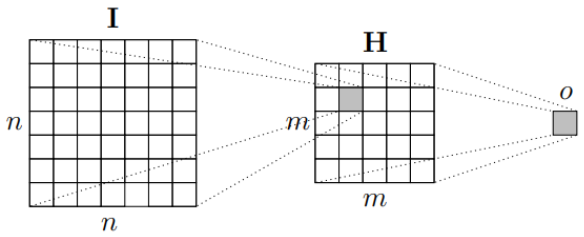
1 2018 Paper 6 Question 2

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2018p6q2.pdf

Evil Robot is updating his visual system. He has a single camera that produces an $n \times n$ matrix $\mathbf{I}$ of pixel values. His visual system is arranged as follows:



The input $\mathbf{I}$ is reduced to an $m \times m$ matrix $\mathbf{H}(\mathbf{I})$. The elements $H_{i,j}$ are

$$H_{i,j}(\mathbf{I}) = \sigma\left(\sum_{k=1}^{n}\sum_{l=1}^{n} w_{k,l}^{(i,j)} I_{k,l} + b^{(i,j)}\right)$$

where σ is an appropriate function, and $w_{k,l}^{(i,j)}$ and $b^{(i,j)}$ are the weights and bias for element (i,j). A single output $o(\mathbf{H})$ is computed as

$$o(\mathbf{H}) = \sigma\left(\sum_{k=1}^{m}\sum_{l=1}^{m} w_{k,l} H_{k,l} + b\right)$$

(a) If Evil Robot has a training example $(\mathbf{I}', y')$ and is using an error $E(\mathbf{w})$ where $\mathbf{w}$ is a vector of all weights and biases available, derive an algorithm for computing $\frac{\partial E}{\partial \mathbf{w}}$ for the example.

In the following question, let:

$$a \triangleq \sum_{k=1}^{m}\sum_{l=1}^{m} w_{k,l} H_{k,l} + b \tag{1}$$

$$a^{(i,j)} \triangleq \sum_{k=1}^{m}\sum_{l=1}^{n} w_{k,l}^{(i,j)} I_{k,l} + b(i,j) \tag{2}$$

$$\frac{\partial E}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial E}{\partial w_{1,1}^{(1,1)}} & \frac{\partial E}{\partial w_{2,1}^{(1,1)}} & \cdots & \frac{\partial E}{\partial w_{n,n}^{(m,m)}} & \frac{\partial E}{\partial w_{1,1}} & \frac{\partial E}{\partial w_{m,m}} \end{pmatrix}$$

Now solve this for each individual weight. Case split on whether the weight is a weight for the output node or for the hidden layer:

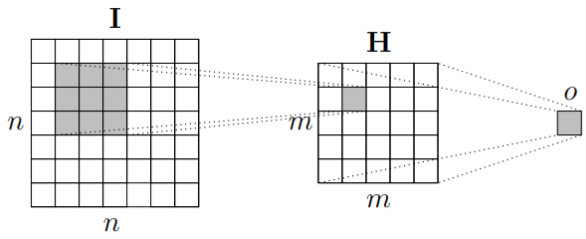
- Case $w_{k,l}$

$$\begin{aligned}
\frac{\partial E}{\partial w_{k,l}} &= \frac{\partial E}{\partial o(\mathbf{H})} \cdot \frac{\partial o(\mathbf{H})}{\partial a} \cdot \frac{\partial a}{\partial w_{k,l}} \\
&= \frac{\partial}{\partial o(\mathbf{H})} \left(y' - o(\mathbf{H})\right)^2 \cdot \frac{\partial}{\partial a}\sigma(a) \cdot \frac{\partial}{\partial w_{k,l}} \left(\sum_{k'=1}^{m}\sum_{l'=1}^{m} w_{k',l'} H_{k',l'} + b\right) \\
&= -2\left(y' - o(\mathbf{H})\right) \cdot \sigma'(a) \cdot H_{k,l}
\end{aligned}$$

