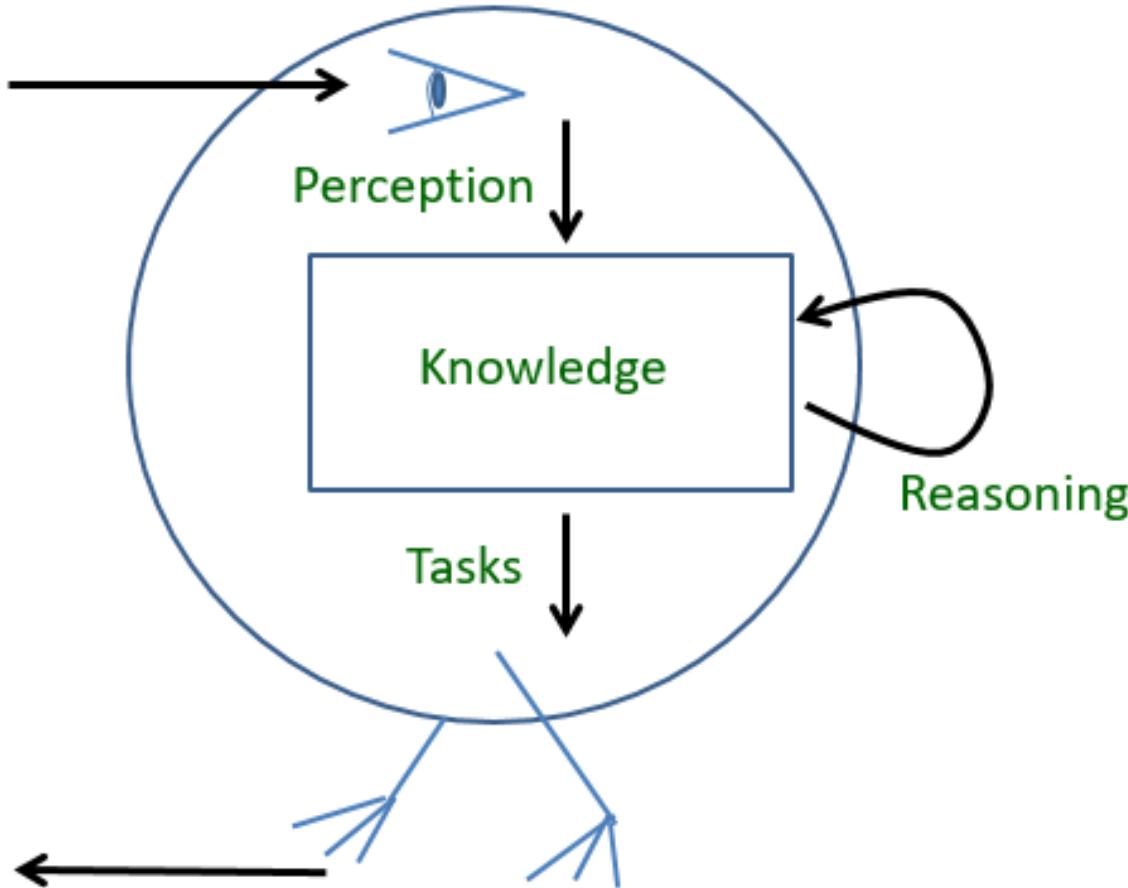


Introduction to Artificial Intelligence

Lecture 5: Propositional Logic



The Knowledge-Based Agent



Reasoning Agents



Today's Program

- [10:00-11:00] **Propositional Logic**
 - Fundamental Concepts in Logic
 - Syntax and Semantics
 - Inference
 - Entailment
 - Logical equivalence
 - Inference rules
 - Formal proofs
- [11:10-12:00] **Efficient Inference Algorithms**
 - Resolution
 - Inference with definite clauses
 - Efficient SAT checking
 - DPLL
 - WalkSat
 - Phase transition
- [12-14] **Exercises**



Fundamental Concepts of Logic



The Purpose of Logics

- **Logics** are formal languages for representing information such that conclusions can be drawn
- Natural language is too complex and **ambiguous**
 - “John saw the diamond through the window and stole it”
 - Reading 1: John stole the diamond
 - Reading 2: John stole the window
- **Sentences** in logics are assertions about a world that are either true or false

Logic: Syntax and Semantics

- Syntax defines the written form of legal sentences in the language
- Semantics define the truth-value of sentences in a world
- World is the setting or environment in which you derive the truth of sentences
- E.g., the language of inequalities
 - $x+2 \geq y$ is a sentence; $x+y \wedge \{\}$ is not a sentence
 - $x+2 \geq y$ is true in a world where $x = 7, y = 1$
 - $x+2 \geq y$ is false in a world where $x = 0, y = 6$



Entailment and Inference

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Definition: $KB \models \alpha$ if and only if α is true in all worlds where KB is true
 - Ex., KB containing *the-apple-is-red* and *the-apple-is-sweet* entails *the-apple-is-sweet*
 - Ex., KB containing “ $y \geq 4$ ”, “ $y \leq 4$ ”, “ $x \neq 21$ ” entails $y = 4$
- Inference is to decide whether $KB \models \alpha$

Models

- A **model** is a formal description of a **possible world** used to decide truth-value of sentences

Example:

Possible world = strength of Corona vaccines

Model = state $\{strong, weak\}$ of vaccine A, B , and C

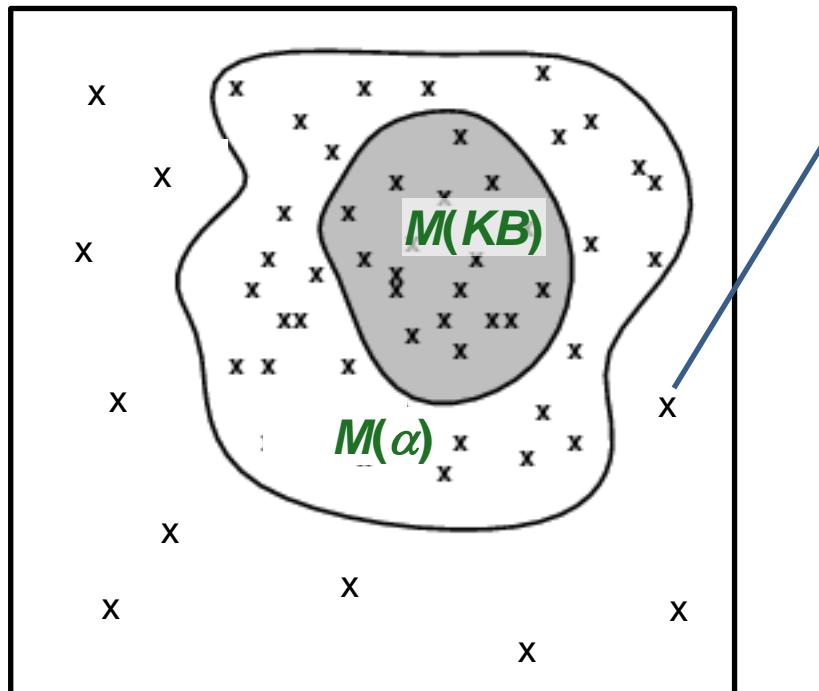
- We say m is a **model** of a sentence α if α is true in m
- $M(\alpha)$ is the set of all models of α



Models

- $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

Model Checking is to explicitly check this relationship



Each "x" is a model:
E.g.: {A=weak,B=strong,
C=weak}

Inference Algorithms

- $KB \vdash_i \alpha$: sentence α can be derived from KB by procedure i
- Soundness: i is sound if $KB \vdash_i \alpha$ implies $KB \models \alpha$
 - Any sentence derived by i from KB is truth preserving
- Completeness: i is complete if $KB \models \alpha$ implies $KB \vdash_i \alpha$
 - All the sentences entailed by KB can be derived by procedure i
 - That is, the procedure will answer any question whose answer follows from what is known by the KB

Propositional Logic



Syntax

- Atomic sentences
 - Proposition symbols P, Q, \dots are sentences
 - The two constants *True* and *False* are sentences
- Complex sentences
 - If S_1 and S_2 are sentences then so are (in order of precedence)
 - $\neg S_1$ *negation* \neg *not* $\neg Q, Q$ *literals*
 - $(S_1 \wedge S_2)$ *conjunction* \wedge *and* S_1, S_2 *conjuncts*
 - $(S_1 \vee S_2)$ *disjunction* \vee *or* S_1, S_2 *disjuncts*
 - $(S_1 \Rightarrow S_2)$ *implication* \Rightarrow *implies* S_1 *premise*
 - $(S_1 \Leftrightarrow S_2)$ *biimplication* \Leftrightarrow *if-and-only-if* S_2 *conclusion*



Semantics (1)

- Each model m assigns **truth value *true* (1) or *false* (0)** to each proposition symbol

E.g.

	P	Q	R
	<i>false</i>	<i>true</i>	<i>false</i>

- The constants *True* and *False* have the expected values

	<i>True</i>	<i>False</i>
	<i>true</i>	<i>false</i>

- This specifies the truth value of atomic sentences



Semantics (2)

- Rules for evaluating truth with respect to a model m :

$\neg S$ is *true* iff S is *false*

$S_1 \wedge S_2$ is *true* iff S_1 is *true* and S_2 is *true*

$S_1 \vee S_2$ is *true* iff S_1 is *true* or S_2 is *true*

$S_1 \Rightarrow S_2$ is *true* iff S_1 is *false* or S_1, S_2 are *true*

$S_1 \Leftrightarrow S_2$ is *true* iff $S_1 \Rightarrow S_2$ is *true* and $S_2 \Rightarrow S_1$ is *true*

- Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P \Rightarrow (Q \wedge R) = \neg \text{false} \Rightarrow (\text{true} \wedge \text{false}) = \text{true} \Rightarrow \text{false} = \text{false}$$

Validity

A sentence is **valid** if it is true in **all** models

e.g.,

True, $A \vee \neg A$, $A \Rightarrow A$, ...

Validity is connected to entailment via the

Deduction Theorem:

$\alpha \models \beta$ if and only if $(\alpha \Rightarrow \beta)$ is valid



Satisfiability

A sentence is **satisfiable** if it is true in **some** model

e.g., $A \vee B$

A sentence is **unsatisfiable** if it is true in **no** models

e.g., $A \wedge \neg A$, *False*

Satisfiability is connected to entailment via the following:

$\alpha \models \beta$ if and only if $(\alpha \wedge \neg \beta)$ is unsatisfiable



Inference by Enumeration (model checking)

- Inference: decide whether $KB \models \alpha$

function TT-ENTAILS?(KB, α) **returns** true or false

inputs: KB , the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

symbols \leftarrow a list of the proposition symbols in KB and α

return TT-CHECK-ALL($KB, \alpha, symbols, \{ \}$)

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** true or false

if EMPTY?(*symbols*) **then**

if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)

else return true // when KB is false, always return true

else do

$P \leftarrow$ FIRST(*symbols*)

rest \leftarrow REST(*symbols*)

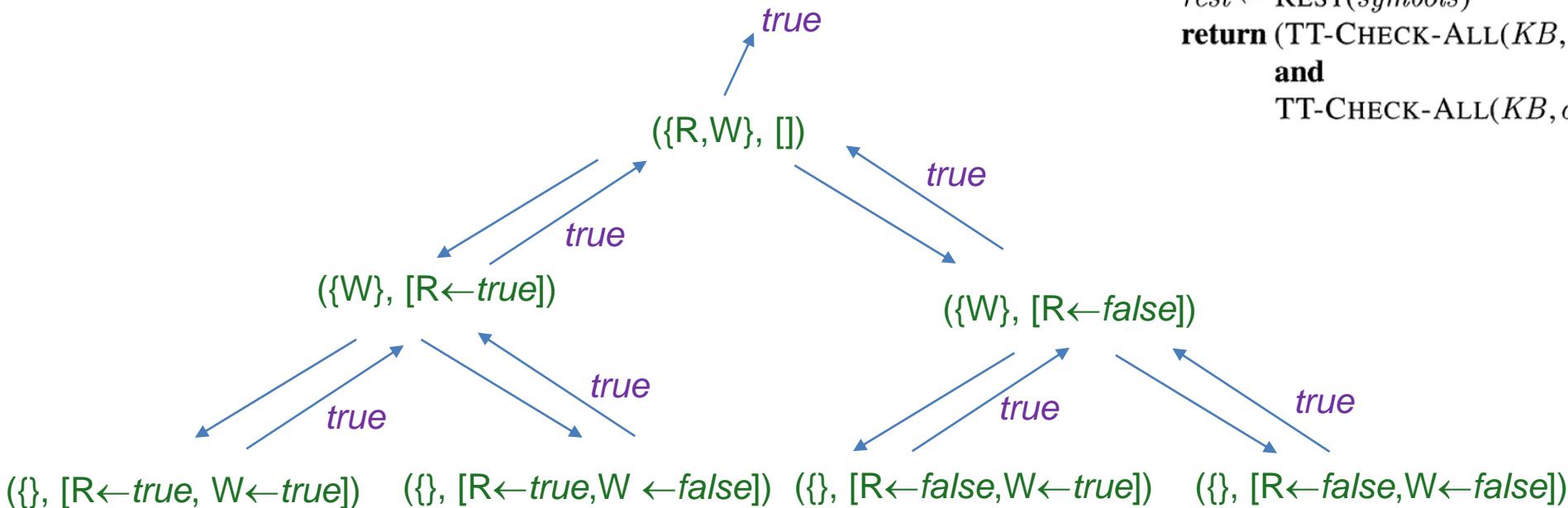
return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)

and

 TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

TT-Entails Example

$$\{(R \Rightarrow W), R\} \models W$$



```
function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false
  if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)
    else return true // when KB is false, always return true
  else do
     $P \leftarrow \text{FIRST}(symbols)$ 
    rest  $\leftarrow \text{REST}(symbols)$ 
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup \{P = \text{true}\}$ )
           and
           TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup \{P = \text{false}\}$ ))
```

Properties of TT-ENTAILS

- DFS enumeration and check of all models
- Sound? yes checks if α is true when KB is true
- Complete? yes checks all models
- For n symbols
 - time complexity is $O(2^n)$
 - space complexity is $O(n)$



Logical Equivalence

- Two sentences are **logically equivalent** iff they are true in the same models:

$$\alpha \equiv \beta \text{ iff } M(\alpha) = M(\beta) \text{ iff } \alpha \models \beta \text{ and } \beta \models \alpha$$

- Simple equivalences:

- $\alpha \wedge \neg\alpha \equiv \text{False}$
- $\alpha \vee \neg\alpha \equiv \text{True}$
- $\alpha \wedge \text{True} \equiv \alpha$
- $\alpha \vee \text{False} \equiv \alpha$
- $\alpha \wedge \text{False} \equiv \text{False}$
- $\alpha \vee \text{True} \equiv \text{True}$
- $\alpha \wedge \alpha \equiv \alpha$
- $\alpha \vee \alpha \equiv \alpha$



Standard Equivalences

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \text{ contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ de Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ de Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge$$

Formal Proof

- Determine logical equivalence between sentences using standard rules

- Ex.

$$(a \vee (b \Rightarrow a))$$

$$\equiv (a \vee (\neg b \vee a))$$

impl. elim.

$$\equiv (a \vee (a \vee \neg b))$$

commutativity of \vee

$$\equiv ((a \vee a) \vee \neg b)$$

associativity of \vee

$$\equiv a \vee \neg b$$

$a \vee a \equiv a$

$$\equiv \neg b \vee a$$

commutativity of \vee

$$\equiv b \Rightarrow a$$

impl. elim.

Check

a	b	$b \Rightarrow a$	$a \vee (b \Rightarrow a)$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	1	1

Inference Rules

numerator (premises) \models denominator (conclusion)

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta} \quad \text{Modus Ponens}$$

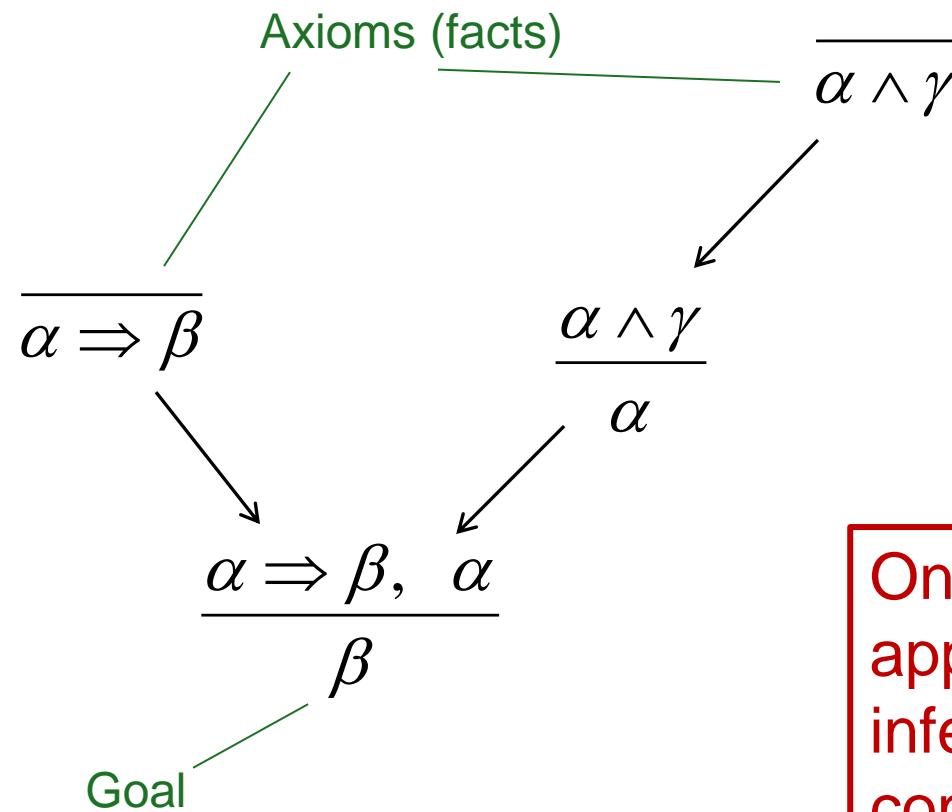
$$\frac{\alpha \wedge \beta}{\alpha} \quad \text{And-Elimination}$$

$$\frac{\alpha \Rightarrow \beta}{\neg \alpha \vee \beta} \quad \frac{\neg \alpha \vee \beta}{\alpha \Rightarrow \beta} \quad \text{All Equivalence rules}$$



Inference Proof

- Search for inference rules to chain “goal” with axioms



Only complete
approach if set of
inference rules is
complete!

Break



Efficient Inference Algorithms



Conjunctive Normal Form (CNF)

Definition

- A **literal** is a symbol or a negated symbol
- A **clause** is a disjunction of literals
- **CNF** is a conjunction of clauses

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Translate Arbitrary Sentence to CNF

1. Eliminate \Leftrightarrow , replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Eliminate \Rightarrow , replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$
3. Move \neg inwards using de Morgan's rules and double-negation
4. Apply distribution law (\wedge over \vee) and flatten
5. Eliminate symbol duplicates in clauses (**factoring**)



Translate to CNF Example

$$P \Leftrightarrow (Q \vee R)$$

$$\equiv P \Rightarrow (Q \vee R) \quad \wedge \quad (Q \vee R) \Rightarrow P$$

$$\equiv (\neg P \vee Q \vee R) \quad \wedge \quad \neg(Q \vee R) \vee P$$

$$\equiv (\neg P \vee Q \vee R) \quad \wedge \quad (\neg Q \wedge \neg R) \vee P$$

$$\equiv (\neg P \vee Q \vee R) \quad \wedge \quad (\neg Q \vee P) \quad \wedge \quad (\neg R \vee P)$$

Translate Arbitrary Sentence to CNF

1. Eliminate \Leftrightarrow , replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Eliminate \Rightarrow , replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
3. Move \neg inwards using de Morgan's rules and double-negation
4. Apply distribution law (\wedge over \vee) and flatten
5. Eliminate symbol duplicates in clauses (factoring)



Resolution: A Complete Inference Rule

$$\frac{P \vee Q, \quad \neg Q \vee R}{P \vee R}$$



Resolution: A Complete Inference Rule

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where ℓ_i and m_j are complementary literals ($\ell_i \equiv \neg m_j$).

Resolution Algorithm

- Remember: $KB \models \alpha$ iff $KB \wedge \neg \alpha$ is unsatisfiable
- To check $KB \models \alpha$
- Keep doing resolution on clauses of $KB \wedge \neg \alpha$ until:
 - () derived
return *true* *Why?*
 - fixpoint reached with no empty clause ()
return *false* (**ground resolution theorem**)

$$\frac{(\neg A), \quad (A)}{()}$$

Resolution Algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*

inputs: KB , the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

$new \leftarrow \{ \}$

loop do

for each pair of clauses C_i, C_j **in** $clauses$ **do**

$resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)

if $resolvents$ contains the empty clause **then return** *true*

$new \leftarrow new \cup resolvents$

if $new \subseteq clauses$ **then return** *false*

$clauses \leftarrow clauses \cup new$



Example: PL-Resolution($\{P \Rightarrow R \wedge Q, \neg Q\}$, $\neg P$)

$$P \Rightarrow R \wedge Q \equiv \neg P \vee (R \wedge Q) \equiv (\neg P \vee R) \wedge (\neg P \vee Q)$$

$$\text{Clauses}_0 = \{(\neg P \vee R), (\neg P \vee Q), (\neg Q), (P)\}$$

$$\text{New}_0 = \{\}$$

$$\text{New}_1 = \{(R), (\neg P), (Q)\}$$

$$\text{Clauses}_1 = \{(\neg P \vee R), (\neg P \vee Q), (\neg Q), \\ (P), (R), (\neg P), (Q)\}$$

