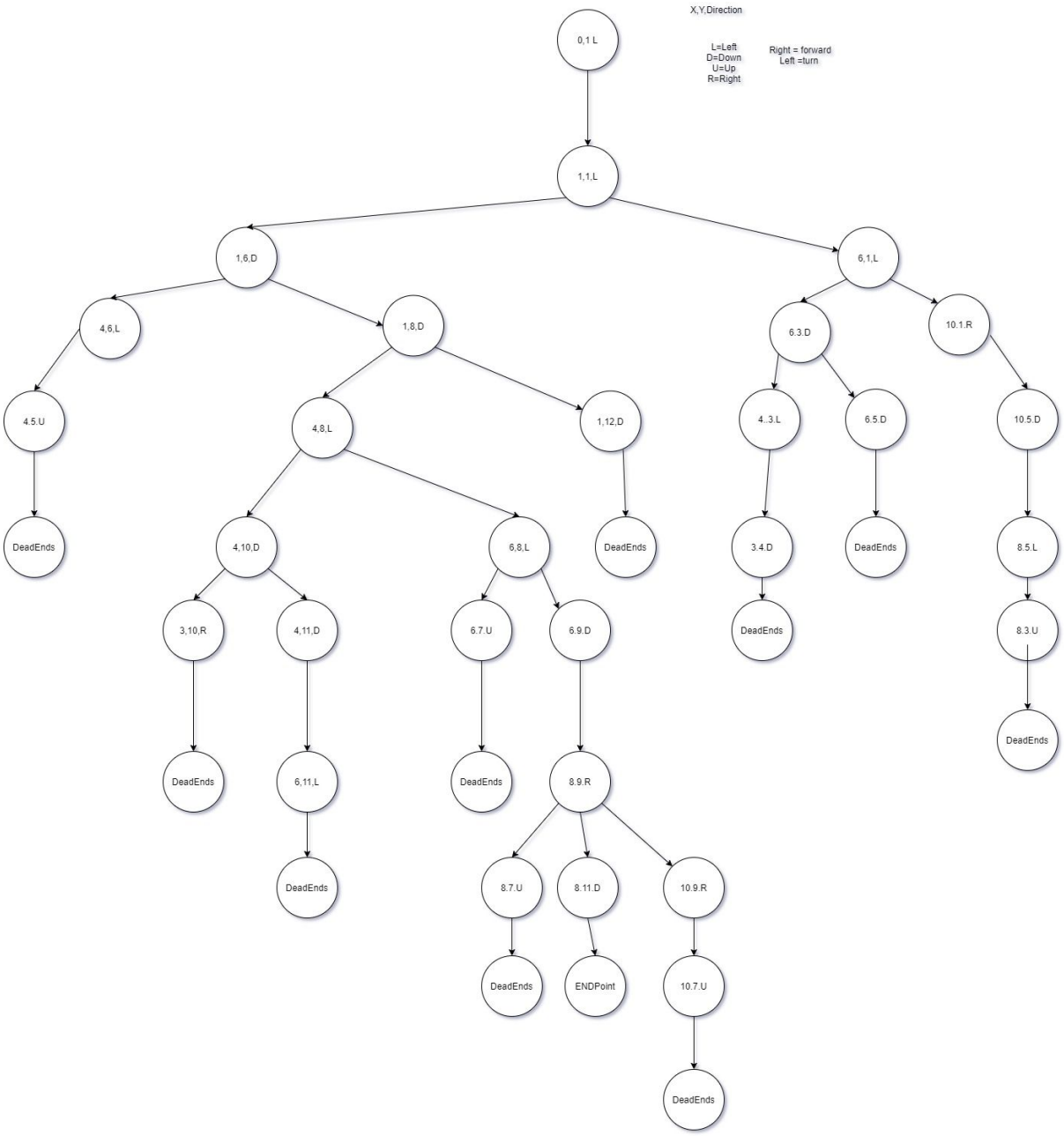
Asignment 2.1

Björn Bengtsson
Nicklas Branding
Welemhret Welay
Hiwot Girma

Search Tree

1.



1.2 Branching factor

Because the children nodes are not equally distributed, the average branching factor needs to be used. 29 edges(non root) and 20 non-leaf nodes 29/20 =1.45

1.3 In general, which properties of the maze can not be represented through a search tree?

Problems that can't be represented at the moment are loops, if there is a loop in the labyrinth the tree will keep going for infinity because there isn't a constraint to not visit old tiles, not to just go back. The solution is to save all visited tiles in a data structure and then look if the new tile has already been visited.