- Case $w_{k,l}^{(i,j)}$

$$\begin{aligned}\frac{\partial E}{\partial w_{k,l}^{(i,j)}} =& \frac{\partial E}{\partial o(\mathbf{H})} \cdot \frac{\partial o(\mathbf{H})}{\partial a} \cdot \frac{\partial a}{\partial H_{k,l}} \cdot \frac{\partial H_{k,l}}{\partial a^{(i,j)}} \cdot \frac{\partial a^{(i,j)}}{\partial w_{k,l}^{(i,j)}} \\ =& \frac{\partial}{\partial o(\mathbf{H})}(y' - o(\mathbf{H}))^2 \cdot \frac{\partial}{\partial a}\sigma(a) \cdot \frac{\partial}{\partial H_{k,l}} \left(\sum_{k'=1}^{m} \sum_{l'=1}^{m} w_{k',l'} H_{k',l'} + b \right) \cdot \frac{\partial}{\partial a^{(i,j)}} \sigma(a^{(i,j)}) \cdot \\ & \frac{\partial}{\partial w_{k,l}^{(i,j)}} \left(\sum_{k'=1}^{n} \sum_{l'=1}^{n} w_{k',l'}^{(i,j)} I_{k',l'} + b^{(i,j)} \right) \\ =& -2(y' - o(\mathbf{H})) \cdot \sigma'(a) \cdot w_{k,l} \cdot \sigma'(a^{(i,j)}) \cdot I_{k,l}\end{aligned}$$

(b) A modification to the system works as follows:



The mapping from $\mathbf{I}$ to $\mathbf{H}$ is replaced by an $n' \times n'$ *convolution kernel*. This has a single set of parameters $v_{k,l}$ and c used to compute every element of $\mathbf{H}$ as the weighted sum of a patch of elements in $\mathbf{I}$

$$H_{i,j}(\mathbf{I}) = \sigma \left(\sum_{k=1}^{n'} \sum_{l=1}^{n'} v_{k,l} I_{i+k-1,j+l-1} + c \right)$$

Provide a detailed description of how the algorithm derived in Part (a) must be updated to take account of the modification.

Same as before, case split on whether the weights is in the final layer or not:

- Case $w_{k,l}$

  This case is unchanged – the convolution kernel only affects the earlier layer.

$$\frac{\partial E}{\partial w_{k,l}} = -2(y' - o(\mathbf{H})) \cdot \sigma'(a) \cdot H_{k,l}$$

- Case $v_{k,l}$

$$\frac{\partial E}{\partial w_{k,l}^{(i,j)}} = \frac{\partial E}{\partial o(\mathbf{H})} \cdot \frac{\partial o(\mathbf{H})}{\partial a} \cdot \left(\sum_{i=1}^{m} \sum_{j=1}^{m} \frac{\partial a}{\partial H_{i,j}} \cdot \frac{\partial H_{i,j}}{\partial a^{(i,j)}} \cdot \frac{\partial a^{(i,j)}}{\partial v_{k,l}} \right) \tag{3}$$

$$= \frac{\partial}{\partial o(\mathbf{H})} (y' - o(\mathbf{H}))^2 \cdot \frac{\partial}{\partial a} \sigma(a) \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{\partial}{\partial H_{i,j}} \left(\sum_{k'=1}^{m} \sum_{l'=1}^{m} w_{k',l'} H_{k',l'} + b \right) \cdot \tag{4}$$

$$\frac{\partial}{\partial a^{(i,j)}} \sigma(a^{(i,j)}) \cdot \frac{\partial}{\partial v_{k,l}} \left(\sum_{k'=1}^{n'} \sum_{l'=1}^{n'} v_{k',l'} I_{i+k'-1,j+l'-1} + c \right) \tag{5}$$