$$\text{New}_2 = \{\dots, (), \dots\}$$

Returns true i.e. $\{P \Rightarrow R \wedge Q, \neg Q\} \models \neg P$

```
clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
new ← {}
loop do
    for each pair of clauses  $C_i, C_j$  in clauses do
        resolvents ← PL-RESOLVE( $C_i, C_j$ )
        if resolvents contains the empty clause then return true
        new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Properties of the Resolution Algorithm

- **Sound**, yes
- **Complexity**
 - Time, Space: $O(2^n)$, due to blow-up of number of clauses
- **Complete**, yes
Ground resolution theorem
if a CNF with set of clauses S is unsatisfiable, then the **resolution closure** of those clauses, $RC(S)$, contains the empty clause $()$.

Proof of ground resolution theorem

We must show: if S is unsatisfiable then $() \in RC(S)$

Prove log. equiv. sentence: if $() \notin RC(S)$ then S satisfiable

Satisfying assignment to propositions P_1 to P_k in $RC(S)$

For $i = 1$ to k

- $P_i = False$ if there is a clause $(False \vee \dots \vee False \vee \neg P_i)$
- $P_i = True$ otherwise

Why is this assignment satisfying?

Assume conflict in iteration i

$i > 0$ since $() \notin RC(S)$

In iteration $i > 0$, we must have two clauses
 $(False \vee \dots \vee False \vee P_i)$ and $(False \vee \dots \vee False \vee \neg P_i)$

But due to resolution clause $(False \vee \dots \vee False)$ also exists.
This is impossible since then conflict happens in iteration $i-1$.



Definit Clauses

- Definit clause: *exactly* one positive literal

$$\begin{array}{c} P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow C \\ \Rightarrow F \end{array}$$

head
body
fact

- Modus Ponens (for Definit clause)

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Inference possible in linear time!

Forward chaining inference

KB:

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

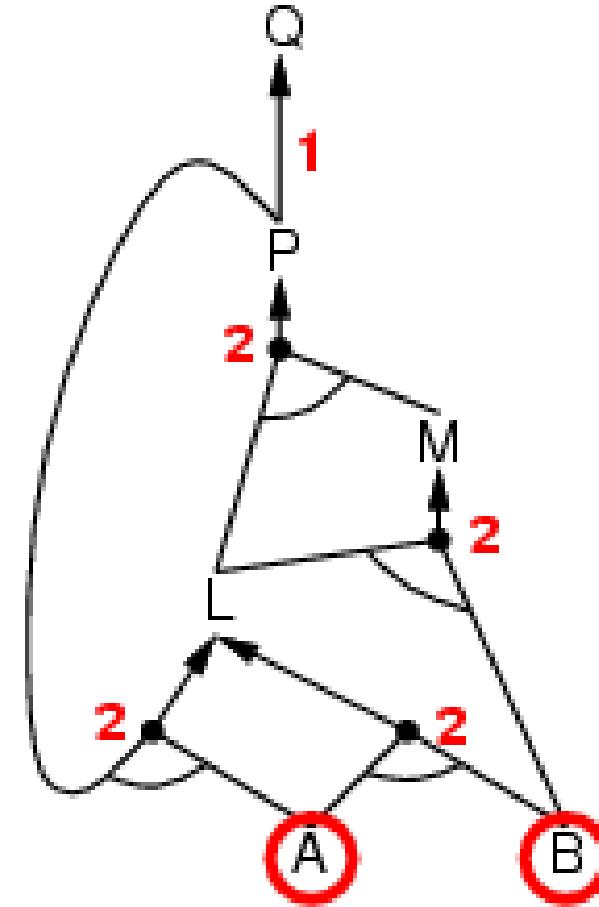
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

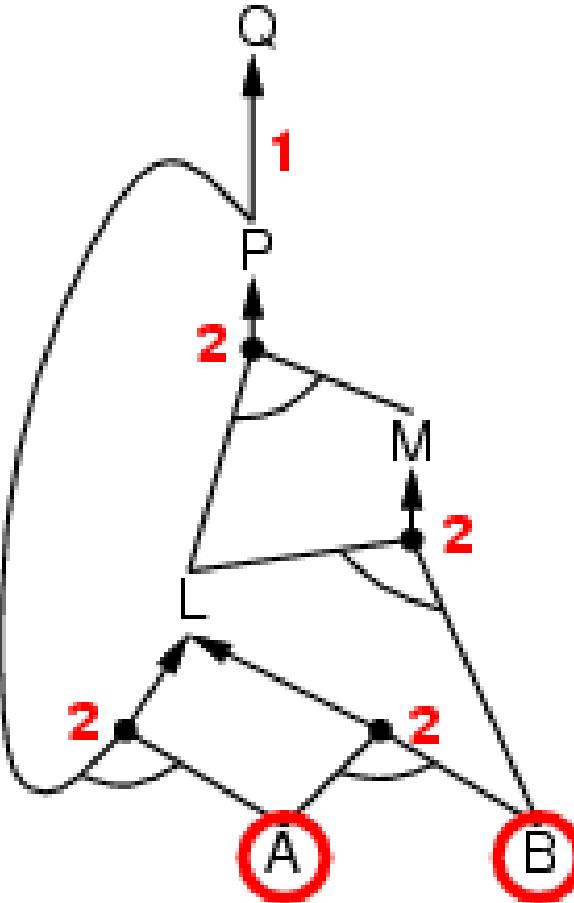
$$A$$

$$B$$

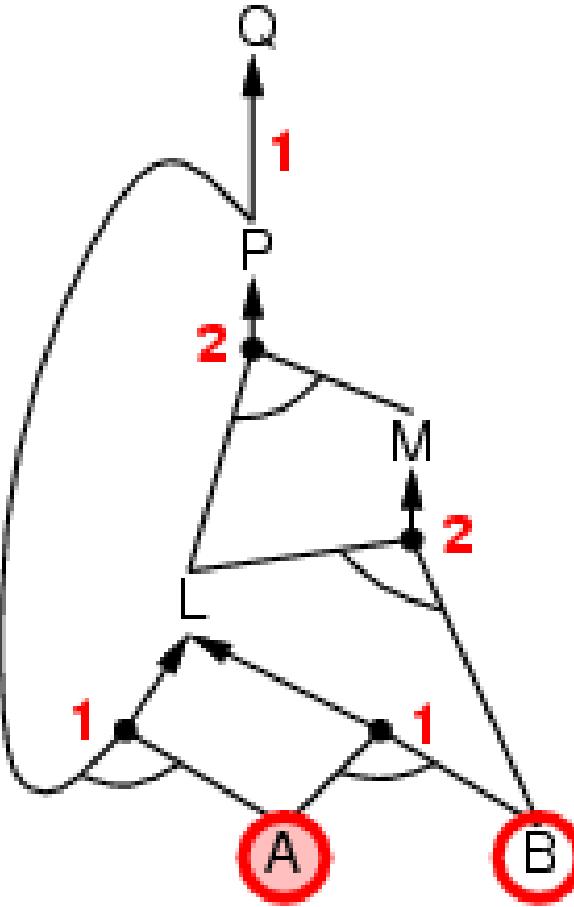
$KB \vDash Q ?$



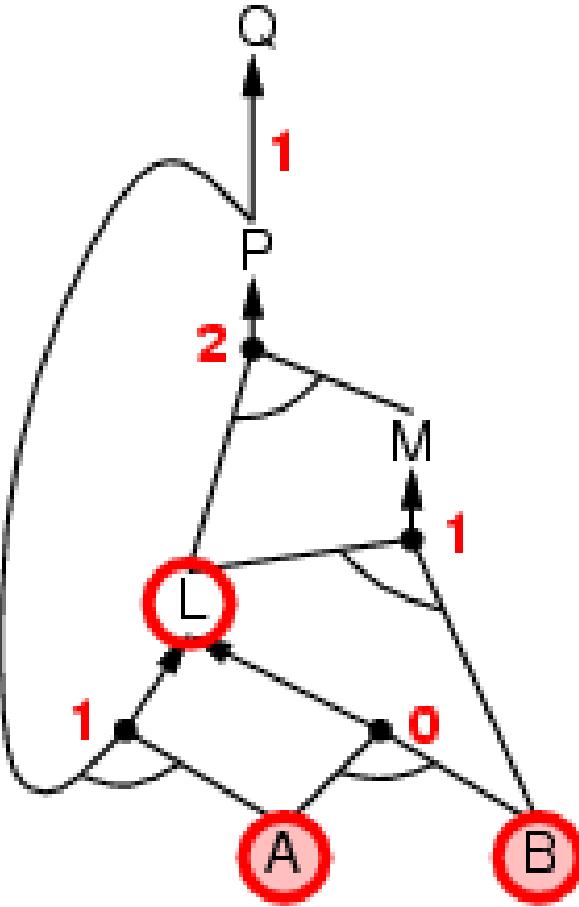
Forward chaining example



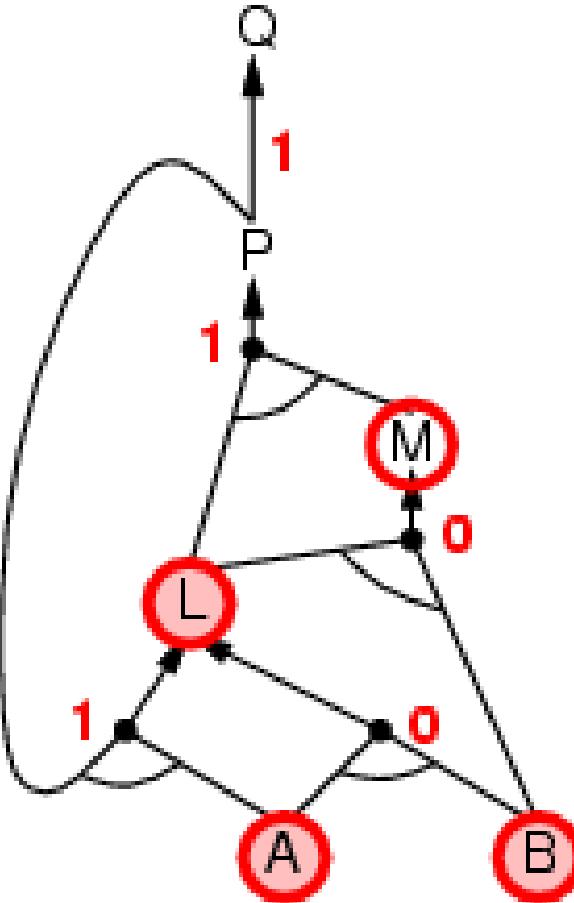
Forward chaining example



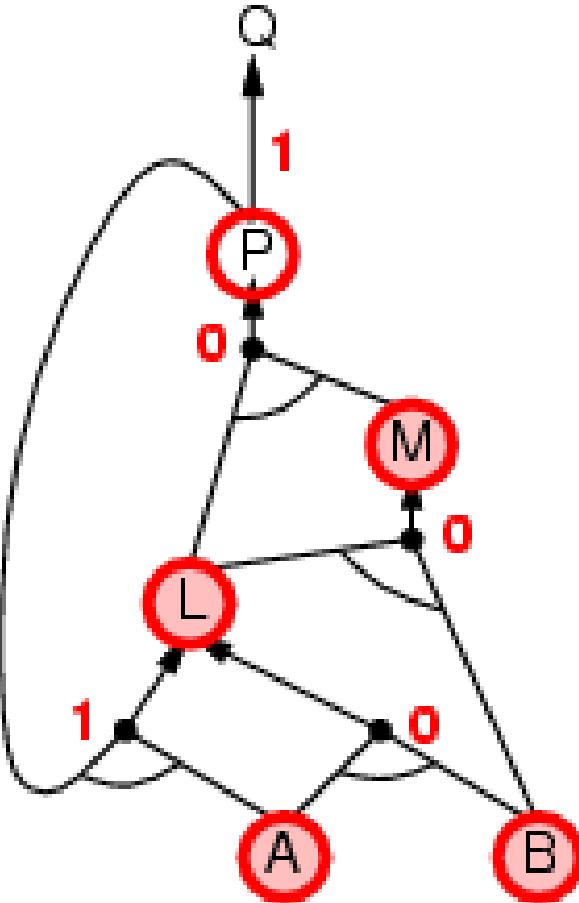
Forward chaining example



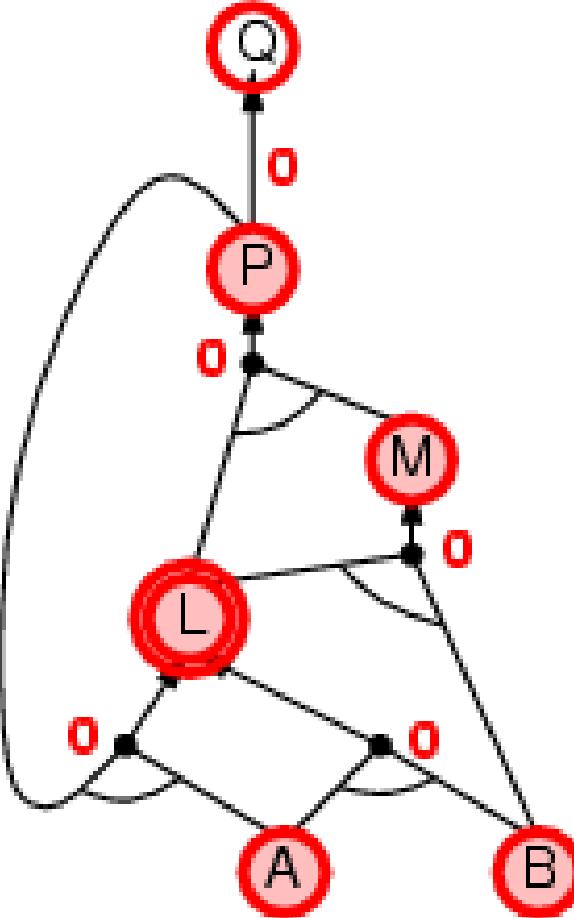
Forward chaining example



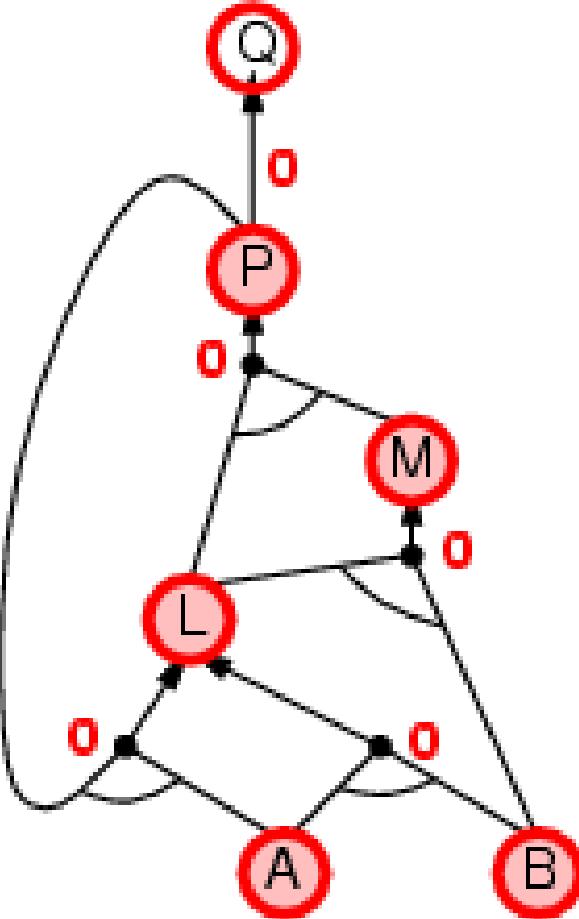
Forward chaining example



Forward chaining example

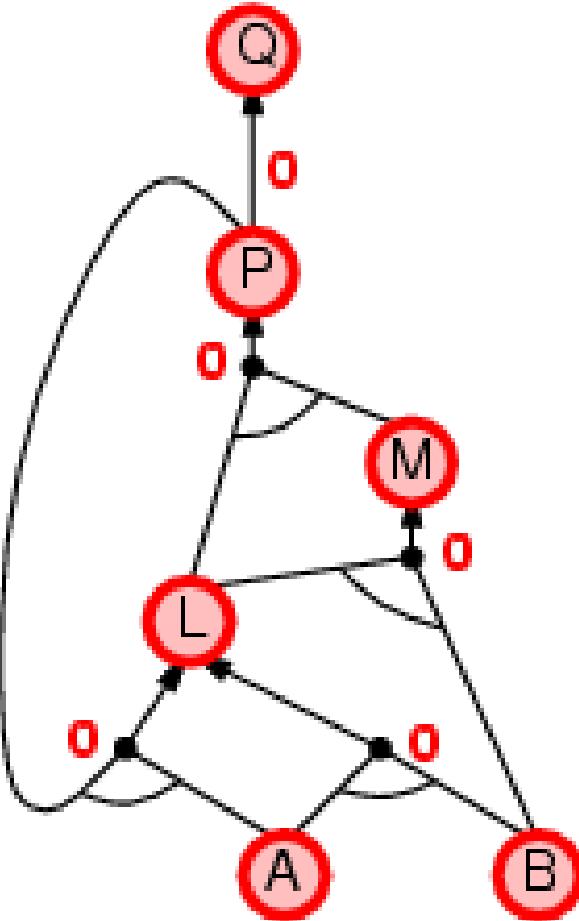


Forward chaining example



Forward chaining example

$KB \models Q$
Yes!



Properties of Forward Chaining

- **Sound**, yes since Modus Ponens is sound
- **Complete**, yes
- **Space and time**: $O(n)$, where n is the total number of literals in the clauses



Efficient SAT-Checking

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)

Incomplete local search algorithms

- WalkSAT algorithm



The DPLL Algorithm

Determines whether a CNF sentence is satisfiable

Improvements over truth table enumeration:

1. Early termination

A clause is true if any literal is true

A sentence is false if any clause is false

2. Pure symbol heuristic

Pure symbol: always appears with the same "sign" in all clauses

e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure

Make a pure symbol literal true

3. Unit propagation

Unit clause: only one literal in the clause

The only literal in a unit clause must be true

e.g., $(\text{False} \vee \neg B)$: B must be false



The DPLL Algorithm

function DPLL-SATISFIABLE?(s) **returns** true or false

inputs: s , a sentence in propositional logic

$\text{clauses} \leftarrow$ the set of clauses in the CNF representation of s

$\text{symbols} \leftarrow$ a list of the proposition symbols in s

return DPLL($\text{clauses}, \text{symbols}, \{\}$)

function DPLL($\text{clauses}, \text{symbols}, \text{model}$) **returns** true or false

if every clause in clauses is true in model **then return** true

if some clause in clauses is false in model **then return** false

$P, \text{value} \leftarrow \text{FIND-PURE-SYMBOL}(\text{symbols}, \text{clauses}, \text{model})$

if P is non-null **then return** DPLL($\text{clauses}, \text{symbols} - P, \text{model} \cup \{P=\text{value}\}$)

$P, \text{value} \leftarrow \text{FIND-UNIT-CLAUSE}(\text{clauses}, \text{model})$

if P is non-null **then return** DPLL($\text{clauses}, \text{symbols} - P, \text{model} \cup \{P=\text{value}\}$)

$P \leftarrow \text{FIRST}(\text{symbols}); \text{rest} \leftarrow \text{REST}(\text{symbols})$

return DPLL($\text{clauses}, \text{rest}, \text{model} \cup \{P=\text{true}\}$) **or**

DPLL($\text{clauses}, \text{rest}, \text{model} \cup \{P=\text{false}\}$))



DPLL Example

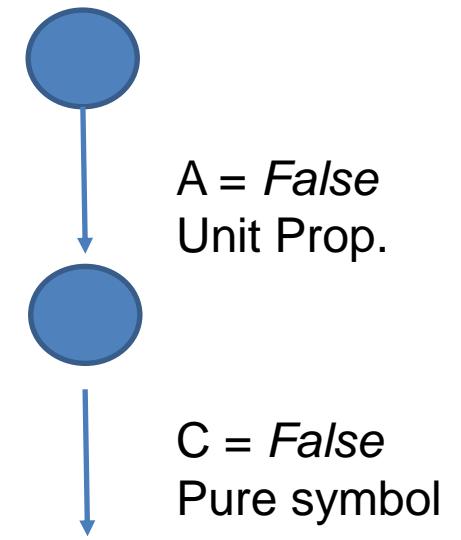
$$(\neg A) \wedge (A \vee \neg C) \wedge (\neg A \vee B) \wedge (C \vee \neg B \vee \neg A)$$

$$\cancel{(\neg A)} \wedge (\text{False} \vee \neg C) \wedge \cancel{(\neg A \vee B)} \wedge \cancel{(C \vee \neg B \vee \neg A)}$$

$$\cancel{(\neg A)} \wedge \cancel{(\text{False} \vee \neg C)} \wedge \cancel{(\neg A \vee B)} \wedge \cancel{(C \vee \neg B \vee \neg A)}$$

function DPLL(*clauses*, *symbols*, *model*) returns true or false

if every clause in *clauses* is true in *model* **then return** true
if some clause in *clauses* is false in *model* **then return** false
P, value \leftarrow FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, *model* \cup {*P*=*value*})
P, value \leftarrow FIND-UNIT-CLAUSE(*clauses*, *model*)
if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, *model* \cup {*P*=*value*})
P \leftarrow FIRST(*symbols*); *rest* \leftarrow REST(*symbols*)
return DPLL(*clauses*, *rest*, *model* \cup {*P*=true}) **or**
DPLL(*clauses*, *rest*, *model* \cup {*P*=false}))



The WalkSAT algorithm

Determine if a CNF sentence is satisfiable

Algorithm

- Start with a random complete assignment
- In each iteration:
 - Pick random false clause
 - With probability p
 - flip random literal in clause
 - Else
 - flip literal in clause that makes most clauses true
- Stop when reaching given max number of iterations



Phase Transition

- Consider random 3-CNF sentences. e.g.,

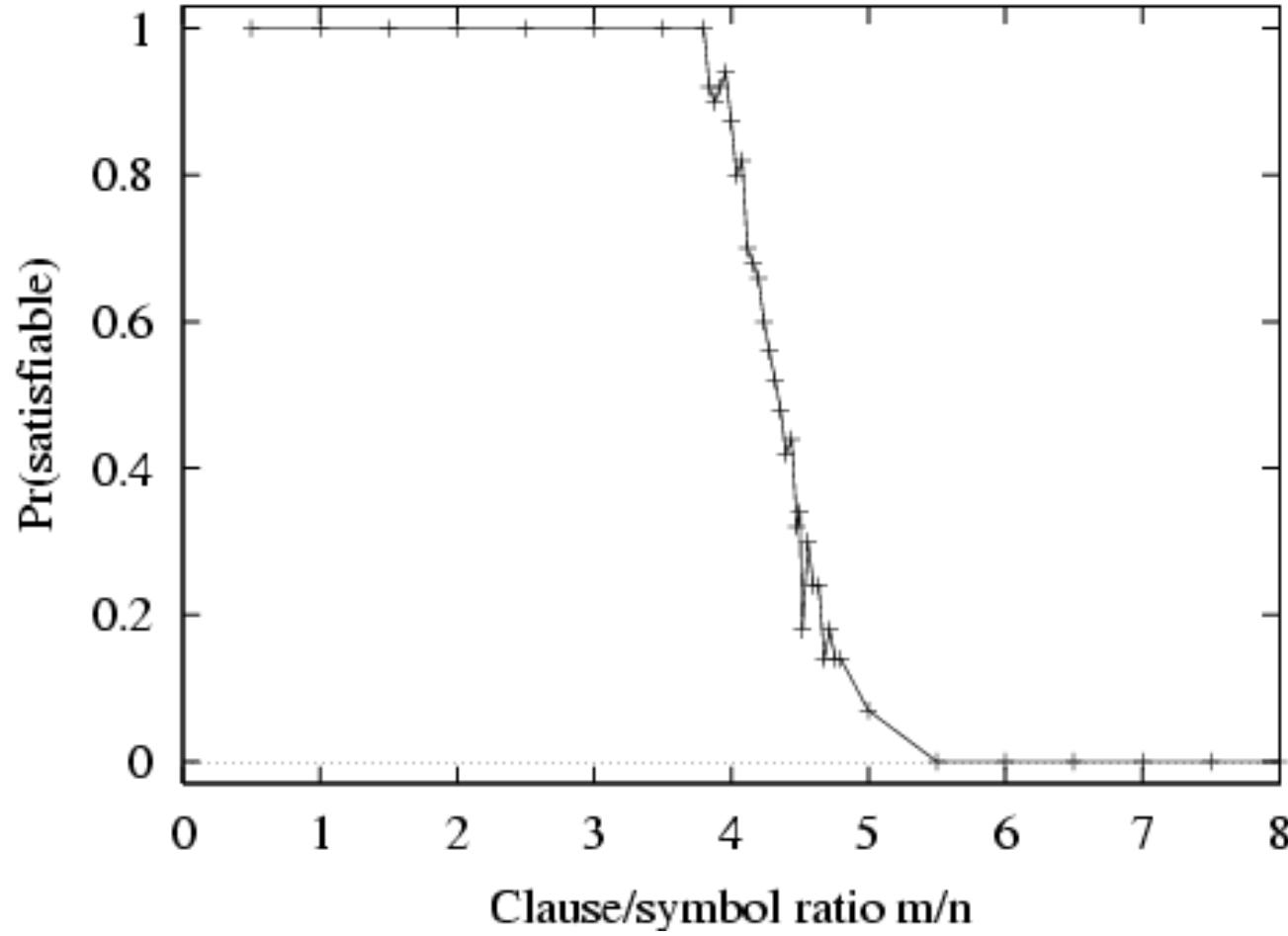
$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

m = number of clauses

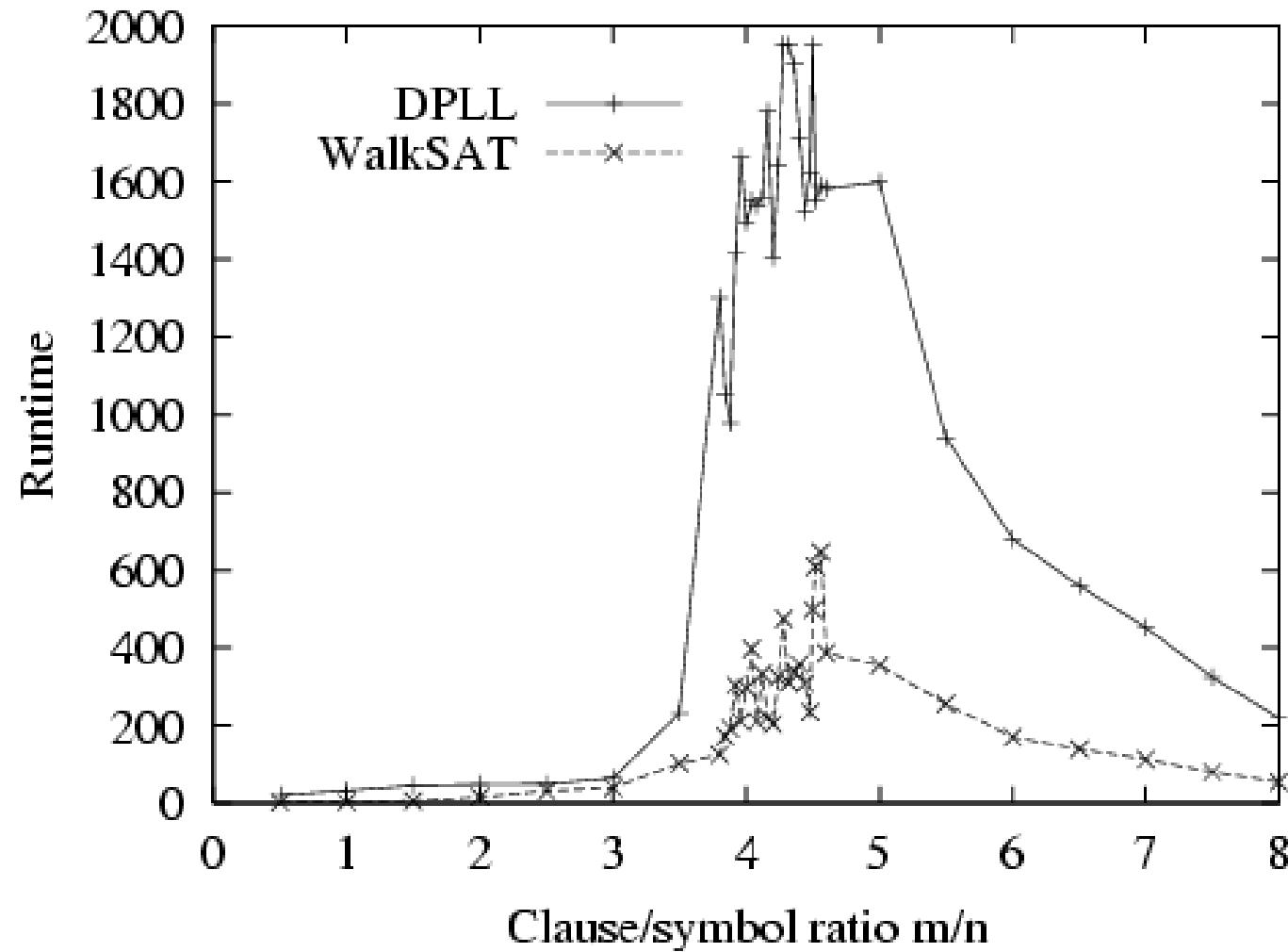
n = number of symbols

- Hard problems seem to cluster near $m/n = 4.3$
(phase transition)

Phase Transition



Phase Transition



Introduction to Artificial Intelligence

Lecture 6: Representations of Boolean Expressions & Binary Decision Diagrams (BDDs)



HOW TO
AVOID A
CLIMATE
DISASTER
THE SOLUTIONS WE HAVE AND THE
BREAKTHROUGHS WE NEED
BILL GATES

Today's Program

- **Representations**
 - Boolean expressions and Boolean functions
 - Desirable properties of representations of Boolean functions
 - Examples: Truth tables, cCNF.
- **Binary Decision Diagrams**
 - If-then-else normal form (INF)
 - Decision trees
 - Ordered Binary Decision Diagrams (OBDDs)
 - Reduced Ordered Binary Decision Diagrams (ROBDDs / BDDs)
- **Exercises**



Boolean Expressions

- Boolean expressions

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \Rightarrow t \mid t \Leftrightarrow t$$

- Precedence

$$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$

- Equivalence: \equiv

- Set of truth values: $B = \{0, 1\}$

- Substitution, e.g. $x \Rightarrow y [0/x, 1/y] \equiv 0 \Rightarrow 1$

Boolean Functions

- A Boolean (n-ary) function

$$f : \underbrace{B \times B \times \dots \times B}_{n} \rightarrow B$$

- Examples

Order of variables matter!

$$f(x_1, x_2) = x_1 \Rightarrow x_2$$

$$f(x_1, x_2, x_3) = x_1 \vee \neg x_3$$

Non-essential variable



Properties of Boolean Functions

- Equality

$$f \equiv g \text{ iff } \forall x . f(x) = g(x)$$

- Several expressions may define same function

$$f(x,y) = x \Rightarrow y$$

$$= \neg x \vee y$$

$$= (\neg x \vee y) \wedge (\neg x \vee x)$$

$$= \dots$$

- Thus, we want to represent Boolean functions rather than Boolean expressions in the machine

Number of Boolean Functions

Number of Boolean functions $f : \mathcal{B}^n \rightarrow \mathcal{B}$

x_1	...	x_n	f
0	...	0	$f(0, \dots, 0)$
0	...	1	$f(0, \dots, 1)$
0	...	0	$f(0, \dots, 0)$
0	...	1	$f(0, \dots, 1)$
...
1	...	0	$f(1, \dots, 0)$
1	...	1	$f(1, \dots, 1)$
1	...	0	$f(1, \dots, 0)$
1	...	1	$f(1, \dots, 1)$
...

$$2^{(2^n)}$$

$$n = 0$$

2 : False , True

$$n = 1$$

$$4: \quad f(x)$$

x	False	x	$\neg x$	True
0	0	0	1	1
1	0	1	0	1

$$n = 8$$

$$10^{80} !!$$



Representation of Boolean functions

Desirable properties:

1. Compact
2. Equality check easy
3. Evaluating truth-value of an assignment easy
4. Boolean operations efficient
5. SAT check efficient
6. Tautology check efficient
7. **Canonicity**: exactly one representation of each Boolean function. - Solves 2, 5, and 6, why?



Compact representations are rare

- $2^{\binom{2^n}{2}}$ Boolean functions in n variables...
 - How do we find a single compact representation for them all?
- The fraction of Boolean functions of n variables with a polynomial size in $n \rightarrow 0$ for $n \rightarrow \infty$



Curse of Boolean function representations:

This problem exists for all representations we know!

Truth tables

- Compact  table size $O(2^n)$
- Equality check easy  canonical
- Easy to evaluate the truth-value of an assignment
 log m or constant
- Boolean operations efficient  linear
- SAT check efficient  canonical
- Tautology check efficient  canonical

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

CNF-representation

- There exists a CNF of every expression
- Given a truth table representation of a Boolean function, can we easily define a CNF of the function?

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



CNF from off-tuples

- Example CNF of f - use *off-tuples*

x	y	z	f	$f(x,y,z) =$
0	0	0	0	$\neg(\neg x \wedge \neg y \wedge \neg z) \wedge$
0	0	1	1	
0	1	0	0	$\neg(\neg x \wedge y \wedge \neg z) \wedge$
0	1	1	1	
1	0	0	0	$\neg(x \wedge \neg y \wedge \neg z)$
1	0	1	1	
1	1	0	1	
1	1	1	1	

CNF from off-tuples

- Example CNF of f - use *off-tuples*

x	y	z	f	$f(x,y,z) =$
0	0	0	0	$(\neg \neg x \vee \neg \neg y \vee \neg \neg z) \wedge$
0	0	1	1	
0	1	0	0	$(\neg \neg x \vee \neg y \vee \neg \neg z) \wedge$
0	1	1	1	
1	0	0	0	$(\neg x \vee \neg \neg y \vee \neg \neg z)$
1	0	1	1	
1	1	0	1	
1	1	1	1	

CNF from off-tuples

- Example CNF of f - use *off-tuples*

x	y	z	f	$f(x,y,z) =$
0	0	0	0	$(x \vee y \vee z) \wedge$
0	0	1	1	
0	1	0	0	$(x \vee \neg y \vee z) \wedge$
0	1	1	1	
1	0	0	0	$(\neg x \vee y \vee z)$
1	0	1	1	
1	1	0	1	
1	1	1	1	

cCNF

- The special CNF-representations produced from *off*-tuples are **canonical** and called **cCNF**
- Are cCNF minimum **size** CNF representations?
 - Hint: look at representation of False
- Easy accessibility?



BREAK



Binary Decision Diagrams



If-then-else operator

- The *if-then-else* Boolean operator is defined by

$$x \rightarrow y_1, y_0 \stackrel{\text{def.}}{\equiv} (x \wedge y_1) \vee (\neg x \wedge y_0)$$

- We have

$$(x \rightarrow y_1, y_0) [1/x] \equiv (1 \wedge y_1) \vee (0 \wedge y_0) \equiv y_1$$

$$(x \rightarrow y_1, y_0) [0/x] \equiv (0 \wedge y_1) \vee (1 \wedge y_0) \equiv y_0$$

- What is $x \rightarrow 1, 0$ equivalent to?



If-then-else operator

- All Boolean operators can be expressed using **only** $x \rightarrow y_1, y_0$ operators with:
 - *if-then-else* expression, 0, or 1 for y_1 and y_0
 - tests on un-negated variables
- What are these *if-then-else* expressions for
 - $\neg x$ $x \rightarrow 0, 1$
 - $x \wedge y$ $x \rightarrow (y \rightarrow 1, 0), 0$
 - $x \vee y$ $x \rightarrow 1, (y \rightarrow 1, 0)$
 - $x \Rightarrow y$ $x \rightarrow (y \rightarrow 1, 0), 1$

If-then-else Normal Form (INF)

An *if-then-else* Normal Form (INF) is a Boolean expression build entirely from the if-then-else operator and the constants 0 and 1 such that all test are performed only on un-negated variables

- **Proposition:** any Boolean expression t is equivalent to an expression in INF



If-then-else Normal Form (INF)

- **Proposition:** any Boolean expression t is equivalent to an expression in INF

Proof: Induction on number of variables in t .

Induction start: 0 variables, t is either equivalent to 1 or 0.

Induction step: n variables. IH: Prop. holds for $n - 1$ variables

Let x be variable in t .

$t \equiv x \rightarrow t[1/x], t[0/x]$ (Shannon expansion of t)

$t[1/x]$ and $t[0/x]$ has one less variable than t , so is equivalent to an expression in INF according to induction hypothesis (IH)

Construction of INF

Expression t with 3 variables:

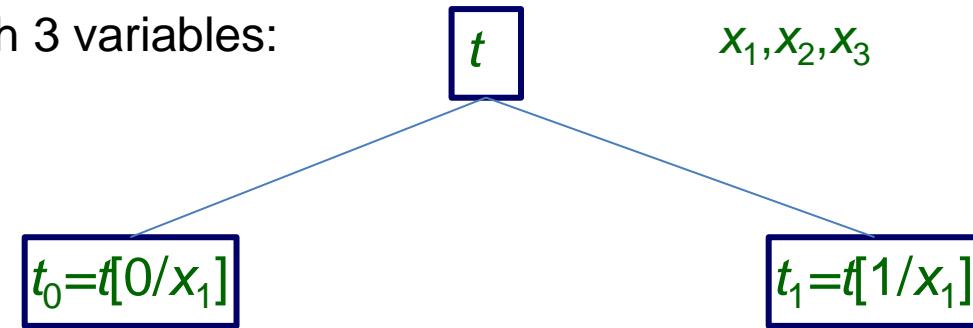
t

x_1, x_2, x_3



Construction of INF

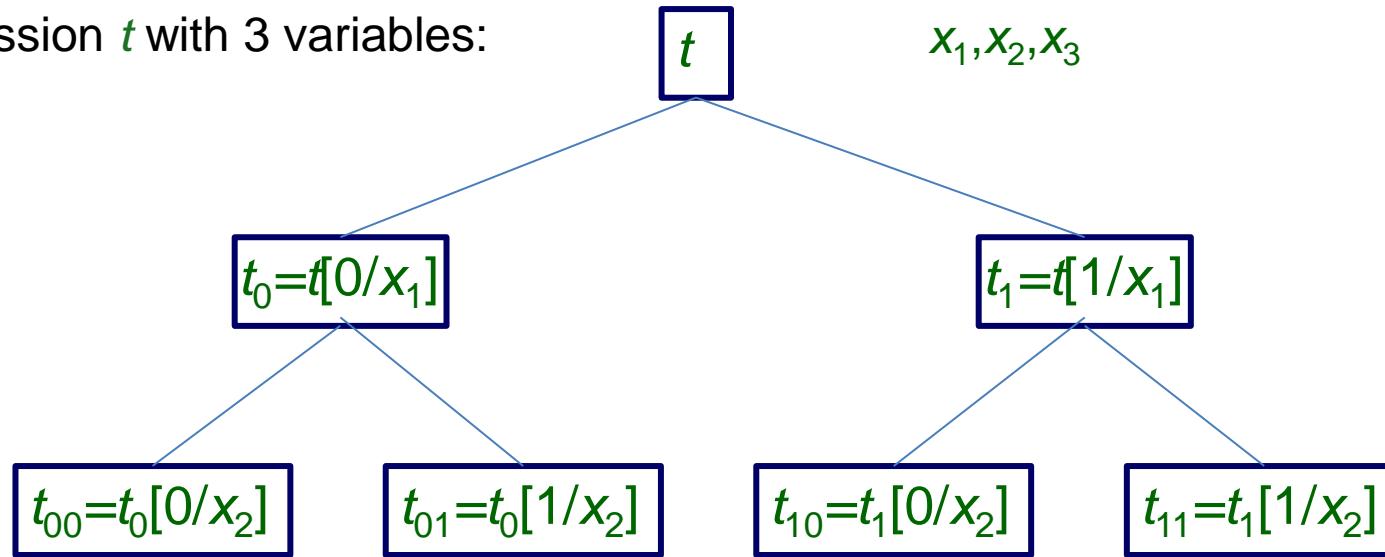
Expression t with 3 variables:



$$t \equiv x_1 \rightarrow t_1, t_0$$

Construction of INF

Expression t with 3 variables:



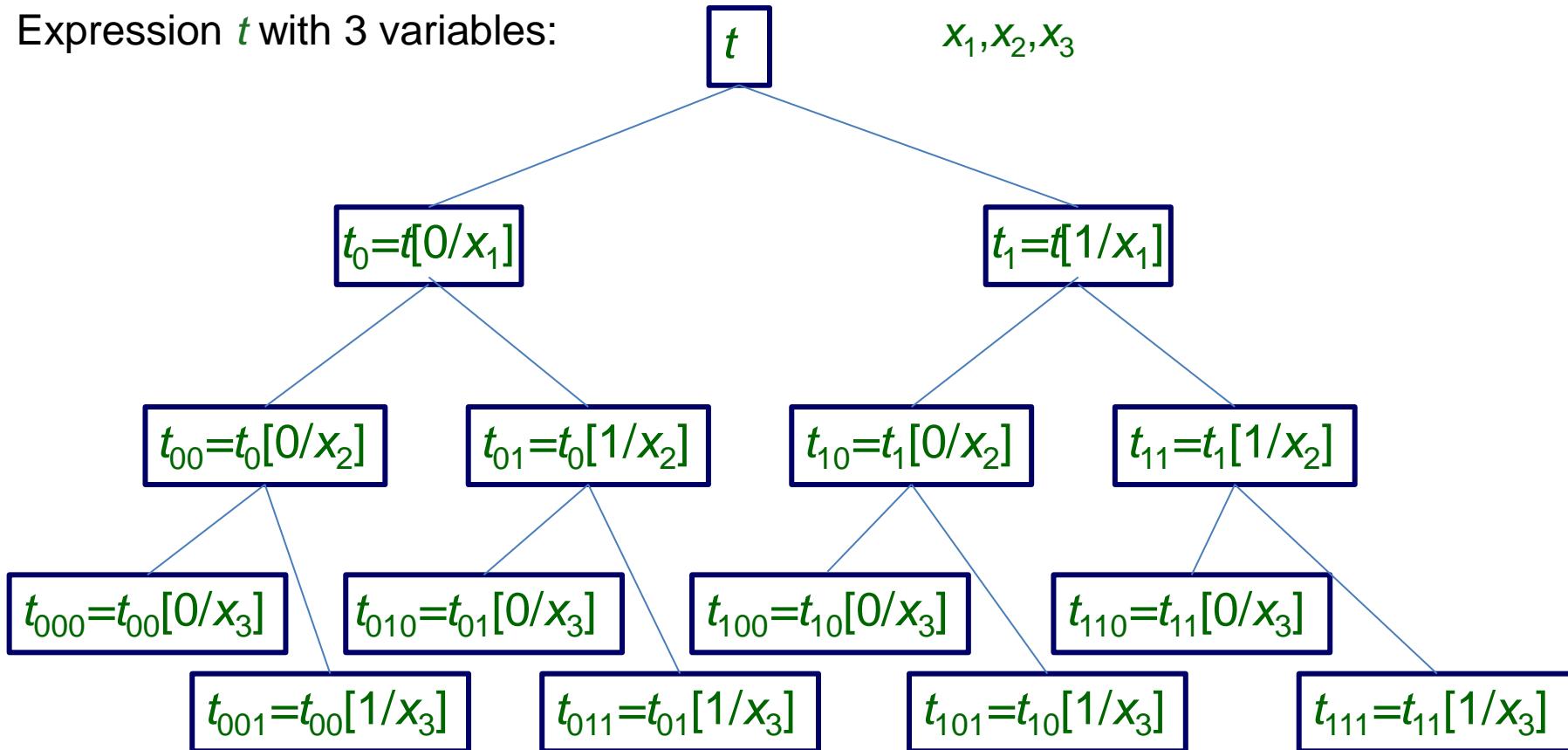
$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv x_2 \rightarrow t_{01}, t_{00}$$

$$t_1 \equiv x_2 \rightarrow t_{11}, t_{10}$$

Construction of INF

Expression t with 3 variables:



$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv x_2 \rightarrow t_{01}, t_{00}$$

$$t_1 \equiv x_2 \rightarrow t_{11}, t_{10}$$

$$t_{00} \equiv x_3 \rightarrow t_{001}, t_{000}$$

$$t_{01} \equiv x_3 \rightarrow t_{011}, t_{010}$$

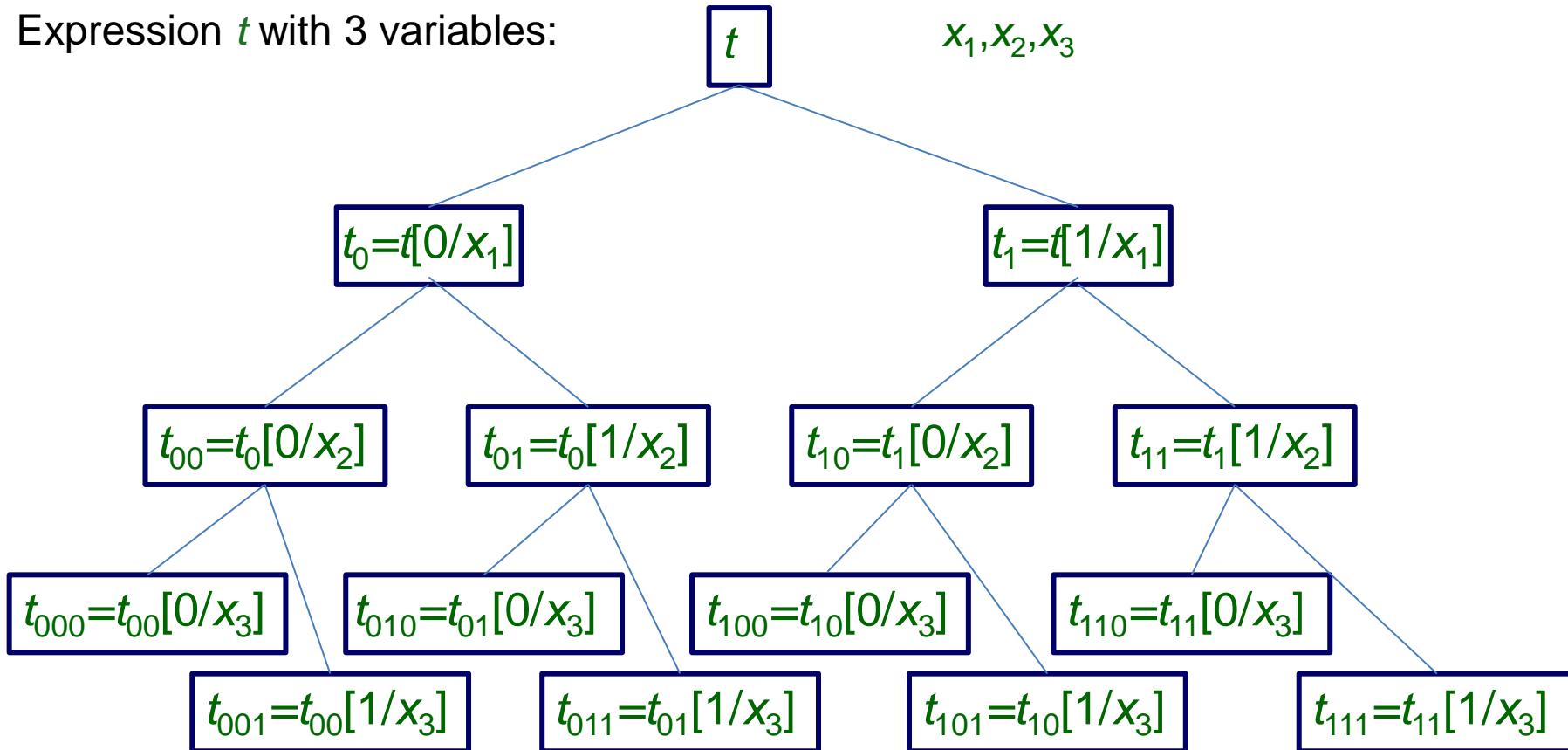
$$t_{10} \equiv x_3 \rightarrow t_{101}, t_{100}$$

$$t_{11} \equiv x_3 \rightarrow t_{111}, t_{110}$$



Construction of INF

Expression t with 3 variables:



$$t \equiv x_1 \rightarrow (x_2 \rightarrow (t_{11}, t_{10}), (x_2 \rightarrow (t_{01}, t_{00}))$$

$$t_{00} \equiv x_3 \rightarrow (t_{001}, t_{000})$$

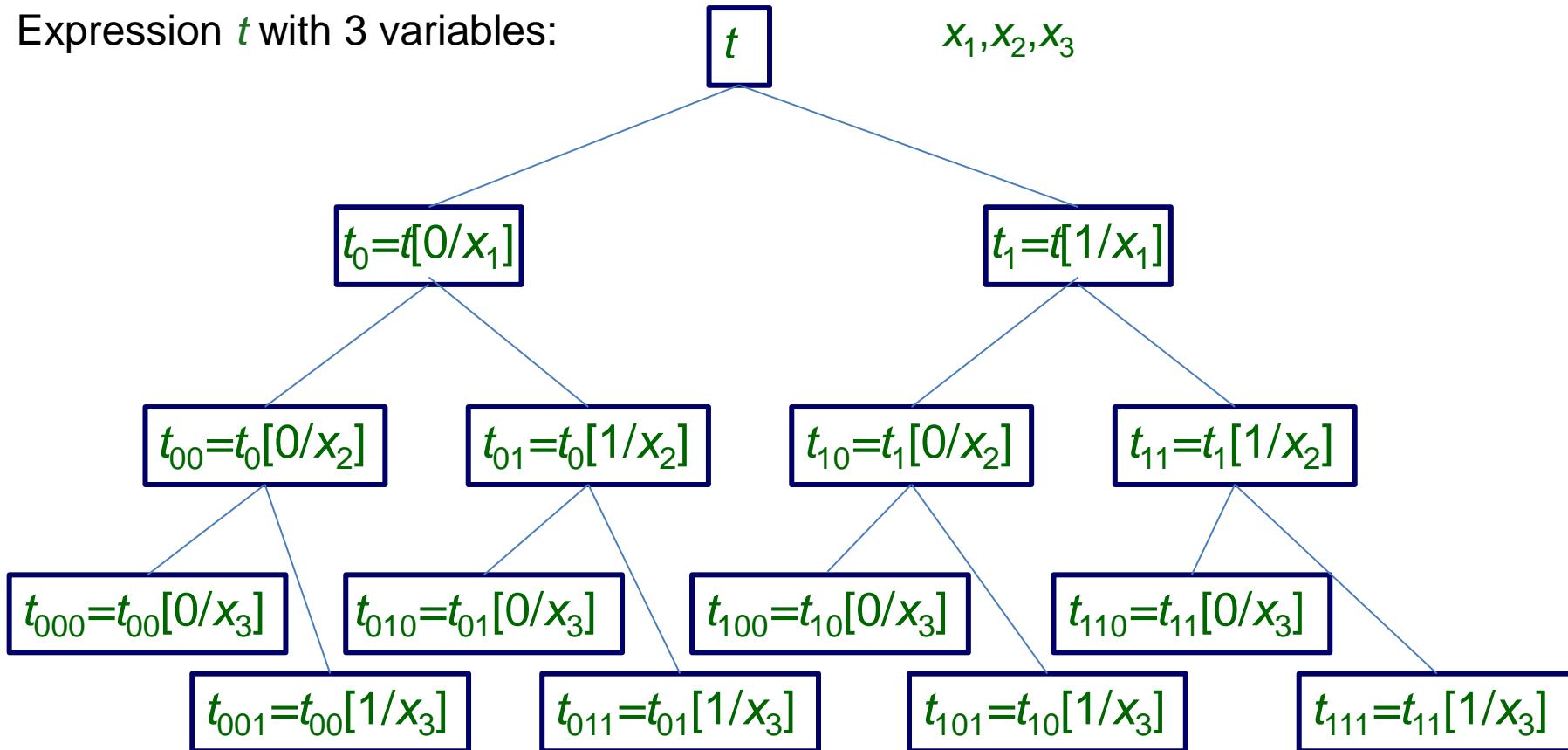
$$t_{01} \equiv x_3 \rightarrow (t_{011}, t_{010})$$

$$t_{10} \equiv x_3 \rightarrow (t_{101}, t_{100})$$

$$t_{11} \equiv x_3 \rightarrow (t_{111}, t_{110})$$

Construction of INF

Expression t with 3 variables:



$$t \equiv x_1 \rightarrow (x_2 \rightarrow (x_3 \rightarrow t_{111}, t_{110}), (x_3 \rightarrow t_{101}, t_{100})), (x_2 \rightarrow (x_3 \rightarrow t_{011}, t_{010})), (x_3 \rightarrow t_{001}, t_{000}))$$

BREAK



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

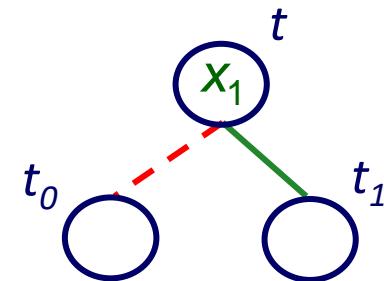


Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0$

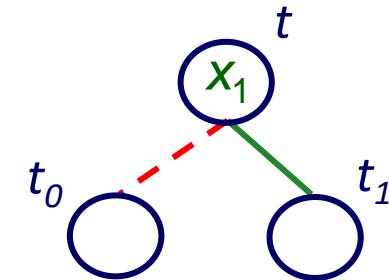


Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$



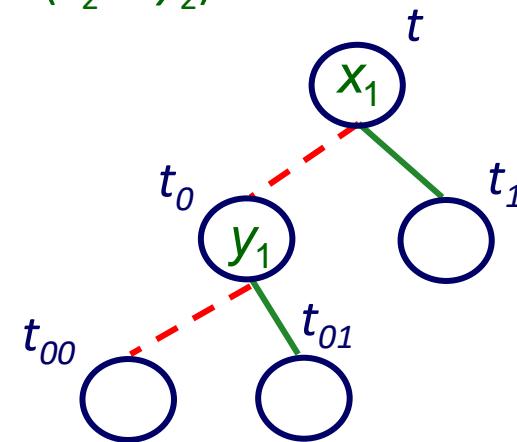
Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$

$t_0 \equiv y_1 \rightarrow t_{01}, t_{00} \equiv y_1 \rightarrow (0 \wedge 1) \vee (x_2 \wedge y_2), (0 \wedge 0) \vee (x_2 \wedge y_2)$



Example

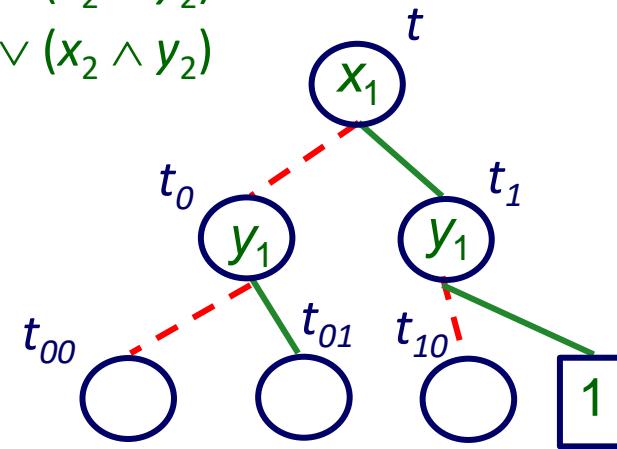
Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$$

$$t_0 \equiv y_1 \rightarrow t_{01}, t_{00} \equiv y_1 \rightarrow (0 \wedge 1) \vee (x_2 \wedge y_2), (0 \wedge 0) \vee (x_2 \wedge y_2)$$

$$\begin{aligned} t_1 &\equiv y_1 \rightarrow t_{11}, t_{10} \equiv y_1 \rightarrow (1 \wedge 1) \vee (x_2 \wedge y_2), (1 \wedge 0) \vee (x_2 \wedge y_2) \\ &\equiv y_1 \rightarrow 1, t_{10} \end{aligned}$$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$

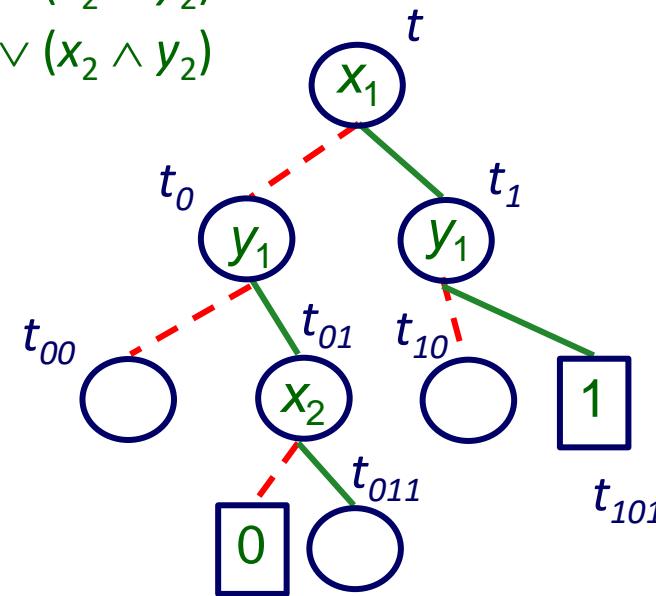
$t_0 \equiv y_1 \rightarrow t_{01}, t_{00} \equiv y_1 \rightarrow (0 \wedge 1) \vee (x_2 \wedge y_2), (0 \wedge 0) \vee (x_2 \wedge y_2)$

$t_1 \equiv y_1 \rightarrow t_{11}, t_{10} \equiv y_1 \rightarrow (1 \wedge 1) \vee (x_2 \wedge y_2), (1 \wedge 0) \vee (x_2 \wedge y_2)$

$\equiv y_1 \rightarrow 1, t_{10}$

$t_{01} \equiv x_2 \rightarrow t_{011}, t_{010} \equiv x_2 \rightarrow t_{011}, (0 \wedge 1) \vee (0 \wedge y_2)$

$\equiv x_2 \rightarrow t_{011}, 0$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$

$t_0 \equiv y_1 \rightarrow t_{01}, t_{00} \equiv y_1 \rightarrow (0 \wedge 1) \vee (x_2 \wedge y_2), (0 \wedge 0) \vee (x_2 \wedge y_2)$

$t_1 \equiv y_1 \rightarrow t_{11}, t_{10} \equiv y_1 \rightarrow (1 \wedge 1) \vee (x_2 \wedge y_2), (1 \wedge 0) \vee (x_2 \wedge y_2)$

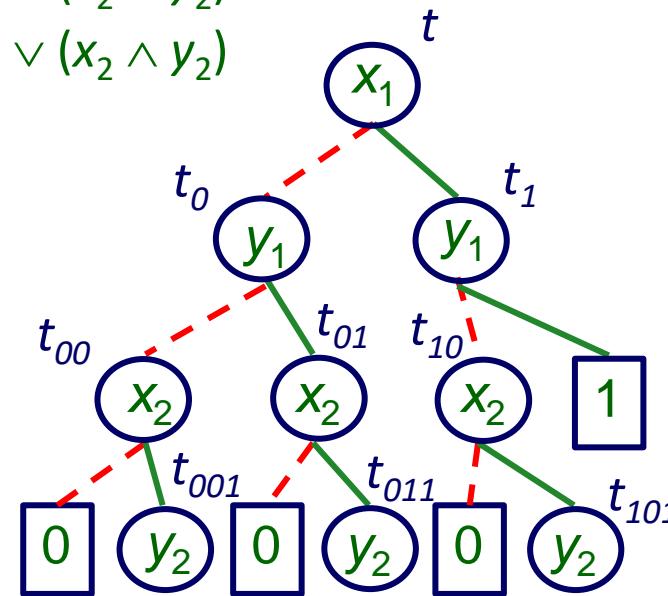
$\equiv y_1 \rightarrow 1, t_{10}$

$t_{01} \equiv x_2 \rightarrow t_{011}, t_{010} \equiv x_2 \rightarrow t_{011}, (0 \wedge 1) \vee (0 \wedge y_2)$

$\equiv x_2 \rightarrow t_{011}, 0$

$t_{00} \equiv x_2 \rightarrow t_{001}, 0$

$t_{10} \equiv x_2 \rightarrow t_{101}, 0$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow t_1, t_0 \equiv x_1 \rightarrow (1 \wedge y_1) \vee (x_2 \wedge y_2), (0 \wedge y_1) \vee (x_2 \wedge y_2)$

$t_0 \equiv y_1 \rightarrow t_{01}, t_{00} \equiv y_1 \rightarrow (0 \wedge 1) \vee (x_2 \wedge y_2), (0 \wedge 0) \vee (x_2 \wedge y_2)$

$t_1 \equiv y_1 \rightarrow t_{11}, t_{10} \equiv y_1 \rightarrow (1 \wedge 1) \vee (x_2 \wedge y_2), (1 \wedge 0) \vee (x_2 \wedge y_2)$
 $\equiv y_1 \rightarrow 1, x_2 \wedge y_2$

$t_{01} \equiv x_2 \rightarrow t_{011}, t_{010} \equiv x_2 \rightarrow t_{011}, (0 \wedge 1) \vee (0 \wedge y_2)$
 $\equiv x_2 \rightarrow t_{011}, 0$

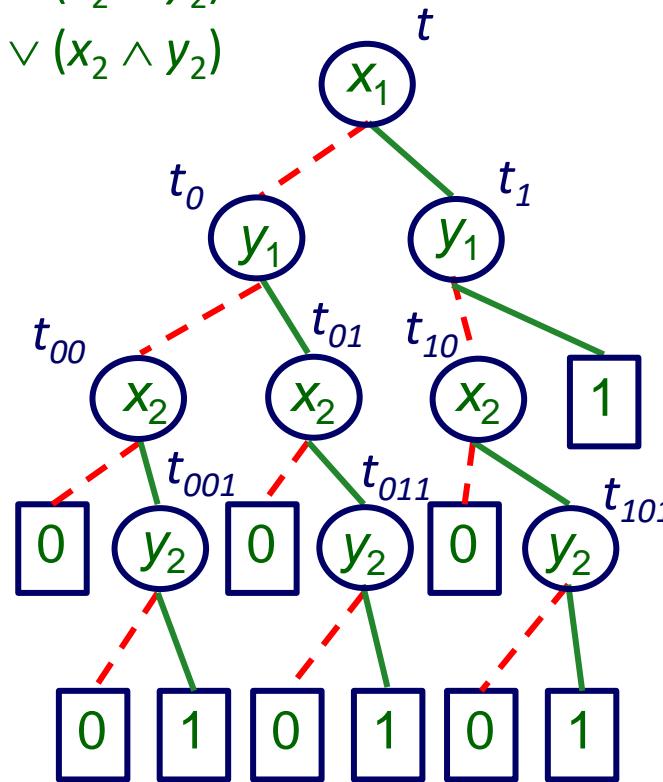
$t_{00} \equiv x_2 \rightarrow t_{001}, 0$

$t_{10} \equiv x_2 \rightarrow t_{101}, 0$

$t_{011} \equiv y_2 \rightarrow (0 \wedge 1) \vee (1 \wedge 1), (0 \wedge 1) \vee (1 \wedge 0)$
 $\equiv y_2 \rightarrow 1, 0$

$t_{001} \equiv y_2 \rightarrow 1, 0$

$t_{101} \equiv y_2 \rightarrow 1, 0$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv y_1 \rightarrow t_{01}, t_{00}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{10}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

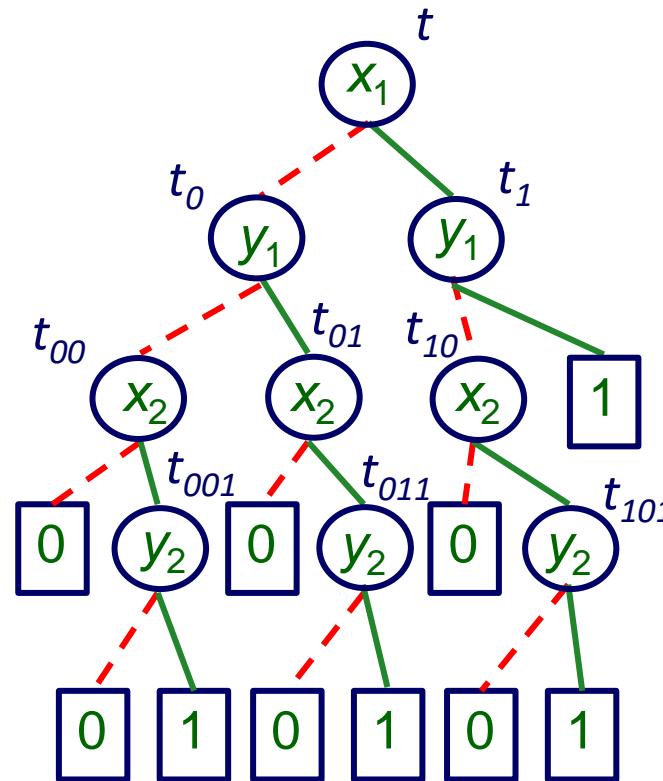
$$t_{00} \equiv x_2 \rightarrow t_{001}, 0$$

$$t_{10} \equiv x_2 \rightarrow t_{101}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$

$$t_{001} \equiv y_2 \rightarrow 1, 0$$

$$t_{101} \equiv y_2 \rightarrow 1, 0$$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow (y_1 \rightarrow 1, t_{10}), (y_1 \rightarrow t_{01}, t_{00})$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

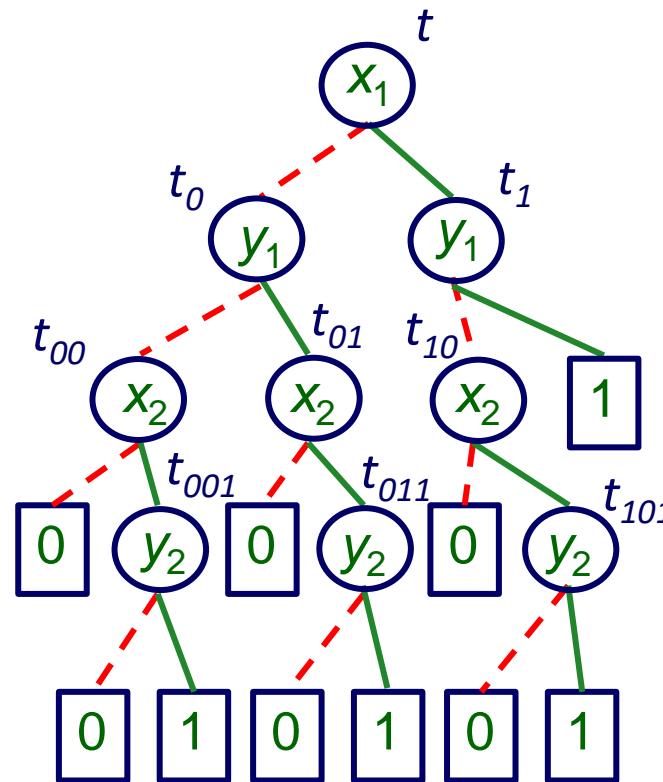
$$t_{00} \equiv x_2 \rightarrow t_{001}, 0$$

$$t_{10} \equiv x_2 \rightarrow t_{101}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$

$$t_{001} \equiv y_2 \rightarrow 1, 0$$

$$t_{101} \equiv y_2 \rightarrow 1, 0$$



Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

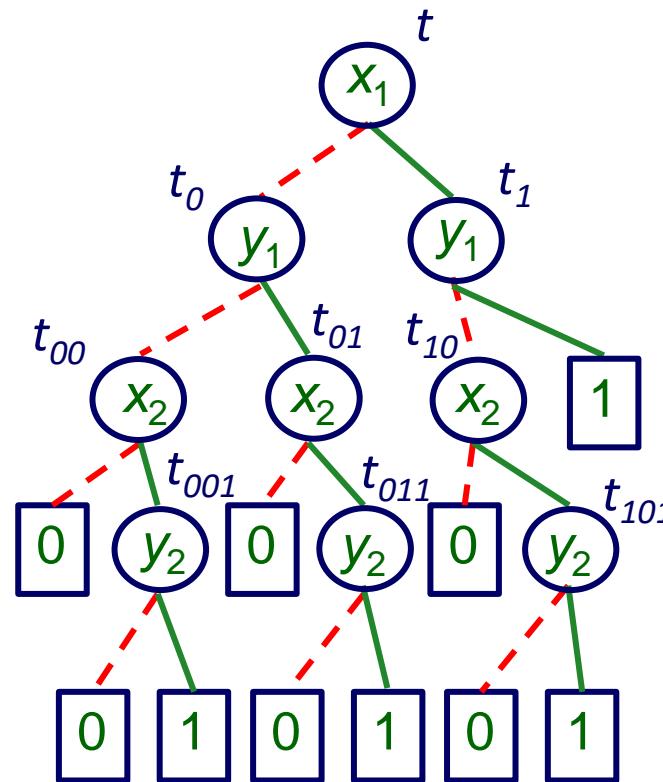
Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow (y_1 \rightarrow 1, (x_2 \rightarrow t_{101}, 0)), (y_1 \rightarrow (x_2 \rightarrow t_{011}, 0), (x_2 \rightarrow t_{001}, 0))$

$t_{011} \equiv y_2 \rightarrow 1, 0$

$t_{001} \equiv y_2 \rightarrow 1, 0$

$t_{101} \equiv y_2 \rightarrow 1, 0$

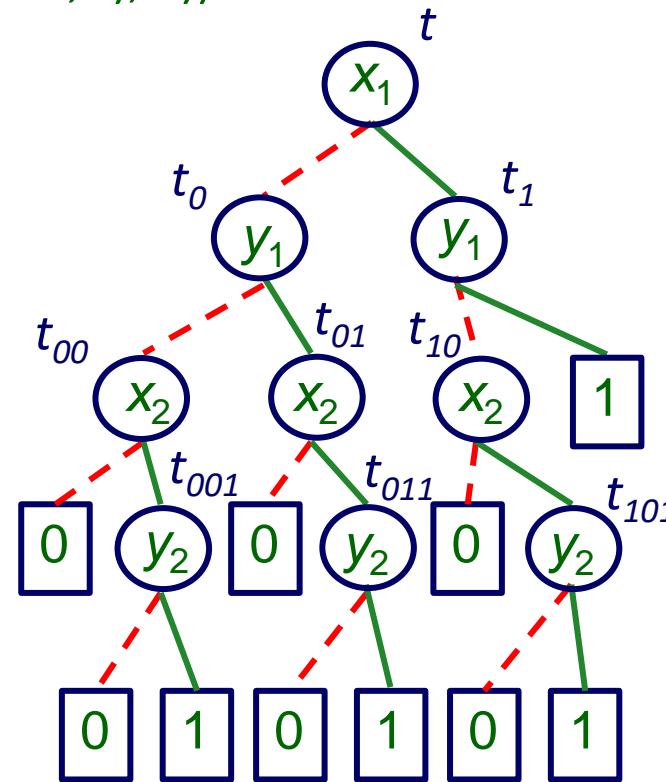


Example

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$t \equiv x_1 \rightarrow (y_1 \rightarrow 1, (x_2 \rightarrow (y_2 \rightarrow 1, 0), 0)),$
 $(y_1 \rightarrow (x_2 \rightarrow (y_2 \rightarrow 1, 0), 0), (x_2 \rightarrow (y_2 \rightarrow 1, 0), 0))$



Decision Tree

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv y_1 \rightarrow t_{01}, t_{00}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{10}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

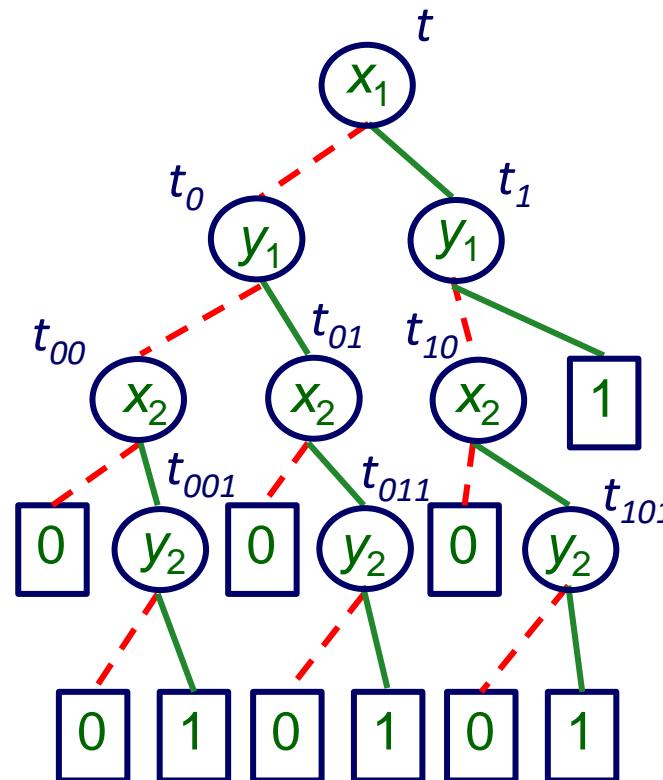
$$t_{00} \equiv x_2 \rightarrow t_{001}, 0$$

$$t_{10} \equiv x_2 \rightarrow t_{101}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$

$$t_{001} \equiv y_2 \rightarrow 1, 0$$

$$t_{101} \equiv y_2 \rightarrow 1, 0$$



Decision Tree

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv y_1 \rightarrow t_{01}, t_{00}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{10}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

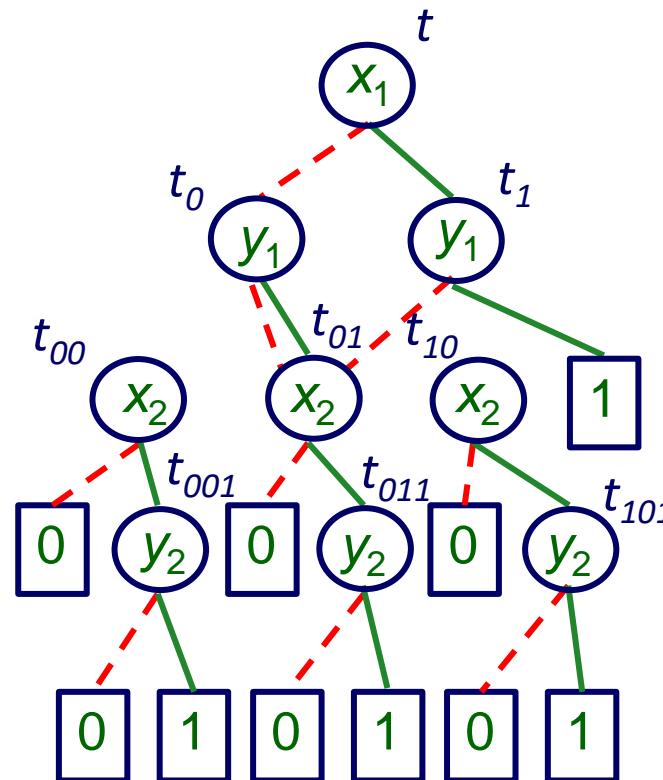
$$t_{00} \equiv x_2 \rightarrow t_{001}, 0$$

$$t_{10} \equiv x_2 \rightarrow t_{101}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$

$$t_{001} \equiv y_2 \rightarrow 1, 0$$

$$t_{101} \equiv y_2 \rightarrow 1, 0$$



Decision Tree

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

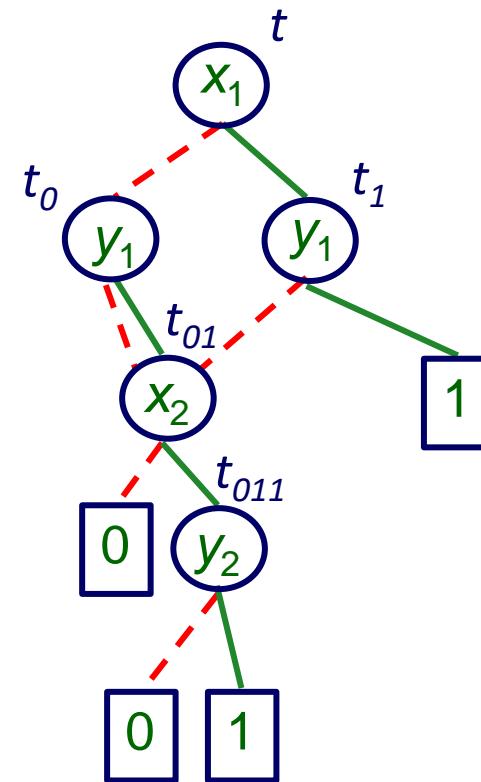
$$t \equiv x_1 \rightarrow t_1, t_0$$

$$t_0 \equiv y_1 \rightarrow t_{01}, t_{01}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{01}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$



Reduction I: Substitute Identical Subtrees

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

$$t \equiv x_1 \rightarrow t_1, t_0$$

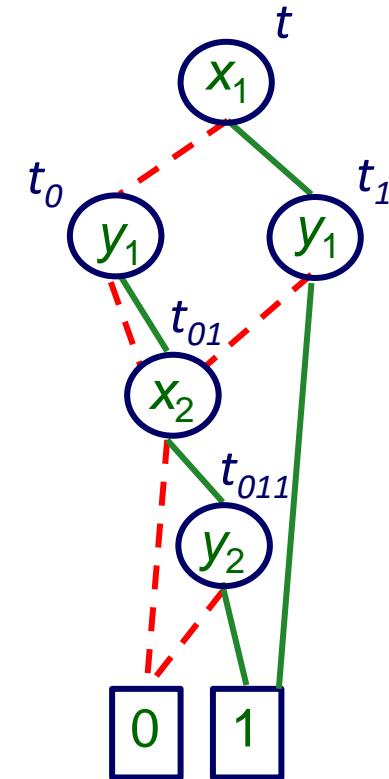
$$t_0 \equiv y_1 \rightarrow t_{01}, t_{01}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{01}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

$$t_{011} \equiv y_2 \rightarrow 1, 0$$

Result: an Ordered Binary Decision Diagram (OBDD)



Reduction II: remove redundant tests

Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Shannon expansion of t in order x_1, y_1, x_2, y_2 :

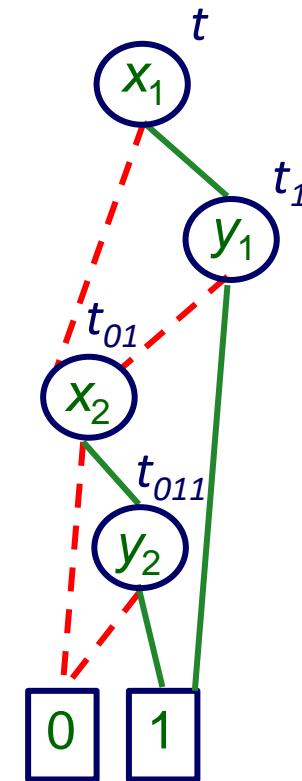
$$t \equiv x_1 \rightarrow t_1, t_{01}$$

$$t_1 \equiv y_1 \rightarrow 1, t_{01}$$

$$t_{01} \equiv x_2 \rightarrow t_{011}, 0$$

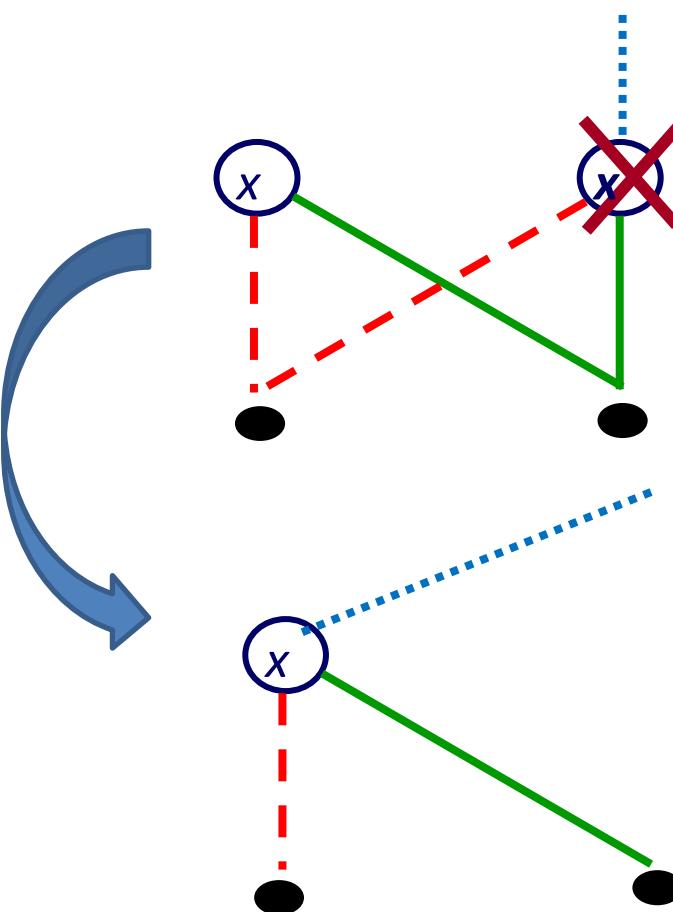
$$t_{011} \equiv y_2 \rightarrow 1, 0$$

Result: a Reduced Ordered Binary Decision Diagram (ROBDD)
[often called a BDD]

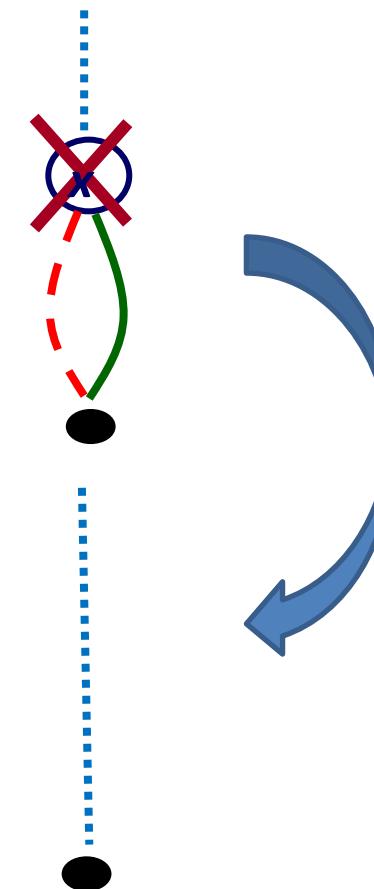


Reductions

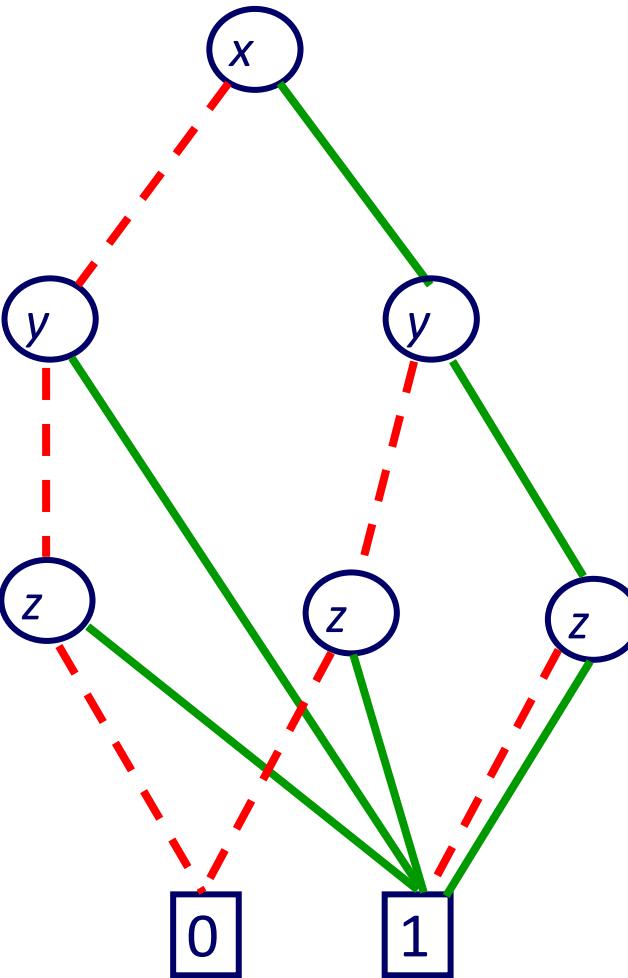
*Uniqueness
requirement*



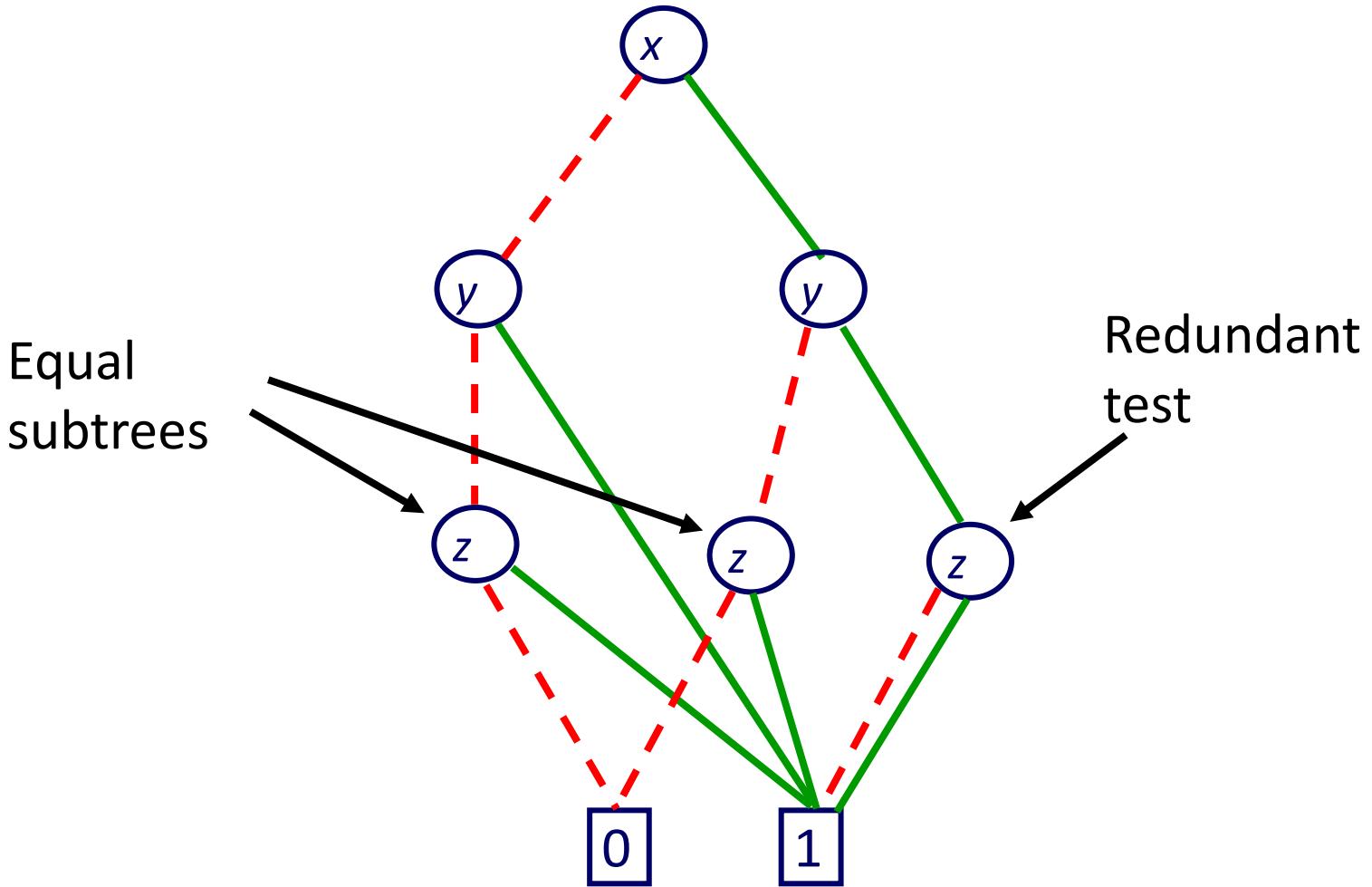
*Non-redundant
tests requirement*



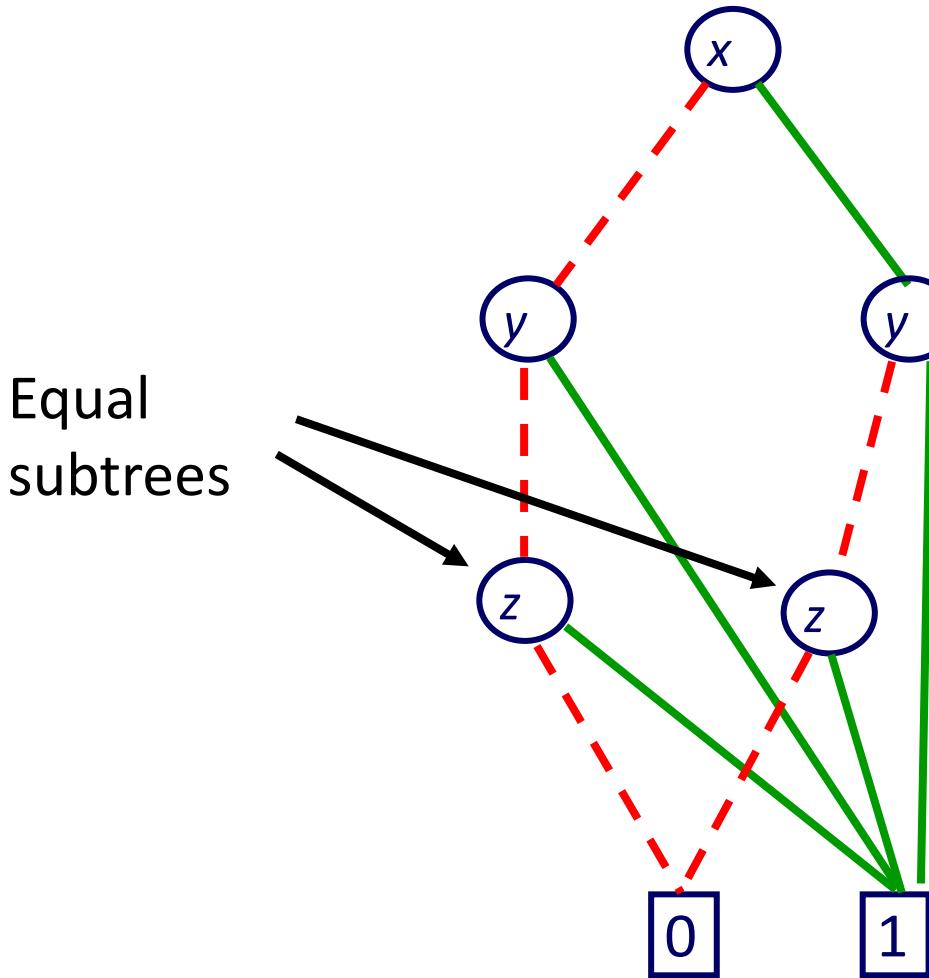
Another reduction example



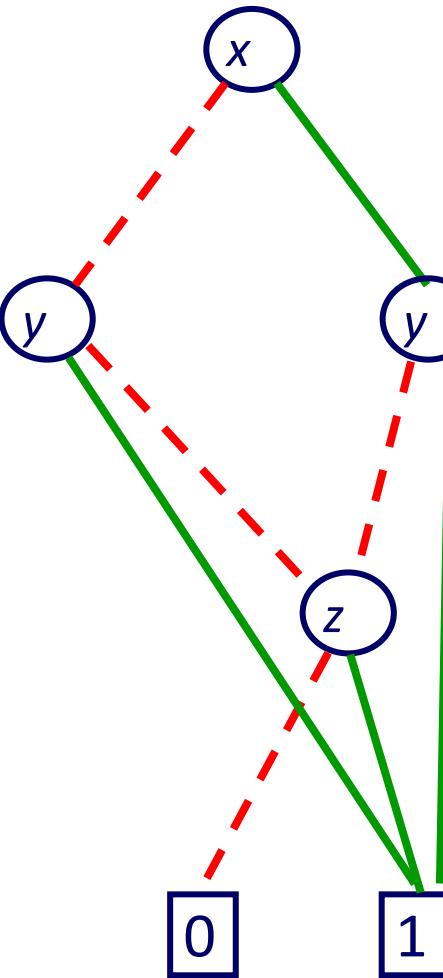
Another reduction example



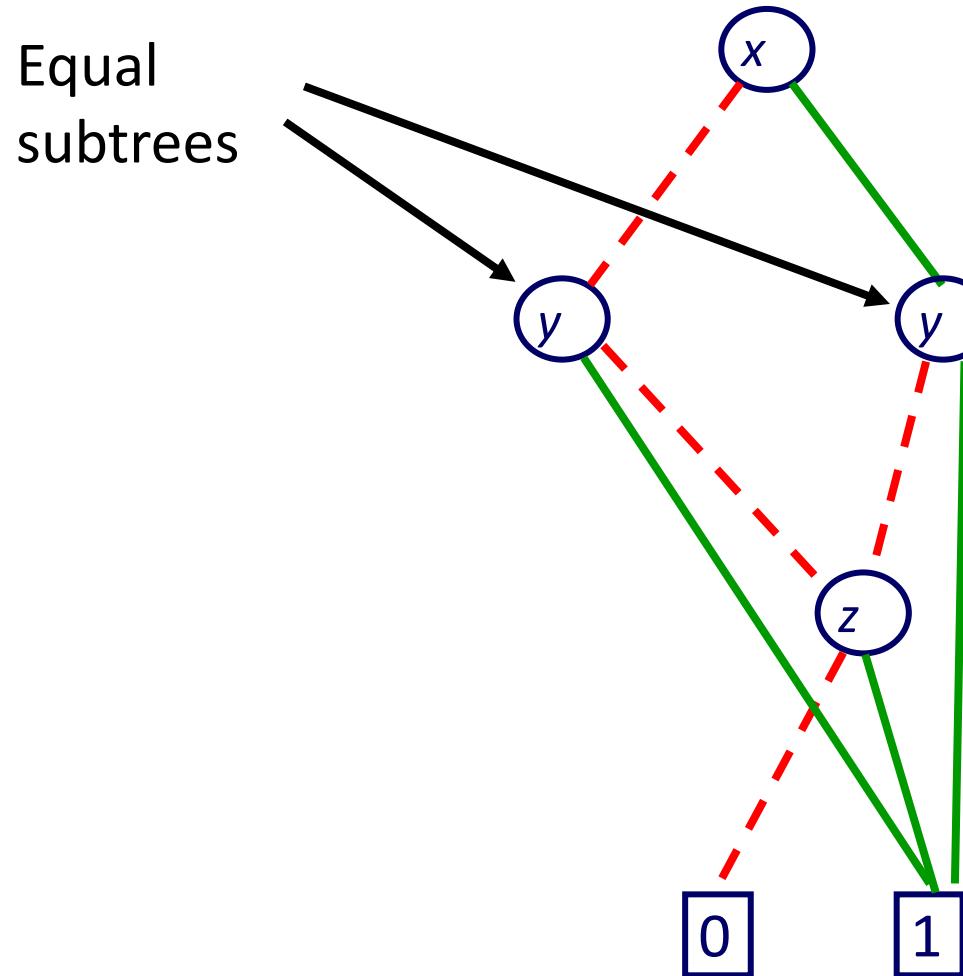
Another reduction example



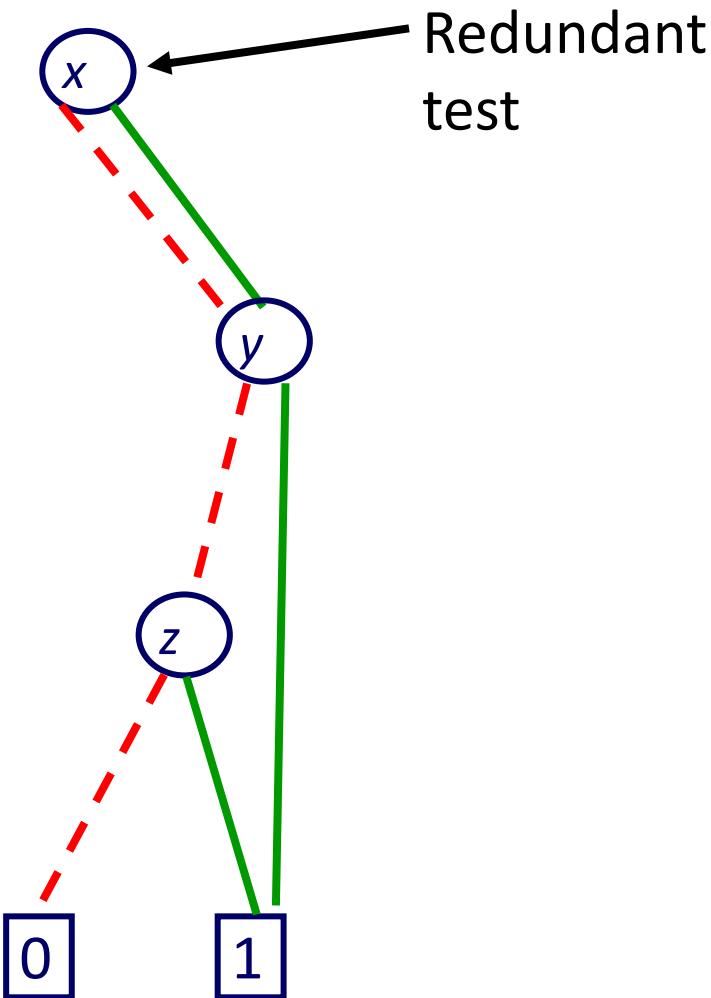
Another reduction example



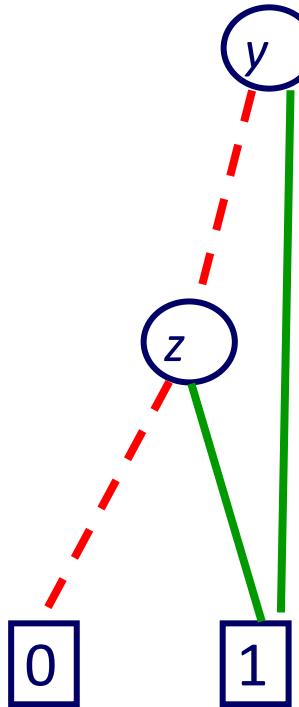
Another reduction example



Another reduction example



Another reduction example



Canonicity of ROBDDs

- **Canonicity Lemma:** for any function $f: \mathcal{B}^n \rightarrow \mathcal{B}$ there is exactly one ROBDD u with a variable ordering $x_1 < x_2 < \dots < x_n$ such that u represents $f(x_1, \dots, x_n)$

Proof (by induction on n)

Read on your own!



Practice

- What are the ROBDDs of

- x

- 1

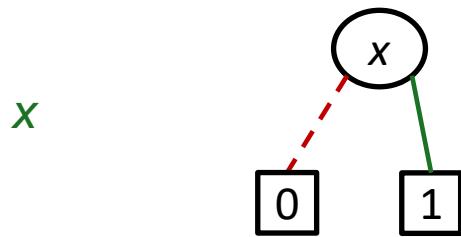
- 0

- $x \wedge y$ order: x, y

- $(x \Rightarrow y) \wedge z$ order: x, y, z

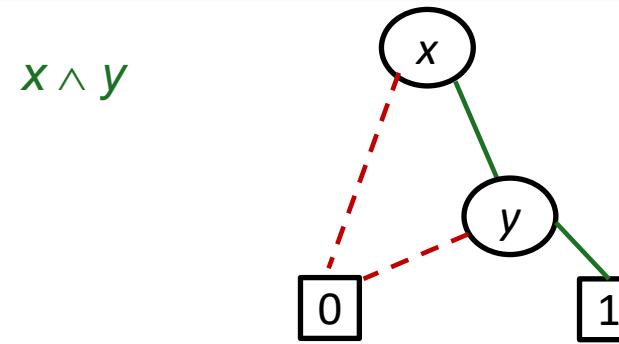


Practice

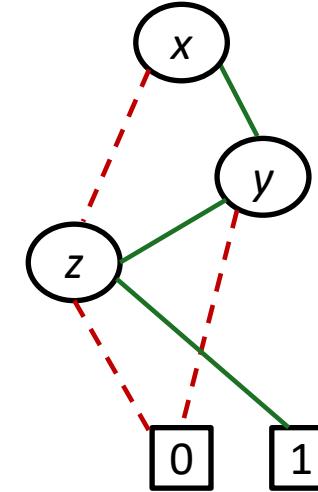


1 1

0 0



$(x \Rightarrow y) \wedge z$



Size of ROBDDs

- ROBDDs of many practically important Boolean functions are small
- Do all functions have polynomial ROBDD size? NO
 - ROBDDs do not escape the **curse of Boolean function representation**



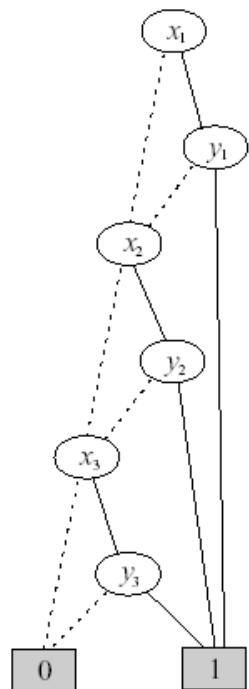
Size of ROBDDs

- The size of an ROBDD depends heavily on the variable ordering
- Example: $t = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$
- Build ROBDD of t in order x_1, x_2, y_1, y_2

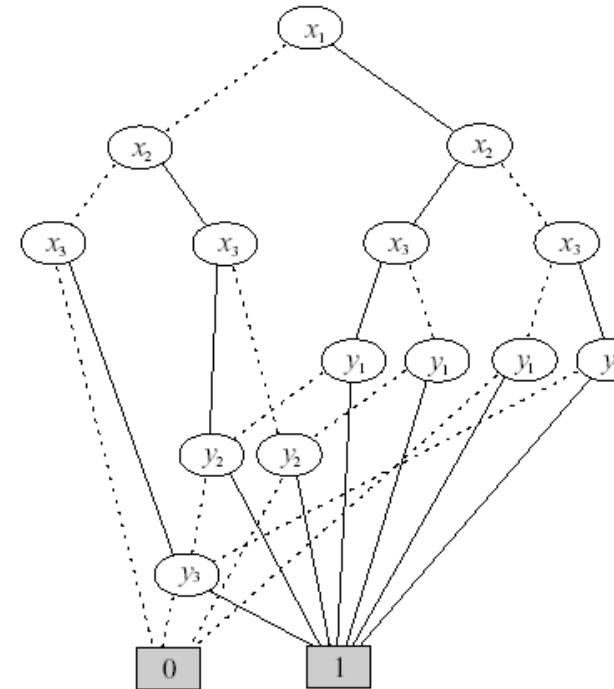
Size of ROBDDs

- The size of an ROBDD depends heavily on the variable ordering

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \dots \vee (x_n \wedge y_n)$$



$$x_1 < y_1 < x_2 < y_2 < \dots < x_n < y_n$$

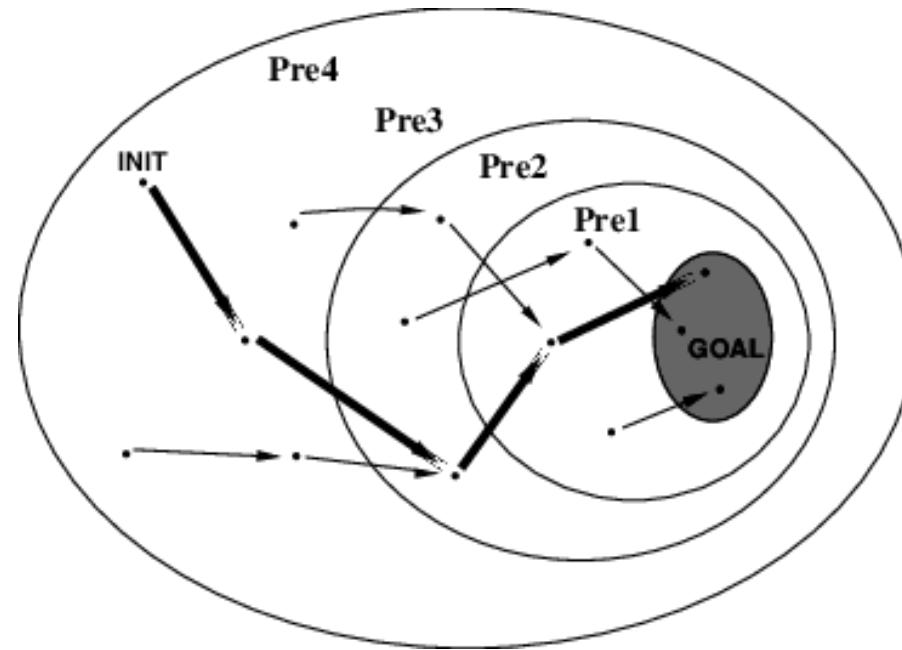


$$x_1 < x_2 < \dots < x_n < y_1 < x_2 < \dots < y_n$$

Universal Planning with BDDs

Map: $state \rightarrow action$

Characteristic function: $\text{Universal Plan}(state,action) : True$ if $action$ should be applied in $state$



State space given by transition function $T(state,action,state') : True$ if $action$ applied in $state$ leads to $state'$

Introduction to Artificial Intelligence

Lecture 7: BDD Construction and Manipulation

1. BDD construction
2. Boolean operations on BDDs
3. BDD-Based configuration (Configit)



Today's Program

- [10:00-10:50]
 - Unique table
 - $\text{Build}(t)$
 - $\text{Apply}(op, u_1, u_2)$
- [11:00-12:00] Configit, BDD-based Configuration



Henrik Hulgaard

CTO



Henrik Reif Andersen

CSO (Chief Strategy Officer)

BDD Construction

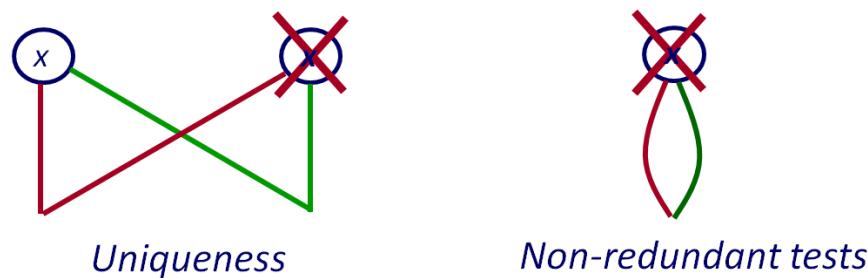


BDD construction

Last week:

1. Make a Decision Tree of the Boolean expression
2. Keep reducing it until no further reductions are possible

This week:



- Reduce the decision tree to a BDD **while building it**

Build BDDs recursively bottom-up, reduce on the way

- Represent BDD on variables x_1, x_2, \dots, x_n by a **table of unique nodes (UT)**
- To add a new node u in UT :
 1. Recursively build $high(u)$ and $low(u)$ and store in UT
 2. Maintain BDD reductions when adding u to UT :
 - a) Only extend UT with u if $high(u) \neq low(u)$ (**non-redundancy test**)
 - b) Only extend UT with u if $u \notin UT$ (**uniqueness**)

Unique Table Representation

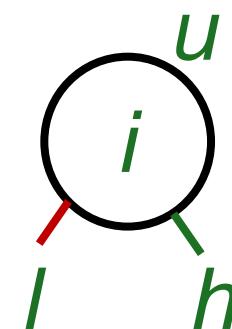
Node Attributes

u unique node identifier $\{0,1,2,3,\dots\}$

i variable index $\{1,2,\dots,n,n+1\}$

l node identifier of low

h node identifier of high



Represent Unique Table by two tables T and H

$$T : u \rightarrow (i, l, h)$$

H is the inverse of T :

$$H : (i, l, h) \rightarrow u$$

$$T(u) = (i, l, h) \Leftrightarrow H(i, l, h) = u$$

Primitive Operations on T and H

$T : u \mapsto (i, l, h)$

$init(T)$

initialize T to contain only 0 and 1

$u \leftarrow add(T, i, l, h)$

allocate a new node u with attributes (i, l, h)

$var(u), low(u), high(u)$

lookup the attributes of u in T

$H : (i, l, h) \mapsto u$

$init(H)$

initialize H to be empty

$b \leftarrow member(H, i, l, h)$

check if (i, l, h) is in H

$u \leftarrow lookup(H, i, l, h)$

find $H(i, l, h)$

$insert(H, i, l, h, u)$

make (i, l, h) map to u in H



Unique Table Interface: MakeNode (Mk)

Mk[T, H](i, l, h)