Another problem is labyrinths were backtracking is needed and where the labyrinth is changin, for example the robot must push a button that opens a door in already visited area, because the robot can't go backward i won't be able to go back to the door and reach the goal.

1.4 Which of the traversing strategies (search strategies) covered in the lectures would you choose to find the exit (goal state)? Explain why you chose this search strategy over the other search strategies.

A* algorithm since it is the most optimal algorithm in terms of computational time. The algorithm selects a node in the tree and calculates the cost from the start node and the goal node to the selected node called g(n) + h(n). Which means that we will always find the cheapest solution. A* is only optimal if h(n) is admissible, meaning that it never overestimates the cost to reach the goal. Breadth-first search is an algorithm that expands on the root node and then expands on its successors from left-to-right until it reaches the goal node. While it expands all the tree nodes it will have many redundant paths which could cost a lot of time, but the biggest problem is the required memory requirement since it expands all the nodes. The branch complexity of breadth-first search is O(b^d), where b is the branching factor and d is the depth of the tree. Depth-first search expands the deepest node until it reaches the leaf of the path and backs up to the parent node to check for unexplored child nodes and this goes on until the whole tree has been explored. The depth-first search follows two strategies: graph-search and tree-search. The tree-search follows a non-compete meaning that if loops occur in the tree there will be problems. The graph-search based is complete, which means that it will avoid redundant and repeated steps in the tree. The main problem for both strategies is that if an infinite non-goal path would be present both would fail. The alternative to depth-first-search is depth-limited search. It works exactly the same as depth-first search, however the depth-limited search has a limited set on the depth the algorithm can expand, this is to alleviate the failure if infinite state spaces are used.

1.5 Would you use a different search strategy if you would not know the layout of the maze in advance, i.e. you would be the black square trapped in the maze? Provide Arguments for your decision!

Yes we would change the search strategy to breadth-first search if there is a possibility of infinite state space because if depth-first search would be used, there is a big chance that it wouldn't reach an exit. It would probably follow an infinity path, but breadth first search would look at all possible outcomes, and then always find the exit.

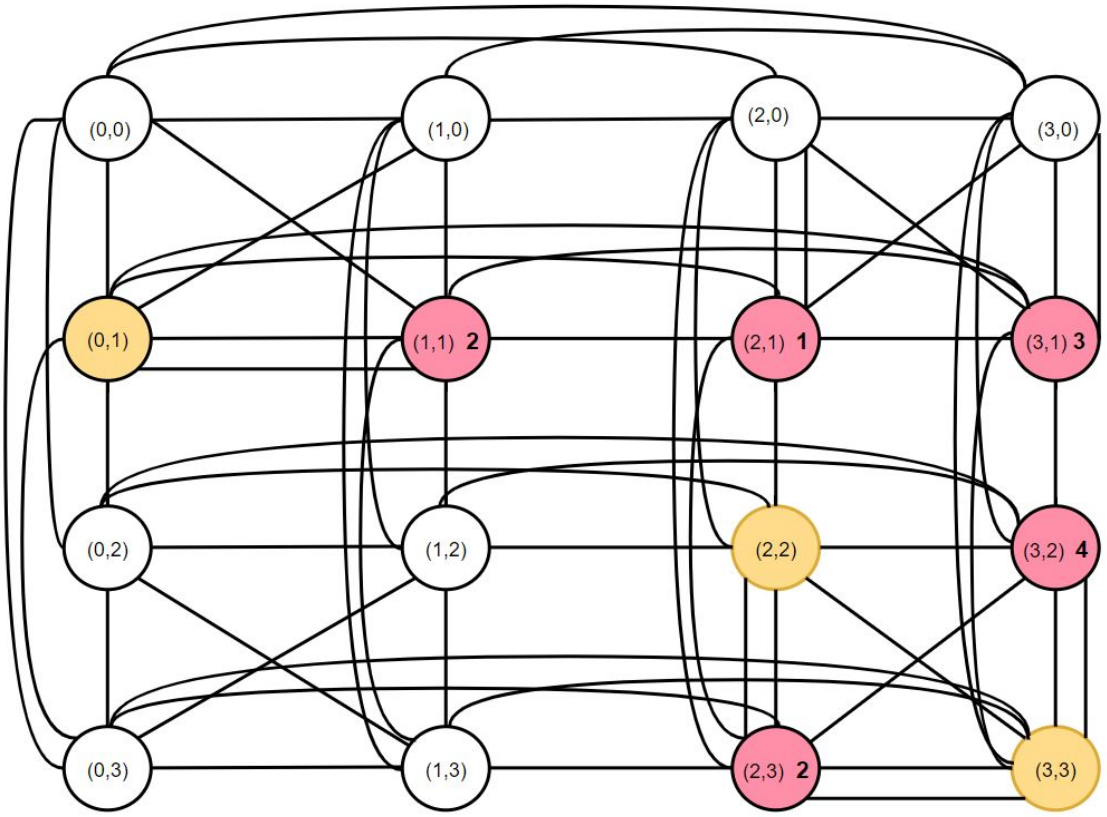
However, if there were finite state space in the maze we would choose depth-first search, since it is less computational heavy compared to breadth-first search since after a path is searched through it is dropped in the memory, and a new path is assigned to the memory to see if the goal is existing in the given path. We would not chose A* star algorithm since this algorithm must know the paths beforehand.

