Final Year Project Report

Full Unit – Interim Report

A Sudoku Solver Using Constraint Satisfaction (& Other Techniques)

Thomas Paul Clarke

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Dave Cohen

Department of Computer Science
Royal Holloway, University of London

February 21, 2014

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Thomas Paul Clarke

Date of Submission: 04/12/2013

Signature:

Table of Contents

Abstract

This Interim report consists of a colaberation of reports that have been made thoughout the first term. Each report has a subject matter that is related to the final project which will consist of a Sudoku solver using constraint satisfaction. These reports also discribe concept programs that have been created in order to demonstrate key algorithms in both the programs and the reports.

SVN Archive

Submission Location

https://svn.cs.rhul.ac.uk/personal/zwac016/CS/Projects/Submissions/ZWAC016/

Entire Project SVN

https://svn.cs.rhul.ac.uk/personal/zwac016/Year%203/Final%20Project/

Project Aims & Objectives

My aim is to create a constraint satisfaction problem in the form of a Sudoku solver; the program will contain a Sudoku board input system and allow the puzzle to be solved using constraint satisfaction algorithms. Along side this I will write a report that will consist of key explanations and diagrams analysing the project's significant concepts and outcomes and the algorithms used to build the Sudoku Solver.

The Aims & objectives within the program:

1. Write a report and program of a Constraint Satisfaction Problem in the form of a Sudoku Solver.
2. Design the program using Data Structures & Design Patterns that will benefit the program
3. Create Graphical User Interface using layout Managers
4. To explore and implement new algorithms to solve the Sudoku and provide other functions.
5. Implement a simple backtracking algorithm on the Sudoku to solve it
6. Implement Generalised Arc Constancy to provide a next best cell function.
7. As an extension I want to incorporate a save and load function to allow many boards to be saved if they are not finished. This will be done via an XML parser.

My Diary, Plan and Time Scale

Date	Deliverable	Kept To
Term 1		
WEEK 1 (23-29/9/13)		
10/9/13	Start project plan	
WEEK 2 (30-6/10/13)		
3/10/13	Write specification And layout reports	3/10/13 Started
4/10/13 2pm	Finish Project Plan	4/10/13 Finished
5/10/13	Detailed Specification Written	5/10/13 Finished
5/10/13	Design and plan the GUI	5/10/13 Started Sketches were made and so were computer built ones
WEEK 3 (7-13/10/13)		
8/10/13	Program 1: Simple colourful GUI with an active button	8/10/13 Started 3 GUI's built within the SVN
8/10/13	Report 1: User Interface design for a Sudoku solver	10/10/13 Started
WEEK 4 (14-20/10/13)		
17/10/13	Finish Program 1	17/10/13 Finished
17/10/13	Finish Report 1	17/10/13 Finished
18/10/13	Report 2: The use of Layout Managers for resizable GUI applications Solvers.	21/10/13 Started
17/10/13	Program 2: data structures including, for instance, dynamic trees and priority queue, populated with large random data sets.	22/10/13 Started
WEEK 5 (21-27/10/13)		
27/10/13	Finish Program 2	29/10/13 Finished
27/10/13	Finish Report 2	29/10/13 Finished
	If feeling up to date start main project design	(Few days behind did not)
WEEK 6 (28-3/11/13)		
28/10/13	Program 3: Game tree for Tic-Tac-Toe	29/10/13 Started
28/10/13	Reports 3: Design Patterns.	1/11/13 Started
WEEK 7 (4-10/11/13)		
10/11/13	Finish Program 3	1/12/13 Finished, had problems implementing the Alphabet algorithm.
10/11/13	Finish Report 3	20/11/13 Finished
WEEK 8 (11-17/11/13)		
11/11/13	Program 4: Eight Queens using Backtracking	11/11/13 Started
13/11/13	Reports 4: Constraint Satisfaction, particularly consistency techniques.	21/11/13 Started
WEEK 9 (18-24/12/13)		
24/11/13	Finish Program 4	1/12/13
24/11/13	Finish Report 4	1/12/13
WEEK 10 (25-1/12/13)		
25/11/13	Report: Complexity, NP hardness and the big O notation	25/11/13 Started
25/11/13	Report: Techniques used by human Sudoku	Have Not Completed
25/11/13	Catch-up Week finish needed	
25/11/13	Start final project if possible	Have Not Done
1/12/13	Finalise reports & programs	1/12/13 Completed
WEEK 11 (2-8/12/13)		
4/12/13 2pm	Term 1 Reports deadline	
7/12/13	Prep for review	
WEEK 12 (9-13/12/13)		
9-13/12/13	December Review Viva	

Holiday	Final project coding and design	
	Outline of report	
Term 2 13/1/14		
14/1/14	Continue final program design	
14/1/14	High-Level Design	
22/1/14	Final report draft start	
1/2/14	First Prototype Program	
22/2/14	Draft Report Finish	
10/3/14	Finish program coding **Final Deliverables** • The program must have a full object-oriented design • A full implementation life cycle using modern software engineering principles • The program will have a splash screen and two other user interaction screens. • The program will have a Graphical User Interface that can be used to generate, load, save, solve and help with Sudoku solving. The report will describe the software engineering process involved in generating your software and describe interesting programming techniques and data structures used (or able to be used) on the project.	
12/3/14	Extra extensions **Suggested Extensions** Porting the program to a mobile device Solving and benchmarking public domain sets of puzzles using XML based files to store and load Sudoku puzzles **Ideas of extensions:** Save results of time to database to have a leader board (Store puzzles as xml) allow random Sudoku puzzles playable by user Solve puzzles user puts in solve in different ways using different algorithms	
13/3/14	Review notes from Algorithms and complexity for final reports	
15/3/14	Testing of code	
20/3/14	Finish report and proof read	
20/3/14	Finalise code and create working file	
22/3/14	Specification of Testing Procedures	
26/3/14 2PM	Final Report	
26/3/14 2PM	Final Project Program	
27-28/3/14	Project Demo Day	

Chapter 1: **Graphical User Interface**

The Graphical User Interface is an essential element to a program that's aiming to be user friendly. The user interface will allow the program to show a window or a series of windows, allowing the user to see all the data that can be output, whilst also permitting the user to input the data. This has become a common process in computer technology and is used by all programs to create an accessible interface for all.

The history of User Interfaces for programs (operating systems in particular) has changed radically since the first Interface was displayed in 1968 by Douglas Englebart[1]. The project involved implementing a display of everything a user would require, whilst making the interface instantly (visually) understandble. It also aimed to allow users to contribute their desired inputs, which outlines the key advantge of the GUI.

1.1 Displaying Inputs/Outputs

This practise also needs to be followed within the GUI for the Sudoku Solver. This involves finding the simplest way to display the inputs and outputs, as well as transforming the dull and monotonous appearance of data into a visually interesting and acessible Interface.

For this project the user will need a very simple layout involving a function that allows the buttons to be displayed in a simple and ordered way. The Sudoku grid is going to be created in a recursive way that will allow each cell to be created individually, which means they will all be equivalent. However, some of the design questions concern what I'll be using as the user enabled part of the cells, i.e. how can a user put a number into a cell?

The whole structure of the grid and user interface will be made in a recursive way to enable the user to edit the grid. By filling the grid with buttons that can change colour when clicked on by the user will show they are editable. The user could then type in the desired number input by using the keyboard.

[1] http://arstechnica.com/features/2005/05/gui/ last Accessed: 31/10/13

Another method is to similarly make the buttons clickable to show an editable state. The user will then be able to use some numbers displayed on the screen to enter into the buttoned cells within the Sudoku grid. This could be a more advantageous method than my previous because it restricts the user, only allowing them to place limited numbers (1-9) in the cells. By programming the numbers for the solver it will prevent certain errors from occurring, which would be more likely to arise by the user using a keyboard for manual input

If the button method proves an insufficient strategy then there's a way to allow the user to pick up and drag a frame that represents a number, which they can then drop it into the cell using an event handler. Therefore, by enabling the user to select pre-programmed numbers in the cell will provide a more efficient user-friendly system.

1.2 User Friendliness

For a user interface to be effective to the user it needs to provide every aspect a user needs to complete the task the program is designed for. The design process is the key to analysing the users needs and requirements; the effectiveness of the design to fulfil those requirements is the measurement of usability.

To satisfy the specification allows user interface to satisfy functionality needs, however to provide a user with the best possible application interface to increase functionality, other examples include the appeal of the application within it's layout and colours.

1.3 Design Concepts

Technical drawings have been made to draft the appearance and feel of the User Interface. I completed these by using different styles and concepts, which would aim to increase the Sudoku solver's functionality and the accessibility of the interface.

Technical Drawing (One Figure 1)

This is a very simple design and was the original just to give an idea of what kind of inputs and outputs there will be and allow them to be visually displayed in a simple format. This design used simple buttons that would be linked to event handlers and through them methods to act on the desired function.

The grid for the Sudoku uses a very simple recursive grid layout that puts the buttons into a correct format. This keeps the design very simple and the desired input for the numbers would be via

Technical Drawing (Two Figure 2)

This design includes different features for example a note box which will be an editable box that allows the user to put notes into if needed, there is also the 1-9 buttons that allow the user to select a cell and input the number from this sequence.

Technical Drawing (Three Figure 3)

This design implements the drag and drop function, as well as another window, which allows the user to save multiple puzzles. When the load button is pressed it displays a window viewing the saved puzzles, which allows them to be selected and loaded. For added simplicity this can also be achieved within a drop down box, which is illustrated in the technical design drawings.

1.4 Design Implementation

Each of these designs has been replicated within window builder to show and create a useable experience. It saves time by implementing this structure because it can display and test the essential features within the Sudoku solver, as well as assessing their practicality. Therefore the best examples of the features used, which were the essential design elements in the technical plans, can be displayed within the final project.

To create the design elements and provide an easy to build on process I have used a model view controller design pattern with the GUI. This will allow the methods and functions to work without being implemented into the design code, which promotes new functions, as they will be separated and processed independently though the controller.

Therefore the project takes what it needs from the program and the computations and will output them.

The concepts and designs for the user interface have been developed to create a user-friendly and accessible alternative to the current plain structure of buttons and large one window interface. By using a task bar as an alternative to buttons and by keeping the window tight (only displaying the Sudoku grid) will keep the options available in a hidden menu, which will make the design cleaner and the Sudoku solver simpler to operate.

Heuristics

Using a resource[2] which represents the process of evaluation of a user interface, these heuristics can be found on Heuristic Evaluation (Nielsen and Molich, 1990; Nielsen, 1994). I will summarise the relevant heuristic and relate them to the GUI of the Sudoku Solver.

1.4.1 Status Of The System

This means to inform the users of the program's inner workings, in terms of the time constraints. For example, if needed there could be a loading bar on a splash screen. As well as a progress bar when the user requires the puzzle to be solved.

The system differs from a real life experience. The system was originally a game played with pen and paper, which allowed the user to make notes and edit their own mistakes. Whilst mistake handling is possible, there are functions that can be implemented that *are* possible in real life. One of these achievable processes could be the filling in of squares.

1.4.2 The User Experience

This is achieved by allowing the user to save and load previous puzzles. This acts as a help function to assist the user in solving the puzzles by providing a few visual steps.

---

[2] Heuristic Evaluation, Usability Evaluation Materials by Darryn Lavery, Gilbert Cockton and Malcolm Atkinson at the Department of Computing Science, University of Glasgow

http://www.dcs.gla.ac.uk/asp/materials/HE_1.0/materials.pdf

1.4.3 Consistency

This is vital to the process it handles. But mostly, the appearance needs to be constant and clear, where the buttons need to be equally sized (especially the ones within the grid) to maintain a uniformed structure.

1.4.4 Error Prevention

It is important to prevent the program from having errors and to implement safegaurds as a precaution in cases they may occur. If the user will input a number higher than 9 into the grid then it will result in an error. However, if the safegaurd was implemented into the program, the user would only be able to choose a button between 1-9, which woulprevent an error from occuring.