```
1:   if  $l = h$  then return  $l$ 
2:   else if member( $H, i, l, h$ ) then
3:       return lookup( $H, i, l, h$ )
4:   else  $u \leftarrow add(T, i, l, h)$ 
5:       insert( $H, i, l, h, u$ )
6:   return  $u$ 
```

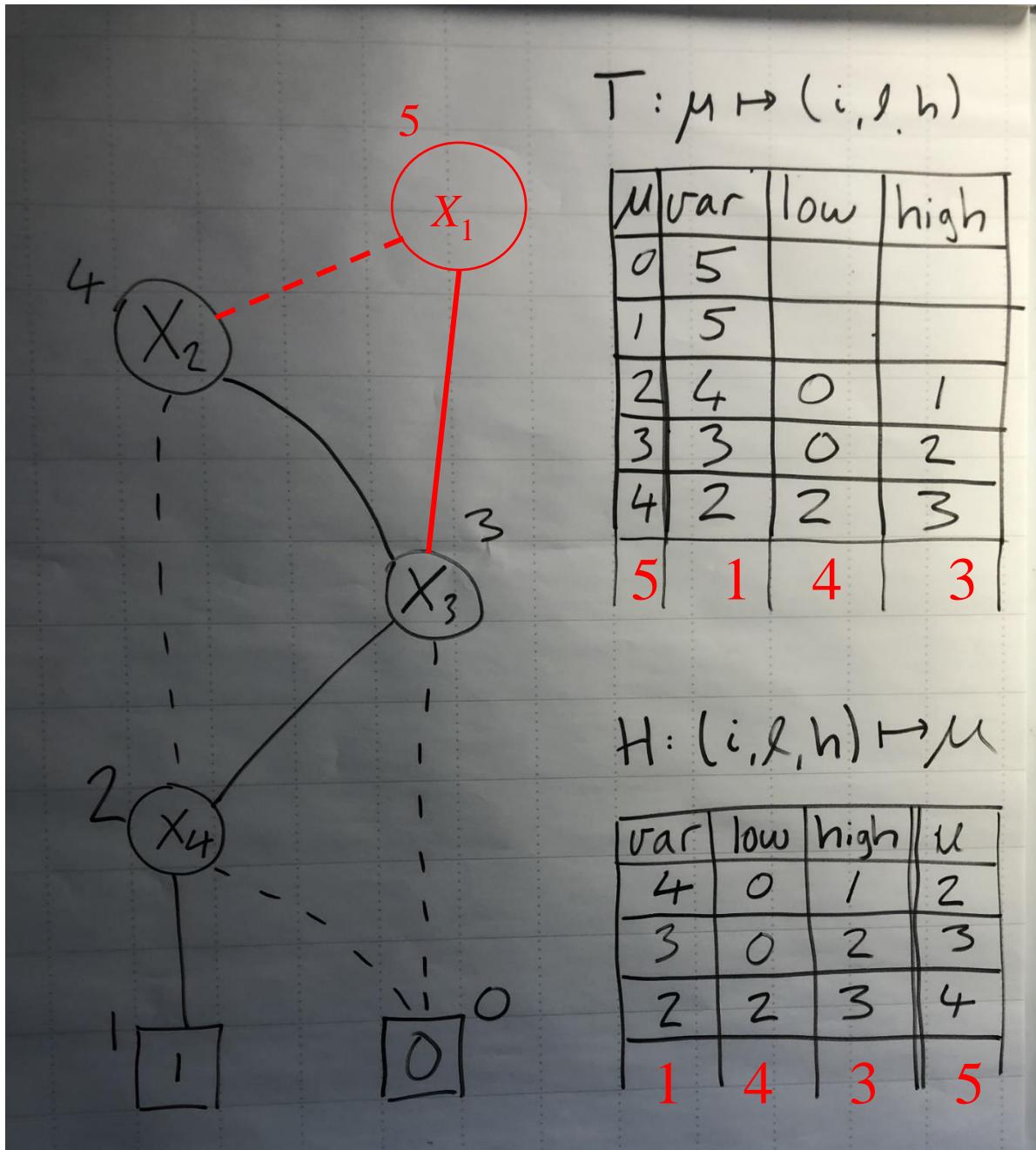
Let's do an example on T, H and Mk!



$Mk(1,4,3) \rightarrow 5$

$Mk(1,4,4) \rightarrow 4$

$Mk(3,0,2) \rightarrow 3$



$T: \mu \mapsto (i, l, h)$

i	var	low	high
0	5		
1	5		
2	4	0	1
3	3	0	2
4	2	2	3
5	1	4	3

$H: (i, l, h) \mapsto \mu$

i	var	low	high	μ
4	0	1		2
3	0	2		3
2	2	3		4
1	4	3		5

$MK[T, H](i, l, h)$