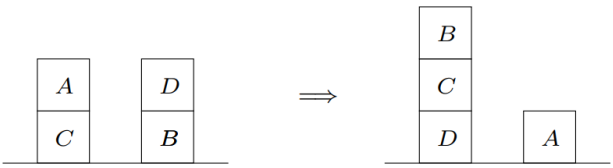
$$= -2(y' - o(\mathbf{H})) \cdot \sigma'(a) \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} w_{i,j} \cdot \sigma'(a^{(i,j)}) \cdot I_{i+k-1,j+l-1} \tag{6}$$

$$\tag{7}$$

2 2019 Paper 6 Question 1

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2019p6q1.pdf

Evil Robot has been kidnapped by experimental psychologists, who are forcing him to solve problems involving the stacking of blocks. For example, given the start state on the left, he is asked to re-arrange the blocks into the state shown on the right.



You are to help him solve these problems by designing a system using *planning graphs*. A block can only be moved if it does not have another block on top of it. Only one block can be placed directly on top of another, although stacks of multiple blocks are allowed.

(a) Explain how this problem can be represented as a planning problem, such that it can be analyzed using a planning graph. Describe how state should be represented, and how actions should be represented, giving a specific example relevant to the stated problem in each case.

I propose representing this problem as a propositional planning problem. This means the problem is represented as a start state, a goal state and a set of propositional rules.

Furthermore, I consider the **order of the stacks to be unimportant**. My initial attempt at answering this question assumed that order was important and rapidly reached a point where (b) became infeasible.

I represent the position of each node by its position relative to other nodes. This is a set of predicates of the form `on(X, Y)` where `on(X, Y)` can be interpreted to mean "block `X` is on top of block `Y`". There are predicates for every pair of nodes $(\texttt{X}, \texttt{Y}) \in \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{None}\}^2$. For conciseness, I use the closed world assumption – any predicate I do not explicitly state to be true may be assumed false.

The start state would be represented as:

$\{$`on(None, A)` `on(A, C)` `on(C, None)` `on(None, D)` `on(D, B)` `on(B, None)`$\}$

The goal state would be represented as:

$\{$`on(None, B)` `on(B, C)` `on(C, D)` `on(D, None)` `on(None, A)` `on(A, None)`$\}$

The following rules are **not first order logic** – rather they are propositional rules with abstract blocks to indicate the structure of the rules – we require separate rules of this structure for every combination of $(\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}) \in \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}^4$ separate

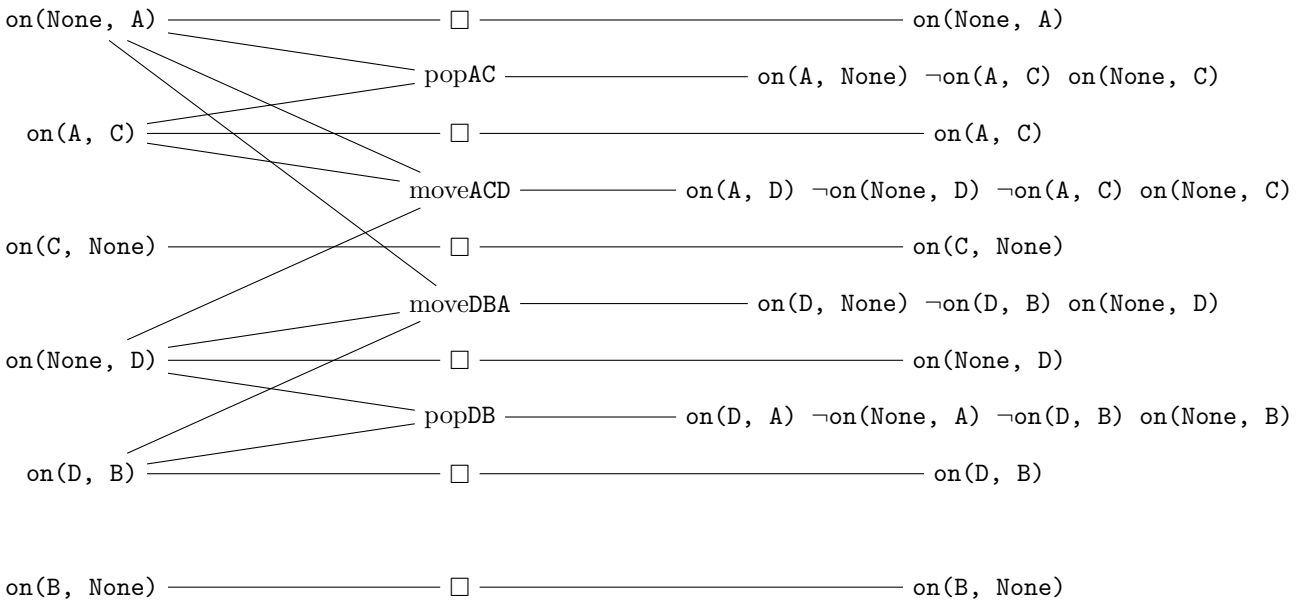
$$\frac{\texttt{on(None, X)} \quad \texttt{on(X, Y)}}{\texttt{on(X, None)} \quad \neg\texttt{on(X, Y)} \quad \texttt{on(None, Y)}} (\text{pop}xy)$$