1.4.5 User Relevant Information

Everything the user needs to know should be visually clear and the trivial or complicated data should be kept from the user. This will enable the user to easily focus on the things they want to. For example, there could be time and date stamps on each saved Sudoku puzzle, so they'll be identifiable for the user. There could also be a smaller visual example of what that puzzle actually looks like for the user's viewing.

1.4.6 Design Accessibility

I aim to make the user experience as simple as possible by preventing the user from experiencing long processes whilst using the Suduko solver, which could easily be made into quick and managable functions. This can be achieved by allowing the user limiting usage by giving them pre-programmed buttons and inputs to fill the cells in the Suduko grid. They could also use the keyboard as it will allow faster adding of integers or a 'button like' tab to move to the next cell.

1.4.7 The Design

It is vital that only neccassary and inportant information should be displayed to the user, which will prevent them from getting confused and unindated with large quanities of data and functions. For example, a program could be given a specific structure. The display needs to be minimal and uncluttered, which will limit non-vital information and only display it at relvant times.

1.4.8 Error Handling

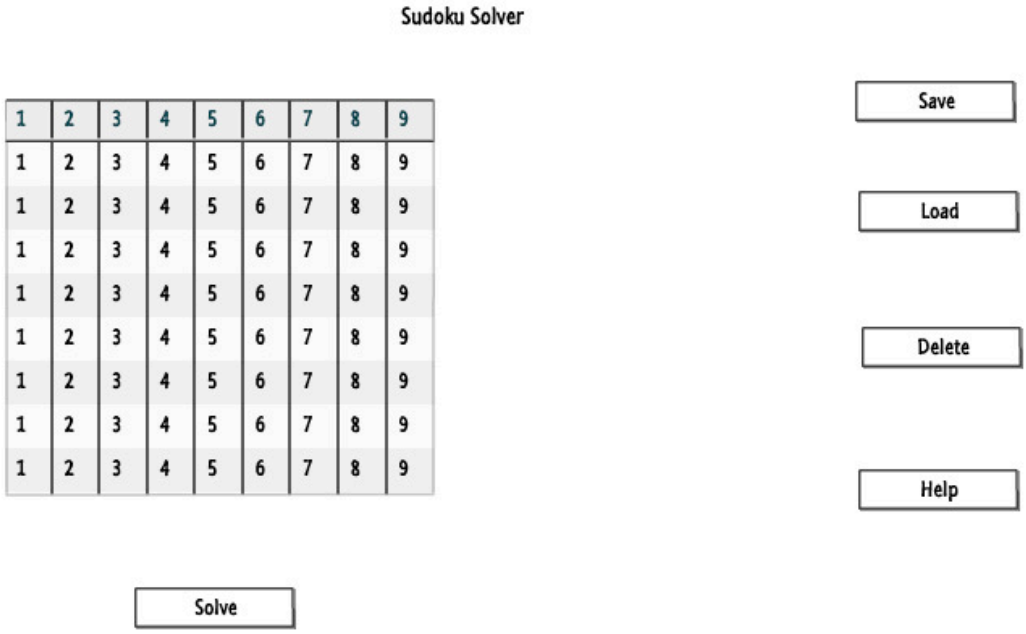
Error messages can be used to explain to the user what the fault of the program is instead of it crashing, which can allow the user to correct their error if it was caused by them.

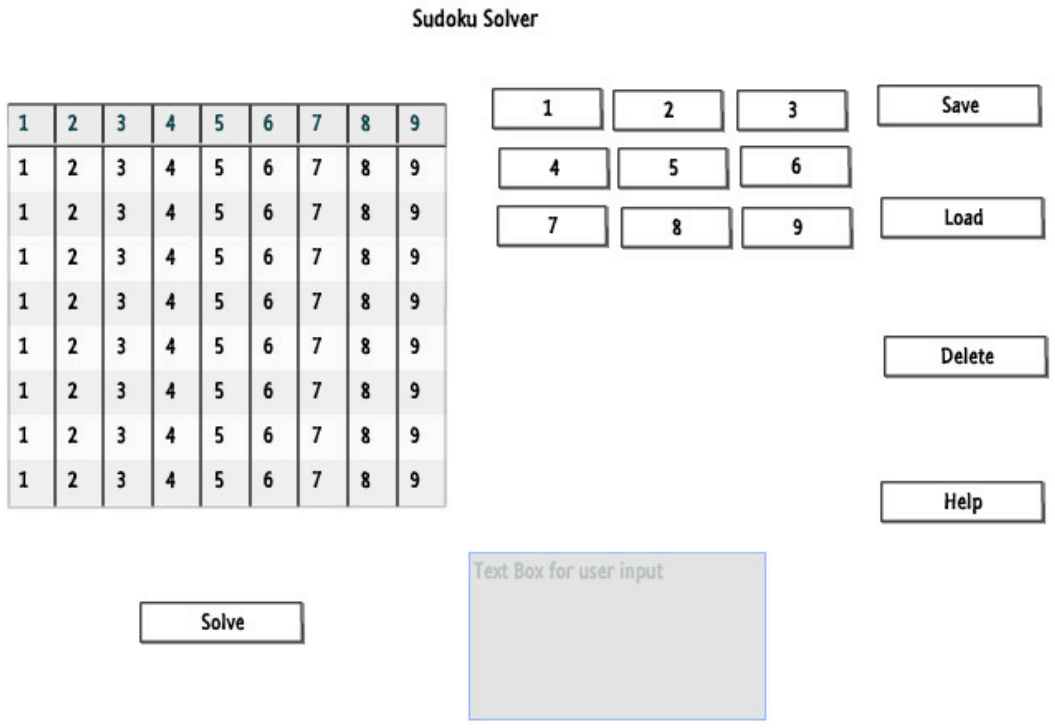
1.4.9 User Assistance

A help function will be implemented to explain the game and the user interface, which can assist a completely new user (with no understanding of the program or the game) on how to work and use it. This could be done by using temporarily displayed boxes that contain useful rules and instructions about what the user needs to do. For example, a drop down box could be used to show the game rules and processes. Or an alternative could be the display of pre-written manual. However these are not the most modernistic or practical assistance tools because users often want to get started on the program as soon as possible or expect a visual tutorial.

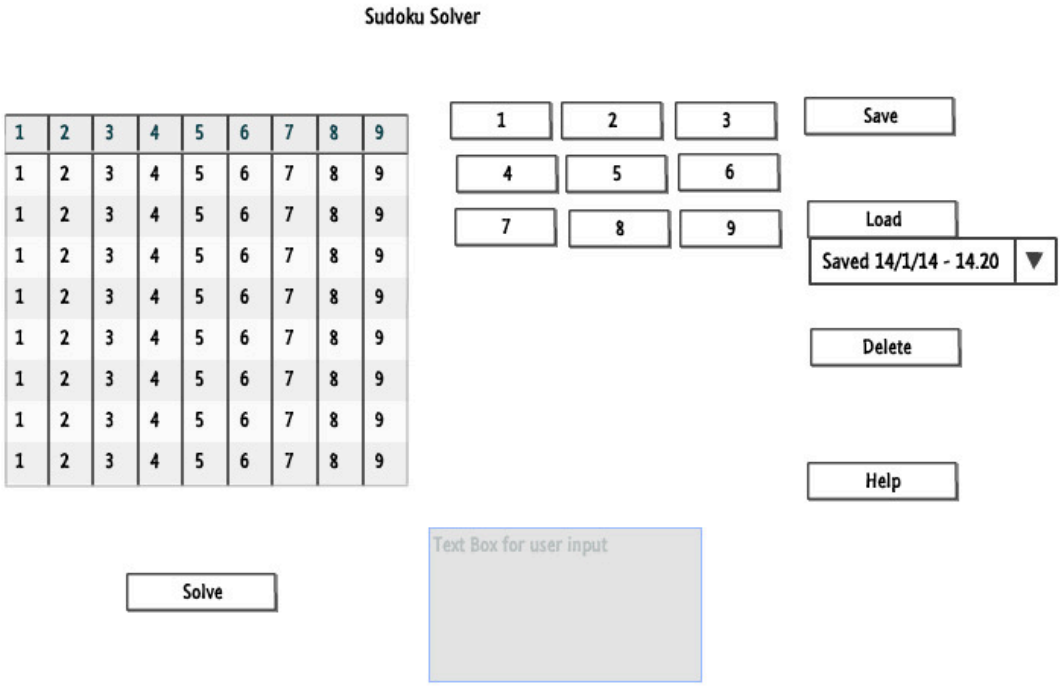
1.5 Evaluation Of Designs

After completing these heuristic trials it has encouraged me to make several changes within the design stages of my final project. There will need to be a few more functions to allow the user to get the best experience. The previous designs have been altered due to the heuristics. Therefore they allow the designer to take a step back from what they want from a system and the real reason it is made. I will be making these changes to my final design to incorporate them into the final report.





Technical Drawing Three Figure 3 _____ *18*



Chapter 2: **Layout Managers**

Layout managers are used to prevent user interfaces from changing dramatically when the window size is reduced or increased. They are needed so that the placement of a button or object position is not effected when the window is changed. There is a possibility that the button can be lost behind the window frame and a hidden part of a user's view. Another problem could be organising the buttons in a way that allows them to be changed, my project will be focusing on the grid and group layout.

Swing will automatically use the absolute layout, which means it will allow the object to be set statically, whilst setting the position of the object using integers to specify location. However this can cause problems when resizing the window later on in the process.

2.1 The General layouts

A brief summary of the general layout I could implement in my Suduko Solver are as follows.

2.1.1 Border Layout

The Boarder layout is used for the simple allocation of components to the areas within the window. It will include five main areas; the title top bar (page start), the 3 main information blocks (that allow the object to be set to the line start), centre and end and the bottom page end with the last bar. This is all achieved by using the following code[3]:

```
pane.add(button, boarderlayout.PAGE_START);.
```

2.1.2 Box Layout

The Box layout is used to display buttons in the same axis whilst making their centres or a set position inline. This allows buttons to be in a row or within the same line. The box layout arranges all the components inside in an ascending order and equal size, which is the desired size of the object to fill the space. The following code sets an object to a box layout:

[3] http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html Last accessed 7/11/2013

pane.setLayout(new BoxLayout(newPane, BoxLayout.X_AXIS));.

2.1.3 Flow Layout

The Flow layout puts the components within a line, following one after another. It is used to allow a series of buttons that are tightly organised and follow the same flow. This is achieved by adding a new object to the flow layout.

2.1.4 Group Layout

The group layout can be used to group components (like buttons) together by connecting the content, which involves adding them into groups. This is completed by using different groups for items and generic settings, which affect all of the items in that group. In this example, there is a content pane from a user interface built by window builder. It creates a group that uses the.addGroup function to add items, which group and present it using the .addGap function.

.addGroup(contentPane.createSequentialGroup() //creates a group that follow in a sequence.
.addGap(100) //the size of the gap between components.
.addComponent(btnSolve, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)// adds the button and puts it into the group layout using default sizes.

2.1.5 Grid Layout

The grid layout is used in this project as a way of keeping the buttons in a visual grid. This is done by adding components to the grid layouts 2D array, which allows it to display everything in the correct position including the specification of the grid location. To create the grid a new instance of the layout is made and then the components can be added to it.

GridLayout sudokuGrid = new GridLayout(0,2);
sudokuGrid.add(button("1"));

This method proved the best way for me to create a Sudoku grid. It involves filling in a 2D array of buttons, which the user can utilize to enter and select the integers used to play the games.

2.2 The Features Of Layout Managers

There are some extra features offered by the layout mangers, which allow the user to customize the visual appearance of the program. These methods are used within selected managers:

[4] Glue is used within layouts such as the box layout, allowing the user to add in a glue object between two components. This provides a gap between the components that can be replaced with a flexible link, which is unlike a rigid link that is unable to change. Unlike the glue function that allows the components be kept at a distance, this function allows the layout not to be affected if they are moved to other positions.