```

1: if  $l = h$  then return  $l$ 
2: else if  $member(H, i, l, h)$  then
3:   return  $lookup(H, i, l, h)$ 
4: else  $u \leftarrow add(T, i, l, h)$ 
5:    $insert(H, i, l, h, u)$ 
6: return  $u$ 

```

Build

Idea: Construct the BDD **recursively** using the
Shannon Expansion $t = x \rightarrow t[1/x], t[0/x]$



Build

Idea: Construct the BDD **recursively** using the
Shannon Expansion $t = x \rightarrow t[1/x], t[0/x]$

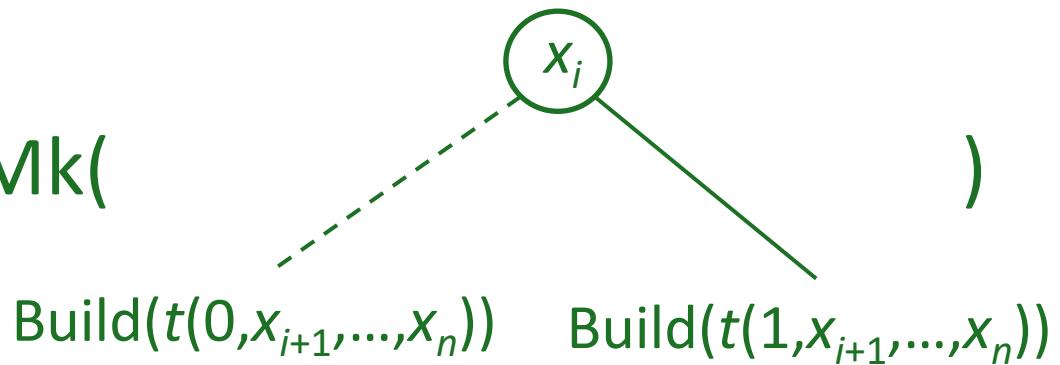
- Terminal cases

$$\text{Build}(0) = \boxed{0}$$

$$\text{Build}(1) = \boxed{1}$$

- Recursive case

$$\text{Build}(t(x_i, x_{i+1}, \dots, x_n)) = \text{Mk}($$



Build

BUILD[T, H](t)

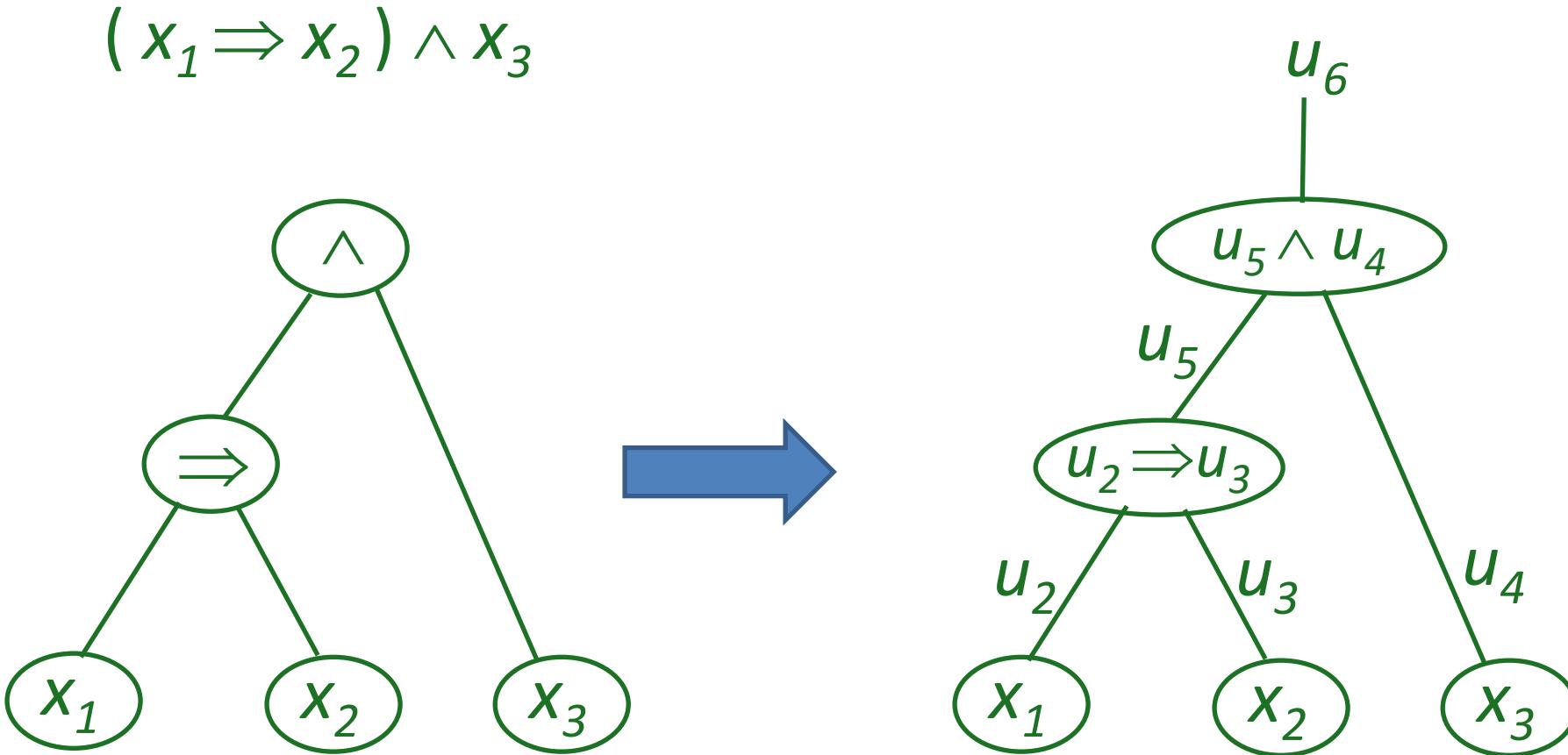
```
1: function BUILD'( $t, i$ ) =
2:     if  $i > n$  then
3:         if  $t$  is false then return 0 else return 1
4:     else  $v_0 \leftarrow \text{BUILD}'(t[0/x_i], i + 1)$ 
5:          $v_1 \leftarrow \text{BUILD}'(t[1/x_i], i + 1)$ 
6:         return MK( $i, v_0, v_1$ )
7: end BUILD'
8:
9: return BUILD'( $t, 1$ )
```



BDD Manipulation

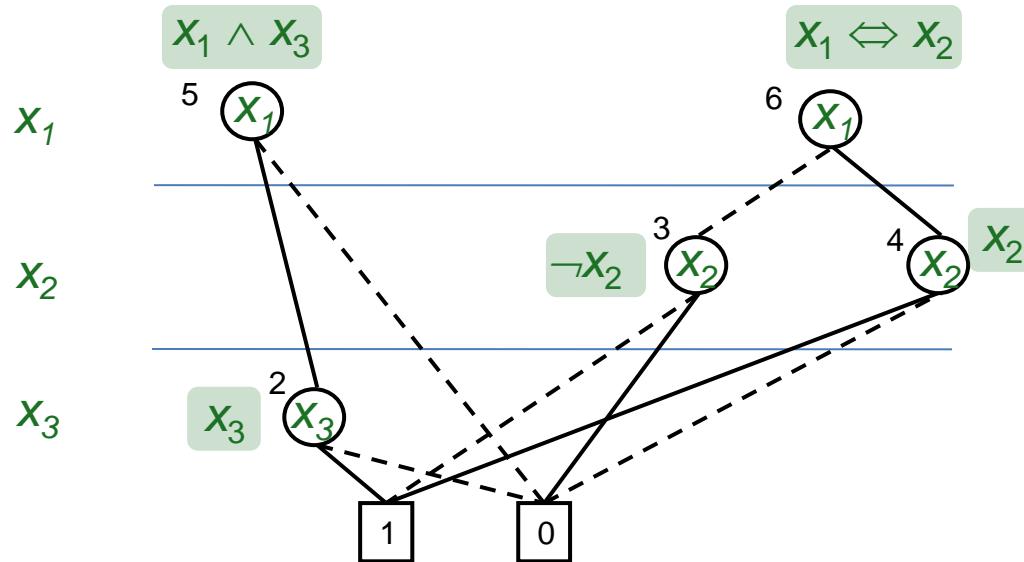


Construct BDDs from expression tree



Multi-Rooted BDD

Unique Table contains many BDDs



Apply

- $\text{Apply}(op, u_1, u_2)$: computes the BDD of

$$u_1 \text{ op } u_2$$

where

op : any of the 16 Boolean operators

u_1, u_2 : root nodes of BDDs

- Relies on the Shannon expansion properties:

$$(x \rightarrow t_1, t_0) \text{ op } (x \rightarrow t'_1, t'_0) \equiv x \rightarrow (t_1 \text{ op } t'_1), (t_0 \text{ op } t'_0)$$

$$(x \rightarrow t_1, t_0) \text{ op } t \equiv x \rightarrow (t_1 \text{ op } t), (t_0 \text{ op } t)$$



Apply with $op = \wedge$

- **Terminal case:** $u \in \{0,1\}$
 $u' \in \{0,1\}$

$$\mathbf{App}(u \wedge u') = u \wedge u'$$

- **Recursive case:** $u = x_v \rightarrow u_1, u_0$
 $u' = x_w \rightarrow u'_1, u'_0$



Apply with $op = \wedge$

- **Terminal case:** $u \in \{0,1\}$
 $u' \in \{0,1\}$
 $\text{App}(u \wedge u') = u \wedge u'$
- **Recursive case:** $u = x_v \rightarrow u_1, u_0$
 $u' = x_w \rightarrow u'_1, u'_0$
 $\text{App}(u \wedge u') =$
 $\text{Mk}(x_v, \text{App}(u_0 \wedge u'_0), \text{App}(u_1 \wedge u'_1)) \quad \text{if } v = w$
 $\text{Mk}(x_v, \text{App}(u_0 \wedge u'), \text{App}(u_1 \wedge u')) \quad \text{if } v < w$
 $\text{Mk}(x_w, \text{App}(u \wedge u'_0), \text{App}(u \wedge u'_1)) \quad \text{if } w < v$