$$\frac{\texttt{on(None, X)} \quad \texttt{on(X, Y)} \quad \texttt{on(None, Z)}}{\texttt{on(X, Z)} \quad \neg\texttt{on(None, Z)} \quad \neg\texttt{on(X, Y)} \quad \texttt{on(None, Y)}} (\texttt{move}xyz)$$

$$\frac{\texttt{on(None, X)} \quad \texttt{on(X, None)} \quad \texttt{on(None, Y)}}{\texttt{on(X, Y)} \quad \neg\texttt{on(None, Y)} \quad \neg\texttt{on(X, None)}} (\text{push}xy)$$

$$\vdots$$

(b) Using the start state in the diagram above, draw the initial planning graph for the problem, including the initial state level, the first action level, and the state level resulting from the first action level. Do not add any mutex links at this stage.

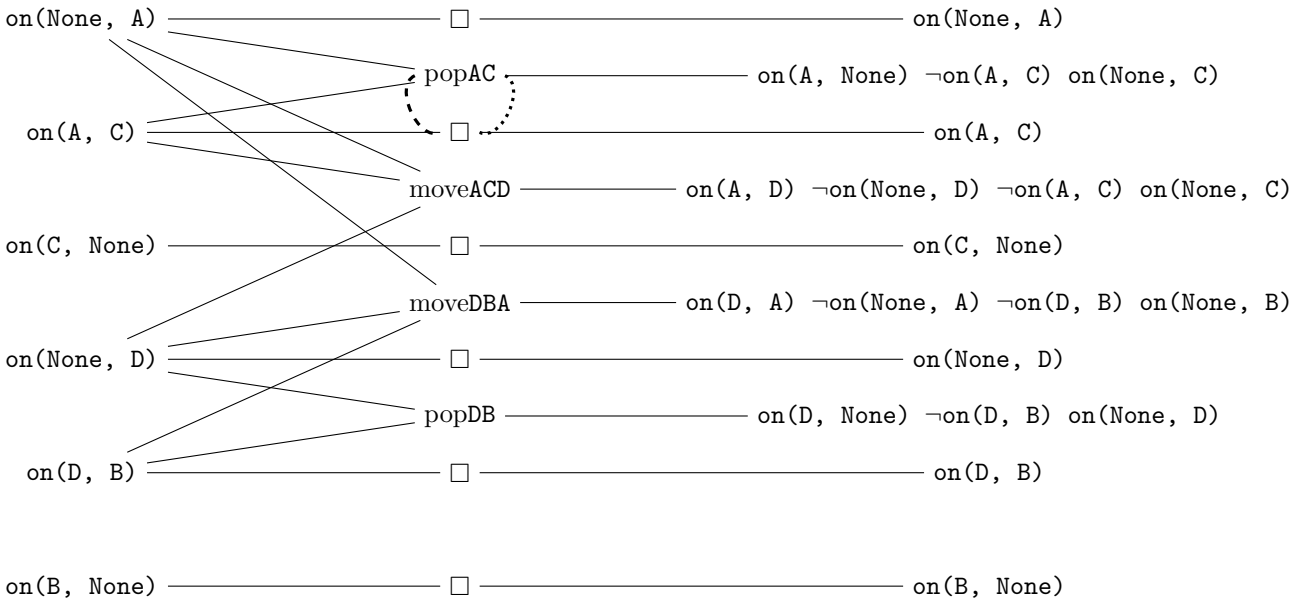


(c) Define an *inconsistent effects mutex* and an *interfering actions mutex*. Add to your diagram for Part (b) a single example of each, or explain why this is not possible.

An *inconsistent effects mutex* is placed between two `actions` when the effect of one action is `P` and the effect of another is ¬`P`.

An *interfering actions mutex* is placed between two actions when the effect of one is `P` and the *precondition* for the other is ¬`P`.

I denote *inconsistent effects mutex* with dotted lines; and *interfering actions mutex* with dashed lines.



(d) Define a *competing for preconditions mutex*. By adding a small number of actions to the second action level of your planning graph, give a single example of such a mutex, or explain why this is not possible.

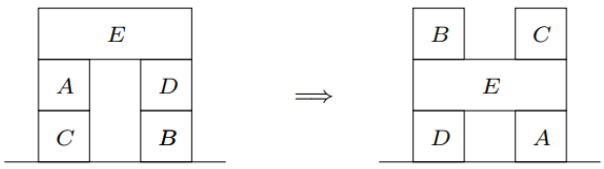
A *competing for preconditions mutex* occurs when the preconditions of two actions are inconsistent.

It's not possible to add this to the diagram. All the possible actions from the start state have the predicates in the start state as preconditions. Since the start state *is* as state, its conditions must be mutually consistent. Therefore there are no actions with inconsistent preconditions