There is a function that allows the alignment of the components to be set to enable the layout to be viewed in a different way, which is done by:

```
button.setAlignmentX(Component.LEFT_ALIGNMENT);
```

Layout mangers are more commonly used because they provide more advantages than absolute placements. It serves in the design stages and creating the GUI, but also assists the resizable windows in the program. The layout managers allow the buttons to be dynamically changed if needed so they remain the same size and within relative distances to each other given the boundaries.

Layouts allow provide as much of a visual aid in terms of grouping the components and allowing them to be set quickly and within the correct placements showing them in the order they are added as well as positioning them with the correct glue in between each component.

[4] http://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

2.3 Advantages and Disadvantages

One of the biggest advantages of layout managers is that they provide a simple template for each window. The templates can be exactly the same on each window and (no matter the size) there will always have the same positioning. This helps to create a consistent and well presented user interface with minimal effort as the layout manager can handle a lot.

However, there are also problems that occur when using them and these limitations concern what each layout manager can actually allow. There are different managers that can be used for many distinct layouts. Yet the use of one could hinder the Developer, as there might need to be a use to keep a component somewhere that the manger does not permit. This can be resolved with absolute positioning but then the Developer loses the benefits of the layouts resizing capabilities.

2.4 Conclusion

During my experience of using a layout manager I found that when adding another component into the later stages you risk changing the rest of the layout. For example, within a button system, I featured 6 buttons that were perfectly arranged with correct spacing in-between. Later, when I added another button and a object next to the previously displayed buttons, the spacing between the buttons changed, which left the display looking dishevelled and untidy. The buttons were shifted much lower than expected. However this was easy to remedy by changing the code and the gaps.

Within the GUI program concept I have only used two of the layout mangers, the grid and group layout. However this will not be the case within the final project, as I will be expecting to use a few others to enable the program to look consistent and accessible for users.

Chapter 3: **Data Structures**

Data structures are concepts and processes for handing the data within a system and each provides its own benefits and reasons to be used. Using these structures allows programmers to use them to increase performance within their programs, which could include processes that allow effective searching and sorting. However, each type has its disadvantages.

3.1 Examples Of Data Structures

To explain data structures and what they are I will use the example of an array, which is a basic way to store data. It is a fixed size, therefore, only a specific number of elements can be within it, and the data is then held. However if the size of the array needs to be increased, the array has to be drained of its elements and re-created in the right size. Although there are other alternatives to using an array, for example, array lists, which are detailed later in my report. The functionality of a list (that helps the array problem) is that it doesn't have a specific size and it adapts to the size of the data as it's put in.

The example above proves that sometimes data structures need to be used in place of others to provide functionality, which otherwise cannot be there. Alternatively, it will provide a much easier way to write the program with the data structure that is best suited. In the case of this array, when the size of it will be changed throughout the program, the array list can be used. This will be treated the same way by using get and set methods:

```
ans = list.get(index);
```

This is Instead of an array, which is done by using:

```
ans = array[index];
```

It would cause problems for the user for the program to be without this structure because the old array would have to be replaced and the size altered, whilst also repopulating it. However this solution allows the program to work without those issues and this is what the data structures are for.

3.1.1 Linked Lists

Another example of a data structure is a linked list, which can be used instead of an array list. It gives similar functionalities that allows the set methods, delete and element functions to be implemented. However it does not provide a function for getting a specific item without traversing through the list. Using this as a linked list over an array list is provided it's not used to source a specific element. It's a much faster way to add and remove elements by simple traversing. Therefore, it highlights how important choosing the correct data structure is and having to know what the program will be doing to decide what is needed.

Throughout my concept programs and main project I will be using many different data structures to allow my program to work effectively and correctly. Therefore, the right structures have to be used and well executed. The concept programs have provided the foundation of what is needed with the final program and each has highlighted one or more structure.

3.1.2 Tree Structures

A tree structure conceptualizes that all data is linked and can be traced via traversing down the tree. Its method is to create by nodes that contain data within the object, which are then stored within the tree. Each node is connected to the next because it stores the next node within the current.

The use of a tree structure is to allow the input data to be easily searched or found, which can be done by adding the nodes that contain the data set and traversing though them. This can reduce search time, therefore each operation requires O(log(n)) time when n is the size of that data set. This data structure also allows efficient removal and insertion allowing the tree to be traversed to add a new node.

Dynamic Trees allow the process for creating a tree during runtime, which will be subject to constant change. It can be added to after the process of creation and this will allow the tree to be made larger. It can also be cut when needed and by using the board states that are no longer required it is possible to trim that branch and dispose of the unnecessary leaf. This highlights the features within the tree of a tree that allow nodes to be added, searched for and removed. These features are valuable to tree functions and this also allows limitations to be placed. For example, when a node is removed, it is known and recognised, and the tree will remake itself to shift the other nodes around the tree if needed.

The advantage of using a Tree is that the data set is easily entered into it, which (once within) can be sorted and compared to another. It does not matter if the tree is unbalanced because the dynamic tree allows the tree to be traversed quickly in O(log(n)) time. Therefore, it does not lose its functionality due to the use of the dynamic tree methods.

This process is more flexible in comparison to the arrays because when a new object is added, the tree is sized to the current number of nodes instead of a set number. For example, an array is set to an integer value, and when an object is removed or added a temp array has to be created to add the new values, so the new array is re-created to correct size. However this is built dynamically and will never be smaller or larger than needed.

3.1.3 Stacks

Stacks are an abstract data type, meaning that they are defined with a set number of operations. In this case, pop, push and peek allows them to be categorised as abstract, which is a form of simplifying their concepts because they can be limited and described with these functions.[5] They are a way of storing elements in an ordered fashion because there is a strict order to the stack on how elements can be inserted or taken from the stack. This can be beneficial when a program needs to be able to store elements but only when dealing with one at a time or when storing that element for later.

This is done by using the simple commands of push and pop, which essentially to input the element into the array one at a time. It is a last in first out data structure, which only allows the top element to be taken from the stack. The push method simply adds the element to the stack and the pop does the reverse where the element is then removed.

The logic in using a stack is due to its limitations actually helping the functionality of a program to be automated. For example, there is the Towers of Hanoi problem, which uses a stack to push and pop the elements or disks in this case.

3.1.4 Queues

A Queue is linked and relates very closely to a stack. However it is created using a linked list and the operations it uses differ slightly. The add operation adds the elements to the tail of

[5] Cay Horstmann Big Java Forth Edition Page 629-631

the queue and the other operations remain the same, peek and remove. This allows the reverse functionality between a stack and a queue, which neither are able to do without the operation. The stack cannot return its bottom element until that element is next and the queue will not return.

Priority Queues include elements that have an assigned number, which relates to the order the elements can be arranged in. This is done by adding each element with its priority number, which gives the order the data is output. It allows the items to be stored in order when they are added to the queue. The queue will then extract the minimum number in the stored elements when the q.remove() method is called.

This will be used to store the data, which will be assigned priority. This can be achieved by using a process which scores the most effective next move, so the board can be scored and stored as a priority queue. This means that only the highest score will be an output, which could provide a quick and easier way than comparing each board.

3.1.5 Heaps

Heaps are structured differently to trees because each sub tree is higher than the root. Yet, there are no restrictions on the value of each node and its position. In the case of a binary heap each parent only has two nodes and in other heaps there are restrictions on the amount of children each parent can have. Heaps are related closely to priority queue and these are defined as the abstract data type of a heap.[6]

A heap is populated by adding a node to the end of the tree, demoting the parents' slots if it is larger than the element to be inserted. This is done to move up the node, so that at the end, the new node will be in a position where both of its children are smaller than its element. However this example has been putting the smallest element at the root, which can also be the largest element as the root.

In order to remove the element at the top of the heap, you use the function that will take away the root node and move the latest node into the root. However this causes

[6] http://en.wikipedia.org/wiki/Heap_(data_structure)

problems as it likely that the heap will not match. It will have to be fixed by resorted it, which is achieved by adding a new element.[7]

An advantage of heaps is their ability to be versatile within using different data types. For instance, it is possible to use an array or an array list to store the elements, using the layers of the heap to be stored in the arrays. This increases its efficiently as the elements are set in their representative locations, which are predefined.

3.1.6 Hash tables

Hash tables are created using the hash function, which creates a code for objects that are different. For example, if there are many elements within a data set that are likely to be reproduced within it, then it is worth using a hash table that uses the hash function to create a unique code. It is important to remember that in some cases there are exception, but they are not common. Yet more commonly the function will be used within the hash table to be a position within an array that is limited to the size a user specifies. If there is a duplicate then it will be stored within a linked list with the other hash coded elements.[8]

To find an object within the hash table each element has to calculate its hash code and reduced modulo to the size of the table because this provides the location within the table. Each element becomes within the location check if they are equal to the given element. If found then x is within the set. To delete an element the same process is repeated. However, if it is found within the set it's removed.

The advantages of hash tables are their rapid increase in efficiency. This means (within the best case) it takes O(1) time to add, remove and delete the hash table elements. This provides a huge advantage to any program that has a large data and a set of related data that needs to be found within the data set.

It is possible to use this within the Sudoku solver to identify each board/game state that is stored in the hash table. This is a much faster process to retrieve the boards than using a tree or any other data structure.

[7] Cay Horstmann Big Java Forth Edition Page 652-663

[8] Cay Horstmann Big Java Forth Edition Page 652-663

3.2 Conclusion

These data structures will be a key part within my finial program. They will aid with the fundamental aspects of handling the data, whilst storing it in the most effective way possible. They'll also assist in outputting that data in the best way possible and (in some cases) in a system that allows it to be automated within the program, which will enable it to remain consistent and constant.

Chapter 4: **Design Patterns**

Design patterns provide the structure for a program, which is done by giving a concept or design a layout that allows it to work correctly. A design pattern does not specify code but is used to simplify a complicated problem by spreading it into defined classes or methods, which are predetermined by the pattern. There are huge varieties of patterns and many different designs for the same problems. This is due to the few limitations that are in place, as it just needs to work for the programmer and fix the problem, which often helps to make the code more understandable with a set layout the pattern defines.

Each pattern will define a structure. It will have to be designed (often) in conjunction with other classes and objects, whilst also having their inheritance defined to the programmer. Therefore it is made easier by knowing what needs to be made. They are designed in a specific way, so that if they do specify an object or class they will be open to change. They will not need to be set exactly because they are defined through each program being built for a different purpose.

Some of the design patterns that I have researched are linked to what I'll be needing within the Sudoku program. There will be several design patterns incorporated within the program. They will all be for their own individual reasons and playing on their unique strengths.

4.1 Using Different Design Patterns

4.1.1 Model View Controller

There is a user interface, that will contain the buttons, and this will be linked to the rest of the program. It is necessary to have a design pattern that will allow a separation of the user interface and its event listeners, which allow it to be handled independently to the other program functions.

The Model View Controller design pattern is split into three different parts, they are are seperated to provide the layout of the User Interface to be clearer so that there are not many methods within one class implementing an interface and relaying information from the other program stright to the view.

The view is that interface code that is generated to provide buttons and a the view for the Interface, this will also contain listeners to allow button presses to be recognised within the program.

The view is linked to the controller which takes the information from the view that a button has been pressed and activates the corresponding listener event that is a method within the controller, this allows the distict seperation between the code for the GUI and the code that is implemented within the program. The controller is linked to the view and the model.

The model implements the action that is requested by the button press, the view is not linked to the model instead the controller is the “middle man”. This allows the model to take the inputs from the controller and then implements the rest of the program to retrieve the outputs, that can then be parsed back to the view.

The sudoku has a view that will contain Jlabels with key listeners and event listeners, this amounts to a lot of possible instructions for each button that are within the controller to prevent clutter and allow a simpler way to pasrse data to and from the user interface to the program to be calculated and returned. In this case the imported board needs to be solved when the “Solved” button is clicked. The controller will register the click and run the solved method that will parse the nessesary data to the model. The model will run the CSP part of the program and return the solved board.