```

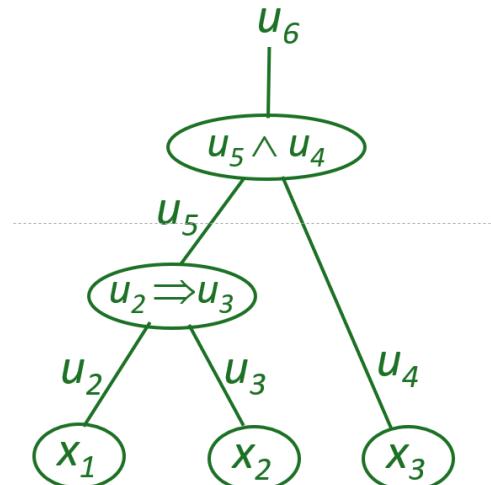
APPLY[ $T, H$ ]( $op, u_1, u_2$ )
1: init( $G$ )
2:
3: function APP( $u_1, u_2$ ) =
4:   if  $G(u_1, u_2) \neq empty$  then return  $G(u_1, u_2)$ 
5:   else if  $u_1 \in \{0, 1\}$  and  $u_2 \in \{0, 1\}$  then  $u \leftarrow op(u_1, u_2)$ 
6:   else if  $var(u_1) = var(u_2)$  then
7:      $u \leftarrow MK(var(u_1), APP(low(u_1), low(u_2)), APP(high(u_1), high(u_2)))$ 
8:   else if  $var(u_1) < var(u_2)$  then
9:      $u \leftarrow MK(var(u_1), APP(low(u_1), u_2), APP(high(u_1), u_2))$ 
10:  else (*  $var(u_1) > var(u_2)$  *)
11:     $u \leftarrow MK(var(u_2), APP(u_1, low(u_2)), APP(u_1, high(u_2)))$ 
12:   $G(u_1, u_2) \leftarrow u$ 
13:  return  $u$ 
14: end APP
15:
16: return APP( $u_1, u_2$ )

```



Properties of Apply

- Improvements?
 - Early termination. E.g., no reason to keep recursing if the left side in a conjunction is 0
- Complexity : $O(|u_1||u_2|)$, due to dynamic programming
- So a BDD of any expression can be computed in **poly** time?



Introduction to Artificial Intelligence

Lecture 8: Constraint Programming



Today's Program

- [10:00 – 10:50] **Definition and Constraint Propagation**
 - Constraint Satisfaction Problems (CSPs)
 - Constraint Propagation
- [10:00 – 12:00] **CSP Algorithms**
 - BACKTRACKING
 - Variable and value selection
 - FORWARD CHECKING & MAINTAINING ARC CONSISTENCY (MAC)
 - Global constraints
 - Modern constraint propagation systems
- [12:00 – 14:00] **Exercises**



Constraint Satisfaction Problems (CSPs)

			2	6		7		1
6	8			7			9	
1	9				4	5		
8	2		1				4	
		4	6		2	9		
5				3		2	8	
		9	3			7	4	
4				5		3	6	
7		3		1	8			

Container Vessel Slot Planning



Definition of a Constraint Satisfaction Problem (CSP)

A **CSP** is a triple $\langle X, D, C \rangle$, where:

$X = \{X_1, \dots, X_n\}$ is a finite set of **variables**.

$D = \{D_1, \dots, D_n\}$ is a set of **domains** of possible values for each variable, where $D_i = \{v_1, \dots, v_{k_i}\}$

$C = \{C_1, \dots, C_m\}$ is a set of **constraints**, where $C_i = \langle \text{scope}, \text{relation} \rangle$

e.g. $X_1 \in \{A, B\}, X_2 \in \{A, B\}$

Implicit constraint representation: $\langle (X_1, X_2), X_1 \neq X_2 \rangle$

Explicit constraint representation: $\langle (X_1, X_2), [(A, B), (B, A)] \rangle$

Typical short notation: $X_1 \neq X_2$



CSP Solutions

- **Partial Assignment** : Values assigned to only some of the variables.
- **Complete Assignment** : Each variable has a value assigned.
- **Consistent Assignment** : Each constraint, where all variables in its scope are assigned, is satisfied.
- **Solution** : Complete consistent assignment.



Types of Constraints

Different Arity

- **Unary constraints** involve a single variable.
 - e.g. $X \neq 12$, $P \in \{ Red, Black \}$
- **Binary constraints** involve pairs of variables.
 - e.g. $X > Y$, $(P = Trump) \Rightarrow (C = Orange)$
- **Global constraints** involve arbitrary number of variables.
 - e.g. *AllDifferent*



Assume all constraints binary or unary

BEGIN

Why: any n-ary constraints can be made unary and binary!

Example:

original constraint and variables:

$$X+Y=Z$$

$X::[1,2]; Y::[3,4]; Z::[5,6]$

encapsulated variable and reduced domain:

$$U::[(1,4,5),(2,3,5),(2,4,6)]$$

$\text{arg1}(U,X) \quad \text{arg2}(U,Y) \quad \text{arg3}(U,Z)$

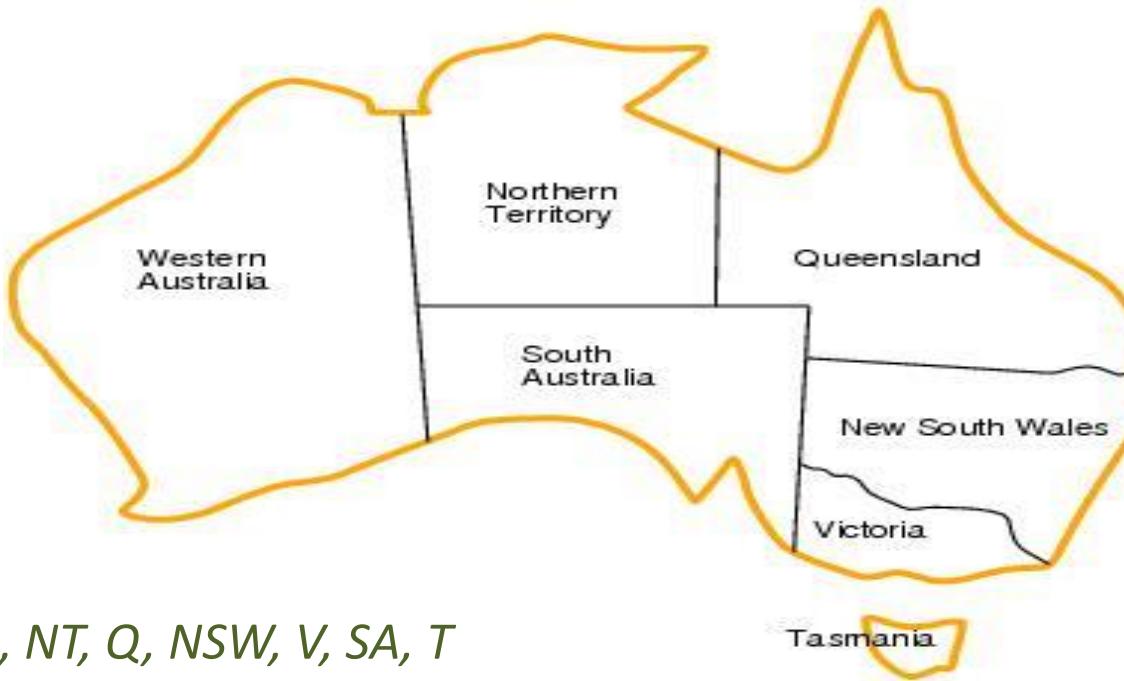
$\text{arg1}((1,4,5), 1) : \text{True}$

$\text{arg2}((1,4,5), 4) : \text{True}$

$\text{arg3}((1,4,5), 5) : \text{True}$



CSP Example: Map Coloring

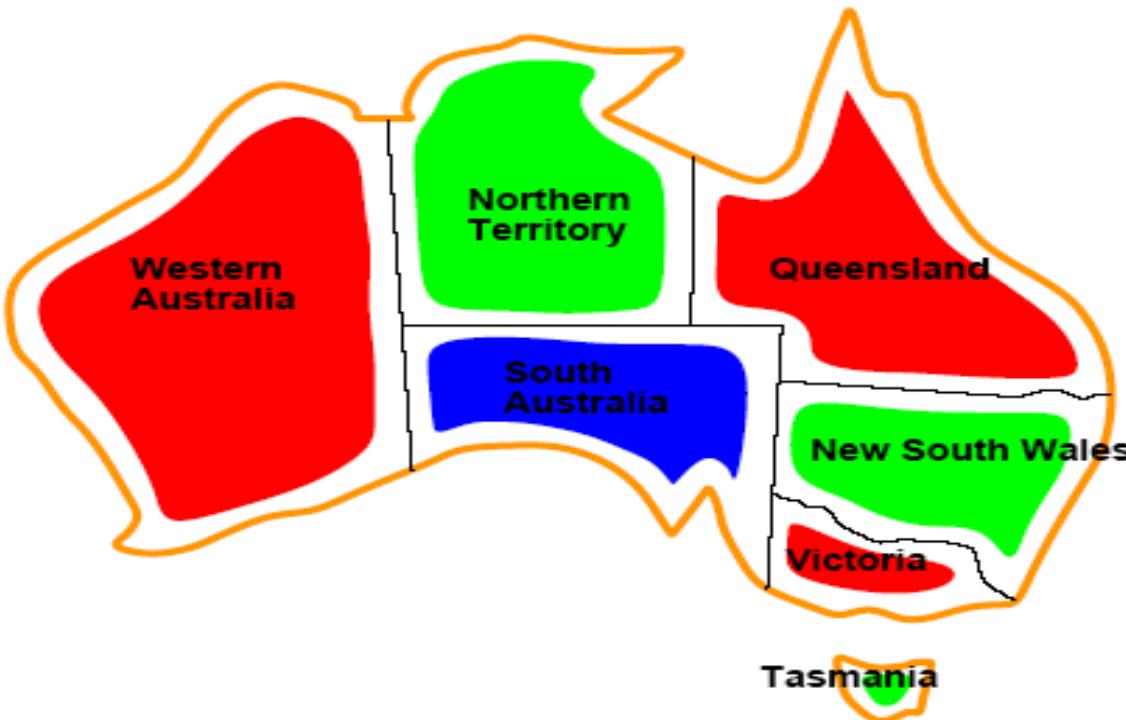


- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D_i = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors.

e.g. $\langle(WA,NT), WA \neq NT\rangle$

$\langle(WA,NT) , [(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)]\rangle$

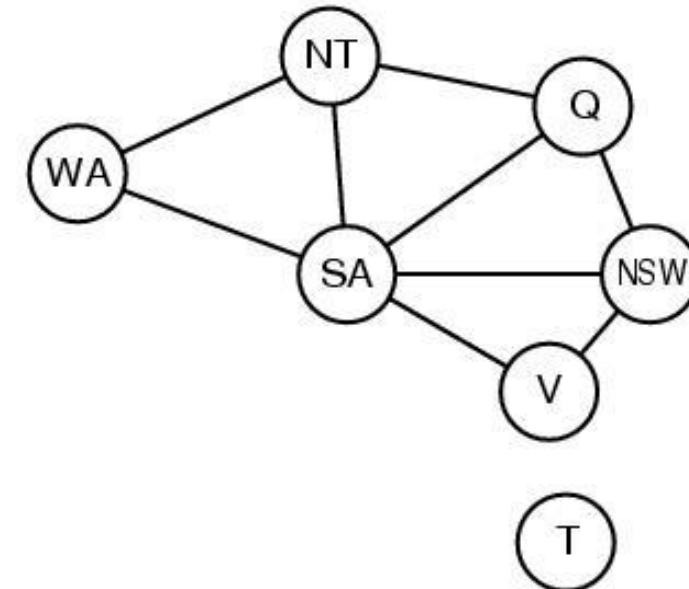
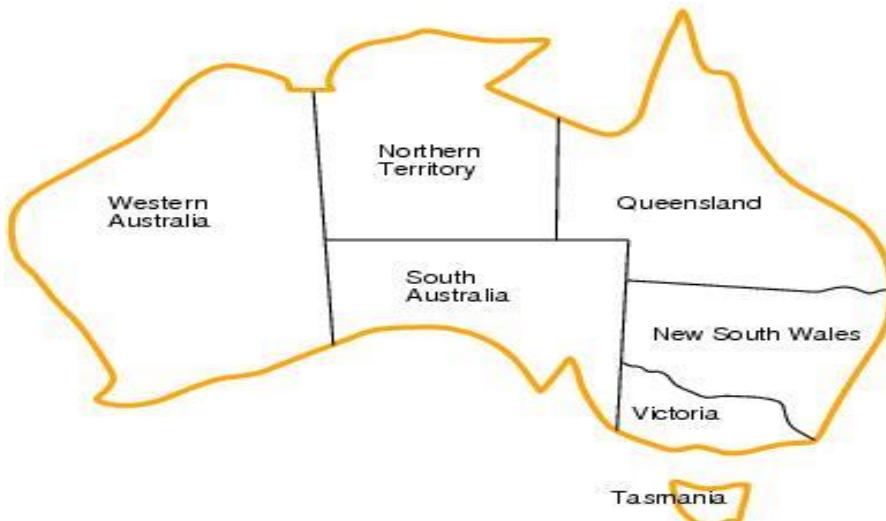
CSP Example: Map Coloring



- Solutions are assignments satisfying all constraints, e.g.:
 $\{WA=\text{red}, NT=\text{green}, Q=\text{red}, NSW=\text{green}, V=\text{red}, SA=\text{blue}, T=\text{green}\}$

Constraint Graph Representation of CSP

- Nodes are variables
- Edges are **binary constraints**



Constraint Propagation (Rule Inference)

CSPs are solved combining **search**
and **constraint propagation**



Constraint Propagation

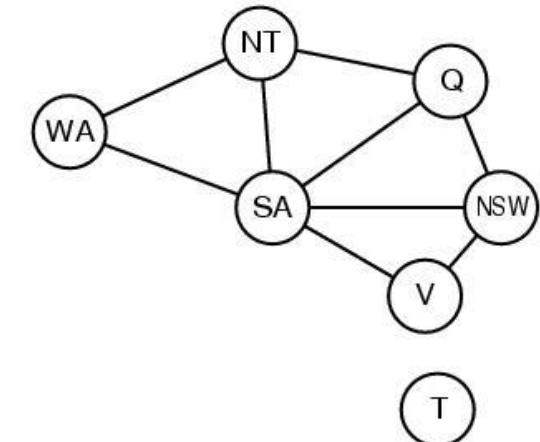
Goal: Local Consistency in the Constraint Graph

- **Node consistency** : for every value v_i of variable X , all unary constraints of X are satisfied.

Example

$SA \in \{red, green, blue\}, SA \neq \{green\}$

SA node consistent: $SA \in \{red, blue\}$



- **Arc consistency**: for every $X \rightarrow Y$ arc, every value v_i in X has a support value u_j in Y .

Arc Consistency Examples

1) $SA \in \{red, green, blue\}$, $NT \in \{blue\}$, $SA \neq NT$

$SA \rightarrow NT$ arc-consistent: $SA \in \{red, green\}$

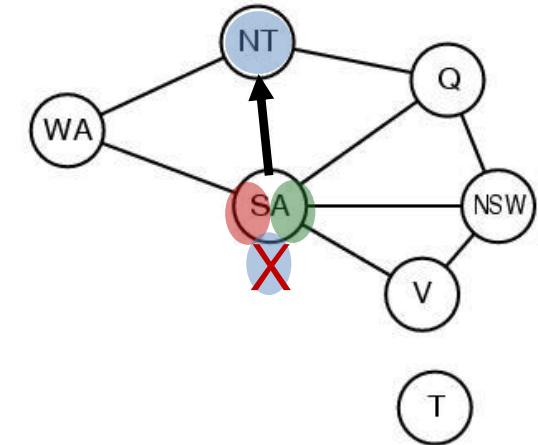
2) $X, Y \in \{0, 1, \dots, 9\}$, $Y = X^2$

$X \rightarrow Y$ arc-consistent : $X \in \{0, 1, 2, 3\}$, $Y \in \{0, 1, \dots, 9\}$

$Y \rightarrow X$ arc-consistent: $X \in \{0, 1, \dots, 9\}$, $Y \in \{0, 1, 4, 9\}$

3) $SA \in \{red, green, blue\}$, $NT \in \{red, green, blue\}$, $SA \neq NT$

$SA \rightarrow NT / NT \rightarrow SA$ arc-consistent: can we prune any values?



Arc Consistency Algorithm AC-3

function $\text{AC-3}(csp)$ **returns** false if an inconsistency is found and true otherwise (+ updated CSP)

inputs: csp , a binary CSP with components (X, D, C)

local variables: $queue$, a queue of arcs, initially all the arcs in csp

while $queue$ is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)$

if $\text{REVISE}(csp, X_i, X_j)$ **then**

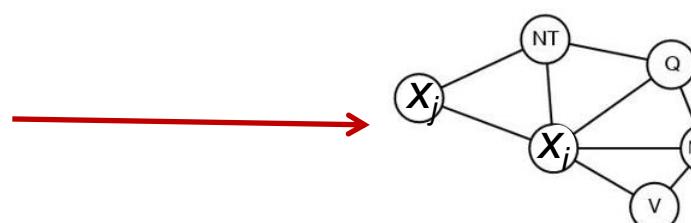
if size of $D_i = 0$ **then return** false

for each X_k **in** $X_i.\text{NEIGHBORS} - \{X_j\}$ **do**

 add (X_k, X_i) to $queue$

return true

Obs: two arcs for
each binary
constraint!



function $\text{REVISE}(csp, X_i, X_j)$ **returns** true iff we revise the domain of X_i

$\text{revised} \leftarrow \text{false}$

for each x **in** D_i **do**

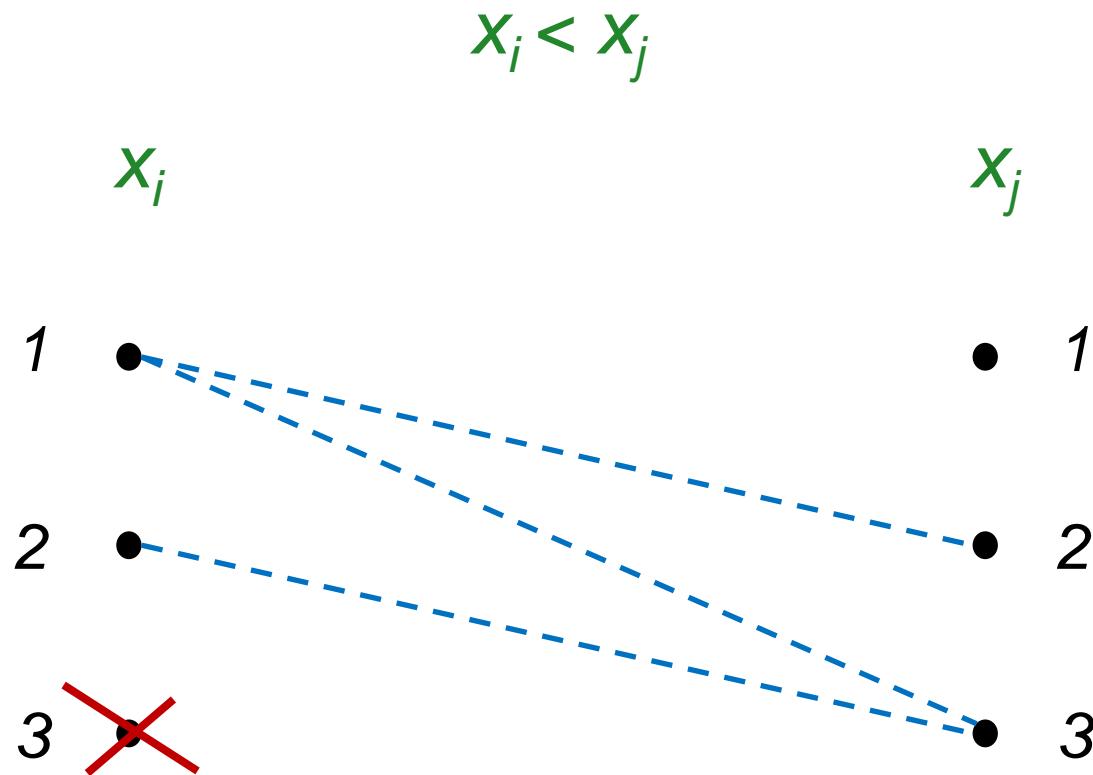
if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**

 delete x from D_i

$\text{revised} \leftarrow \text{true}$

return revised

Revise(x_i, x_j) Example



AC-3 Example

$$x_1, x_2, x_3 \in \{1,2,3\}$$

$$x_1 > x_2$$

$$x_2 > x_3$$

Q

$$\cancel{x_1 \rightarrow x_2}$$

$$\cancel{x_2 \rightarrow x_1}$$

$$\cancel{x_2 \rightarrow x_3}$$

$$\cancel{x_3 \rightarrow x_2}$$

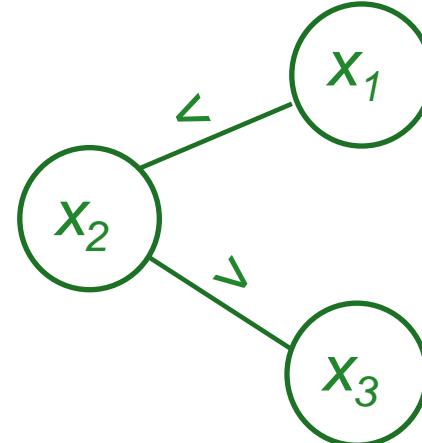
$$\cancel{x_3 \rightarrow x_2}$$

$$\cancel{x_1 \rightarrow x_2}$$

$$x_1 = \cancel{\{1,2,3\}}$$

$$x_2 = \cancel{\{1,2,3\}}$$

$$x_3 = \cancel{\{1,2,3\}}$$



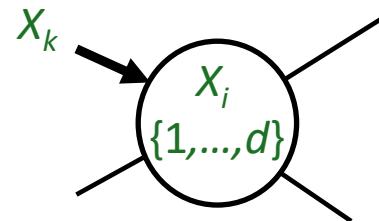
Complexity of AC-3

Assume

- n variables,
- at most d values in domains
- c binary constraints

Observations

- An arc (x_k, x_i) can be added to Q at most d times, since x_i has at most d values
- An arc can be revised in d^2
- Thus, worst case runtime is $O(cd^3)$



Coffee Break



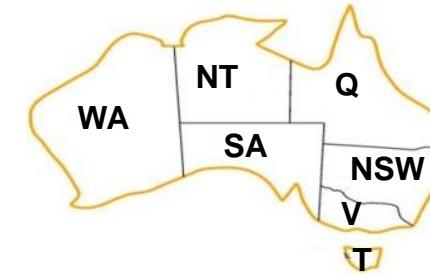
CSP Solving



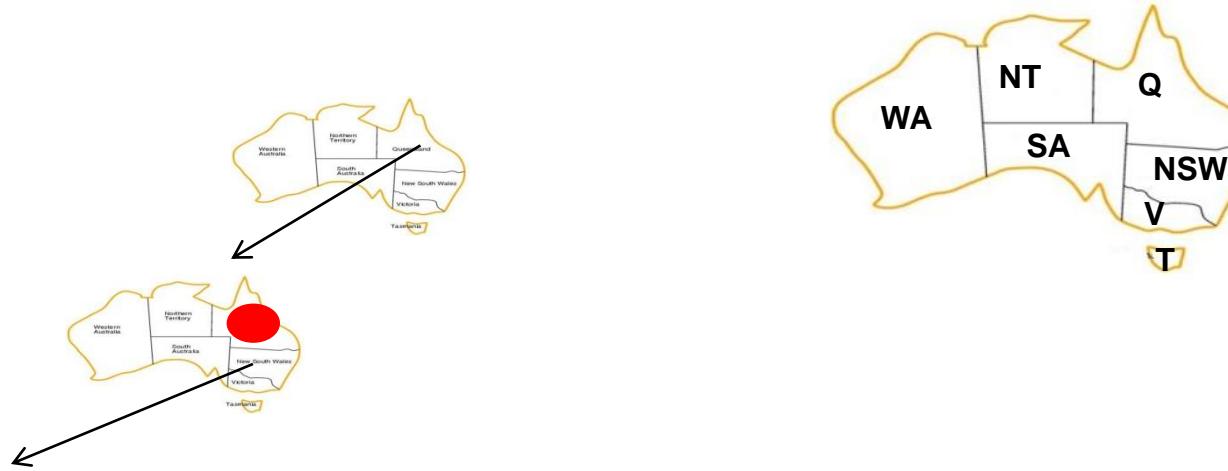
Search in CSP

- Inference is not enough
- Apply depth-first search:
 - State: Partial assignment
 - Action: $var = value$
- **Backtracking Algorithm:**
 - Choose values for one variable at the time.
 - Backtrack when a variable has no legal values left.

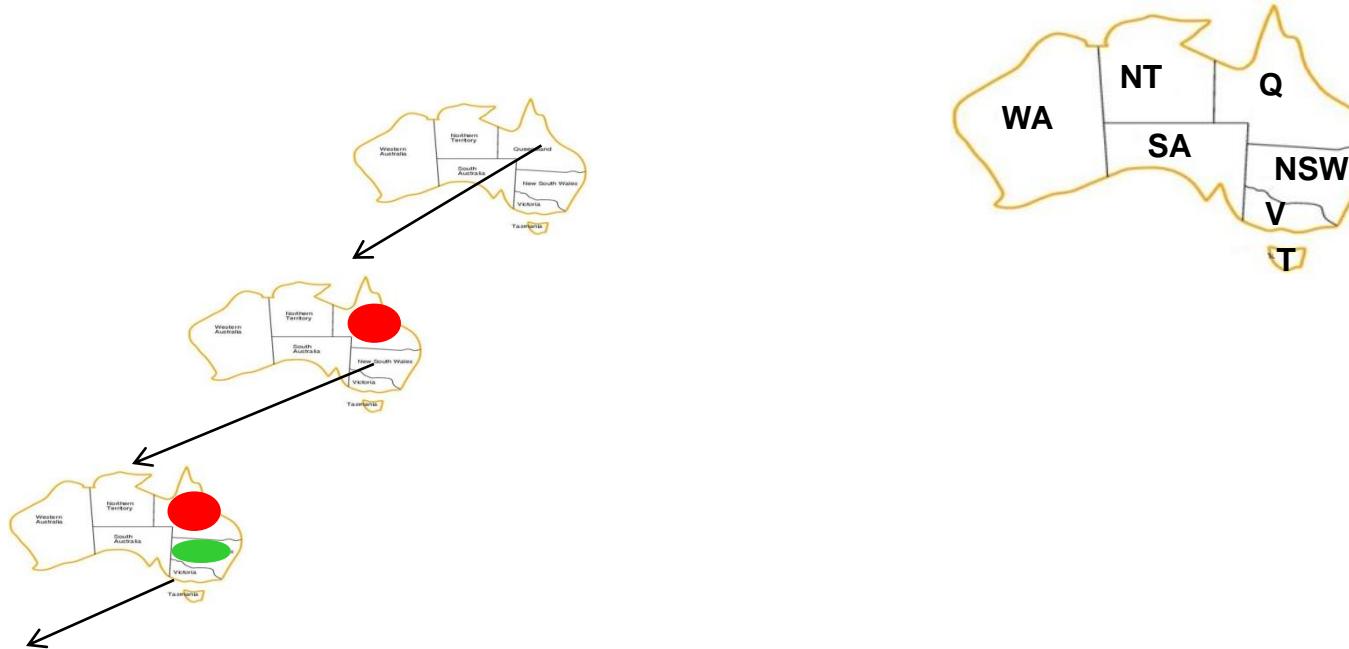
Backtracking Example



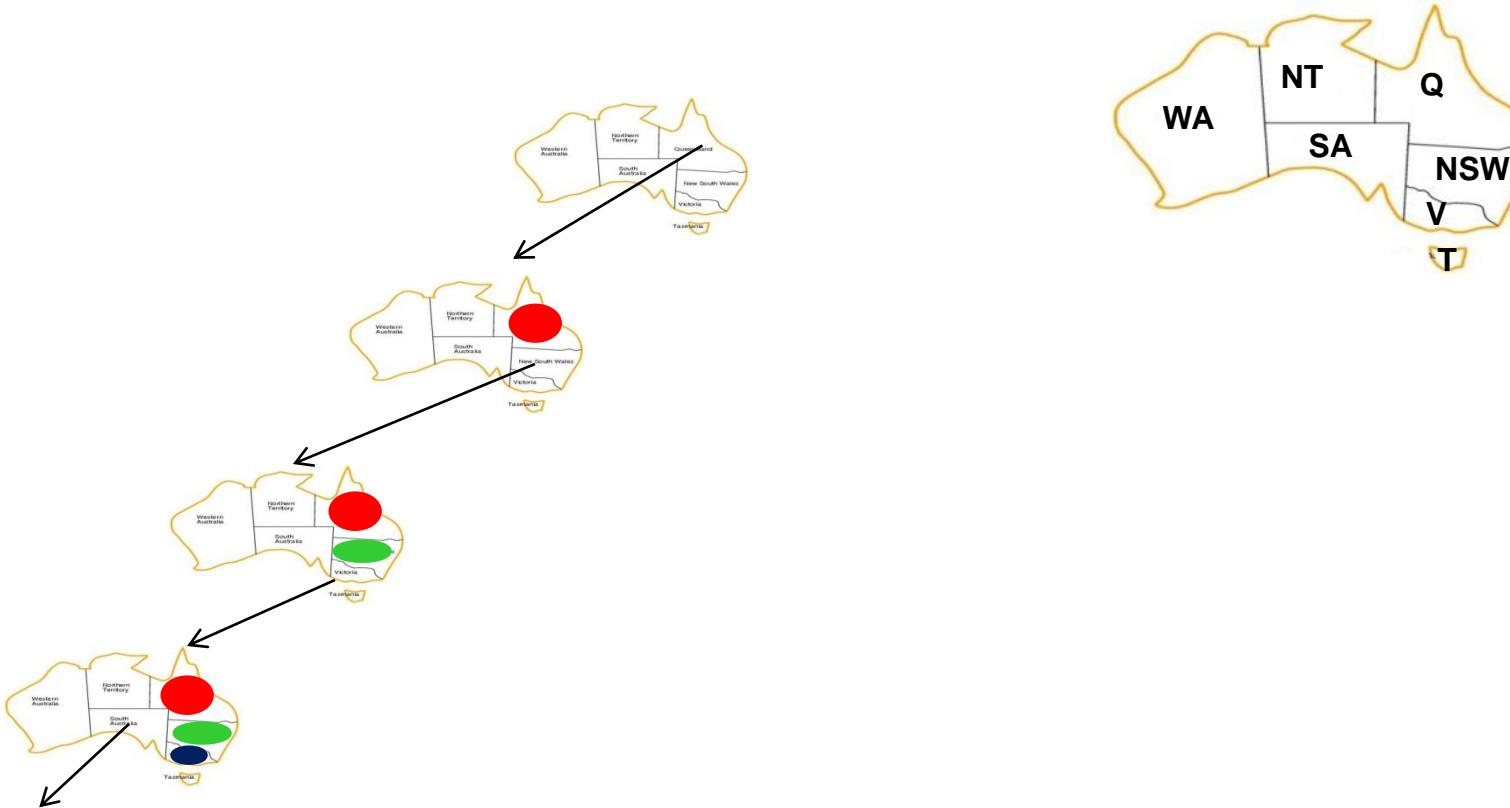
Backtracking Example



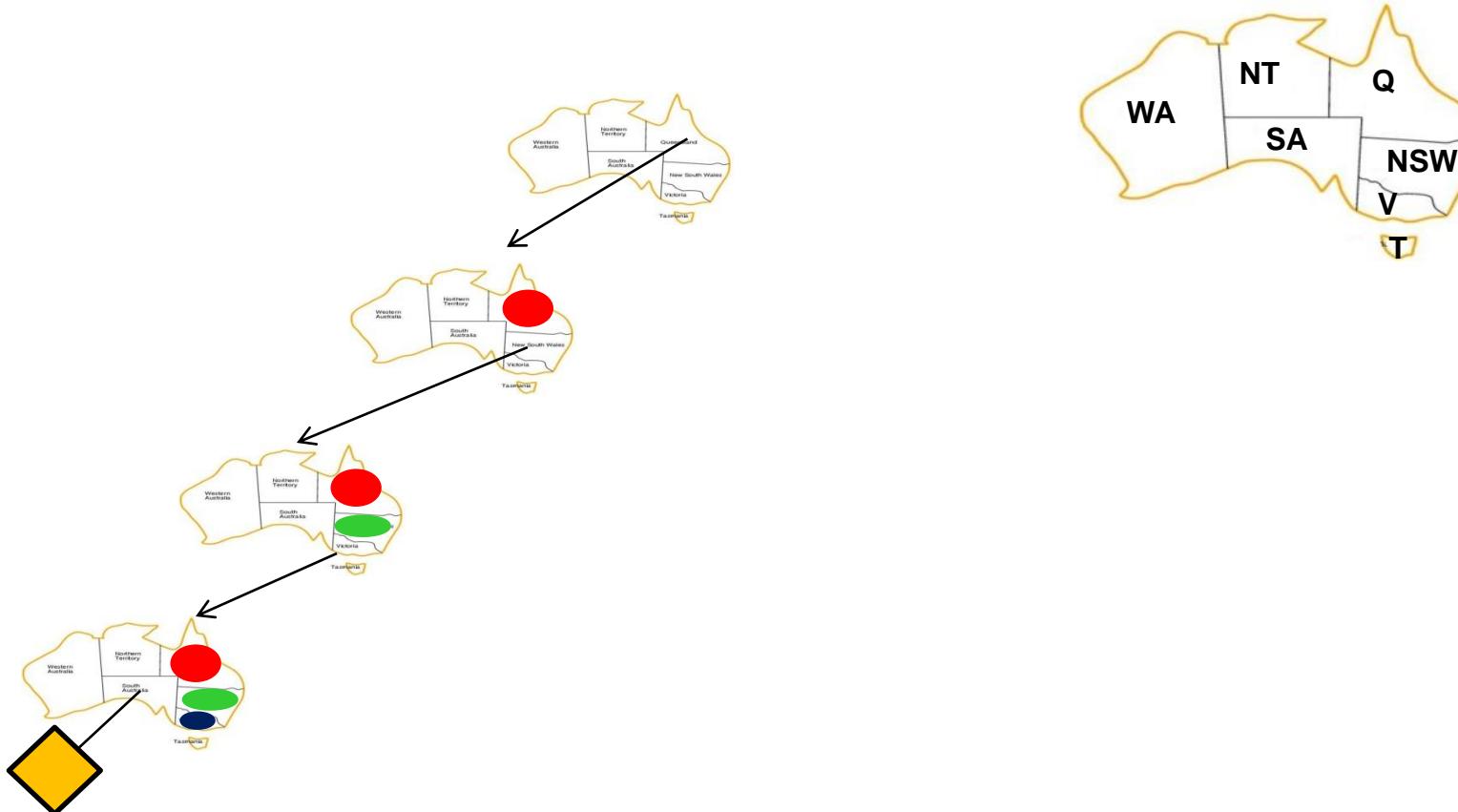
Backtracking Example



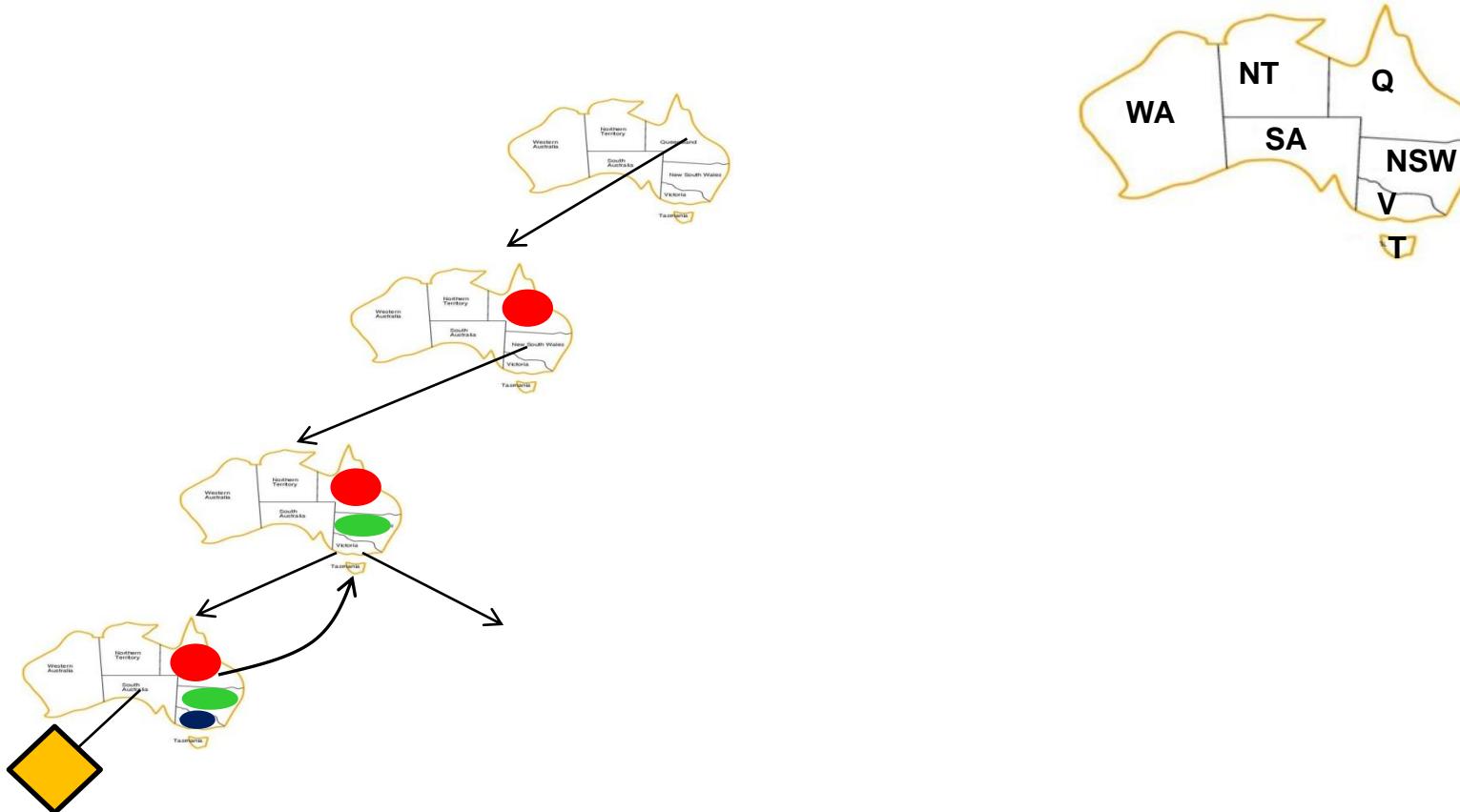
Backtracking Example



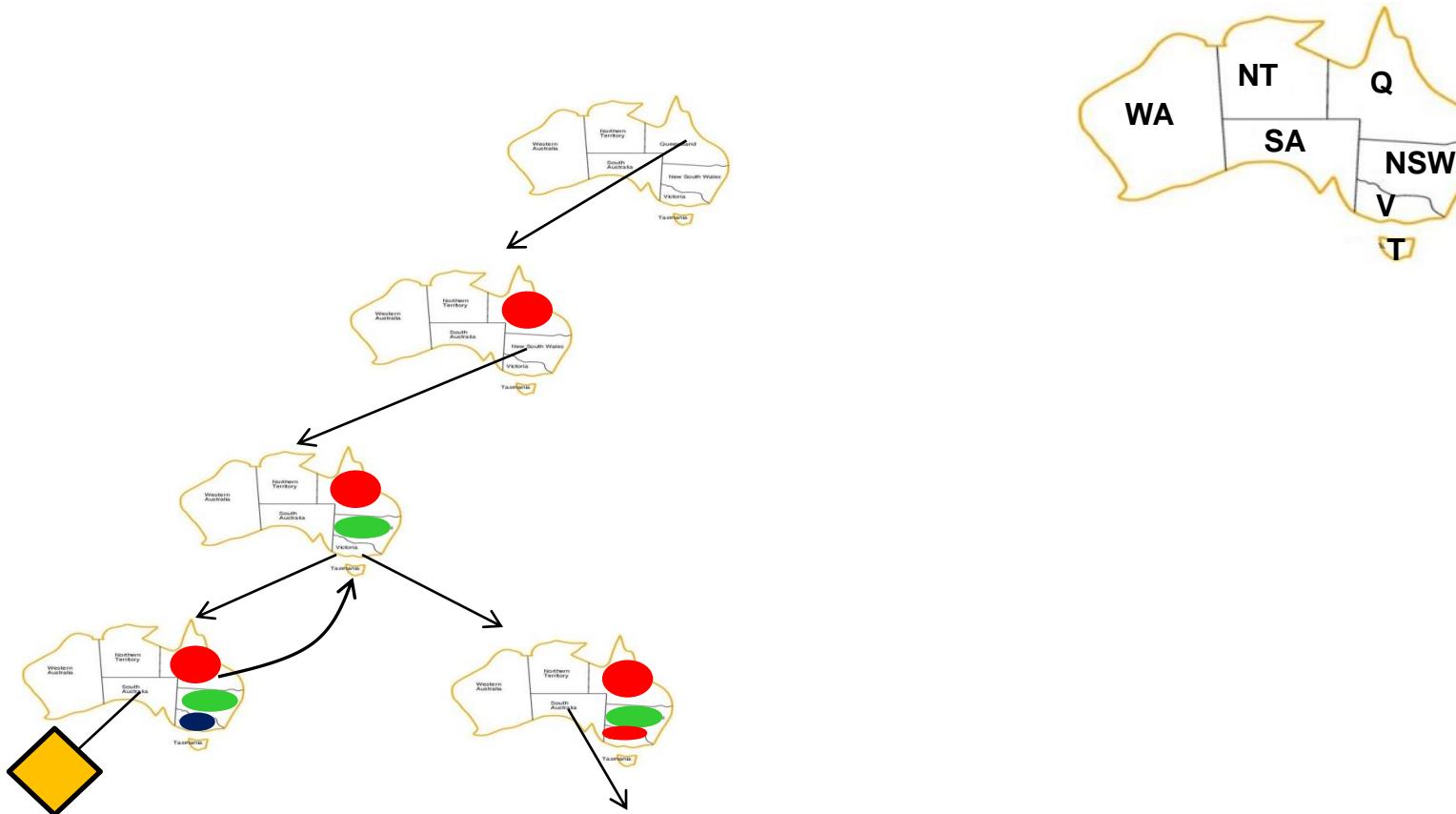
Backtracking Example



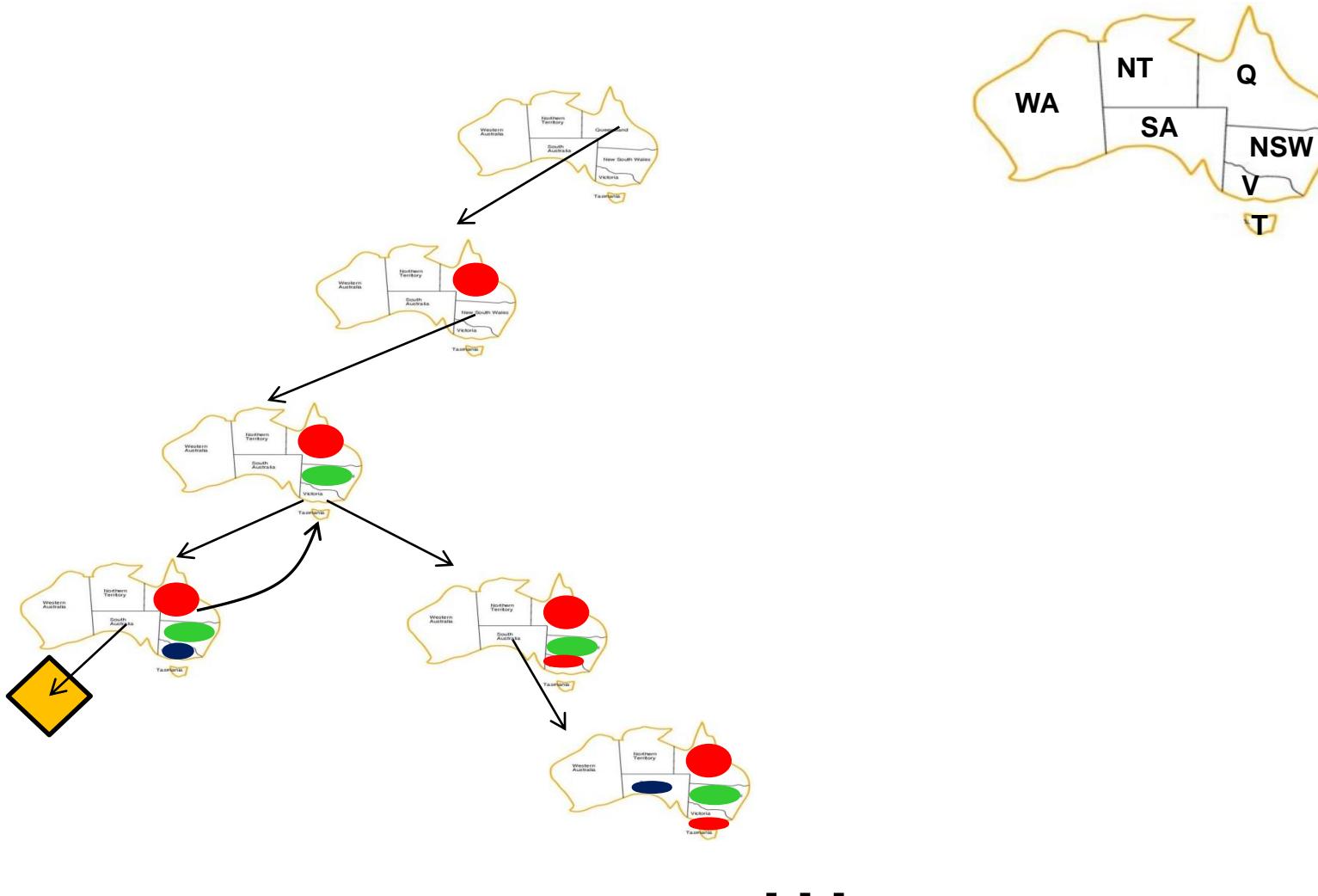
Backtracking Example



Backtracking Example



Backtracking Example

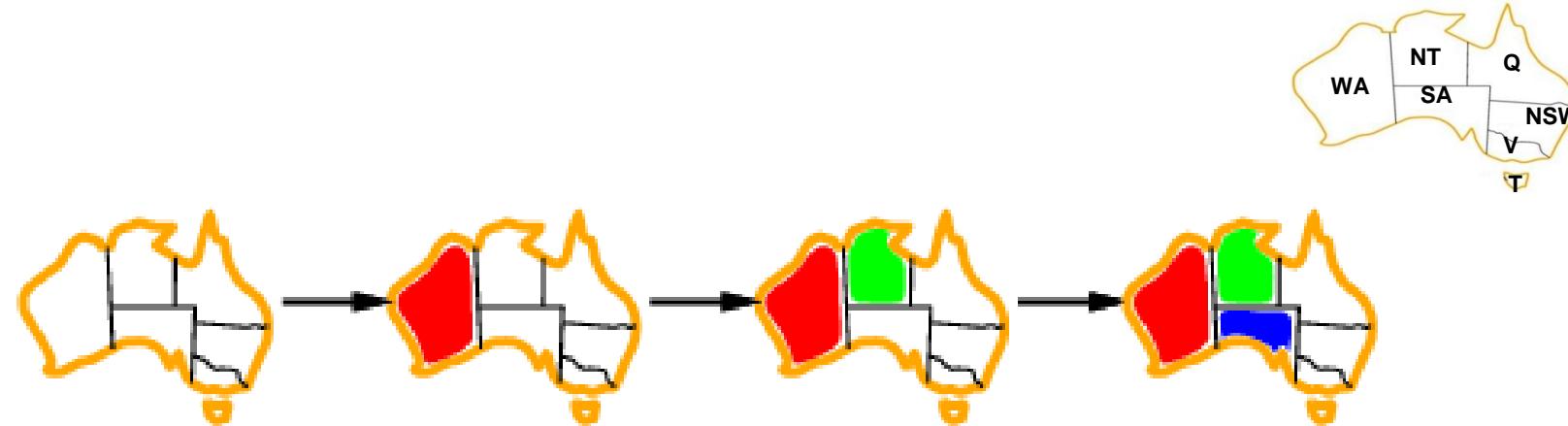


Backtracking Algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)
function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, value)
      if inferences  $\neq$  failure then
        add inferences to csp
        result  $\leftarrow$  BACKTRACK(assignment, csp)
        if result  $\neq$  failure then
          return result
      remove {var = value} and inferences from assignment and csp
  return failure
```



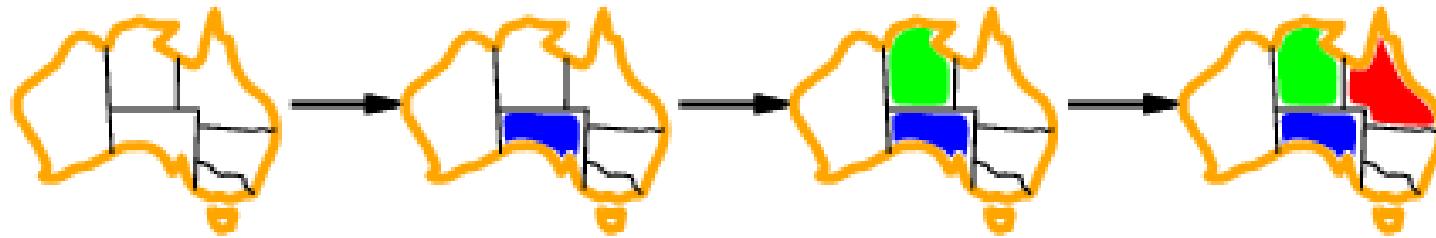
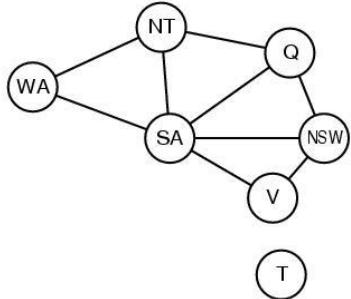
Select-Unassigned-Variable



- Minimum remaining values (MRV)
- Rule: choose variable with the **fewest legal values**

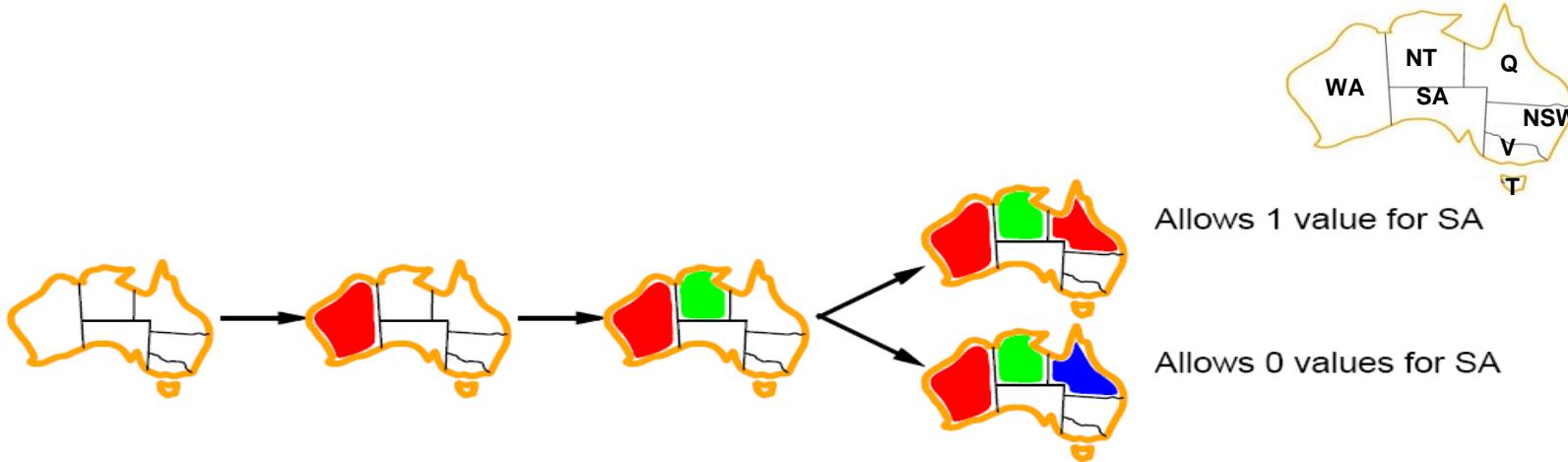


Select-Unassigned-Variable



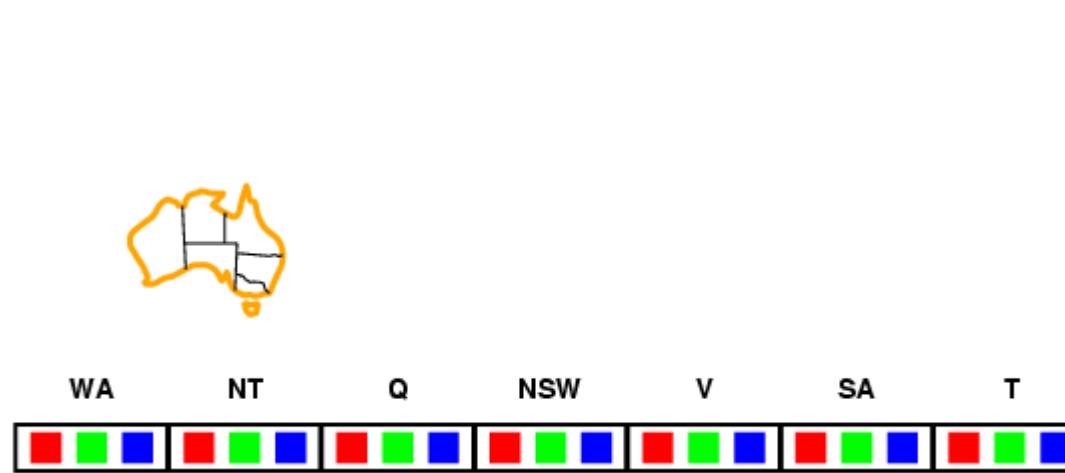
- Degree heuristic
- Rule: select variable that is involved in the largest number of constraints on other unassigned variables.
- Degree heuristic is very useful as a tie breaker: use MRV break ties with degree heuristic

Order-Domain-Values



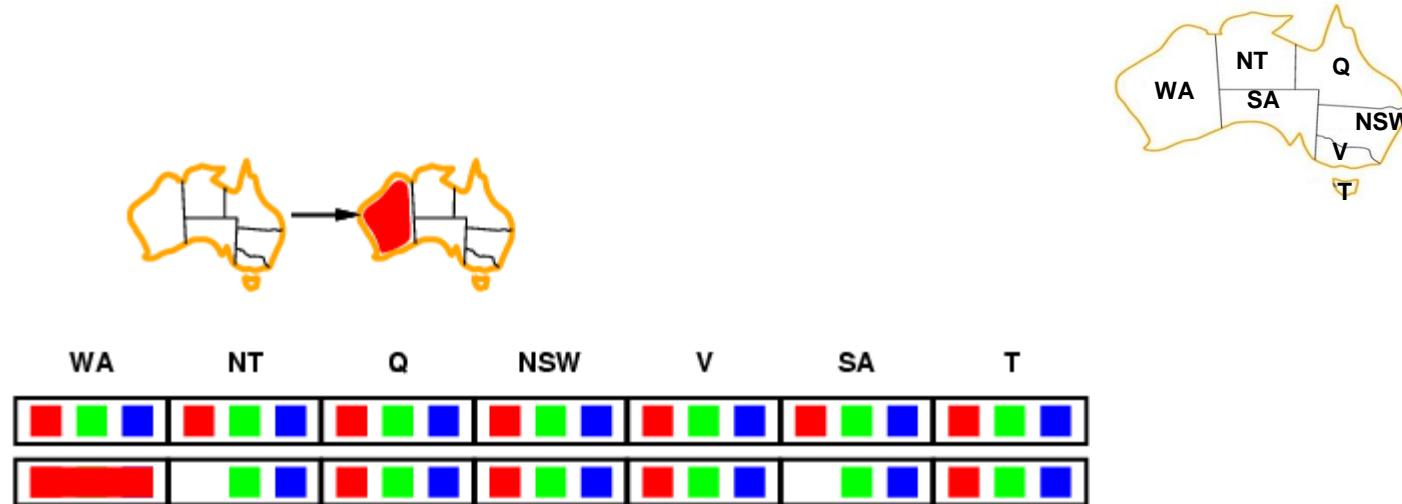
- Least constraining value heuristic
- Rule: given a variable choose the least constraining value i.e., the one that leaves the maximum flexibility for subsequent variable assignments.