2.0 constraint problem

Your task is to model the game Sudokuas a CSP. The goal of the game is to place numbers into a grid in such a way that two equal numbers do not share the same row, column or quarter of the grid.Your description of the CSP has to at least contain the following:

- the set of variables = Each of the 16 Squares, 11 Empty Spaces
- the set of domains = The possible elements in the 4x4 soduku problem space are 1,2,3, and 4. Therefore the domain is=(1, 2,3,4)
- the set of constraints are: each row, column and quarter(square) have unique elements(1-4).

Additionally, you should answer the following questions:
2.1. Draw a graph of the initial game state, given in figure 2, illustrating the connections among the cells with respect to the constraints and highlight the cell(s)/node(s) with the most restrictions.

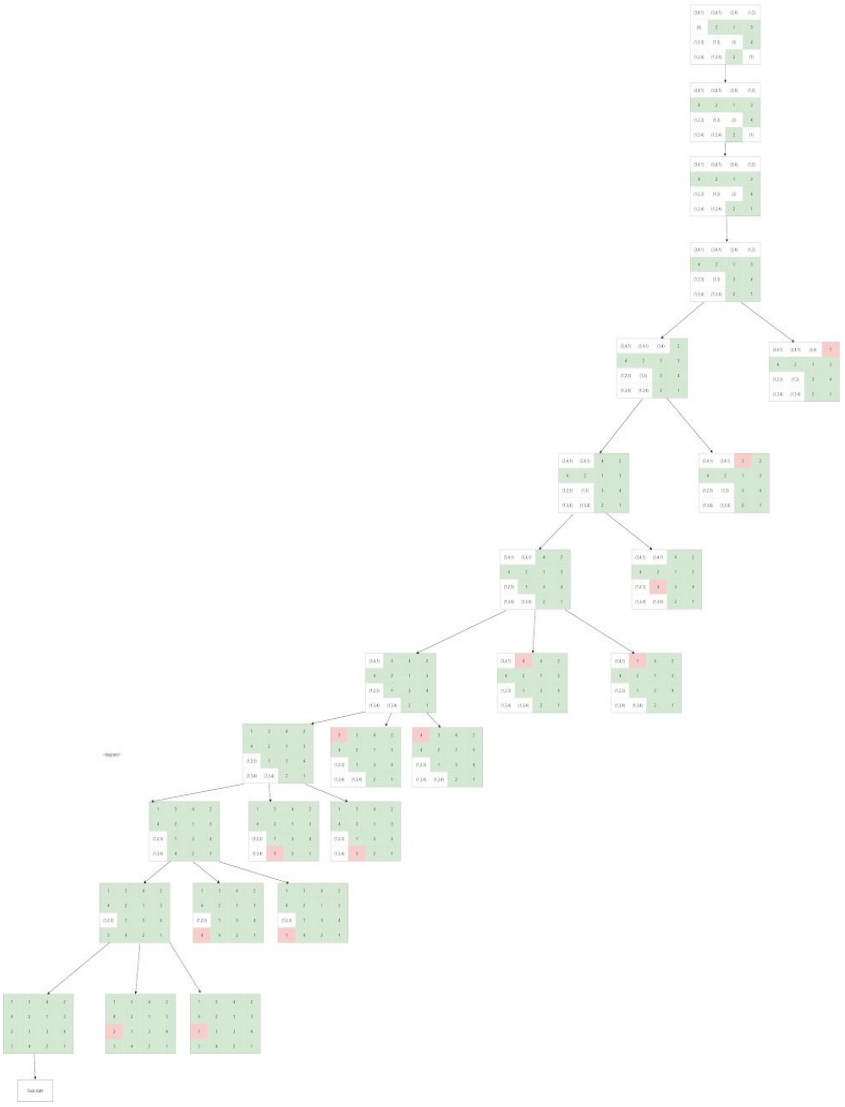


The values of the initial Game state of the Sudoku is summarized in the table below:

(X,Y) position in cells	**Initial Value**
(1,1), (2,3)	2
(2,1)	1
(3,1)	3
(3,2)	4
Other Cells	Empty

The yellow nodes are the most restricted nodes/cells, whereas the red nodes are the cells that have initial values. Highlighting the most restricted cells/nodes is important since it determines the start of the search tree, and the eventual win of the Sudoku game.