4.1.2 Flyweight Pattern

The flyweight pattern creates new objects, but ones that differ from each other. Similar objects are not created as they don’t need to be. For example, within a Sudoku grid at the beginning of the game many of the cells are not given a value until the user sets one. This means, at some point, there are only a few objects that will be set with values that differ from others. Due to this change there will only be 10 possible values, null-9, which allows the pattern to use each number as a pathway to create the cells by only storing the position of each cell. This benefits the program because it is more effective at reducing the number of objects created and decreasing the memory needed, which also increases better performance[9].

[9] http://www.tutorialspoint.com/design_pattern/flyweight_pattern.htm

Flyweight is designed by having a reference to every object that could be considered a flyweight object. For example, in the Sudoku puzzle there are only ten variations of what each cell can be. This means that the flyweight will have the ten different objects stored within a hash map. Therefore the only data needing to be stored is each cell that equals those objects. This will be used when creating the constraints considering the variables and domains.

The Sudoku solver is a puzzle that requires 81 instances of an object. In this case each cell within the grid will have to be created and this will represent the numbers that are input by the user. It is necessary for each of these objects to be duplicated so they can be handled in same manner throughout the whole program.

4.1.3 Composite pattern

Alternatively the Composite design pattern groups objects and treats them in the same way as one would be treated. This could benefit the handling of the variables as there could be many and a lot with the same values such as 0, as that is thier intial state. This allows there to be a group that can change, for example, rows, columns and objects with the same value can be grouped, which allows them to be checked against constraints.

This pattern's process is to put the objects into a tree where they can be added and removed from the group by using simple methods. However, this does limit the functions, yet still allows there to be a link between the objects and their values. With this process comes the ability to put the cells into trees.

4.1.4 Strategy pattern

The Strategy pattern allows the program to use a variety of independent algorithms, which can be applied to any object. It provides a structure that each object can be handled by. The desired methods that could be helpful in the project to restrict unnecessary checks taking place, which could reduce the time taken to complete constraints.

The sudoku solver will be able to use different algorithms to solve the puzzle and this parrtern aids the implementation of different algorithms, which are essential. Each constraint could be used in a different method and there can be a runner method that allows the object to only use the constraints it needs, which prevents them from all running coincidingly.

Within the Sudoku solver there are many data structures that are used in several ways to It would be beneficial to have a pattern that would allow the separation of the data structure from the logic, which would allow it to be independent.

4.1.5 Singleton pattern

There have been a lot of ways to handle and manipulate an object but there are design patterns that allow easy and effective ways to create objects. It is possible to incorporate a design pattern that creates objects but enable more effective ways to do so.

For example, the singleton pattern creates a class that can create an object every time it is invoked, which allows the process to be very quick and easy. It means that creating an object is simply calling the method. It is easy to then limit the number of objects, as the amount of them depends on how many times the class is invoked.

To link this to the Sudoku program will allow the correct number of cells to be created. Allowing other design patterns to be incorporated into this could be necessary to create the objects, using the flyweight pattern. For example, when it creates its objects it could use a singleton, meaning it's possible to get the benefits from both patterns.

4.2 Advantages and Disadvantages

Design patterns give a wide range of functions and assistance throughout a project, from the backend of the program to the GUI. This can really push a programmer to try new patterns to improve their programs. They can be used repeatedly within projects as each pattern can be used for many different uses and reasons.

However there are problems with design patterns. There are so many, that for each problem, there will be a pattern that is linked and can be used for the situation. The problem here, is that you have to find the right pattern, which can be quite labour intensive as it requires research or previous knowledge of the patterns.

Although, when working within a team each person will have their own experience with patterns and their way of programming. Within a group there can sometimes be too many ideas for design patterns that could actually harm the programs design instead of help it.

4.3 Conclusion

These design patterns are chosen to help with the programming and design of the sudoku solver, they each provide useful for the use they are implemented for. Including them within the program will be done at the design stages when a UML diagram will be made. They will help the process of coding by making it understandable, as well as helping the performance within the system by making it as effective as possible.

Chapter 5: **Constraint Satisfaction**

A constraint satisfaction problem is a specific problem that is best solved using strategies that relate to constraints. These are the limitations that are put on an object. The process of creating these objects is linked to what that object will be able to do to validate its value or a value it could take on. There are many ways to provide constraints. They are essentially a set of rules that the objects associated with it must conform to, which will provide the actions needed with the boolean outcomes from these operations.

Use of constraint satisfaction problems are often linked to logic puzzles like Eight Queens and Sudoku. This is because the games rules are all about limiting what values can be placed in specific locations, this is done to prevent collisions or duplicate numbers. There are many different ways to apply constraints and each program needs to have all of the rules in place so it can be applied correctly.

A constraint[10] is split into two parts, the variables and the domain. The variables are the values needing to be completed with the correct value and, in these programs, it will be a location on a board that needs to be checked if it is a correct or a false state. The domain is the parallel part of the constraint and contains a set of values that are the allowed for the variables use. If the variables within the constraint are also within the domain then the constraint can be satisfied.

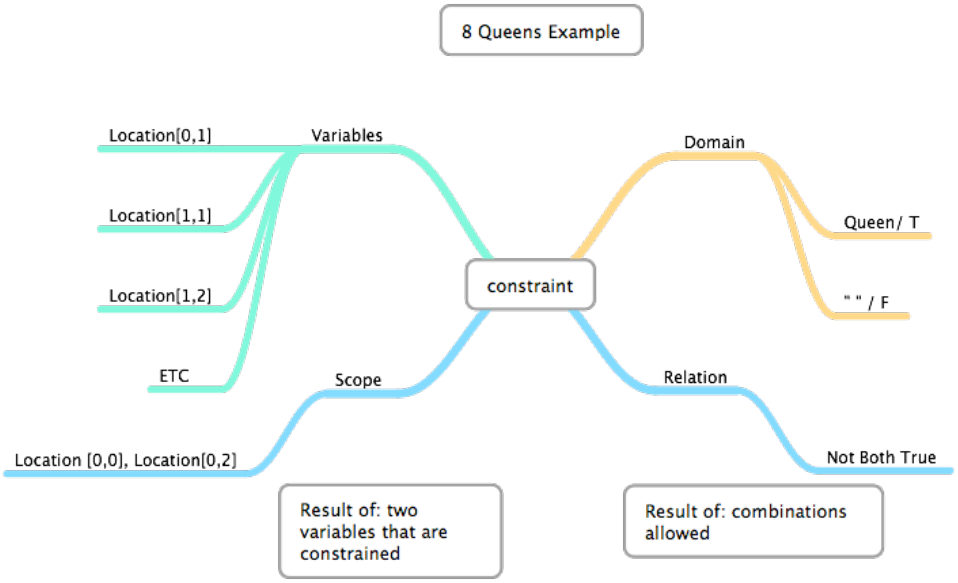
Each constraint has variables and in this example the variable is a Queen from the 8 queen puzzle. The queen has been placed in a location, which now needs to be checked. The method is using constraint satisfaction by using the domain this is given with the variable and contains the value that the Queen can take. If it is satisfied it can be placed as a Q otherwise the cell will be left empty.

Within the constraint there is, what we call, a scope. It is the relation between two variables that are constrained, (coming from the tuple) which assigns the variables to the scope. When the two constraints are matched together we call it a scope and these are then constrained to produce the relation. The scope being currently used is utilised by the relation and it this relationship, in terms of their values, which allows them to combine.

[10] Professor David Cohen. The Complexity of the Constraint Satisfaction Problem. ISG-CS Mini-Conference Slides http://www.rhul.ac.uk/computerscience/documents/pdf/csisgmini-conf/dac.pdf

5.1.1 Constraint Diagram

The diagram below shows this process related to the 8 Queen puzzle as an example to show the way P = (Variable, Domain, Constraint)



5.2 Constraint Techniques

Within constraint satisfaction there are techniques that can be used to increase efficiency. However, these are deemed deterministic, which means, that unlike a search, they will provide the same results each time. So given a particular input there will always be a constant output.[11] This proves to be very useful as it allows the computation to be made as soon as possible. [12]

To improve the constraint it is possible to select the objects that are no longer needed due to a variable being domain consistent. Meaning that within the domain of the variable there contains a value that is matched within the constraints. [13] Therefore this branch, and any others it continues to follow, is not needed because it's not valid. By pruning the domains as much as possible allows the efficiency advantage without creating impossible boards.

[11] http://technet.microsoft.com/en-us/library/aa214775(v=sql.80).aspx
[12] Discrete Mathematics & its Applications, 6th Edition. by Kenneth H. Rosen. Page 836
[13] http://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/CSP3.pdf

5.2.1 Backtracking

I have used a method called backtracking, which unlike forward checking and arc consistency, (discussed later in the report) does not have any propagation. This means that there is no reproductions of the states and it is the same board that is edited and deleted to find the desired state. Backtracking works by searching and if the outcome matches the desired output then it passes.

Backtracking is a recursive algorithm used to find a desired output through use of the brute force approach. It does this by starting at the root of the tree and following a consistent path until the end of that path within a leaf. When the leaf is evaluated into the correct output the algorithm stops and the value will be found. Or an incorrect node is found and then the process is repeated by going back to the parent of the leaf and following and checking another node. This is done until the full tree has been traversed and there are no correct nodes on that tree, or that a correct node is found and the recursion stops. [14]

This is the backtracking algorithm I've used within the 8Queens program:

```
private boolean backtracking(int col) {
        int row = 0;
        if (col == boardSize){
                return true;
        }
        else{
                boolean attempt = false;
                while((row < boardSize) && !attempt){
                        if (check(row,col)){
                                row++;
                        }
                        else {
                                board[row][col] = 1 ;
                                attempt = backtracking(col+1);
                                if (!attempt ){
                                        board[row][col] = 0;
                                        row++;
```

[14] http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html

```
                    }
                }
            }
            return attempt;
    }
    }
```

This algorithm uses the current attempt to see if this passes and if it does not pass then the attempt is withdrawn and another attempt is used.

5.3 Arc Consistency Algorithm[15]

Include description of algorithm here!

Arc Consistency is different because the domains are reduced and there are propagates of each state. The Generalised Arc Consistency algorithm is used to simplify the constraint satisfaction problem by continually iterating through the constraints and their implications of the variables one at a time. The difference between the generalised arc consistently and arc consistency is the use of non binary constraints using tuples of variables instead of a pair.

The algorithm is displayed below with a 'line-by-line' commentary:

5.3.1 Generalized arc consistency algorithm[16]

1: **Procedure** GAC (Variables, Domain, Constraints) [17]

2: **Inputs**

3: *V* // a set of variables

4: *domain*// a function such that *domain (X)* is the domain of variable *X*

5: *C* // set of constraints to be satisfied

6: **Output**

7: //arc-consistent domains for each variable to cut down domains that increase efficiency.

---

[15] A filtering algorithm for constraints of difference in CSP's by Jean-Charles REGIN Pages 362 -367

[16] http://artint.info/html/ArtInt_79.html

[17] http://www.youtube.com/watch?v=Ge-XiX_tP4c - Doug Fisher's video on Generalized Arc Consistency

8: **Local**

9: $\boldsymbol{D}_X$ //is a set of values for each variable *X*

10: *TDA* // is a set of arcs

11: **for each** variable *X* **do**

12: $\boldsymbol{D}_X \leftarrow domain(X)$

13: $TDA \leftarrow \{\langle X,c\rangle \mid c \in C \text{ and } X \in scope(c)\}$ // For all the variables create the scopes, each variable within the domain creates the scope.

14: **while** ($TDA \neq \{\}$) // While the TDA set is not empty, meaning all the constraints have not been satisfied. When they have stop.