Forward Checking



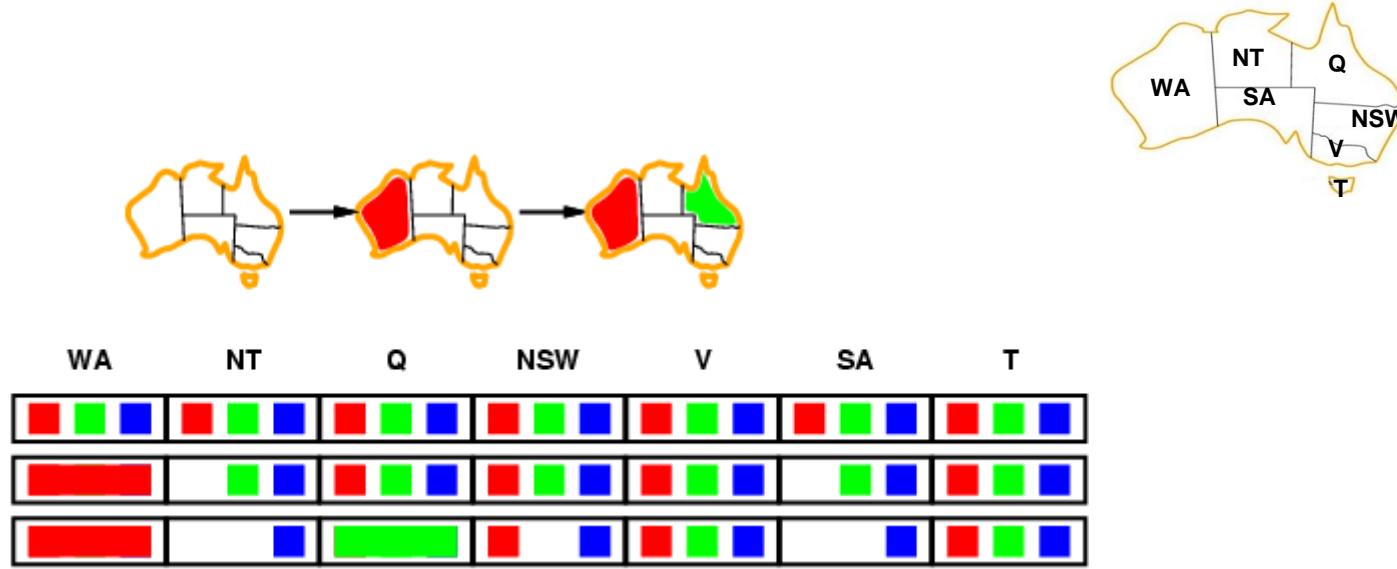
- **Forward checking:** Whenever a value v is assigned to a variable X_i , make all variables **consistent with this assignment**.
- Terminates search when any variable has no legal values.

Forward Checking



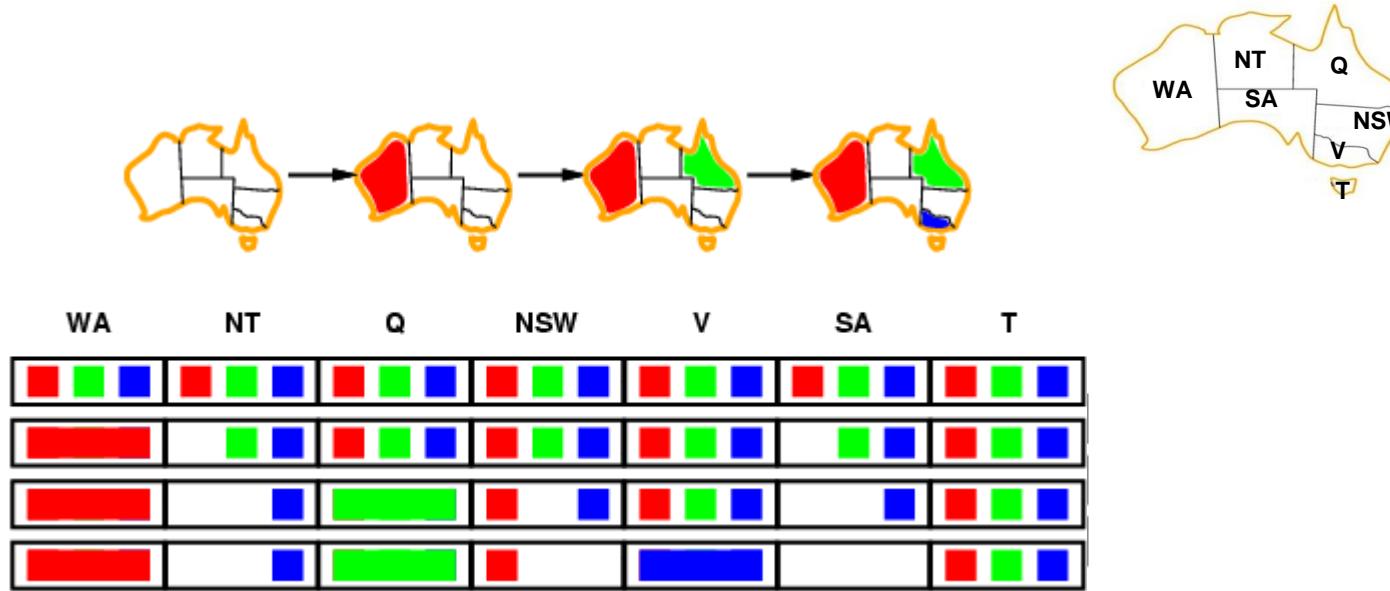
- **Forward checking:** Whenever a value v is assigned to a variable X_i , make all variables **consistent with this assignment**.
- Terminates search when any variable has no legal values.

Forward Checking



- **Forward checking:** Whenever a value v is assigned to a variable X_i , make all variables **consistent with this assignment**.
- Terminates search when any variable has no legal values.

Forward Checking



- **Forward checking:** Whenever a value v is assigned to a variable X_i , make all variables **consistent with this assignment**.
- Terminates search when any variable has no legal values.

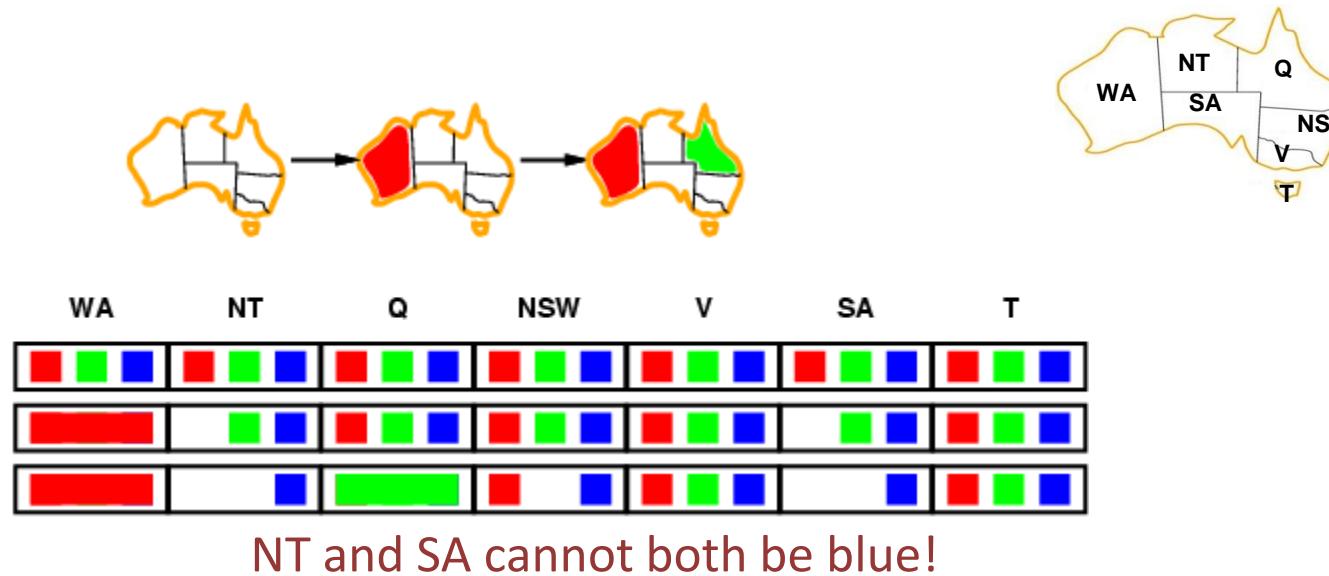
Forward Checking Algorithm

```
function FORWARD-CHECKING-SEARCH(csp) returns a solution or failure
    return RECURSIVE-FORWARD-CHECKING({ },csp)
function RECURSIVE-FORWARD-CHECKING(assignment,csp) returns a solution or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment do
            add{var=value} to assignment
            inferences  $\leftarrow$  remove all domain values of unassigned variables
                inconsistent with {var = value}
            if inferences $\neq$ failure then
                add inferences to assignment and update csp
                result $\leftarrow$ RECURSIVE-FORWARD-CHECKING(assigment, csp)
                if result  $\neq$ failure then
                    result
                    remove {var=value} and inferences from assignment and csp
    return failure
```

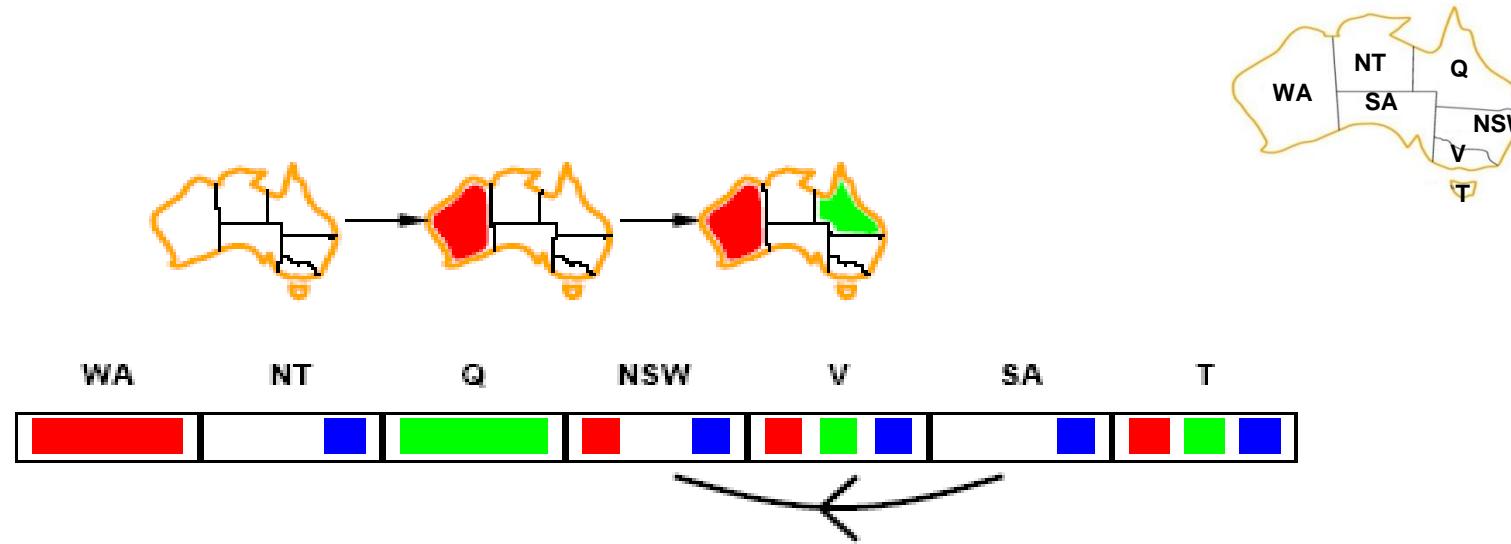


Forward Checking

- **Forward checking** propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

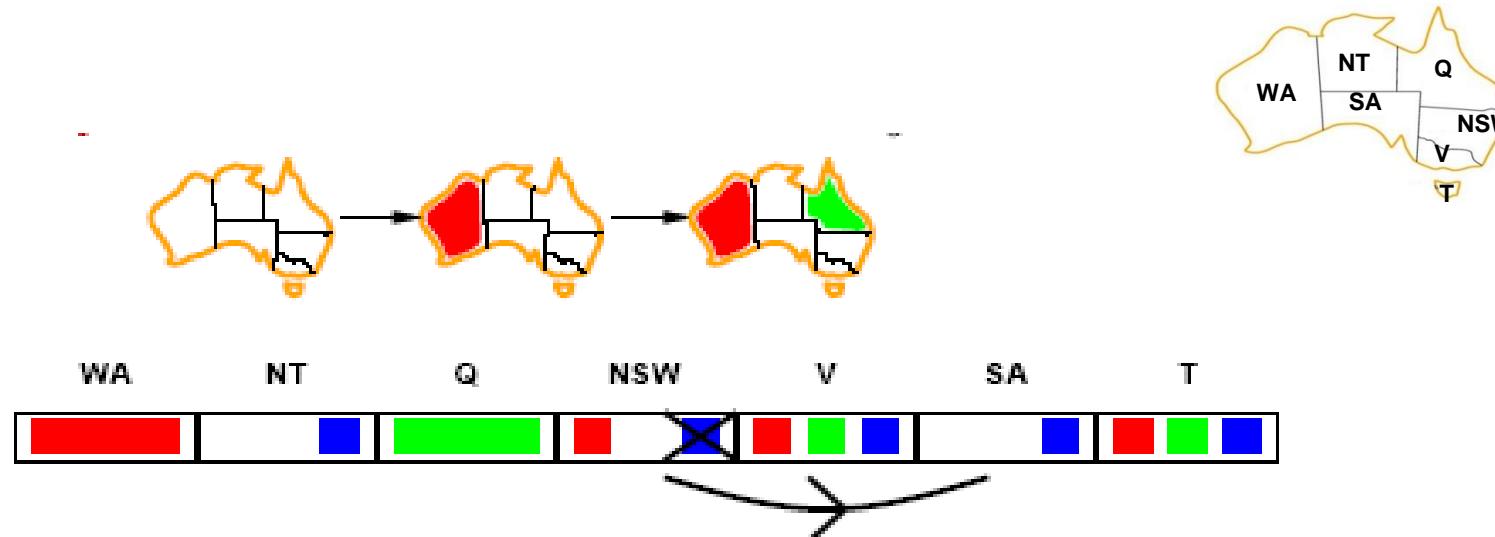


Arc Consistency



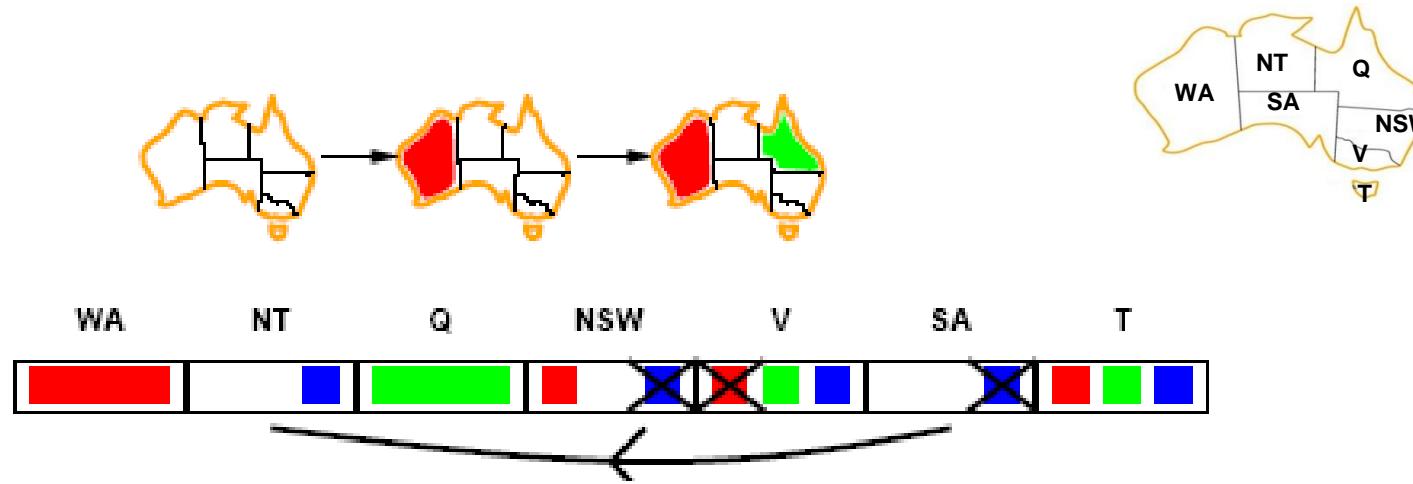
- $X \rightarrow Y$ is consistent iff for **every** value u_i of X there is some allowed v_j value in Y

Arc Consistency



- $X \rightarrow Y$ is consistent iff for **every** value u_i of X there is some allowed v_j value in Y

Arc Consistency



- $X \rightarrow Y$ is consistent iff for **every** value u_i of X there is some allowed v_j value in Y .
- Arc consistency detects failure earlier than FC

MAC

```
function MAC-SEARCH(csp) returns a solution or failure
  run AC-3(csp)
  return RECURSIVE-MAC({ },csp)
```

```
function RECURSIVE-MAC(assignment,csp) returns a solution or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment do
      add {var=value} to assignment
      inferences  $\leftarrow$  AC-3(csp)
      if inferences  $\neq$  failure then
        add inferences to assignment and update csp
        result  $\leftarrow$  RECURSIVE-MAC(assignment, csp)
        if result  $\neq$  failure then
          result result
          remove {var=value} and inferences from assignment and csp
  return failure
```



Assume all constraints binary or unary
END



Global Constraints

Definition

- Depends on arbitrary number of variables

Generalized Arc Consistency (GAC)

- For each value in domain of X_i , there exists a valid assignment for all the remaining variables in the constraint
- $x,y,z \in \{1,2,3\}$, $[x + y = z]: \{(1,1,2), (1,2,3), (2,1,3)\}$
GAC : $x \in \{1,2\}$, $y \in \{1,2\}$, $z \in \{2,3\}$



Alldifferent

- Alldifferent: all $X_i \in$ scope of constraint are assigned to different values.
e.g. $X_1, \dots, X_4 \in \{0, 1, 2, 3, 4\}$ $Alldiff(X_1, X_2, X_3, X_4)$

$X_1=0, X_2=1, X_3=2, X_4=3$ ✓

$X_1=0, X_2=1, X_3=1, X_4=2$ ✗

- How to represent Alldifferent with binary \neq constraints?



What are the AllDiff constraints of Sudoku?

			2	6		7		1
6	8			7			9	
1	9				4	5		
8	2		1				4	
		4	6		2	9		
5				3		2	8	
		9	3			7	4	
4				5		3	6	
7		3		1	8			

Simple approximation to AllDiff GAC

1. If $\#values < \#vars$ then return failure

$X_1=0, X_2 \in \{1, 2\}, X_3 \in \{2, 3\}, X_4=3$ ✓

$X_1 \in \{1, 2\}, X_2 \in \{1, 2\}, X_3 \in \{1, 2\}, X_4=3$ ✗

2. Remove all vars with singleton domains

$X_1=0, X_2 \in \{1, 2\}, X_3 \in \{2, 3\}, X_4=3 \rightarrow X_2 \in \{1, 2\}, X_3 \in \{2, 3\}$

3. Remove singleton values from remaining domains

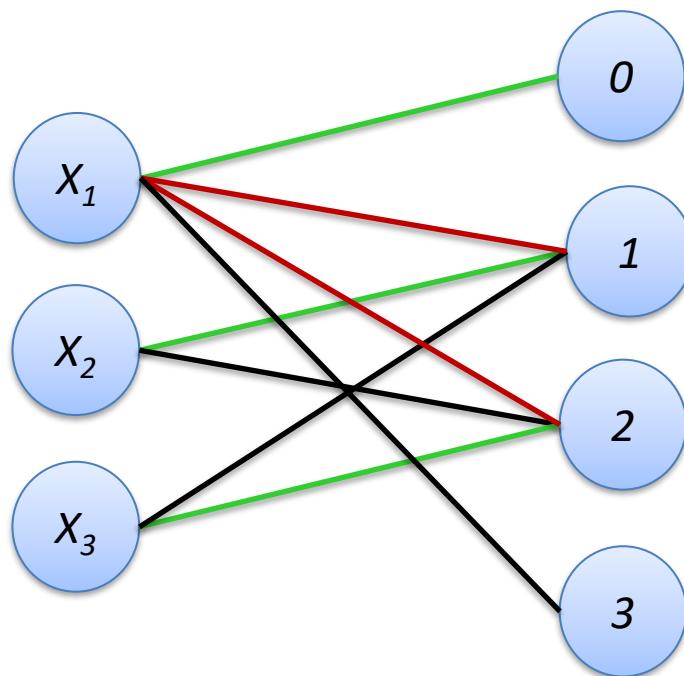
$X_2 \in \{1, 2\}, X_3 \in \{2, 3\} \rightarrow X_2 \in \{1, 2\}, X_3 = 2$

4. Goto 1



Fast Computation of AllDiff GAC

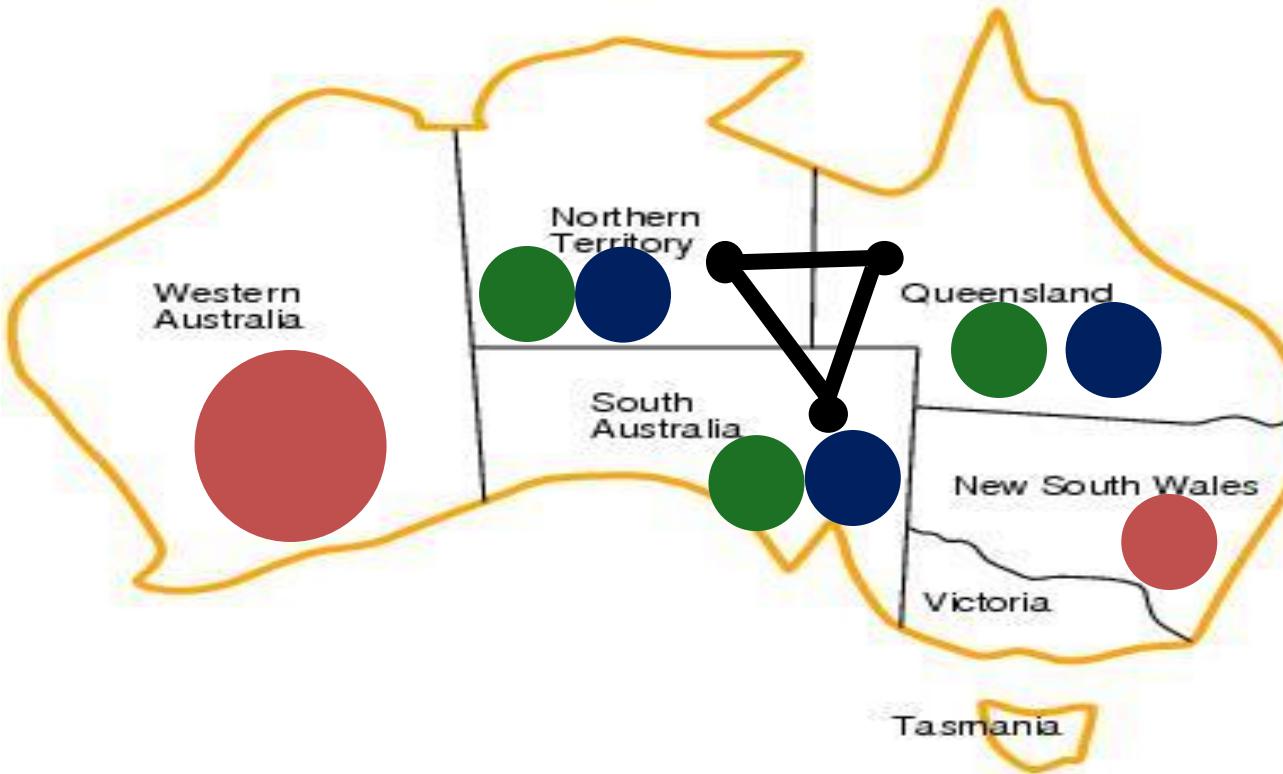
- GAC = Find all max matchings in a bipartite graph
 - e.g. $X_1 \in \{0, 1, 2, 3\}$, $X_2 \in \{1, 2\}$, $X_3 \in \{1, 2\}$



Can be shown to be possible in $O(d\sqrt{n})$

Where AC-3 on $O(n^2)$
≠ constraints takes $O(n^2 d^3)$

AllDiff Stronger than binary constraints



Inconsistency detected by AllDiff GAC,
but not by AC-3 on \neq constraints!

Constraint propagation systems

World's best 08-12 **Gecode** (SAP configurator 2017)

- Constraint Store
- Propagators
- Propagator Loop
- Search



Constraint store

$$x \in \{3,4,5\} \quad y \in \{3,4,5\}$$

- Maps variables to possible values



Constraint store

finite domains

$x \in \{3,4,5\}$ $y \in \{3,4,5\}$

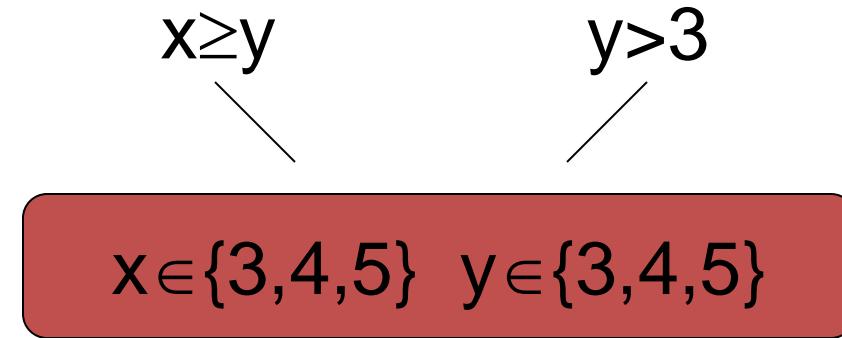
- Maps variables to possible values
- Others: finite sets, continuous domains, trees, ...

Constraints and Propagators

- Extensive use of global constraints
- Propagators **implement** constraints!
 - prune values in conflict with constraint
- Constraint propagation drives propagators for several constraints

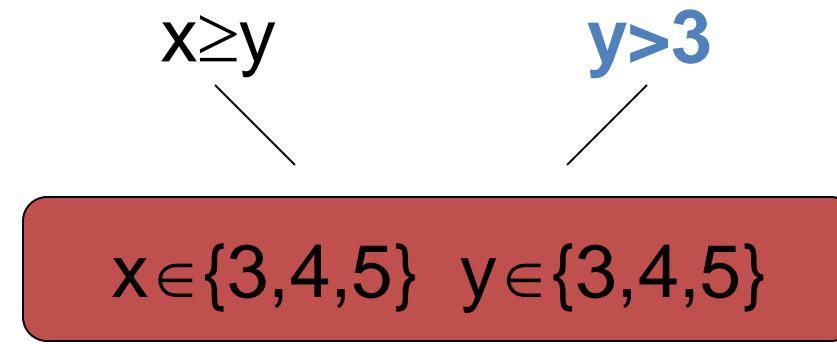


Propagators



- Amplify store by constraint propagation

Propagators



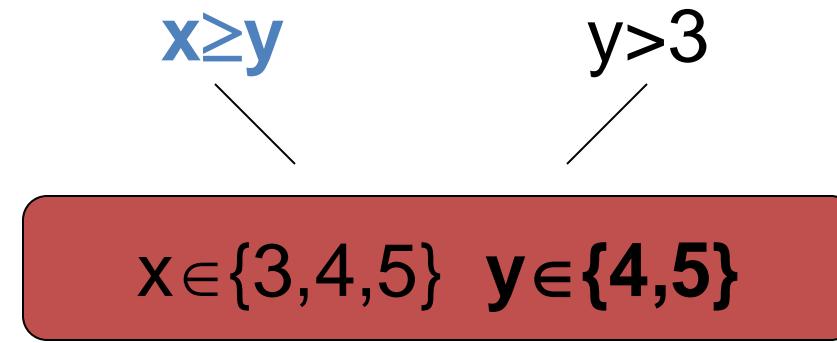
- Amplify store by constraint propagation

Propagators



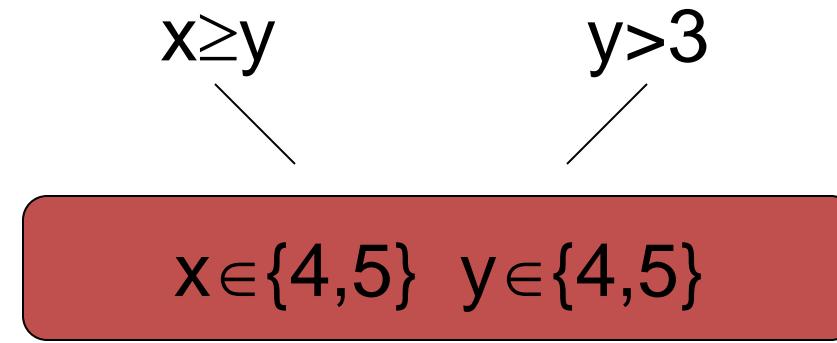
- Amplify store by constraint propagation

Propagators



- Amplify store by constraint propagation

Propagators



- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
 - no more propagation possible

Propagators

 $x \geq y$ $x \in \{4,5\} \quad y \in \{4,5\}$

- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
 - no more propagation possible

Propagation loop

```
function PROPAGATE(csp)
     $Q \leftarrow \{c_1, c_2, \dots, c_n\}$                                 // Queue of propagators from csp
    while  $Q \neq \{\}$  do
         $c \leftarrow \text{REMOVE-FIRST}(Q)$ 
        EXECUTE(c)                                              // run propagator, prune domains of variables
        if any domain in csp is empty then
            return false
        for each  $X_i \in \text{SCOPE}(c)$  s.t.  $D_i$  was narrowed do
            ADD(GET-PROPS( $X_i$ ) - c) to Q                  //Add new propagators
        end for
    end while                                                       // Fixed point reached!!
    return true
```

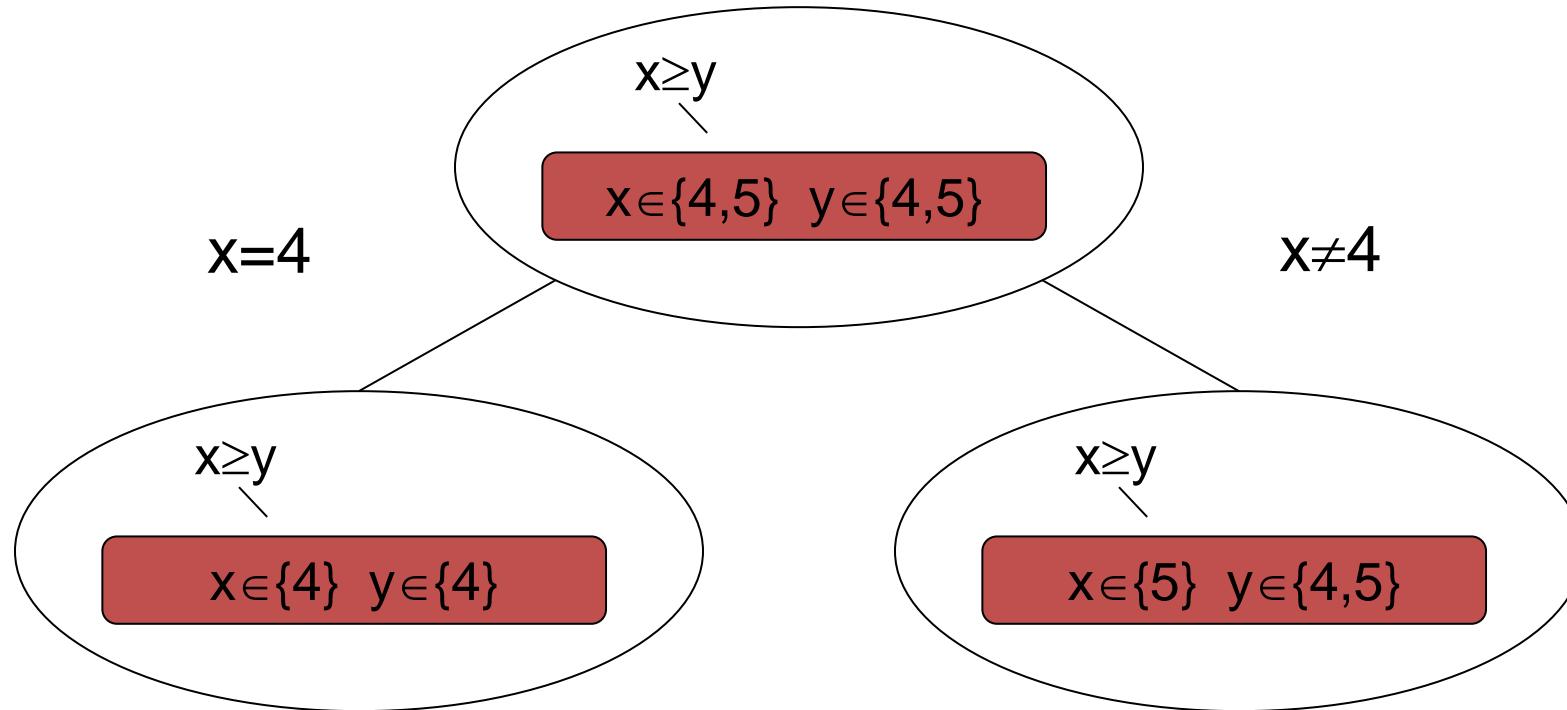


Search

- Propagation **alone** is not enough!
- Search: Explore search tree for solutions (Backtracking)
- Branching: Select variable and value (define search tree)
 - Minimum remaining values
 - Degree
 - Least constraining value



Search: Branching



- Create subproblems with additional information
- Enable further constraint propagation

Constraint propagation systems

- Each node in the search tree has different information.
- Constraint store and propagators are encapsulated in a **Space**.
- Manipulate spaces: Copy , add new constraints, ask for solutions, etc.