2.2. Draw the search tree. Keep in mind to expand the tree at cells/nodes with most restrictions first. In situations in which you can not easily decide (by number of restrictions): come up with a concise description of your decision strategy.Most importantly structure and present your graph in a comprehensible way.



In the above search tree the root node starts from the most restricted cells in the 4x4 Sudoku. After filling the most restricted cells we start selecting the next most restricted cells, this iteration keeps going until the final goal state is reached. The rule for selecting the cells to fill is determined by the number of neighbours in terms of columns, rows and the big square, in this case the 4x4 matrix cells that have already been filled. In special cases; for example the position (3,0) and (2,0) in the sudoku grid have the restrictions (1,2) and (3,4). They also have the same number of neighbours which in this situation we go with randomly selecting the cell to be filled.

2.3. What algorithm would be best to find a solution and why? Argue.

Generally constraint satisfaction problems can be solved efficiently using backtracking algorithm approach where the possible solutions are drawn if not it will backtrack to find another solution. The 4x4 Sudoku is simple and the designed search tree is convenient for the Breadth Search algorithm. Applying Backtracking in small constraint satisfaction problems in our case 4x4 Sudoku is not efficient. Therefore, Breadth first search is the best and the most efficient algorithm to solve this problem. Based on the above search tree the most convenient and efficient algorithm to traverse on the search tree is Breadth First Search algorithm.

2.4. Would the way you solved this problem be a feasible way of solving a similar problem with a large grid size as well? Argue.

The way that we solve the above problem is mostly convenient and efficient for small grid size Sudoku Games. As the size of the grid increases the complexity of the algorithm in terms of memory and time increases. Breadth First Search algorithms space requirement increases exponentially as the grid size of the Sudoku Game Increases. So Backtracking is an ideal algorithm approach to the large size Sudoku constraint satisfaction problem.

3 Probabilities

3.1 Independence TheoremAssumeP(A|B) =P(A), show that the following statements hold:1.P(A∧B) =P(A)P(B)

P(A∧B) =P(A)P(B)

P(A∧B) is equal to P(B|A)*P(A)

P(B|A)*P(A) =P(A)P(B)

P(B|A)=P(B)

replacing P(B|A) with P(B) in P(B|A)*P(A) =P(A)P(B)

P(A)*P(B) =P(A)*P(B)

2.P(B|A) =P(B)

P(B|A) = P(B)

P(B|A) is equal to P(B∧A)/P(A)

P(B∧A)/P(A) = P(B)

P(B∧A)=P(A)*P(B)

Replacing P(B∧A) with P(B)*P(A) in P(B∧A)/P(A) = P(B)

(P(B)*P(A))/P(A)=P(B)

P(B)=P(B)

3.P(¬A|B) =P(¬A)
P(¬A|B) = P(¬A)
P(¬A∧B)/P(B) = P(¬A)
P(¬A∧B)=P(¬A)*P(B)
Replacing P(¬A∧B) with P(¬A)*P(B) in P(¬A∧B)/P(B) = P(¬A)
P((¬A)*P(B))/P(B)=P(¬A)
P(¬A)=P(¬A)

4.P(A|¬B) =P(A)
P(A|¬B)=P(A)
P(A∧¬B)/P(¬B) = P(A)
P(A∧¬B)=P(A)*P(¬B)
Replacing P(A∧¬B) with P(A)*P(¬B) in P(A∧¬B)/P(¬B) = P(A)
(P*(A)*P(¬B))/P(¬B)=P(A)
P(A)=P(A)

3.2 Maximum Likelihood Estimate (MLE) We want to fit a univariate Gaussian distribution N(μ,σ2) to some data D = (x1,x2,...xN).