15: **select** $\langle X,c\rangle \in TDA$; // For each constraint within the TDA

16: $TDA \leftarrow TDA \setminus \{\langle X,c\rangle\}$;

17: $ND_X \leftarrow \{x \mid x \in \boldsymbol{D}_X$ and some $\{X=x, Y_1=y_1,\ldots,Y_k=y_k\} \in c$ where $y_i \in \boldsymbol{D}_{Yi}$ for all

18: **if** ($ND_X \neq \boldsymbol{D}_X$) **then** // if the new domain is not the same as the old domain due to it being reduced.

19: $TDA \leftarrow TDA \cup \{\langle Z,c'\rangle \mid X \in scope(c'), c'$ is not $c, Z \in scope(c') \setminus \{X\}\}$ // Within the TDA set now that the domain has changed it needs to be considered again so that it can be checked due to the new domain changes within a constraint that is in its scope.

20: $\boldsymbol{D}_X \leftarrow ND_X$

5.3.2 Evaluation

This can relate to the Sudoku solver: There are several different ways to create each board, store them and trim the branches of the boards that are not valid due to duplications. However this process can cause unnecessary processing, as each board will have to be tested until one returns false. Although the branch is trimmed it will still have been produced.

This algorithm prevents redundant boards being created as the given domains are reduced to prevent propagation. Instead of trimming the tree the process has prevented them from being created. Therefore, the constraint will increase the efficiency of the program.

5.4 Dynamic Variable Ordering

A process that uses the most constrained variable to be the first one utilised. Its purpose is to allow the dead ends to be discovered as early as possible. Also, without needing to be considered from other variables, this effective heuristic cuts down the search space. [18]

Dynamic ordering means that the choice of the next variable used will be dependant on the state of the search. This is found using forward checking, which is the opposite to backtracking. It means that the current domain is dependant on the previous and current constraints. The reason for this is to use the best possible variable in order to increase the efficiency.[19]

The concept of dynamic variable ordering is to compare all the available variables and control which constraints will be compared first. Using the variable that has the most constraints does this. If the number of constraints is high then it is likely to produce the highest number of fails when tested, as it allows the most constrained. Its function is to get as many of the failed tests done in the early stages so that they do not have to be replicated later on in the process. This will improve the efficiency when searching and creating the rest of the board. [20]

---

[18] www.ics.uci.edu/~dechter/courses/ics-275a/spring.../chapter5-09.ppt slides 26-31

[19] http://ktiml.mff.cuni.cz/~bartak/constraints/ordering.html

[20] http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf

This could benefit the Sudoku Solver because it'll allow the board states to be compared and searched in an effective way. Therefore the states that will fail are found as early as possible. It will be used when the next cell needs to be populated within the grid, using the tightest constraints on the cell to produce the failed states. Please see example below:

```
Cell = variable
Column  = {2,4,3}
Row = {1,2,3,4,5,6,7,8}
Square = {1,2}

For loop {
Constraint 1 = ( cell.value != row.cell[i] && cell.value != column.cell[i])
Constraint 2 = (cell.value != square.cell[i])
}
```

Constraint 1 has more constraints within it, so the dynamic variable ordering will use the variable that has the most constraints. By using the constraints, tests will be able to be run. Failing the tests means that the possibility of that option is ruled out, which (in this case) means that the most restrictive test, constraint 1, can be used and constraint 2 might not even be considered. [21]

5.5 Conclusion

Constraint satisfaction is essential within the Sudoku solver. It will enable the process of board handling, which entails the constraint finding the next cell by using a tree that contains the options. However this will have to be restricted by using these algorithms. It makes searching more effective by reducing the options, which only allows valid boards to be sourced.

[21] http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf

Chapter 6: Complexity, NP Hardness and the Big O Notation

Changes need to be made

-NP Hard Problems

CSP is NP-Hard Proof

.....

Each program has a runtime. Alongside this comes the complexity of the algorithms, the usage of the program, and how efficiently it does its job. There are cases where programs are not able to perform a task due to the constraints on time and the speed in which a problem can be solved.

6.1.1 Complexity

We explore and define the complexity of an algorithm to understand the amount of time it will take to solve a problem. It's important to follow this procedure in order to get an idea of how long the algorithm will take. This can be done using a timer, which can be used to time a small section of the algorithm. For example, a recursion, and this can be incremented by the amount of times the algorithm can be run in a worst-case scenario. [22]

Complexity is not an accurate value and consists of a multiplication of the scenarios. It uses this number to suggest a value to the user about how long the algorithm will take. This is also a way to view how long the computer will demand to complete the task. In some cases a faster computer can help this situation.

An example of this could be the traveling salesman problem.[23] It describes the scenario of a travelling salesman, who jumps between several cities. The program is used to find the shortest route to travel to each city, whilst walking the smallest distance. However, each city has to be checked against the others before each distance is calculated. This problem takes factorial time to be completed and 10 cities will take 10! Time.

6.1.2 Big O Notation

Programmers use the big O notation to specify the number of visits , in order n analyses the performance of an algorithm. The use of the O notation is used to describe how the run time scales in relation to the input of the algorithm. An example of a Big O Notation is O(n!). This actually means, that for every possible element of n, it is explored.

[22] http://www.cosc.canterbury.ac.nz/csfieldguide/student/Complexity%20and%20tractability.html

[23] Time and Space, Designing Efficient Programs. Adrian Johnstone & Elizabeth Scott. Chapter 5,Page 68 – 75

6.1.3 The Class NP

Within the project aspect the main aspect to look at is the problem large sets of possible nodes and the constraint such as time[24]. It is usually beneficial to find the best possible solution to a problem. However, this is not always possible, which is proven by the Traveling Salesman Problem. Therefore it is possible to categorise problems by using a threshold to limit the distance travelled between the cities. This limitation can be a used to enable the program to run more effectively than the threshold. If it is better then the decision problem then it is true, or else it is false.

There are problems that can be similar and their complexity can cause them to be unfeasible in order run in any practical amount of time. However, there are "tractable solutions"[4] that can be adapted to others, and this group is NP-complete. "Complete means they can all be converted into each other"[25] and contains the hardest problems within NP. To be within the NP there needs to be a polynomial time algorithm, which means that all the algorithms within NP have a polynomial time solution.

6.1.4 NP Hard

NP hard are a set of problems that are difficult to calculate. They are within the set NP-Complete but not within NP or P, which is due to the NP-hard set of problems being within the hardest NP-complete problems. The reason for having this set is to distinguish the sets that are hard to be solved. For example, the Traveling Salesman Problem is NP-Hard even when it's cut down to one node to be visited at once.

6.1.5 Sudoku is NP-Complete[26]

Proving that the Sudoku puzzle is within NP-complete[27] was documented in 2002. To summarise these findings, the puzzle contains a n2 x n2 grid that is split into n x n squares, which then contains n values. Each search is done on a value. In order to find the correct answer, if there was to be a search on every cell, there are a minimum of n-1 cells in each block that could be searched, which all have different n values. This could take n·n! Time.

[24] Algorithms and Complexity 2, CS2870 By Gregory Gutin February 16, 2013. Page 93
[25] http://www.cosc.canterbury.ac.nz/csfieldguide/student/Complexity%20and%20tractability.html
[26] Graph Algorithms and NP-Completeness. Kurt Mehlhorn. Page 171- 211
[27] http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/SIGAL87-2.pdf

This proves that the algorithm is hard. However there is a case in which the Sudoku size can be reduced for n = 3, which can then be solved by using a backtracking method[528], known as a Latin Square. This proves that it belongs in the NP class due its ability to be solved in polynomial time. I can apply this to the Sudoku Puzzle, as it can be a section of a grid. It can be performed n times, so it will now be O(n!), which is a vast improvement. Yet, it still proves that have a poor scalability.

6.2 Conclusion

The information I've discovered about the process of solving the Sudoku puzzles will now change the way the program will be designed and made. I have realised, by trial and error, that searching each cell to get the correct answer isn't practical, as there are too many iterations that will have to be made. It has also given me a secondary program to look into, which is the Latin Squares algorithm.

[28] http://www.math.cornell.edu/~mec/Summer2009/Mahmood/More.html

Chapter 7: **Tic-Tac-Toe Program**

A concept program made was a Tic-Tac-Toe game; the reason for this was to be able to understand how to program a game using a game tree structure and to have a basic knowledge of artificial intelligence. The game of Tic-Tac-Toe is very basic, it is a two player turn based game that both payer have a token (O or X) and this is placed within a 3x3 square grid. The winner is the player that gets 3 tokens in a row, and a draw if no one can get 3 in a row.

7.1 Artificial Intelligence

When a user plays the computer game it is necessary to be able to play an artificial player, this is done by using all of the possible moves and choosing the best to be the placement that will challenge the user. The process for doing this is simply using the game rules to get all of the possible moves and using the best which is designed to beat the user, this is selected with another set of rules.

Using the game rules it is possible that each game move can be simulated within the program and the best move is selected by finding out what options can be played once that token has been placed, this is done using steps of if this is placed then the next possible placement if it is good then it scores higher and if it will do nothing then it will score nothing, if it hinders the player or even makes them lose then it is given minus scores. This way it is possible at the end to use the highest scoring move due to the following moves providing the best chance for the user.

```
Node highScoreNode = new Node();
        //high positioned nodes are given a low score
        highScoreNode.setScore(-1);
        int[] highMove = new int [3];
        Node lowNode = new Node();
        //low positioned nodes are given a high score
        lowNode.setScore(1);
```

The score is very important to be assigned to that node so that on the game tree which is used to store all of the boards and allows it to be traversed in a way that when the score is higher than the any other node then that is the next best move. There are algorithms that check each of the nodes and score them. The position from the previous node on the game tree to the best scoring one is the next move location which can then be sent to the setPosition(0,2) method.

7.2 Game Tree

Game tree's are a tree that store game states, in this case it starts with an empty board and each token that is placed can be placed In 1 of 9 squares that are within the 3x3 grid, this allows the game tree to start with an empty board at the root and have 9 options with the depth 2, this contains the options where the first players token has been placed in each of the 9 possible squares.

The next depth allows the second player to be placed in of the 8 remaining squares that the first player did not place the token. Creating a board of each option does this placement, where the token could be placed and then they are all scored. To which the best scored board is the next best placement.

The disadvantage of storing all of the possible boards is that a lot of them will be redundant and are not needed due to some of them being the same there are other reasons to filter the other boards, for example getting rid of the redundant boards that are used when a parent has ended due to it resulting in a win or draw and then the boards after do not need to be stored.

7.2.1 Boards

The number of possible boards calculates as 3^9 is the number of boards that can be considered because there are 3 states to each cell (X, O, Empty) and there are 9 cells, this number is then totalled to 19,683 boards.

However the number of possible games is the number of possible ways to place the tokens within a board, this is calculated as 9! Possible games as there are 362,880. To reduce this number factors have to be taken into place to stop the board production if there is finishing sate, i.e. someone wins.[29]

[29] Steve Schaefer (January 2002). "MathRec Solutions (Tic-Tac-Toe)". MathRec.org

7.2.2 Duplicate Boards

The way to reduce the game tree is to access what is needed and what can be reduced. Obviously cutting out the redundant boards is possible as well as boards that are duplicates of each other, there is a way that a board is the same as there are not many possibilities. The first is to think of each board as a symmetrical cell where when the values of cells around the board it is possible that another board will be the same as that and this will cause the duplicated board redundant. The placements of the values within the board can be moved.

These four boards are cases where they are all the same and when rotated they will give the same case in terms of scoring and anything else, to reduce space a hash table could be used to identify all the boards and each is given a unique ID number so that when a board like this what is identical it will only be included once and not stored several times.

X	O	-
O	X	-
-	-	-

-	O	X
-	X	O
-	-	-

-	-	-
	X	O
	O	X

-	-	-
O	X	-
X	O	-

7.2.3 Symmetric Boards

A way of reducing boards by comparing new boards to the already considered boards and the chances they are the same this is symmetric, and an example of this would be if a board only had 3 values both X's are placed in the top left and top right corners and the O is stored in the centre cell. The board is symmetric as the board has the same placement on both sides; this means two boards do not need to be stored independently as they are the same.

