IT3105 Module 3

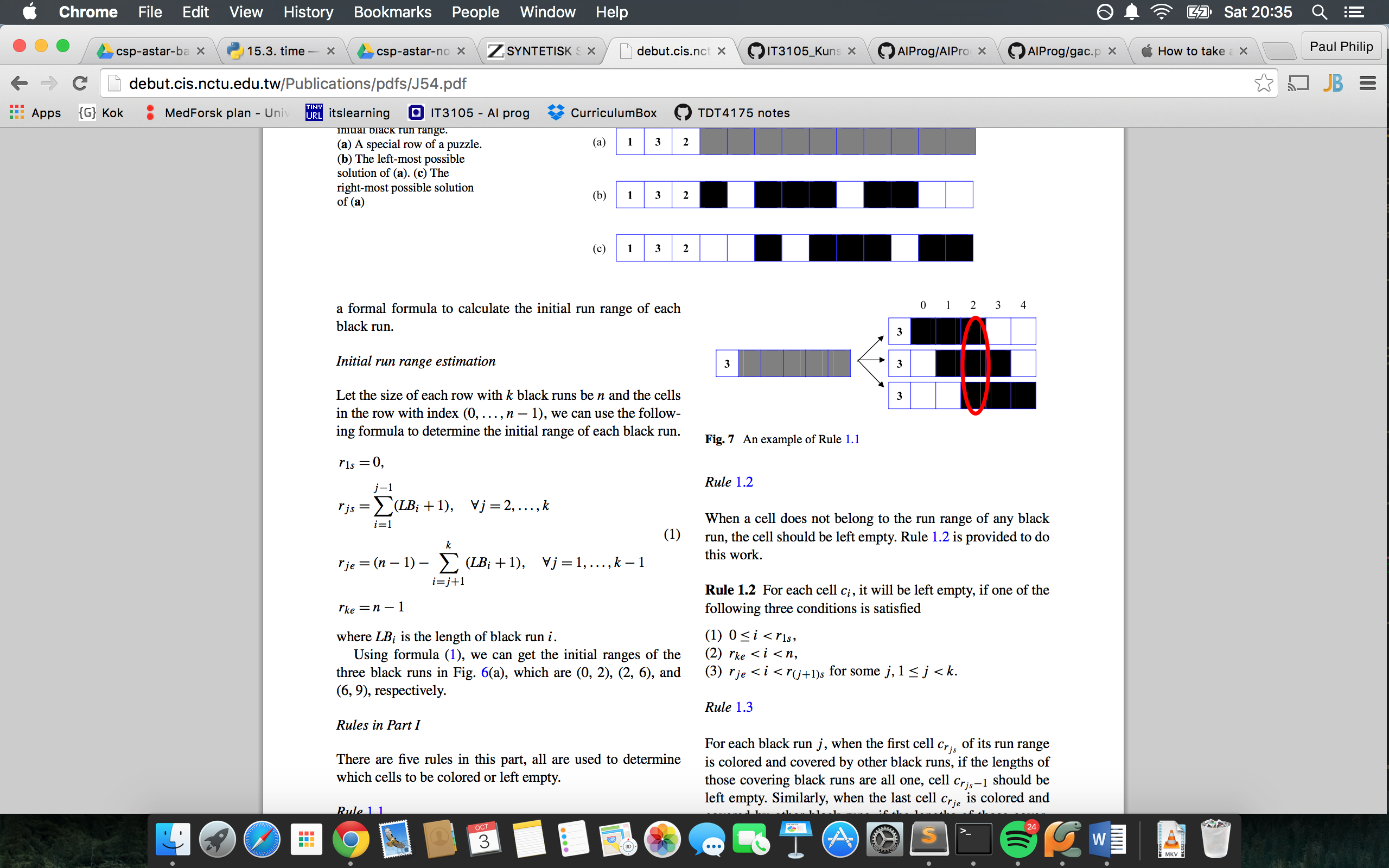
Combining Best-First Search and Constraint-Satisfaction to Solve Nonograms

**Representation of variables, domains and constraints**

We have decided to follow chapter 2.2 Using an Aggregate Representation described in the module 3 description to represent the variables, domains and constraints in this project. We first started using segments and their index as variables, but found out that this caused problems later on.

The variables are represented as entire rows or columns, where a Variable object has three parameters – **index, type and size**, where index is row or column number, type is simply a string stating whether this is a *row* or a *column*, and size is the length of the row or column.

The domain for a variable is a list of all the possible permutations of segment placements this variable has, given a number of segment sizes and the row or column length. A permutation is represented as a list of Booleans, where each index is either True or False based on whether this permutation allows a segment at the appropriate index. We first calculate the initial possible ranges of a segment in a variable using the following formula:



*n = size of row or column, k = number of segments, LBi = length of segment i*

*Function output: [[0, 1, 2, 3], [4, 5, 6]]*

Next, we calculate all the permutations for this row or column using the following function:

**def calculate\_permutations**(self, segment\_domains, segments):  
 permutations = list(itertools.product(\*segment\_domains))  
 **for** list\_element **in** copy.deepcopy(permutations):  
 **for** i **in** range(len(list\_element)-1):  
 **if** isinstance(list\_element, tuple):  
 **if not** list\_element[i] + segments[i] < list\_element[i+1]:  
 **if** list\_element **in** permutations:  
 permutations.remove(list\_element)  
 **break  
 return** permutations

We now have everything we need to create Boolean lists of all possible permutations for a variable.

The constraints are generated by iterating through every variable corresponding to each column for every variable corresponding to each row, and then assigning it the expression *x == y.* A constraint is represented as its own object, Constraint, which holds a list of variables that are involved in this constraint, and a string expression on the form “x==y”.

**Heuristics**

• Explains the heuristics used for this problem. Note that heuristics appear in at least two places in A\*-GSP: a) in A\*’s traditional h function, and b) in the choice of a variable on which to base the next assumption. Both (and others, if relevant) should be mentioned in the report.

**Subclasses and methods needed to specialize the general-purpose A\*-GAC system to handle nonograms**

• Briefly overviews the primary subclasses and methods needed to specialize your general-purpose A\*-GAC system to handle nonograms.

**Other design decisions**

• Mentions any other design decisions that are, in your mind, critical to getting the system to perform well.

Diagrams & code chunks are OK