N(x|μ,σ2) =1√2πσexp(−(x−μ)22σ2)

- state the MLE estimates for both parameters μ and σ
- show your calculations (step by step), ensure your steps are comprehensively

Given the formula of $N(x|\mu,\sigma^2) = (1/\sqrt{2\pi}\sigma) * exp(-(x-\mu)^2/\sigma^2)$ we can estimate σ and μ by first taking the log-likelihood of the exponential since we want to maximize our probability of σ and μ given the dataset and since we are maximizing over real-numbers given of the exponential it is easier to take the log-likelihood of the exponential since the estimator is in the exponential, and this is because if we were to take log x on the y axis and normal x on the x we would get a line which is increasing to the maximum. When changing to log-likelihood we get the new equation $log((2\sigma^2)^{-n/2} exp(-1/2\sigma^2 \sum(x-\mu)^2))$ we can then replace exponential with log $log((2\sigma^2)^{-n/2} log(exp(-1/2\sigma^2 \sum(x-\mu)^2)))$ =

$-n/2 log(2\sigma^2) - 1/2\sigma^2 \sum(x-\mu)^2 = -n/2 * log(2) - n/2 * log(\sigma^2) * \sum(x-\mu)^2$ now we will try to find the estimator of μ and σ, which we do by taking the derivative of log-likelihood

$\partial/\partial\mu(-n/2\ log(2) - n/2\ log(\sigma^2) - 1/2\sigma^2 \sum(x-\mu)^2$ here we can take out the power of 2 and remove the logarithm giving us $1/\sigma^2 \sum(x-\mu) = 1/\sigma^2 \sum(x-n\mu) = 0$ because we have the

expression to equal zero we can change the expression to find the estimate of μ

$(\mu = 1/n \sum x)$

Now that we have derived the estimate of μ we will start with the σ, we will begin again at the log-likelihood and take the derivative of σ^2 *which we set to* β *for simplicity*

$l(\mu, \sigma, D) = ((-n/2\ log(2\sigma^2) - 1/2\sigma^2\ \sum(x-\mu)^2) = -n/2log(2) - nlog(\beta) - 1/2 * \beta^{-1} \sum(x-\mu)^2$

$\partial/\partial\sigma^2 = -n/2\beta\ + 1/2\beta^{-1} \sum(x-\mu)^2 = 0$ *to find maxima, we want to multiply by* β *on the enumerator*

$= -n\beta/2\beta^2 + 1/2\beta^2 \sum(x-\mu)^2 = 0$ we then multiply both sides with $2\beta^2$ gives us

$-n\beta\ + \sum(x-\mu)^2 = 0$ from here we can take $-n\beta$ over to the other side $n\beta\ =$

$\sum(x-\mu)^2\ = \beta = 1/n \sum(x-\mu)^2$ which is our estimate of sigma

$\sigma^2 = 1/n \sum(x-\mu)^2$

To conclude our estimates are $\sigma^2 = 1/n\ \sum(x-\mu)^2$ and $(\mu = 1/n \sum x)$

4..2 Gaussian DistributionN(x|μ,σ2) = 1√2πσexp(−(x−μ)22σ2)The task for you is to implement the Gaussian distribution and plot it's result for a range of xvalues and at least 3 different combinations of (μ,σ) into the same plot. (μ= 1,σ= 1)should be one of the combinations.The signature of you function could look something like this:def myNormal(x, mu, sigma)res = ...add your code here ...return resNote:Your are not allowed to use distribution implementaitions provided by packages such as numpy or similar. Nevertheless, you are allowed to use basic operators provided by numpy4 such asnumpy.exp(x)computingexornumpy.sqrt(x)returning√x. Some hints: You cande fine a range of value withnumpy.linspace(a,b,num)which returnsnumevenly spreadvalues betweenaandbhave a look here:https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.linspace.html- use this to get values to plot. Plotting canbe achieved using:import matplotlib.pyplot as pltplt.plot([1,2,3,4])plt.show()This should show a straight line with a pitch of one, seehttps://matplotlib.org/users/pyplot_tutorial.htmlfor more details.Attention:Ensure that you function works on scalars as inputs for xas well as for vectors-linspace will return such a v4

Implementation of the Gaussian Distribution:

```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
#Define and implement the Gaussian Distribution Algorithm
def myNormal(x, mu, sigma):
    res = (1/ (np.sqrt(2.0 * np.pi * sigma))) * np.exp(-1 * ((x - mu) ** 2)/(2.0 * sigma**2))
```

```
        return res
# Declare list of x values between -10 and 10
x = np.linspace(-10,10, 200)
# Plot the distribution in the same plot.
plt.plot(x,myNormal(x,1,1))
plt.plot(x,myNormal(x,0.5,1))
plt.plot(x,myNormal(x,1,3))
# Display the plot
plt.show()
```