Once these factors have been considered it changes what the number of possible games as the symmetries and rotations the number of games reduces to 26,830.

7.2.4 Positions

Each position within the board will have a preferred positioning, to be able to simulate the AI of the computer player it cannot just add in the top left most cell and that be the best position it needs to be done in a way where each move is given a preferred position, this is done by following a set of rules:[30]

Win- Complete a 2/3 line

Block – If user is going to win then block last value

[30] http://en.wikipedia.org/wiki/Tic-tac-toe

Split – create two rows with a 2/3 line

Block Split – block split

Centre – enter value in centre cell.

Corner- place in opposite corner of user

Empty corner – place in corner

Empty row – place in middle of the row

These rules allow the program to look at the whole board and decide what is the most effective move to be placed and this way increases the chances of winning.

7.3 Algorithms

To provide a next value to be placed within the board an algorithm is used to find the right value by checking the current board and returning the best option, there are several ways of doing this and they range in accuracy of the best possible value.

7.3.1 Brute force

This algorithm will check every possible value by using the processing power of a computer to do so, every possible value is checked by using the value and comparing it to the constraints, if it satisfies then it is allowed to be placed within that cell, If not the next possible value is then considered and this is recursively done until the right value is found. This will allow all of the possible boards to be created, and as long as the correct constraints are specified it will allow the algorithm to alter the results and will not accept the wrong values.

7.3.2 MiniMax

Used to calculate the best possible move for the computer and the player, it scores each of the moves by using the current player that is taking their turn and they are given the priority by choosing a cell that is the highest score.

The MiniMax algorithm can be described by a list of instructions:

If game is over, return the score of current player

Else Get list of new games for every possible move

Create list of score

For each state add MiniMax result of state to the score list

If current player is X, return max score from list

If current player is O, return min score from list

This recursive algorithm uses the current player as the perspective for that initial run of the algorithm. And it will return the move that is desired to provide the best move to beat the opponent. This however provides an unbeatable program due to tic tac toe being a game that if played properly without mistakes it should always end in draw or win.

To improve the MiniMax algorithm by being able to prevent or stop the opponent from winning, however it does not within the current algorithm as it does not consider to block or prevent the other player. This is why the depth is included to adjust the score by the depth, as each level is incremented by one to allow it to adjust the scores within the depths making it more likely to choose the desired board.

7.3.3 Alpha-Beta Pruning[31]

While MiniMax calculates the players best move with the opponents moves within consideration, the MiniMax algorithm is quite naive and does not notice boards that do pose a threat to the current player, and the algorithm will continue to produce tree nodes connected to dead nodes that contain false boards as they do not provide a solution.

AlphaBeta will search for the best move but will stop when there is a state that is deemed to not matter to the rest of the search. This will be more efficient than the MiniMax algorithm due to it greeting the tree that extends further than it needs to due to boards that are redundant for example boards that contain a win and this will allow fewer boards to be created and reduce redundancy.

An example of how this algorithm can dramatically change the amount of processing that needs to be done to consider the game states is that a board of only two values:[32]

X	O	-
-	-	-
-	-	-

MiniMax : explores 8,232 game states

AlphaBeta : explores 47 game states

[31] Algorithms in a Nutshell, by George T. Heineman, Gary Pollice, and Stanley Selkow pages 217-221

[32] Algorithms in a Nutshell, by George T. Heineman, Gary Pollice, and Stanley Selkow pages 223

The checking of these game states actually provides the same answer that the best possible value is to put X into the middle cell, however this is the best case example where the values start with the best solutions are the first to be checked, however this will not always be the case.

7.4 How this relates to Sudoku

With a Sudoku Project to do next the use of this concept program is to get used to programming a style of program that constraints a board object and the uses it has to this style of problem, there are a few common factors between a Sudoku and a tic tac toe board as they both contain cells of 3x3 however the Sudoku also has 9 of these by the concept of checking the values within the cells and using constraints on them to prevent the wrong values from being placed.

Finding the best move was the main objective within this program and the same can be true for the Sudoku puzzle as each value needs to be the correct value so that the puzzle can be satisfied completely.

Chapter 8: **Techniques used by human Sudoku solvers**

8.1 Introduction

Around the world Sudoku puzzles are solved by millions of people, there are many ways that people can solve it by using their basic logic or more complicated processes that are proven to aid and move a step closer to a desired completed board. Users will often only use the very simple methods and often will struggle with anything harder as they will need to use the other methods that simply scanning will not provide an answer.

8.2 Background

How Sudoku became a well known and widely played puzzle game starts with its creation that begun in 1783 when Leonhard Euler the Swizz mathematician created "Latin Squares" [33] which will be discussed later, are the foundation of the grid that contains a Latin Square that is divided into nine and provide some of the limitations that are the constraints within the Sudoku. It became accessible to most people and published as a puzzle game, first in America and then later within the Times Newspaper where it became a hit due to its simplicity and ability to be addictive.

8.2.1 Latin Squares

Already mentioned, Latin Squares are and n x n grid containing numbers that are only allowed to appear once on each row and column.
Here is a Latin Square of size 6 It is possible to see here that each row and column only has one of each number.

0	1	2
1	2	0
2	0	1

This allows the grid to then be used in the same way as the Sudoku but with a distinct change of using a full 9x9 grid with 9 sub-grids that must contain numbers 1-9. This changes

[33]http://www.theguardian.com/media/2005/may/15/pressandpublishing.usnews

a lot, it is not possible for the grid to be a true Latin Square as it cannot have a normal sequence instead it is further limited and by this it does make the puzzle significantly harder. It is also important to mention the 8 Queens program here that was also created as there is a correlation to the Latin Squares and an 8 Queens problem where in that case a Queen must be placed in a cell where no other queen is on the same row, column and diagonal line. Here is an example of a Sudoku:

9	1	2	3	4	5	6	7	8
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7

As you can see the grid does not replicate numbers within columns or rows as well as within each individual sub-grid. This is what makes a Sudoku and these constraints allow the publishers to part fill some of the grid with premade numbers and this is then possible to be completed and must be done in the way it was designed to allow the correct constraints to be satisfied.

8.3 Techniques used by human Sudoku solvers [34]

To use the puzzles that was published in books or newspapers there needed to be different levels of Sudoku's, providing fewer predefined numbers did this. Allowing the users to find the puzzles harder as conventional methods of solving it would often are not enough and other strategies would need to be used.

Methods used by humans to solve Sudoku's range from simplistic to more complicated, here are some of the techniques:

8.3.1 Using Pencil marks

Not exactly a technique however it does allow the user to add their own workings out to each cell to make it easier to use. This is when the user puts in temporary values of what the cell could be at that time and will change a lot often.

[34] http://www.decabit.com/Sudoku/Techniques

8.3.2 Crosshatch Scanning/Naked Single[35]

The most simple method, using all the other numbers in the current grid to see if any can be filled due to the limitations of only one number can be placed within that cell. In this case it could be using the restrictions from the column, row and sub-grid, meaning there could only be one number that could go in that cell.

8.3.3 Hidden Single

Much like the previous but uses the constraint of the row, colour and block must contain numbers 1-9. This could be the case if there were already some numbers defined within a row, column or block and the last must be a number that is not within the set currently. Most human users will use these single methods. [36]

8.3.4 Row & Column Checking [3]

When a number is possible there is only one instance of it within a block, as well as the column and row that it is placed, this means that no other block can have a number within it that is the same on that row. This limitation to the rows and columns can aid in reducing possible numbers.

8.3.5 Elimination of Subsets

When a collection of cells have the same candidate numbers (possible numbers) but a pair or more have only a specific candidates that only appear in those cells all other candidates are not considered from these cells due to these cells being the only ones that could possibly contain those numbers so the other candidates are meaningless. [37]

8.3.6 X Wing

Four cells that contain the same candidate, if their positions are within 2 rows and two columns then this allows the X Wing technique to be used, because the 4 cells cross over in terms of rows and columns their candidates are then eliminated as they would be unable to have the same number as they are in line. [38]

---

[35] http://www.stolaf.edu/people/hansonr/sudoku/explain.htm
[36] http://www.sudoku-solutions.com/solvingHiddenSubsets.php#hiddenSingle
[37] http://www.decabit.com/Sudoku/NakedSubset
[38] http://www.decabit.com/Sudoku/XWing

8.3.7 Swordfish

A technique much like X Wing however it is a bit more advanced as it uses three rows and columns, this is a way of eliminating large numbers of redundant candidates, if all three rows have multiple cells that contain the same candidates within the same column then they are eliminated. [39]

8.3.8 XY Chain

When there are a group of cells that are all linked, meaning they are all within a sequence where the first is on the same row as the second and shares a candidate, as well as this a third cell is within the same block and this also contains a shared candidate with cell two. This creates a chain and can continue, if the chain is created and the end point of the chain is the same then the candidate from the endpoints can be removed [40]

8.3.9 Comparison of Methods

These techniques are very useful however they will not be able to be used every time the puzzle is solved it will depend on what the layout of a Sudoku is and only used when they are available, often determined by the pre-set cells due to the complexity of the Sudoku. The harder the puzzle the more likely one of these techniques will need to be used so that it can be completed.

8.4 How Programs Solve Sudoku's

8.4.1 Backtracking (Recursive)

Much like a human a program can check the basic constrains that are within the Sudoku and check for singles using a version of the cross hatch scanning by looking at the rows and columns and checking, to do this is checks for the candidates within the constraints for that cell and if it has none then it moves to the next cell, if a candidate is found it is entered into that cell temporarily until it is confirmed, this then recursively executed until the grid is full of the numbers and all constraints have been satisfied, this means the Sudoku is solved.

[39] http://www.angusj.com/sudoku/hints.php#sordfish
[40] http://www.decabit.com/Sudoku/XYChain

8.4.2 Exact Cover

This process is a very effective way to complete a Sudoku puzzle by creating a matrix containing all of the data needed to complete the puzzle. The matrix is broken up into four sets of 81. The first set represents one of the cells that are within the Sudoku so there are 81 and is used to show that a number has been placed within that cell.
The next set is defined by the 9 rows and the possible numbers inside of them (1-9) this allows each row to be defined by any of the numbers it could possibly be.
The next set is the same for the 9 columns and the possible numbers within them (1-9) the columns will not be set to the correct number using the possible values.
The final set are the labelled by the 9 boxes and the possible numbers within them (1-9) as each sub-grid needs to contain one of each number this process allows all to be tried if needed.

When a number is placed it is represented in all 4 of the correlating places within the matrix, this way each entry can be clearly positioned where it is with each row having a cell for each possible number and a search tree is used to find the missing numbers. [41]

8.4.3 Brute Force

The brute force algorithm uses the process of going to each cell and assigning it a number 1 for example and checks the value against the constraints, if it could be true then that cell is set to one and move onto the next cell where the same process happens. If the constraints do not permit the use of the number considered for that cell then the number is increased to 2 and the check is made again. If after all possible numbers have been considered then the cell is left empty and is set to a priority of 2. This means that it will need to be given the same treatment when the next pass is made. This process is run until the last number is found.

8.5 Humans vs. Computers

Sudoku is used as a puzzle game, however humans do struggle with the puzzle due to its wide range of possibilities and restrictiveness due to the constraints that provide the rules of the game, however humans are in general slow at completing these puzzles especially when compared to the time a computer program takes to complete the same grid.

The processes used by humans are slow and ineffective however there is not much that can be changed do to the limitations on how much we can compute within such a small

[41] http://www.ams.org/samplings/feature-column/fcarc-kanoodle

time and at the same time. The scanning technique which requires the user to look around the grid to find a cell that could be a number, this scanning does waste time as it is possible that this process might not prove effective in this case.