Search algorithm

function SEARCH(csp) **returns** solution or failure

```
 $csp \leftarrow \text{PROPAGATE}(csp)$ 
if  $csp$  is failure then return failure
if  $csp$  is solved then return SOL( $csp$ )
 $csp' \leftarrow csp$ 
ADD(  $csp'$ , BRANCH( $csp', 1$ ))
 $csp'' \leftarrow csp$ 
ADD(  $csp''$ , BRANCH( $csp'', 2$ ))
 $solution \leftarrow \text{SEARCH}(csp')$ 
if  $solution$  is valid then return  $solution$ 
else return SEARCH(  $csp''$ )
```

function BRANCH($csp, nbranch$)

return the constraint representing the $nbranch$ according to variable
and value heuristic selection.



Introduction to Artificial Intelligence

Lecture 9: Local Search (Meta-Heuristics)



Local Search: Overview & Motivation

- Solves Constraint Optimization Problems (COPs)
- Makes small changes to a **complete solution**
 - Has low memory consumption
 - Objective value is known
 - Anytime algorithm
- Effective at solving large optimization problems
- Easy to implement real-world constraints
- Widely used in practice
- However:
 - Incomplete method, i.e. no guarantees of optimality

Today's Program

1. Local Search Basics
2. Meta Heuristics
 1. Hill Climbing
 2. Simulated Annealing
 3. Tabu Search
 4. Genetic Algorithms
 5. Constraint Based Local Search
 6. Advanced Methods

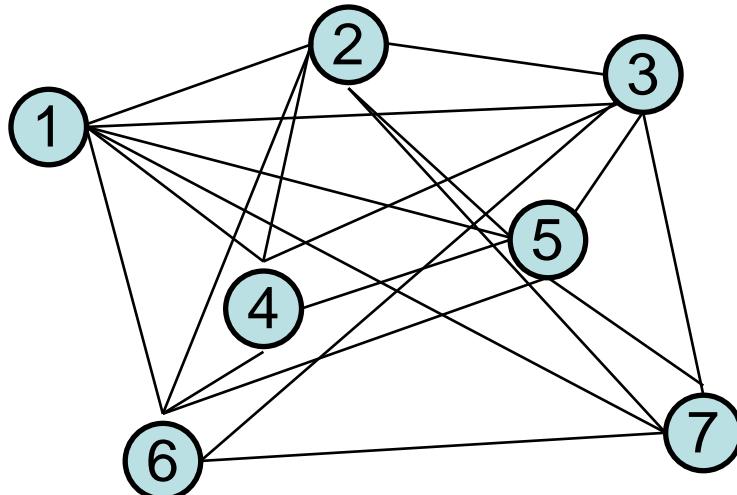
Local Search Algorithm Basics

1. Begin with a complete assignment to variables using a **constructive heuristic**.
 - (A possibly infeasible solution to the problem)
2. **Search** by moving to other **local** complete assignments.
 - (Explore the “neighborhood”)
3. Repeat the previous step until the assignment is “Good enough” wrt. feasible and/or optimal.
 - (Termination condition)



Traveling Salesman Problem (TSP)

- Given: A fully connected undirected graph.
 - Edge costs: distance between nodes



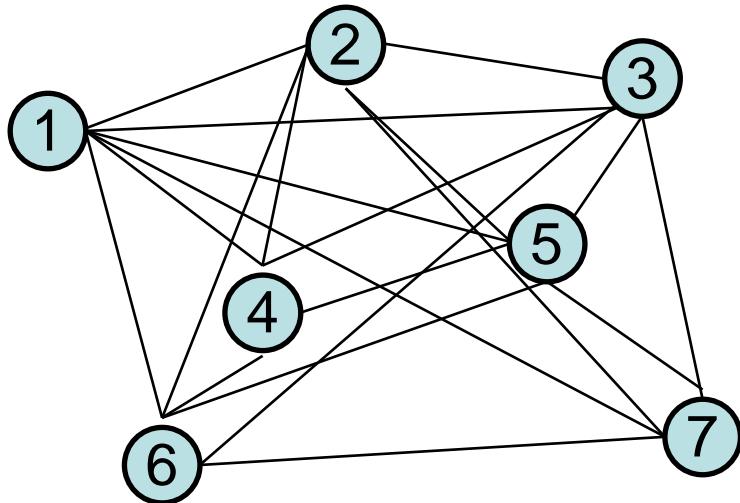
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Task: Find the minimum cost path that visits all nodes and returns to the start.



TSP: Initial Solution

- Local searches need an initial solution.
- We can store a TSP solution as a permutation.



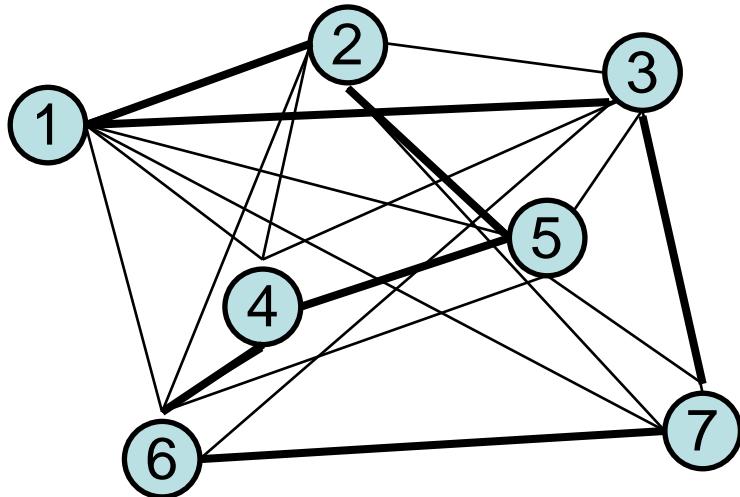
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- How would you construct an initial solution?



TSP: Initial Solution

- Nearest neighbor heuristic



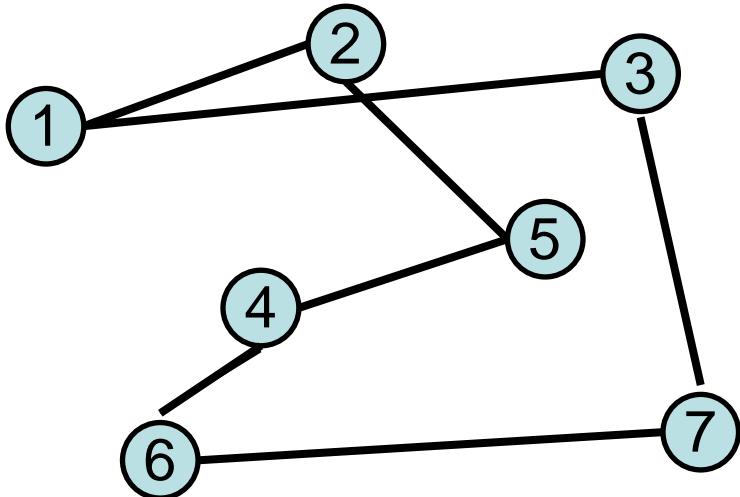
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Initial solution: 1 2 5 4 6 7 3
- Cost: 27.4



TSP: Neighborhoods

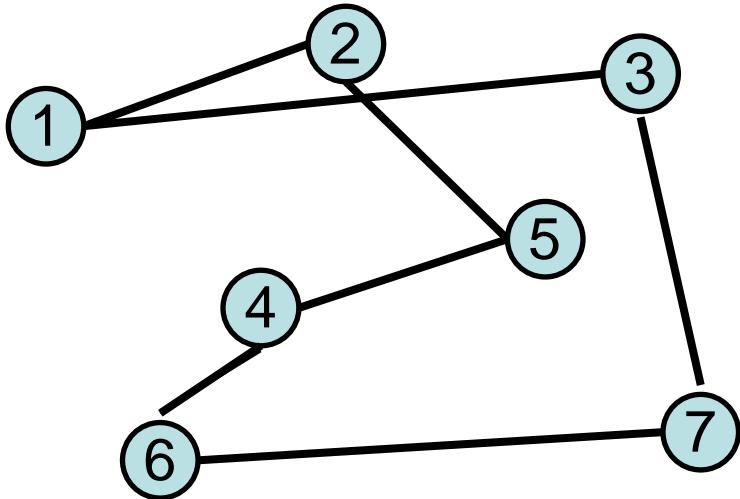
- How can we improve our initial solution?



	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

TSP: 2-Opt Neighborhood

- 2-Opt removes 2 edges that cross each other and replaces them with non-crossing edges.



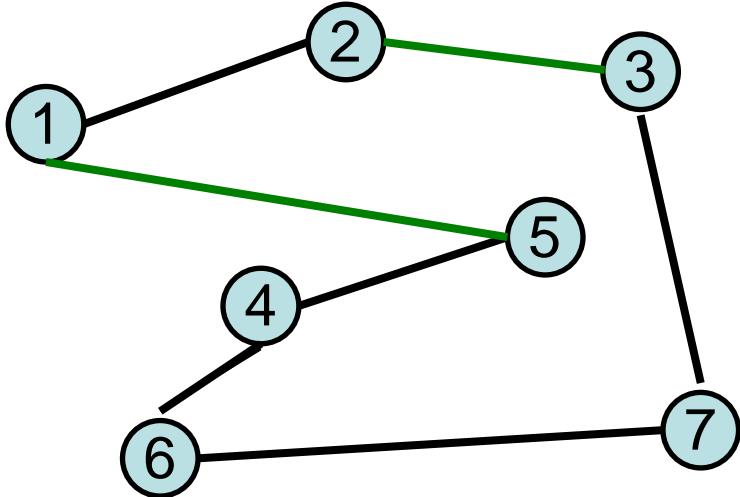
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Which edges should we pick to remove?



TSP: 2-Opt Neighborhood

- 2-Opt removes 2 edges that cross each other and replaces them with non-crossing edges.



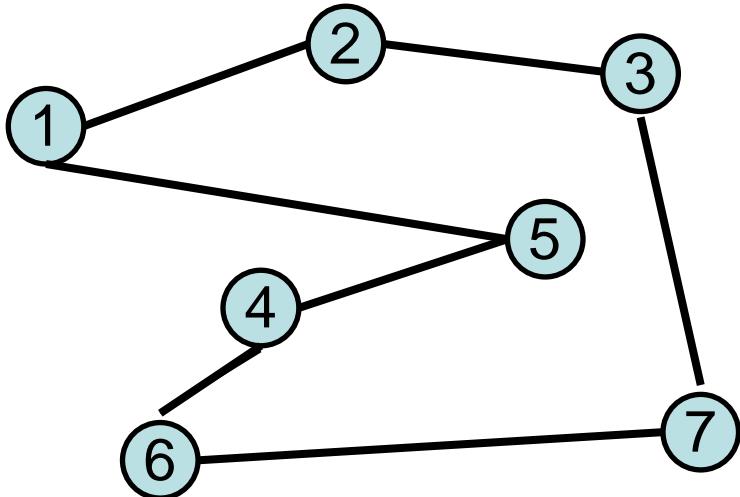
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- New solution: 1 2 3 7 6 4 5
- New cost: 26.7 (previous cost: 27.1)



TSP: k -Opt Neighborhood

- Remove k edges and repair the path.
- Lets try $k=3$

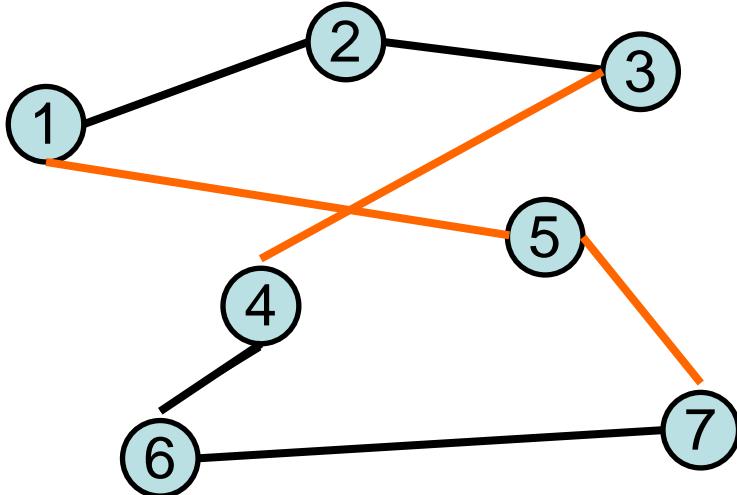


	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0



TSP: k -Opt Neighborhood

- Remove k edges and repair the path.
- Lets try $k=3$



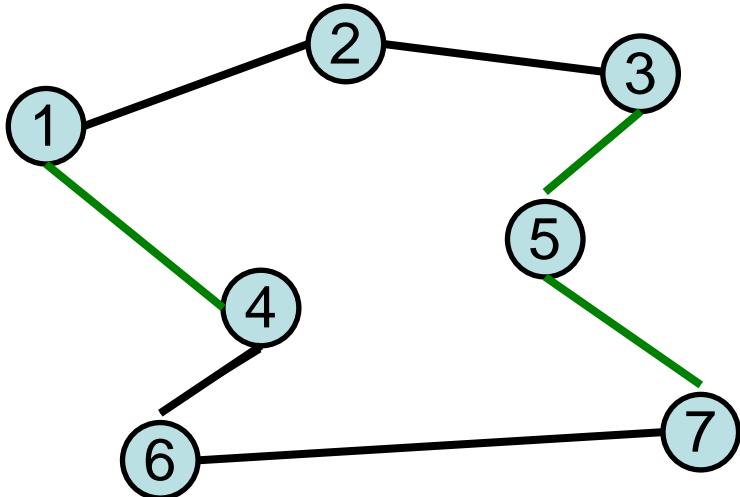
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Possible solution: 1 2 3 4 6 7 5
- Cost: 27.1



TSP: k -Opt Neighborhood

- Remove k edges and repair the path.
- Lets try $k=3$



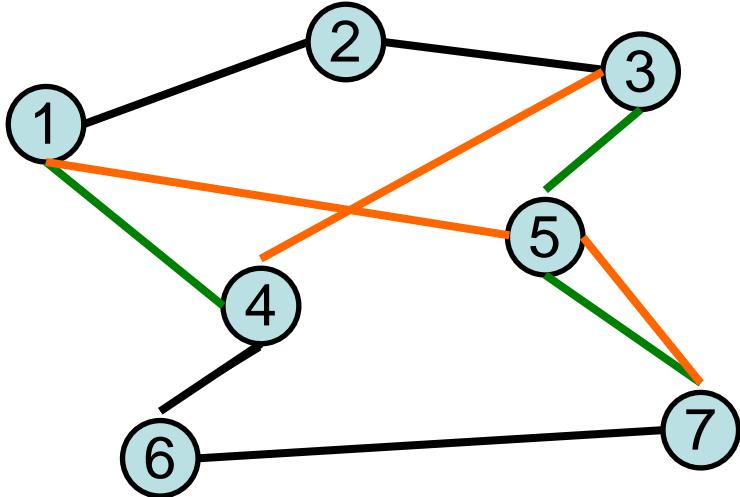
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Possible solution: 1 2 3 5 7 6 4
- Cost: 23.8



TSP: Neighbor Selection

- Which neighbor should we choose?



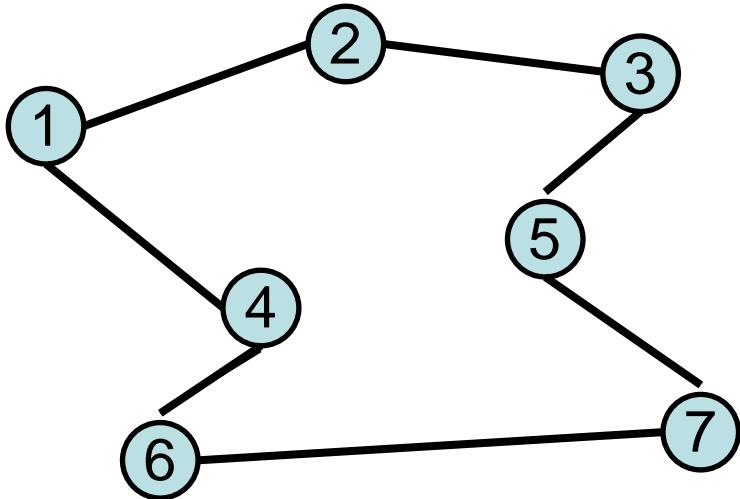
	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- 23.8 vs. 27.1



TSP: Termination

- When should we stop performing improvements?



	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

TSP: Termination

- When should we stop performing improvements?
- Budget based:
 - Number of iterations
 - CPU time
- Solution quality
 - Within $X\%$ of a lower bound
 - Business requirements satisfied
- Convergence criteria:
 - No improvement in last Y iterations.
 - Average improvement below threshold ε

Hill Climbing



Hill Climbing Algorithm

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

current \leftarrow *problem.INITIAL*

while *true* **do**

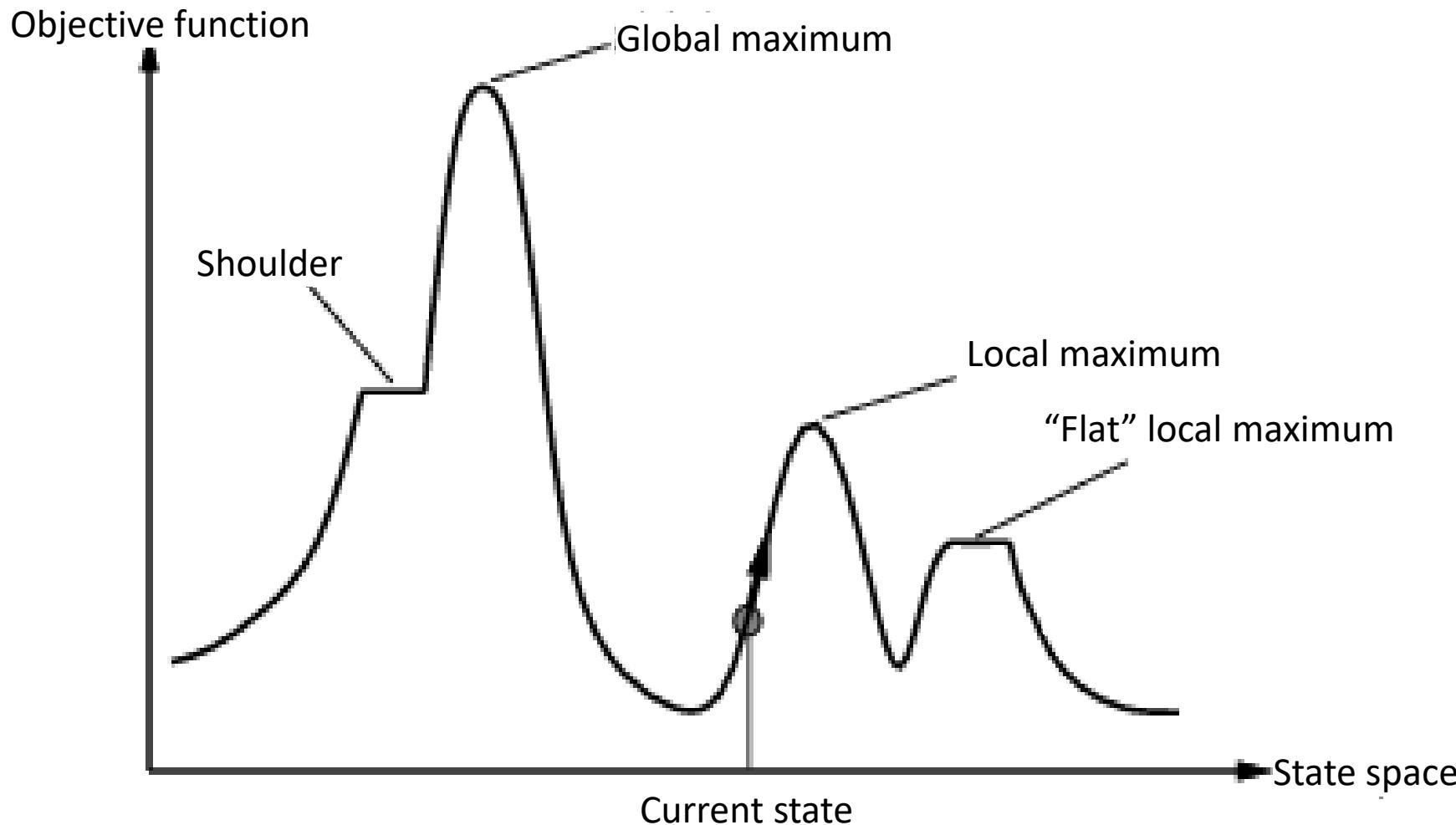
neighbor \leftarrow a highest-value successor state of *current*

if VALUE(*neighbor*) \leq VALUE(*current*) **then return** *current*

current \leftarrow *neighbor*



State-Space Landscape



Hill Climbing: Pro & Con

- Advantages
 - Fast convergence to a local maximum
 - Often results are good (but not optimal) solutions
- Disadvantages
 - Gets stuck in local maxima
 - Gets stuck on shoulders and plateaus



Exploitation vs. Exploration

- **Exploitation** (Intensification)
 - Greedy; always select most improving neighbor
- **Exploration** (Diversification)
 - Also select less improving and non-improving neighbors

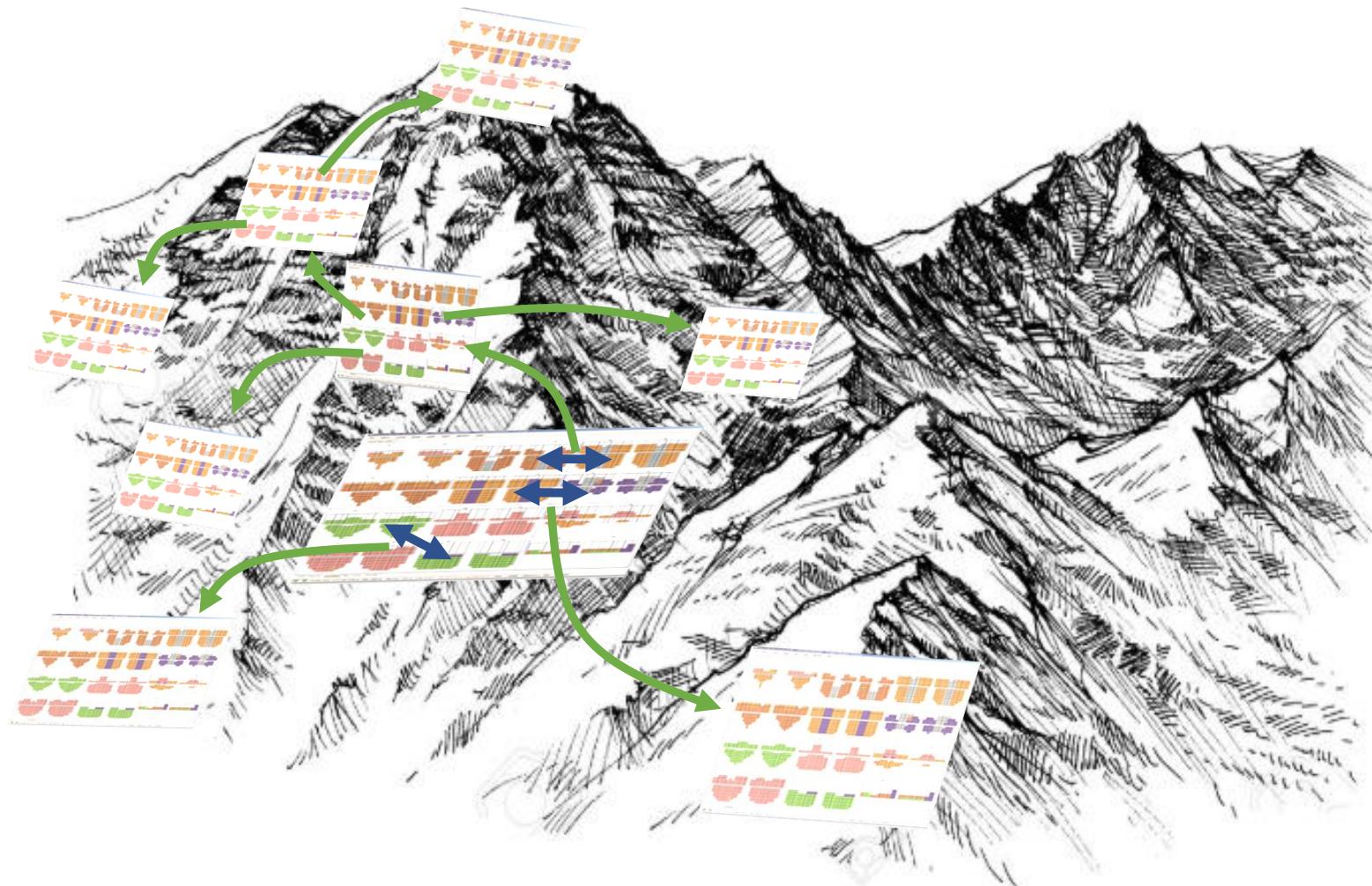


Escaping Local Maxima

- Variations of Hill-Climbing
 - **Sideways move:** allow non-improving moves to traverse plateaus.
 - **Stochastic Hill-Climbing:** random choice of uphill moves.
 - **First-Choice Hill-Climbing:** random generation of neighbors.
 - **Random-restart Hill-Climbing:** restart the search from a different initial state



Example: First-Choice Hill-Climbing



Simulated Annealing



Simulated Annealing



- Inspired by annealing in metals
 - Used to soften metals by heating and then gradually cooling them, allowing atoms to find a low-energy crystalline state.
- Idea:
 - Escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency

Simulated Annealing (obs: minimizing)

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

current \leftarrow *problem.INITIAL*

for $t = 1$ to ∞ **do**

$T \leftarrow \text{schedule}(t)$

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

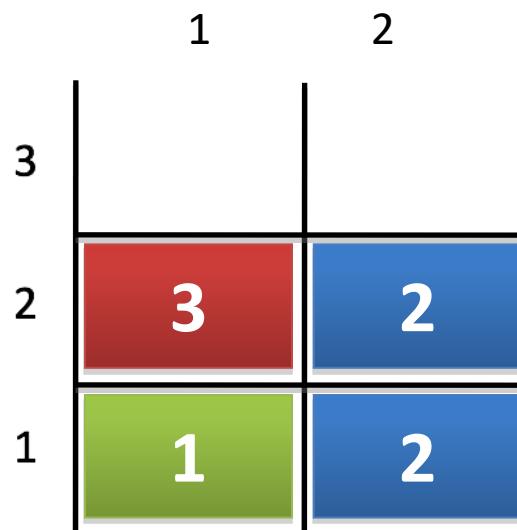
$\Delta E \leftarrow \text{VALUE}(\textit{current}) - \text{VALUE}(\textit{next})$

if $\Delta E \geq 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

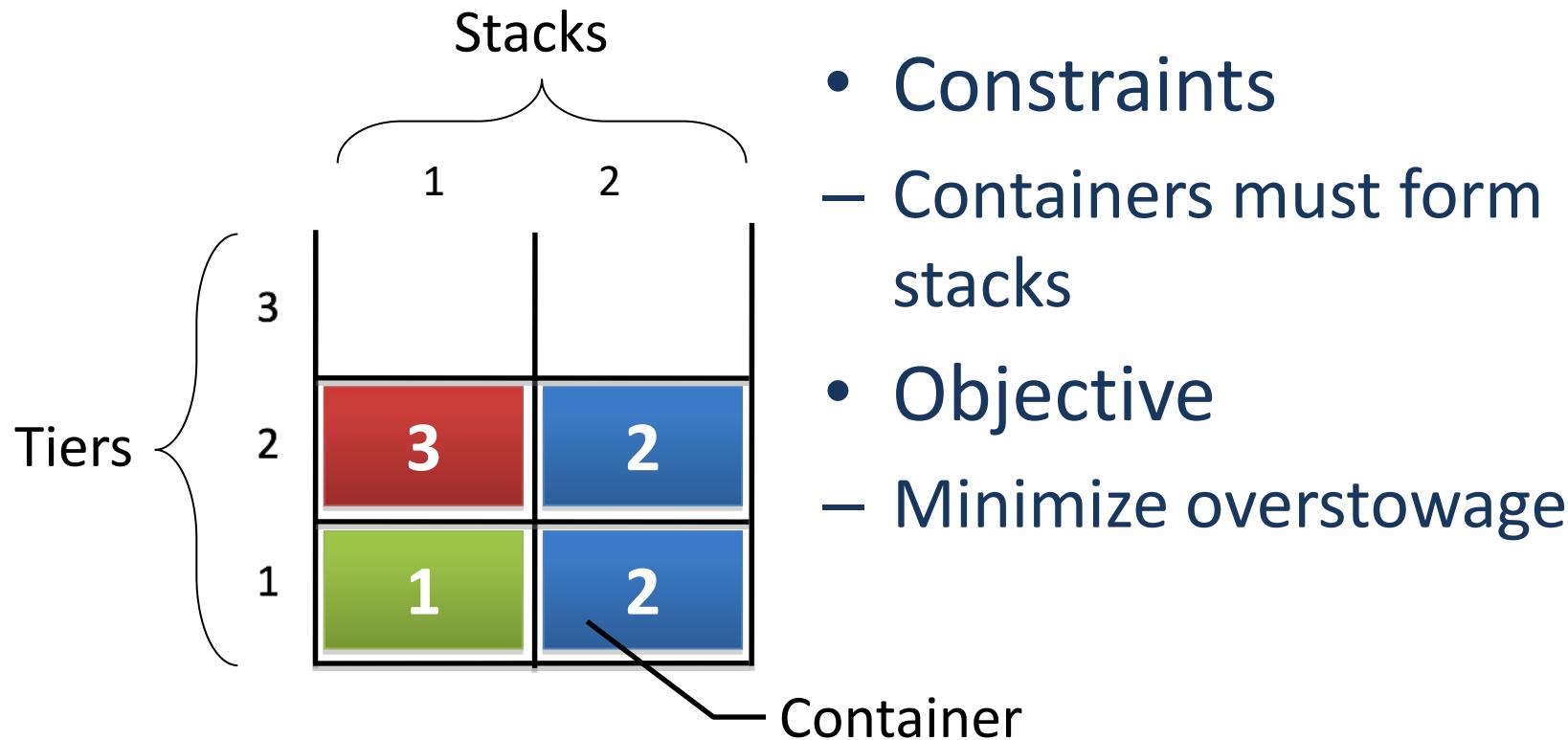


Ex. Container Stowage Problem



Ex. Container Stowage Problem

- Given a feasible container configuration, find the one which minimizes overstowage.



- Constraints
 - Containers must form stacks
- Objective
 - Minimize overstowage

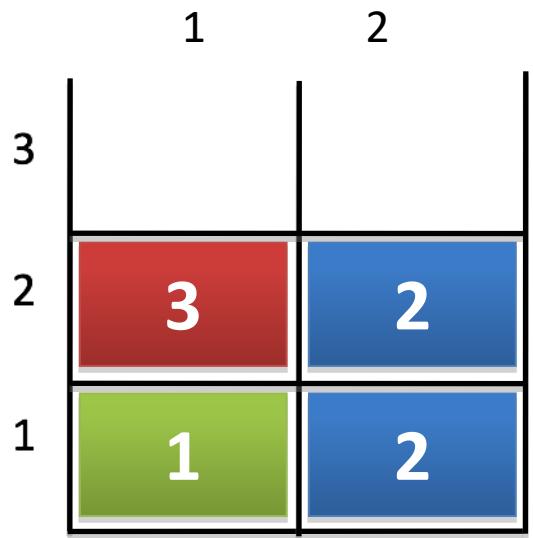


Ex. Container Stowage Problem

- **State:** A container configuration
- **Neighborhood:** Container swaps
- **Objective function (*Value*):** Number of overstowed containers.
- **Termination criteria:** $Value = 0$