(e) How many more action levels would you expect to need before a valid plan could be extracted to solve the problem stated? Explain your answer.

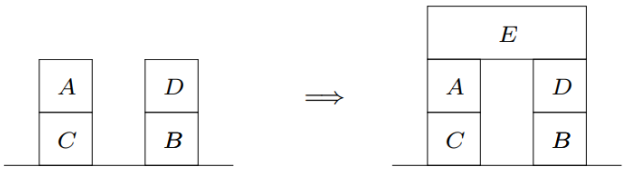
I expect we would need to compute the action graph to depth 3 before a valid plan could be extracted. The plan must go to depth of at least 3 before all the conditions are individually satisfied. At this depth, there is a pair of partial plans which are *not* mutex with each other and can be combined to find the full plan – the plans for pushing `C` onto `D` and the plan for swapping the order of `B` and `D`.

(f) Give two examples of the difficulties that might arise if we also wish to include long blocks as follows:



In each case explain why it might be difficult to address such an extension using planning graphs.

Consider firstly the rule for placing a long block on the top of two stacks:



Long blocks can only be placed on two stacks of equal height. In order to *know* that the stacks are of equal height, we would have to either include the height of a block in its state or have separate rules for every possible permutation of blocks of any height in each tower. Either way means any planning algorithm which used long blocks would have to compute the full plan up to that point and be unable to exploit partial planning. This would be inefficient reducing GraphPlan into a breadth-first search of the possible states.

In order to incorporate long blocks into the plan, we must consider the order of the stacks. This increases the number of rules, greatly increasing the branching factor – consider having to apply a variant of `popAB` to place `A` in every position between stacks.