For a human to advance and be faster and even complete much harder Sudoku's they need to know the techniques that will be needed to be able to find the possible numbers and be able to place their values in the cells. Techniques that would not initially be thought of by a player would be X Wing, it uses a very straight forward process that makes sense, however when the user is solving the puzzle with a moronic approach they would not consider this.

Computers on the other hand are much more effective time wise with their completion of the grid, however saying this there are a variety of algorithms that can be used backtracking and exact cover being a few, these both efficient and would allow a Sudoku to be solved as long as it was a possible grid.

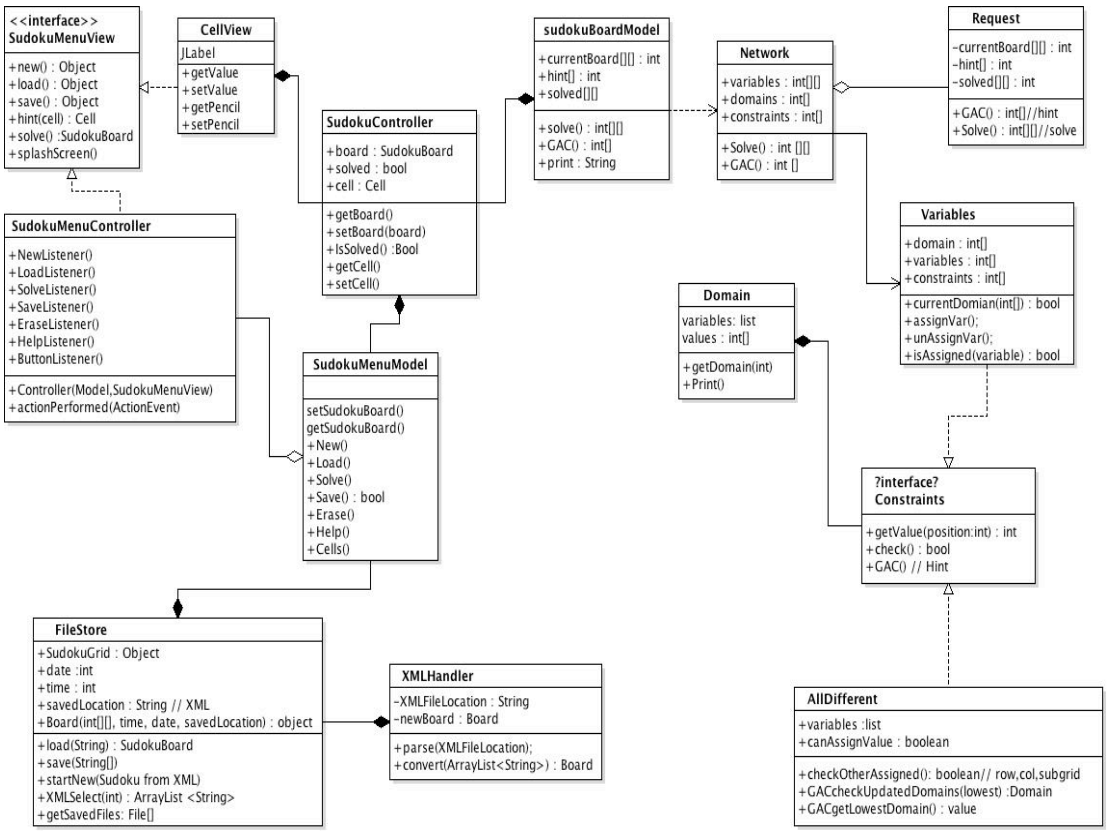
These algorithms are a lot faster to complete the puzzle however this is due to the ability to create large matrixes to hold the data, or the extra processing speed to continuously check each value for every cell. These processes are only as effective as the speed that the algorithm can process all of the data that it needs to, in backtracking this is very little data but has to be replicated many times. With exact cover it is the whole matrix that needs to be computed.

Chapter 9: **Design & Refactoring**

The design of the project has been constantly changing as the concept programs have highlighted new routes and bought increased my confidence with the ability I have to program something without being afraid to branch out and try new techniques, as well as create my own.

Designing the Sudoku program started with a series of UML diagrams that describe the project and its method of doing so, using the classes that are intended to provide every function that is needed as well as mapping out the strategies and to be able to view what needs to be made within the programming stages.

This is the UML diagram that I have created to layout the program and this has allowed me to view what needs to be programmed in order to have a fully functioning program. There is a description of each class below.



9.1 Description of Classes

9.1.1 <Interface> SudokuMenuView Class

This class will be created using Google's WindowBuilderPro that will allow me to quickly implement everything that is needed like buttons and creating auto-code that speeds up the process, it has added functions of adding layout managers. The use of the View is to give the user everything they need to interact with the program, by using actionListeners and keyListeners this could be from the Sudoku Grid where the labels represent the label values, as well as this there are buttons that provide functions that interact with the either the cell values by erasing the values, or loading a new board from an XML are just examples.

9.1.2 SudokuMenuController Class

The controller is used to provide a route to go to implement the listeners actions that are desired, for example if the program wants to display a value in a cell it needs to tell the model that is parsed though the controller and then to the view to be displayed. This process of keeping the listener actions within the controller allows the view to be less cluttered and contain only methods that are needed or provide a function within the view.

9.1.3 SudokuMenuModel Class

This is the link from the program and its inputs and outputs to the MVC that handle the view, this is used to provide the actions to get the values or implement the correct methods if a button is pressed, for example the load button is pressed and a XML date is selected to be loaded, the model passes the date and will call the fileStore class load function.

9.1.4 FileStore Class

File handling is a part of the program that needs to be implemented by the buttons on the view and the forms that allow the user to select the date of boards to be loaded, however this array is used to simply what needs to be taken into account by using methods for each button call.

The load method requires the string of a chosen XML file which was saved, this allows the correct XML to be loaded and also it then does not need to be stored within the program making the program more efficient as it isn't holding many saved files within its own memory.

Save method turns the current board displayed in the Sudoku grid in the view and stores it as a string that can then be passed to the XMLHandler that can export that string of integers into a single XML file.

StartNew loads a random pre-set board that is stored in a folder separate from the saved files, this is created by using the XMLHandler to import the file and then display them on the View. getSavedFiles returns an array of the current files that are within a folder that are saved by their date, this can then be presented to the user to be selected when a board is requested to load.

9.1.5 XMLHandler Class

Designed to parse the XML file and use the <Sudoku> set </Sudoku> placement to be able to correctly identify the placement of the required integers, this would allow the XML to be quickly scanned and only the Sudoku headers will need to be identified to retrieve the set. This will then be converted to a string and given back to the FileStore class that has requested it.

The Build method will take an array list containing items that are within the Sudoku, these will then be grouped with a date of saved in the form of a string, this would allow the file name to be the string date and the data within will consist of the saved board that is stored within an xml.

9.1.6 SudokuController Class

This class will create a board object that consists of Cells that contain the integers that put together the board, these integers will be saved within cell objects that are created within the CellView class and each board will contain 81 of these. The controller is used to set the board cell values as well as retrieve them.

9.1.7 CellView Class

Links the cells to the view as these are created within the view and are set using the controller. And this class allows the text of the labels to be set and the process of setting them. The Cell view observes the model so when the model is updated so can the CellView class, this allows a change whenever it is needed so that the board will also be updated.

9.1.8 SudokuBoardModel Class

Model for the boards implement the methods that can be used on the board, for example this would allow the algorithms to be launched by using GAC or solve. The board can be printed so that it can be viewed when testing.

9.1.9 Network Class

A virtual network is created to deal with the constraint satisfaction aspect of the program that just needs to be parsed the board to be used and then the applications of the algorithms can be applied, this can be done using the methods that will use the Generalised Arc Consistency to supply a next move. The solve method will provide the answer of the Sudoku Puzzle.

9.1.10 Request Class

Uses the request from the view and specifies which aspects of the network need to be considered.

9.1.11 Variables Class

Uses the variables that are the cell values from the SudokuBoard, these values are entered by a user and saved to a new board that is parsed to the variable class where the values of each cell become the variables that are able to be checked if they are assigned or not using a method, as well as the ability to change the assigned value so that they can all be checked, these are accessed by the Constraints class.

9.1.12 <Interface> Constraints Class

The constraints class is used as an interface between the domain and the variable as there is a given variable that will take a value and the domain is the list of possible values that the variable can have, the constraint takes the value and the domain and makes a constraint from it, meaning what possible values the variable can hold.

The constraint class parses its constraints from the AllDifferent Class, where a list of variables is checked to be true and if it is possible to assign the value that is the only possible value for that variable from the domain.

9.1.13 Domain Class

Takes the given variable and displays the values that it could possibly hold due to other factors, for example within the same row if there are already values, they will not need to be within the domain and will not be included. This is then given to the constraint class to deduce the possible values from the constraints.

9.1.14 AllDifferent Class

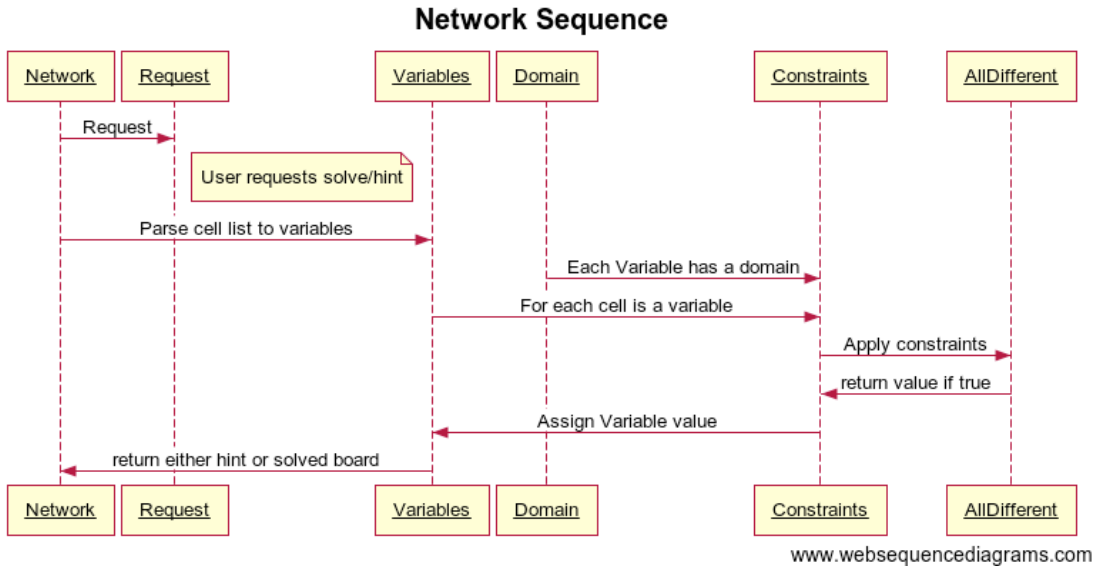
Takes the variables and checks that the values within them are all different so there are not any duplications within the scope, if there are the value for the cell is wrong and there has been an error within the board.

9.1.15 The Network

When a request is made the network is traversed, it takes the given board and uses the constraints that are taken from the classes that make up the constraints class, using the domain and variables, this is then compared to the AllDifferent class that applies the constrains, i.e. that all the values within the scope are different values. These values are then returned if true and then assigned at the SudokuBoardModel that connects to the network.

9.2 Sequence Diagram

This sequence diagram provides a visible representation of this.



9.3 Refactoring

During the design stages and throughout the implementation of writing the code there were several changes that occurred that might have deviated from the plan, this was essential to build the program in a correct way or to overcome obstacles that may not have been considered until they actually had to be implemented.

9.3.1 User interface

The user interface was changed often to allow new ways and improvements to the program, allowing the functionality to be improved by changing the layout of the GUI placing the buttons within a different position and assigning them to different anchors, this allowed the window to be easily changed but more compact so that It was not a full screen size with wasted space, instead everything is now closer and more compact to prevent over usage of space.