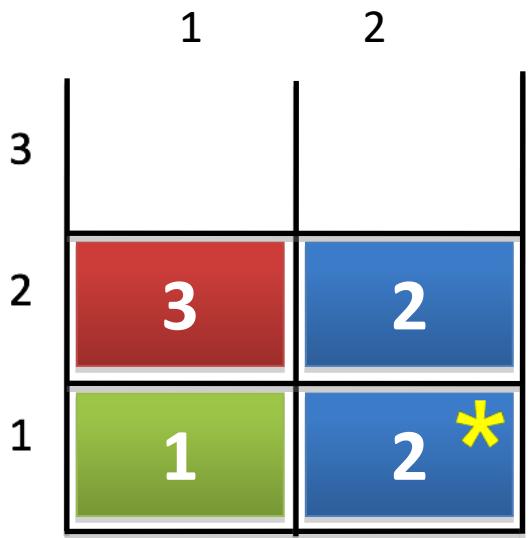
Ex. Container Stowage Problem



Objective: 1

Temperature: 10

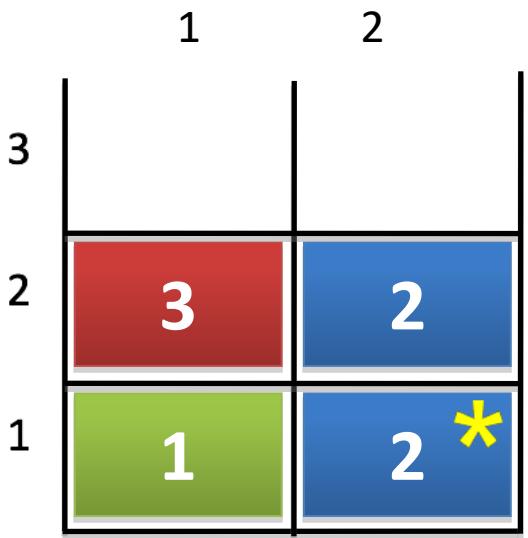
Ex. Container Stowage Problem



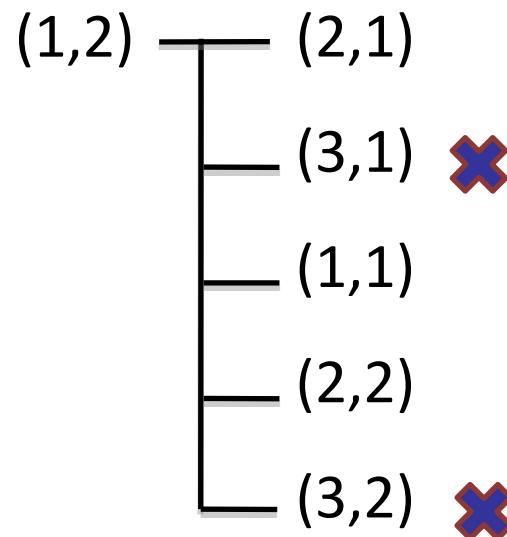
Objective: 1

Temperature: 10

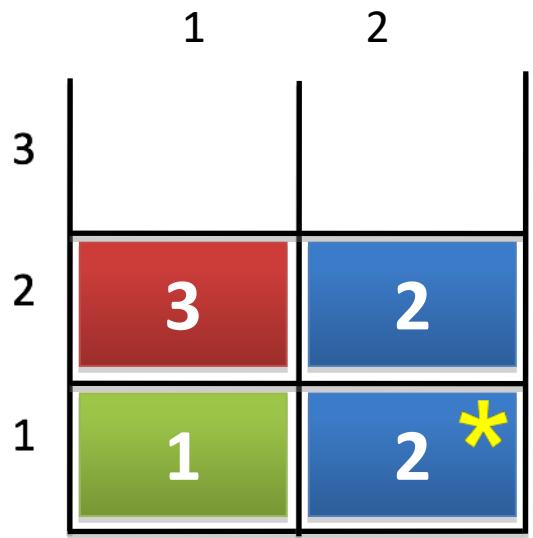
Ex. Container Stowage Problem



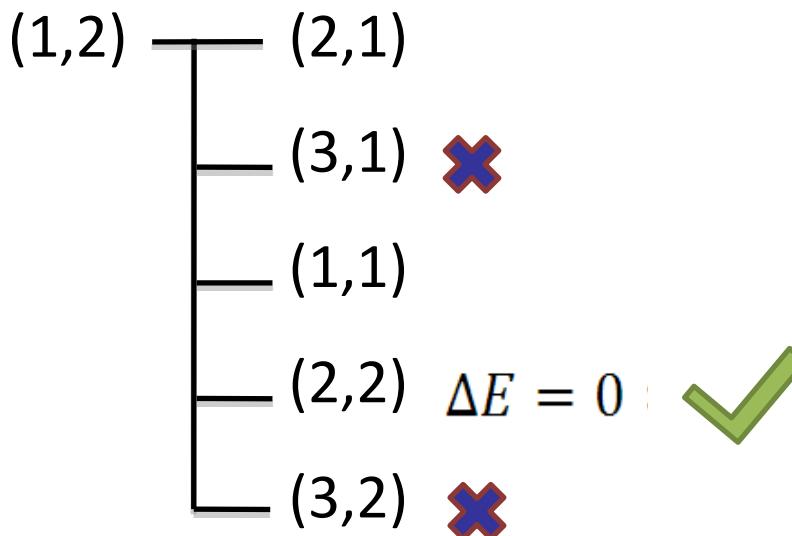
Objective: 1
Temperature: 10



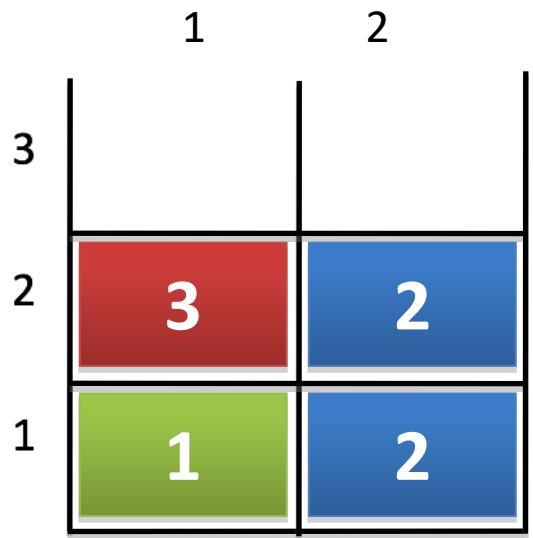
Ex. Container Stowage Problem



Objective: 1
Temperature: 10

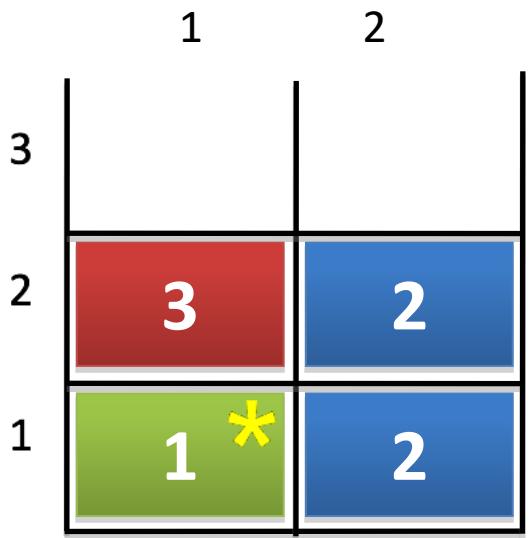


Ex. Container Stowage Problem



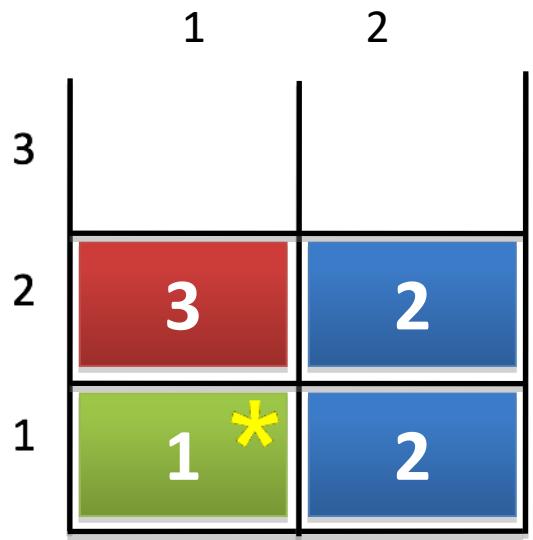
Objective: 1
Temperature: 2

Ex. Container Stowage Problem

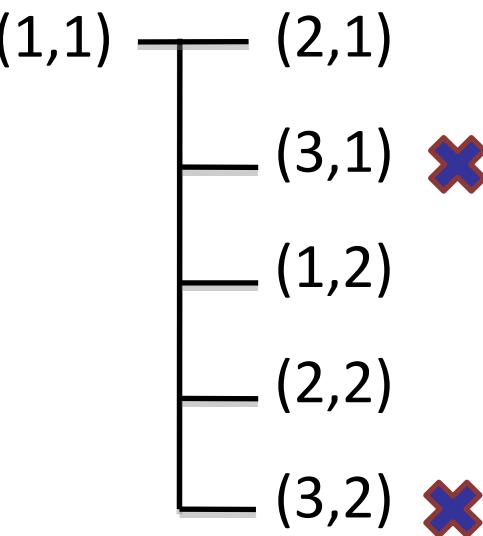


Objective: 1
Temperature: 2

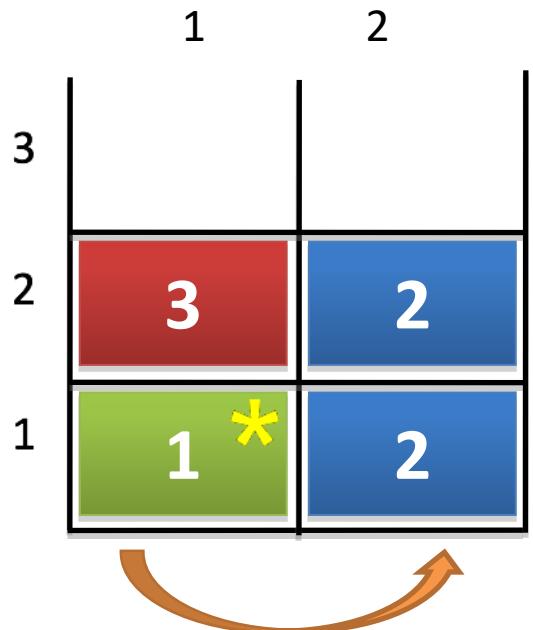
Ex. Container Stowage Problem



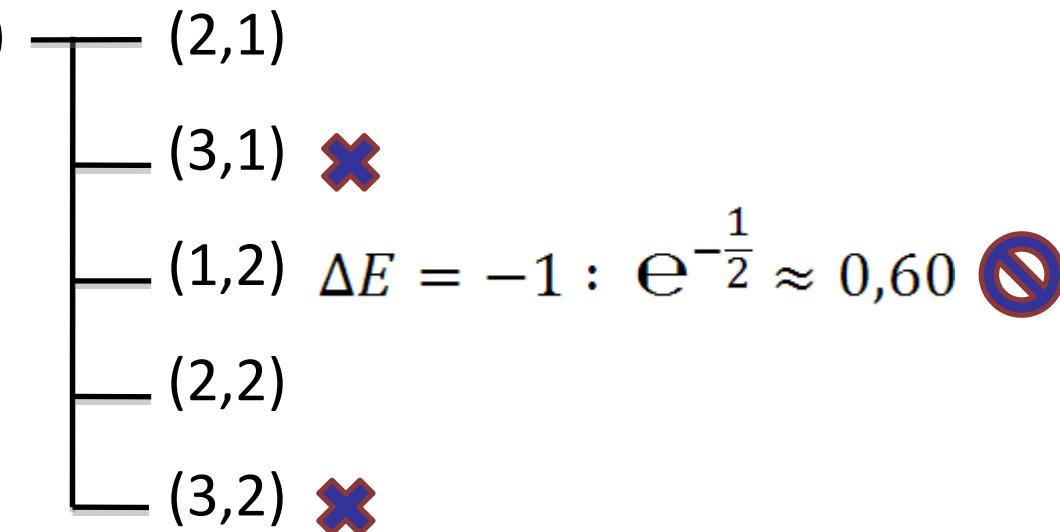
Objective: 1
Temperature: 2



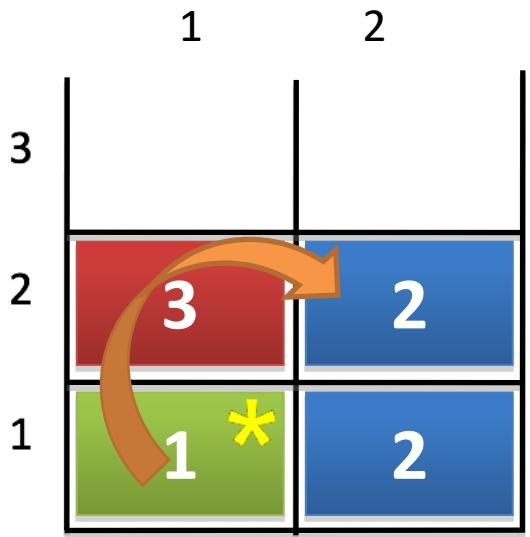
Ex. Container Stowage Problem



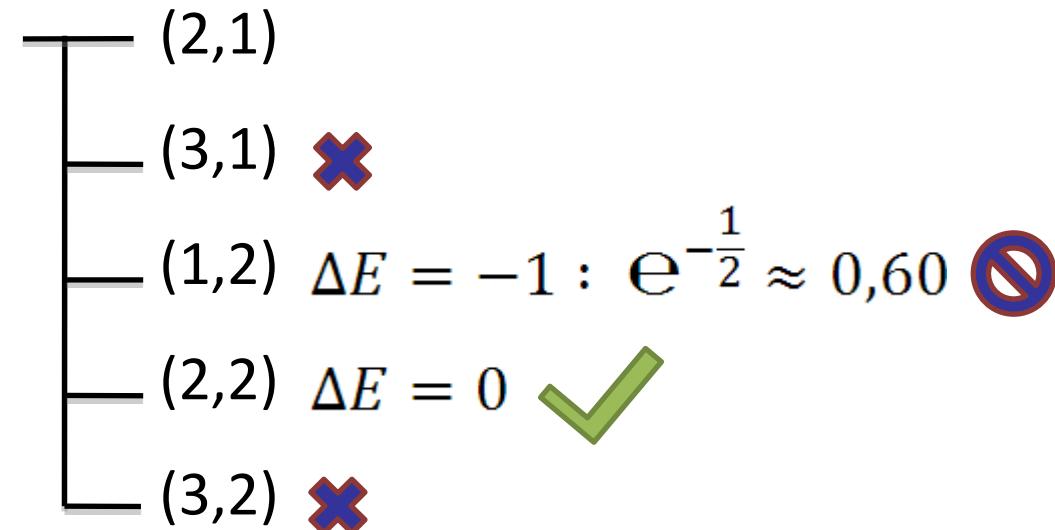
Objective: 1
Temperature: 2



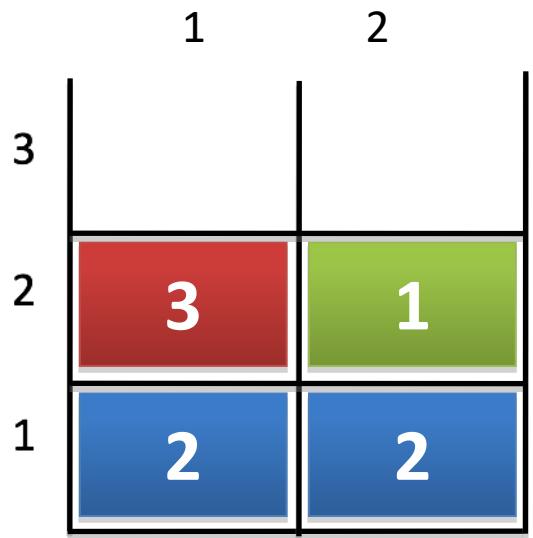
Ex. Container Stowage Problem



Objective: 1
Temperature: 2

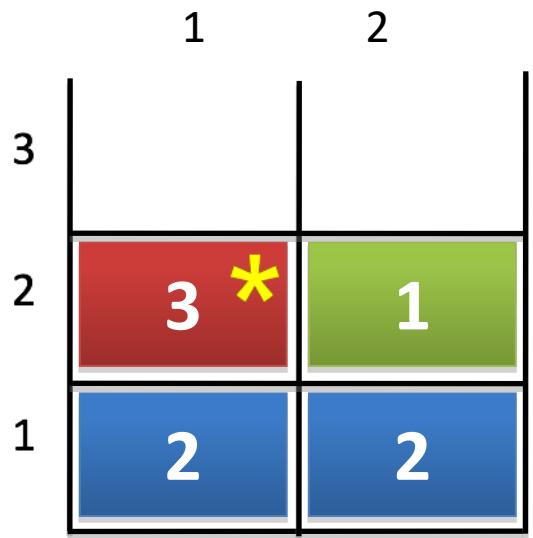


Ex. Container Stowage Problem



Objective: 1
Temperature: 0.5

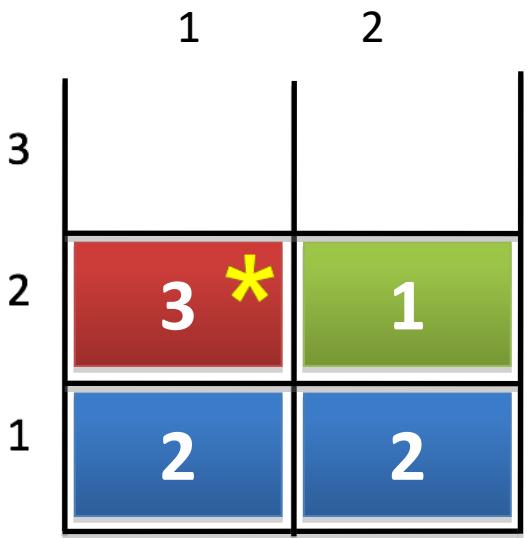
Ex. Container Stowage Problem



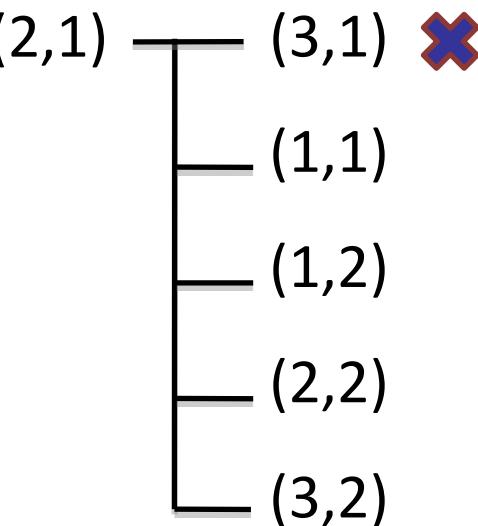
Objective: 1

Temperature: 0.5

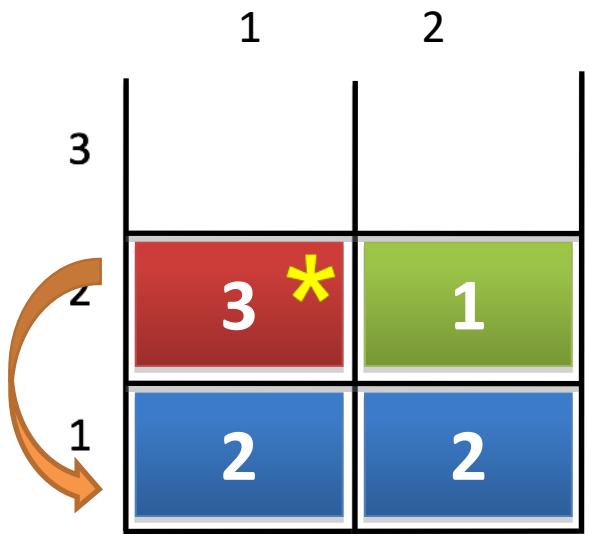
Ex. Container Stowage Problem



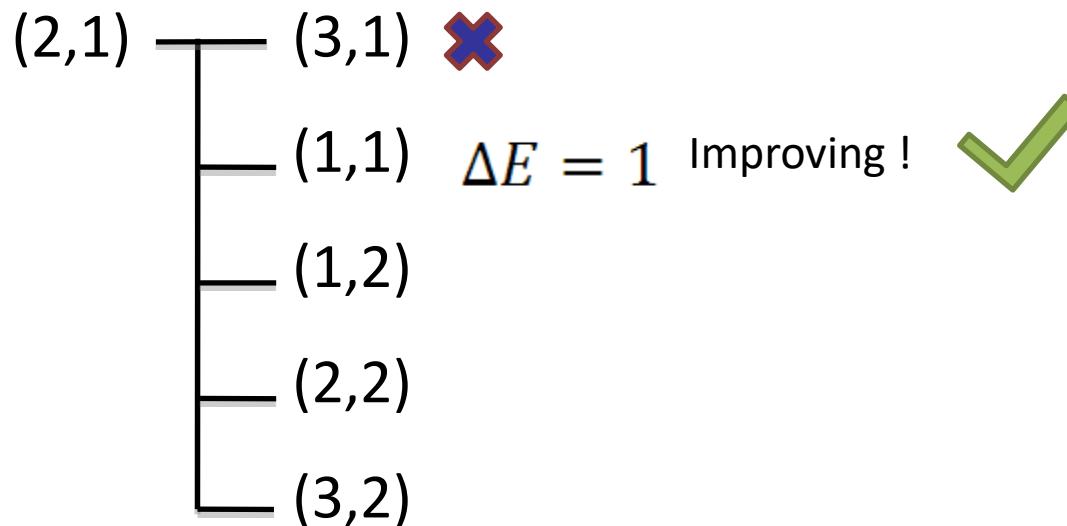
Objective: 1
Temperature: 0.5



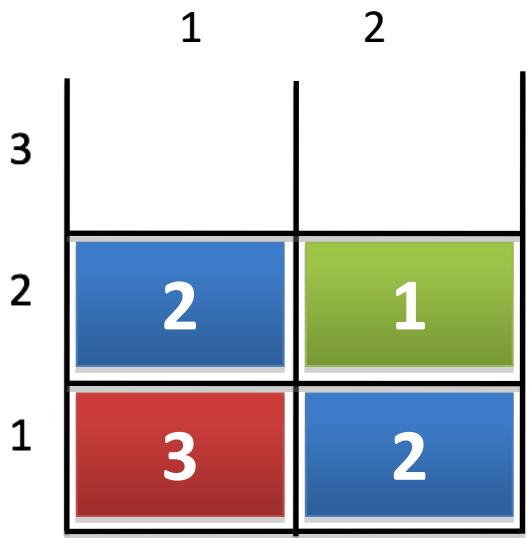
Ex. Container Stowage Problem



Objective: 1
Temperature: 0.5



Ex. Container Stowage Problem



Objective: 0
Temperature: 0.2



Lower bound ! Terminate!

Break



Tabu Search (TS)



Tabu Search (Paper GP10)

- A **tabu** (also spelled taboo) is a strong social **prohibition** (or **ban**) against words, objects, actions, or discussions that are considered undesirable or offensive by a group, culture, society, or community.
 - “Taboo” Wikipedia



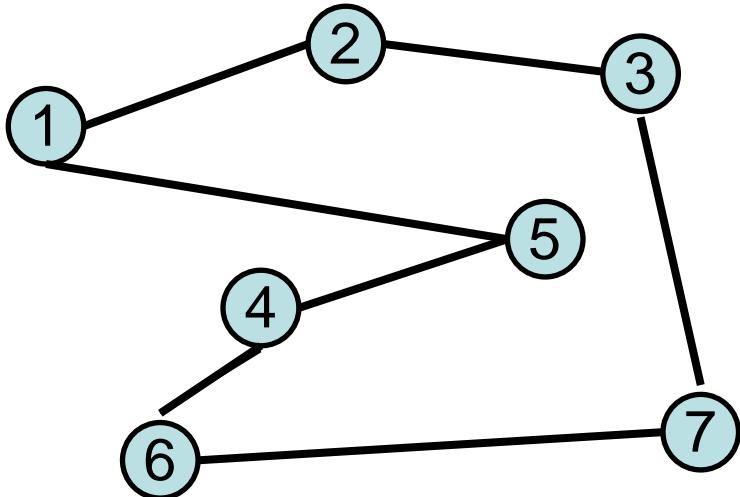
Tabu Search

- Idea:
 - Accept the best neighbor at each iteration
 - Avoid previously seen solutions by keeping a memory (tabu list) of previous states



Tabu Search: TSP Example

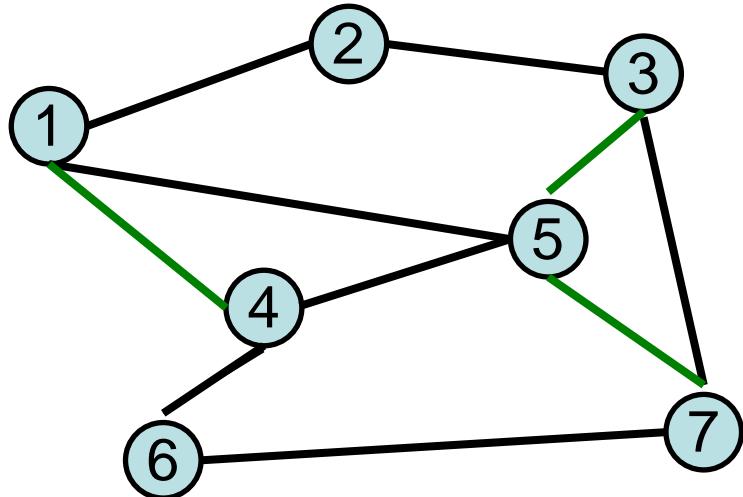
- What could we store in our tabu list?



	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

Tabu Search: TSP Example

- We could store an entire solution
- Or, just store the changes we made



	1	2	3	4	5	6	7
1	0	2.2	5	2.8	4.1	5	8.5
2	2.2	0	3	3	2.8	6	8
3	5	3	0	4.2	2.2	7.2	7
4	2.8	3	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5	5.4
6	5	6	7.2	3.1	5	0	5.1
7	8.5	8	7	5.7	5.4	5.1	0

- Tabu list:
 1. $-(1,5), -(4,5), -(3,7)$
 2. $+(1,4), +(5,7), +(3,5)$

Tabu Search (maximizing)

```
function TABU-SEARCH( problem ) returns a solution state
    inputs: problem, a problem

    current  $\leftarrow$  problem.INITIAL-STATE
    best  $\leftarrow$  current
    T  $\leftarrow$  Empty tabu list
    while ( termination criterion not satisfied ) do
        current  $\leftarrow$  a highest-valued successor of current legal wrt. T
        if VALUE(current) > VALUE (best) then
            best  $\leftarrow$  current
            ADD( T , ACTION-TO( current ) )
    return best
```



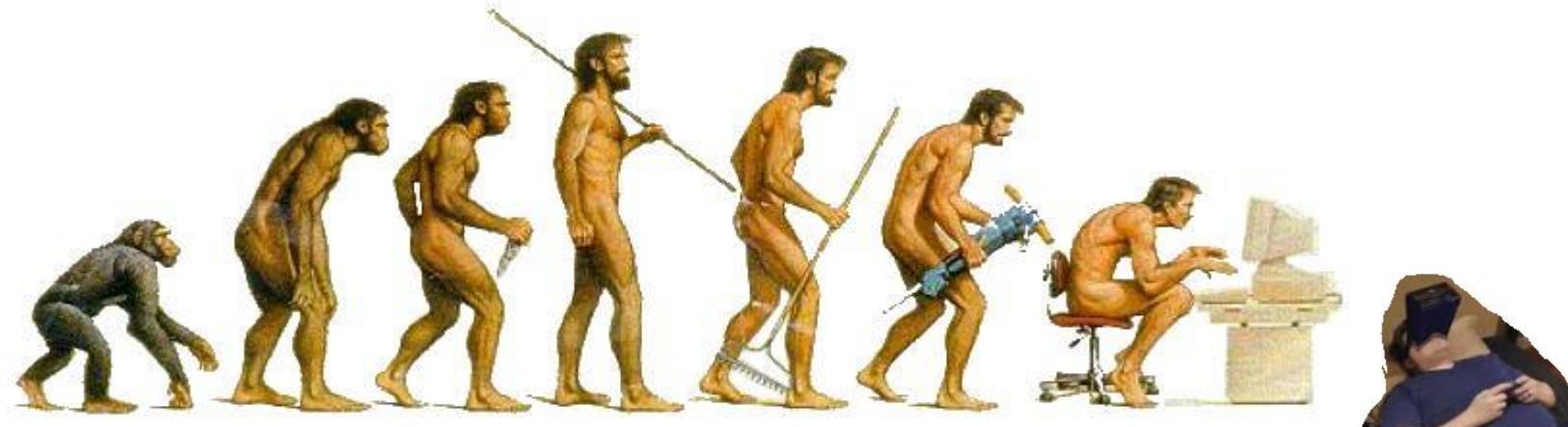
Tabu Search Variations

- Tabu list
 - Variable length
 - Aspiration criteria (override tabu, e.g. if improving move)
- Probabilistic tabu search
 - Only consider a random sample of the neighborhood



Genetic Algorithms (Population based methods)

Evolution



(or is it?)



Genetic Algorithms

- **Individual:** A variable assignment (“Genome”)
 - Can be represented by a bit string
- **Population:** n individuals
 - Initialize to randomly generated individuals
- **Fitness function:** Evaluates the “fitness” of an individual
- **Selection:** Identify the most fit members of a population
- **Crossover:** Form new individuals out of multiple individuals (reproduction)
- **Mutation:** Randomly change a value in an individual’s genome



Example - 1 max

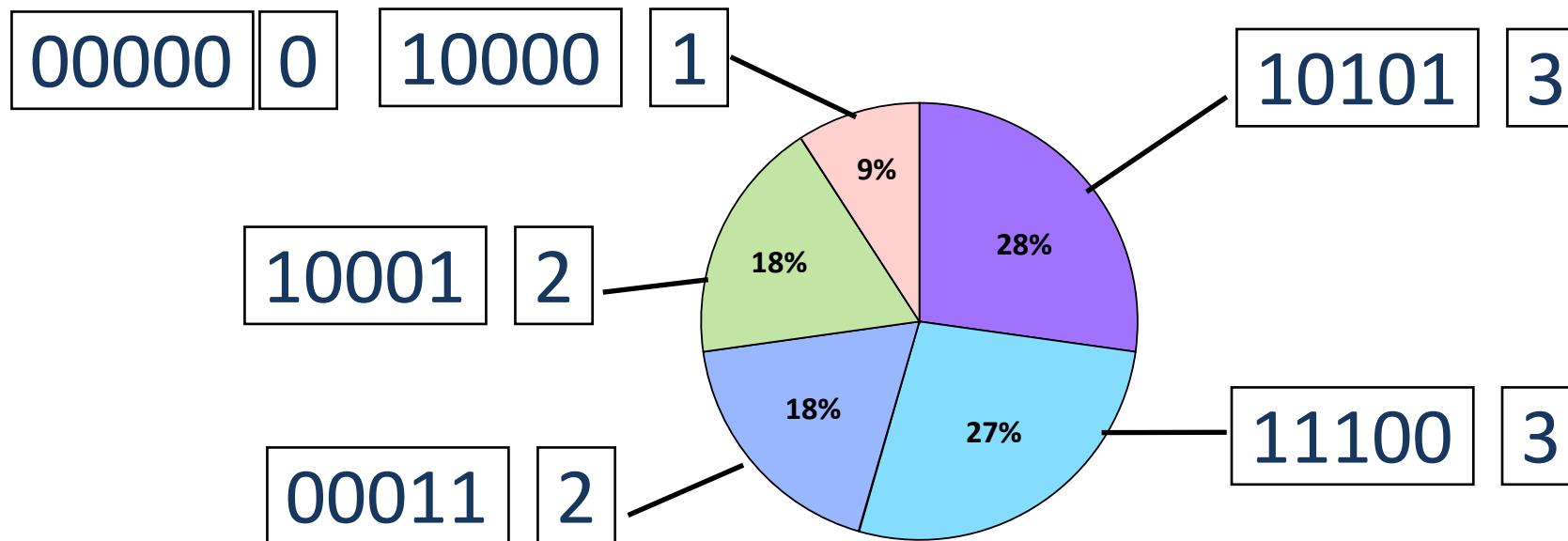
- **Genome:** bit string of length 5
- **Fitness function:** $f(x) = \# \text{ of } 1\text{s in the bit string}$
 - e.g. $f(00110) = 2, f(11111) = 5$
- Goal: Maximize $f(x)$
- Step 1: Initialize population

10000	10101	10001	00011	11100	00000
-------	-------	-------	-------	-------	-------



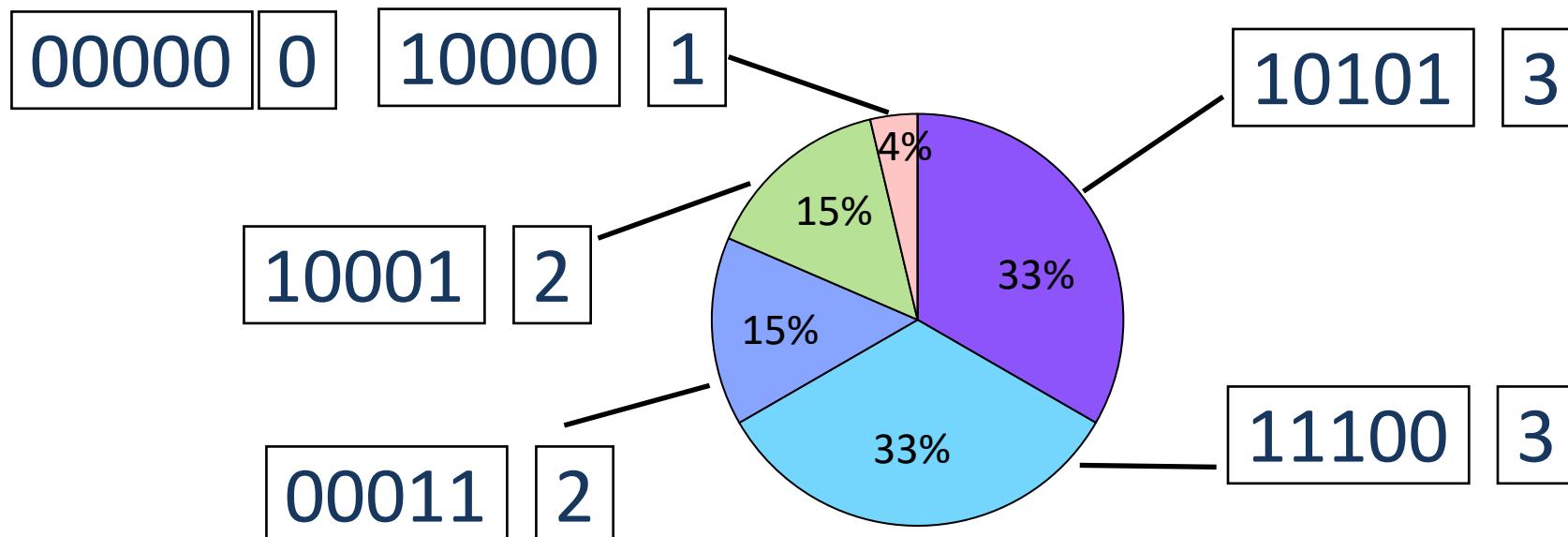
Example - 1 max

- Step 2: Evaluate the population's fitness
- Step 3: Selection. (Roulette wheel selection)
 - Select genomes with probability proportional to their fitness



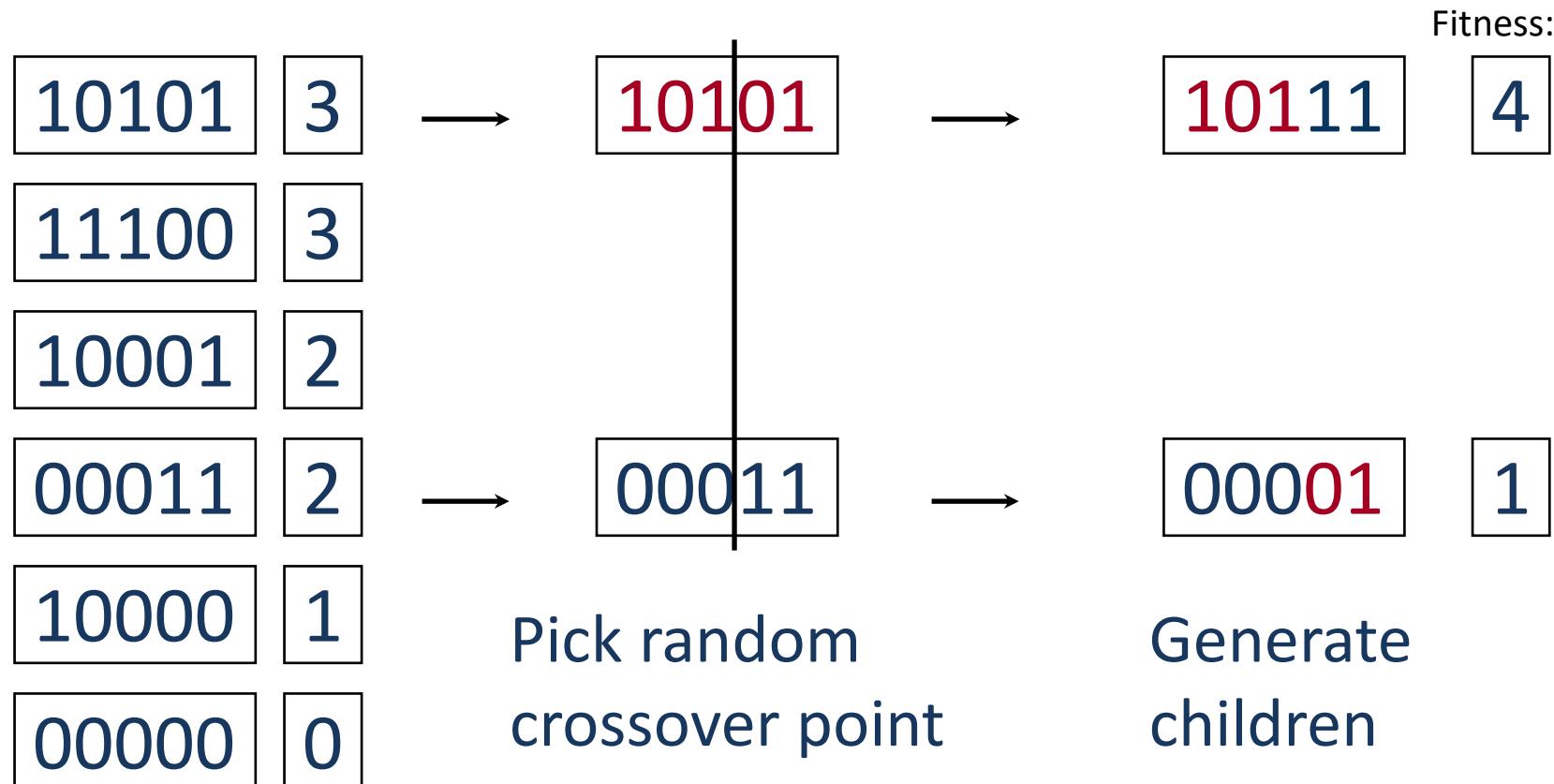
Example - 1 max

- Fitness scaling can improve performance.
 - Square (or cube) the fitness of each individual before performing selection



Example - 1 max

- Select genomes and perform crossover.



Example - 1 max

- Continue selection until new population is formed
 - Individuals are often allowed to be part of multiple selections.
- Step 4: Mutation
 - With some probability, usually < 0.1 make a small change in the genome

00001



10001

Flip random bit



Example - 1 max

- Population at the end of the generation:

10111	4
11100	3
10001	2
10001	2
10000	1
00000	0

- Unless a termination criteria has been reached, continue with the next generation.



Genetic Algorithm

function GENETIC-ALGORITHM(*population, fitness*) **returns** an individual
repeat

weights \leftarrow WEIGHTED-BY(*population, fitness*)

population2 \leftarrow empty list

for *i* = 1 **to** SIZE(*population*) **do**

parent1, parent2 \leftarrow WEIGHTED-RANDOM-CHOICE (*population, weights, 2*)

child \leftarrow REPRODUCE (*parent1, parent2*)

if (small random probability) **then** *child* \leftarrow MUTATE(*child*)

add *child* to *population2*

population \leftarrow *population2*

until some individual is fit enough, or enough time has elapsed

return the best individual in *population*, according to fitness

Genetic Algorithms in Practice

- Necessary that “Genome” forms **meaningful** components of the problem
- Population size difficult to determine
 - Between 25 and 100, depending on the problem
- Terminate criteria vary
 - Little change in average fitness of the population over last n generations, where $n \approx 5$
- Choice of crossover operator extremely important
 - Single point vs. multiple point, etc.

Is multi-restart LS better than population-based LS?



Constraint Based Local Search

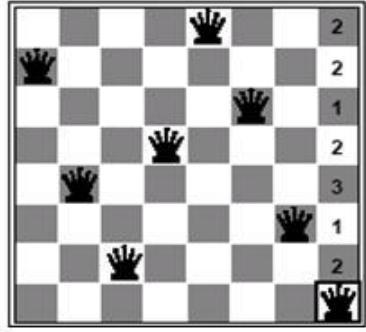


Constraint Based Local Search

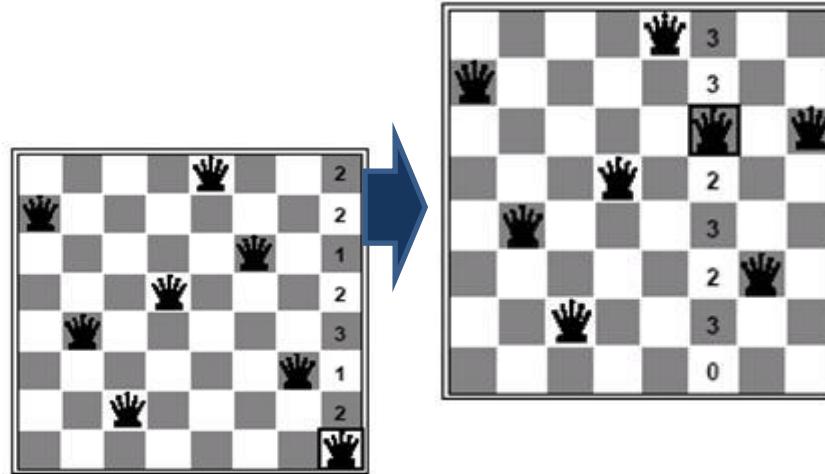
- Constraint satisfaction problems
 - allow states with unsatisfied constraints
 - operators **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
 - Select new value that results in a minimum number of conflicts with the other variables



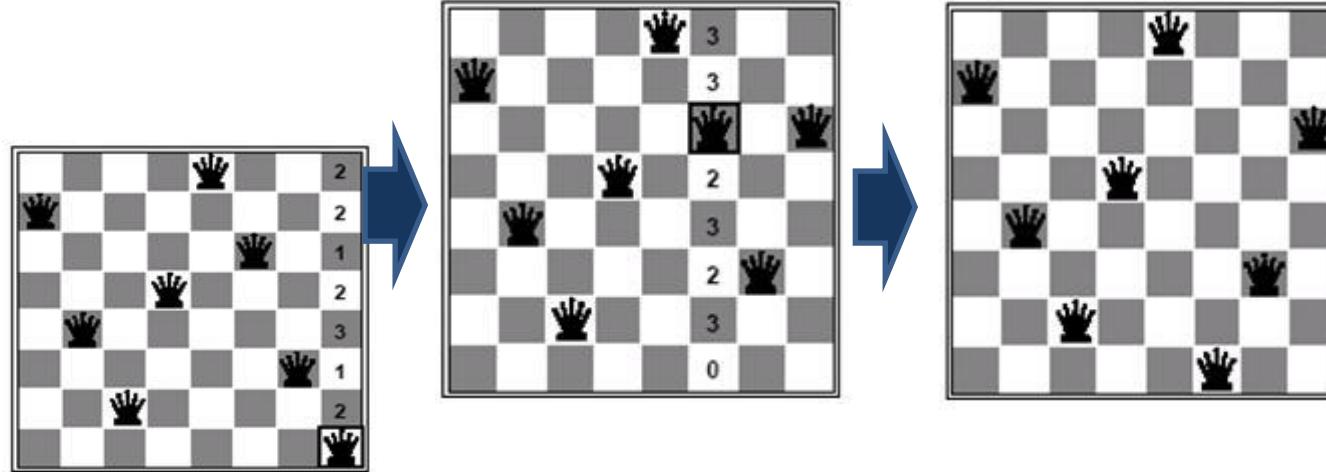
Min-conflict algorithm



Min-conflict algorithm



Min-conflict algorithm



Advanced Methods

- Large Neighborhood Search (LNS)
 - Destroy bad part of solution
 - Reconstruct, e.g. with constraint programming
(or learning Neural LNS, Tierney 2020)
- Variable Neighborhood Search (VNS)
 - Systematic change of the neighborhood during search (diversification)
 - Change level of abstraction during search



Readings

Chapter 2 Tabu Search

Michel Gendreau and Jean-Yves Potvin

Abstract This chapter presents the fundamental concepts of tabu search (TS) in a tutorial fashion. Special emphasis is put on showing the relationships with classical local search methods and on the basic elements of any TS heuristic, namely the definition of the search space, the neighborhood structure, and the search memory. Other sections cover other important concepts such as search intensification and diversification and provide references to significant work on TS. Recent advances in TS are also briefly discussed.

2.1 Introduction

Over the last 20 years, hundreds of papers presenting applications of tabu search (TS), a heuristic method originally proposed by Glover in 1986 [29], to various combinatorial problems have appeared in the operations research literature. In several cases, the methods described provide solutions very close to optimality and are among the most effective, if not the best, to tackle the difficult problems at hand. These successes have made TS extremely popular among those interested in finding good solutions to the large combinatorial problems encountered in many practical settings. Several papers, book chapters, special issues, and books have surveyed

Michel Gendreau
Département de mathématiques et de génie industriel, École Polytechnique de Montréal, and
Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport,
Montréal, QC, Canada
email: michel.gendreau@cirrelt.ca

Jean-Yves Potvin
Département d'informatique et de recherche opérationnelle, Université de Montréal, and Centre
interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Montréal,
QC, Canada
email: potvin@iro.umontreal.ca

M. Gendreau, J.-Y. Potvin (eds.), *Handbook of Metaheuristics*,
International Series in Operations Research & Management Science 146,
DOI 10.1007/978-1-4419-1665-2_2, © Springer Science+Business Media, LLC 2010

Actually a chapter in this great book
(third edition 2019):