One of the factors that have changed is an extra button within the program as a help function, to allow the user a pop up text box, that contains a small set of instructions of how to play Sudoku as well as a hint on the controls, for example to enter a value into a cell simply click and then key in the desired number. However this also includes small descriptions for each button that shows what it does.

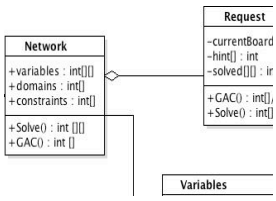
9.3.2 Sudoku Grid

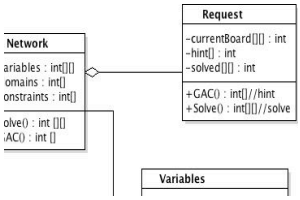
Originally the program was designed to contain buttons within the Sudoku Grid, however upon programming I found that it was more beneficial to use JLabels, as they are more versatile with what can be done to them.

Due to using JLabels it was necessary to implement a Key listener that I wanted to have for each cell so that when the user typed the desired number and clicked on a cell, that cell would then be updated as text for the number that was pressed by the user.

9.3.3 Creating a network

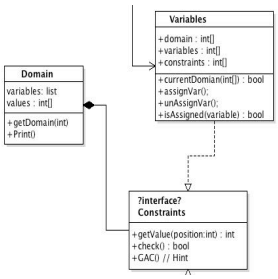
The reasoning to divide the program into different sections by using a network is that one section of the program can handle to objects and user interface and the other, which only needs to know the current board values can be separated. It is practical to involve a Network class that can separated these two functions and allow the program to simply parse the board and then the network will return the requested board.



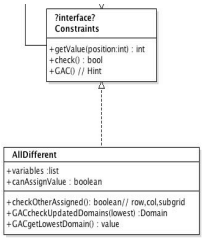


The network involves the classes that interact with the board in terms of constraints and the values assigned to them. Within the network there will be a class that allows the request of the user to included as to what algorithm needs to be implemented.

The main bulk of the network system is are taken place on the board that are then values that are assigned to the board as a board that contains a next possible value



the constraints that reduced to a set of completed board or placement.



The constraints them selves will be taken from the AllDifferent class that specify that each value has to be different that any of the others.

9.3.4 Sudoku board Model View Controller

To allow structure the classes of a board that need to have both a class that contains the cell that are linked to JSwing objects such as JLabels it is essential that there is a structure that allows each of the desired board functionalities to work, it was useful to implement a MVC structure that allowed the view to contain the boards cells that are JLabels and contain the values that are represented within the board.

The board controller class is used as the main part of the board that is where all the data taken from the view that is input and is collaborated with the board and sent to the model to be parsed to the network. Without the controller the view and model would not be able to parse information to each other as the controller links the two.

The use of the model in this case is to ready the board to be sent to the network so that it is the current board that has been taken from the controller and this will then be send to the network, this also needs to be able to handle the output from the network as it will return a new board or value which can then be parsed to the controller.

9.3.5 Observer[42]

As within the program there will be a lot of objects that will be changing often they will need to be observed with an observable method, within the class where the changes are made the

[42] http://docs.oracle.com/javase/7/docs/api/java/util/Observer.html

observed object and the update method is called whenever the object is changed, this allows the class to know when there is a change within the object, in the case of the Sudoku the observable object will be the cells that contain the values of the Sudoku.

9.3.6 Swing Worker

Within the program writing often problems occur that were unforeseen within the design stages, for example I came across an issues with non responsive JLabels that were assigned the desired value but were not updated to do so within the view. To resolve this I used a swing worker.

Concurrency had to be introduced to the user interface as it was essential that the program was also responsive to user updates even when making a calculation, the main reason for needing other threads to allow the program to calculate what the cells values need to be within the board and these need to be updated when they are changed.

The resolution to this is to use a swing worker that uses a worker thread to allow the calculation process to be made within the program so that when it returns the view can be updated to the current board state. The example of the swing worker that I used for the load button.

```
class LoadListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                SwingWorker<SudokuBoard, Void> worker = new SwingWorker<SudokuBoard,
Void>() {

                        @Override
protected SudokuBoard doInBackground() throws Exception {
                                String chosenDate = m_view.loadMenu();
//load the date from the selected by the user
                SudokuBoard board = m_model.Load(chosenDate);;
//load the board and return the new board.
                 return board ;}
                        @Override
                protected void done() {
                                SudokuBoard board = null;
                                        try {
```

```
                                        board = get();
//save value that was returned to a new board
                                    } catch (InterruptedException e) {
                                        e.printStackTrace();
                                    } catch (ExecutionException e) {
                                        e.printStackTrace();}
                                    board.setBoard(board);
//set that board as the current board within the
 view using the SudokuBoard class to publish it.
                                }
                            };
                        worker.execute();
// run the swingWorker
```

9.4 Conclusion

Throughout the design stage everything was changing as there are always better ways to do something and this proves true when coding begins as there are problems and things were not expected so the strategies and methods are amended, like seen within the SwingWorker problem.

Chapter 10: **Showcase of program**

10.1 Graphical User Interface

Description

Image

10.1.1 Load Window

Description

Image

10.2 Algorithm code

Description

10.3 Backtracking

Code Snippit (See Chapter 5.2.1)

10.4 Dynamic Variable Ordering

Code Snippit

10.5 Generalised Arc Consistency

Code Snippit

10.6 XML parser

How to create and take the information stored within the files

10.7 Conclusion of program

Chapter 11: **Testing**

11.1 Test Driven Design

Description

Implementation

Why use it

11.2 GUI

Explanation

11.2.1 Results1

11.3 SudokuBoard Test

Explanation

11.3.1 Results

11.4 Network Constraints Test

11.4.1 Constraints to solve board

11.4.2 Hints for the best next move

Chapter 12: **Professional Issues**

12.1 Software Piracy

The use of the Internet to fulfil both educational and entertainment needs has allowed a huge collection of programs that are easily downloadable from websites without the need to pay or even question their legitimacy. This unfortunately is now how programs are downloaded, due to the large amount of freeware programs available many customers are reluctant to pay for a product.

The programs for Sudoku solvers are all over the Internet, most of which are websites that are purpose built within the page. However the others are a mixture of small programs developed by students or people that are programming for fun and allow their software to be used freely. However the companies that provide the same or more complex program often require payment for the product

To download a program that is required is simple and this can be done using a search engine that are designed to look at what words have been put in place within the website, i.e. the program description, to display the product the search engine thinks best matches the search, this is done by using key words within the site and is called search engine optimisation.

12.1.1 Threats

Searches do not provide any safeguards in terms of the sites visited it is possible that there are harmful programs within a site that can be searched for. When finding a program, for example a Sudoku Solver it is possible to download many solvers from the internet that are freeware, however these all could possibly pose a threat to the computer running it.
Ilegitimate software

The programs that can be downloaded for free can sometimes be cracked versions of other developers work, thus infringing copyright and committing a crime, there are many cases where program code is edited to be able to be used for free. However even when a program is legitimate but edited it does not mean the program is safe to run due to the program code changing it would be very simple to hide a virus within a program that is presumed legitimate.

Software piracy has been a massive issue within the games and entertainment industry as a whole, there are very few things that a dedicated and skilled team would not be able to copy or

at least bypass systems to allowed added functionality, for example with mobile phones that can be jail broken to allow 3rd party software on to the device that has not been verified by the manufacturer.

When free software is introduced to the internet by the developer it is often posted to many sites to allow the product to be as widely accessible as possible, however when some sites do allow downloads of the product it needs to be verified as it would be possible to create a edited harmful version and publish it to one of the download sites and no one would question its legitimacy.

12.2 Prevention

When a file is downloaded there are safeguards protecting the computer however when the user wants to open a file made by an unknown developer then it will tell the user so and allow them to choose if the program should be run, however if a user has downloaded a program often they will just want to use it so there is a possibility they will allow the program to run.

To prevent this taking place companies promote the use of their own products and often enforce this by providing updates within their software that fix some issues, i.e. updating an IPhone that is jail broken will have to be done again once the update is completed. The same is true with programs that are not from the original developer they can have major flaws in security or functionality that need to be patched but are unable to do so.

Another prevention method is to use antivirus software, however not effective all the time, it is the best way to prevent most programs that contain a virus from being introduced to a computers files. This is possible by allowing the antivirus access to the program file, where it will analyse the code within and look for hidden files that might contain harmful data, this data is then compared to the database of already know virus's and will warn the user of the impending threat.

Bibliography

Book Resources

Professor David Cohen, The Complexity of the Constraint Satisfaction Problem, ISG-CS Mini-Conference Slides

Cay Horstmann, Big Java: Forth Edition, Page 629-663

Darryn Lavery, Gilbert Cockton and Malcolm Atkinson, Heuristic Evaluation, Usability Evaluation Materials: The Department of Computing Science, University of Glasgow

Adrian Johnstone & Elizabeth Scott,T'ime and Space' in Designing Efficient Programs. Chapter 5,Page 68 – 75

Algorithms and Complexity 2, CS2870 By Gregory Gutin February 16, 2013. Page 93

Graph Algorithms and NP-Completeness. Kurt Mehlhorn. Page 171- 211

Discrete Mathematics & its Applications, 6th Edition. by Kenneth H. Rosen. Page 836

Algorithms in a Nutshell, by George T. Heineman, Gary Pollice, and Stanley Selkow pages 217-221

Algorithms in a Nutshell, by George T. Heineman, Gary Pollice, and Stanley Selkow pages 223

Web Resources

http://arstechnica.com/features/2005/05/gui/ last Accessed: 31/10/13

http://www.dcs.gla.ac.uk/asp/materials/HE_1.0/materials.pdf

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html Last accessed 7/11/2013

http://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

http://en.wikipedia.org/wiki/Heap_(data_structure)

http://www.tutorialspoint.com/design_pattern/flyweight_pattern.htm

http://www.codeproject.com/Articles/186185/Visitor-Design-Pattern

http://www.rhul.ac.uk/computerscience/documents/pdf/csisgmini-conf/dac.pdf

http://technet.microsoft.com/en-us/library/aa214775(v=sql.80).aspx

http://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/CSP3.pdf

http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html

http://artint.info/html/ArtInt_79.html

www.ics.uci.edu/~dechter/courses/ics-275a/spring.../chapter5-09.ppt slides 26-31

http://ktiml.mff.cuni.cz/~bartak/constraints/ordering.html

http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf

http://www.cosc.canterbury.ac.nz/csfieldguide/student/Complexity%20and%20tractability.html

http://www.cosc.canterbury.ac.nz/csfieldguide/student/Complexity%20and%20tractability.html

http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/SIGAL87-2.pdf

http://www.math.cornell.edu/~mec/Summer2009/Mahmood/More.html

Steve Schaefer (January 2002). "MathRec Solutions (Tic-Tac-Toe)". MathRec.org - http://web.archive.org/web/20130628112339/http://www.mathrec.org/old/2002jan/solutions.html

http://en.wikipedia.org/wiki/Tic-tac-toe

http://www.theguardian.com/media/2005/may/15/pressandpublishing.usnews

http://www.decabit.com/Sudoku/Techniques

http://www.stolaf.edu/people/hansonr/sudoku/explain.htm

http://www.sudoku-solutions.com/solvingHiddenSubsets.php#hiddenSingle

http://www.decabit.com/Sudoku/NakedSubset

http://www.decabit.com/Sudoku/XWing

http://www.angusj.com/sudoku/hints.php#sordfish

http://www.decabit.com/Sudoku/XYChain

http://www.ams.org/samplings/feature-column/fcarc-kanoodle

http://docs.oracle.com/javase/7/docs/api/java/util/Observer.html

Multi-Media Web Resources

http://www.youtube.com/watch?v=Ge-XiX_tP4c-Doug Fisher's video on Generalized Arc Consistency