REPORT

Friday, 22nd September 2023

By

Codie Shannon

30013375

Assignment One

In

COMP.7212

Artificial Intelligence Techniques

Table of Contents

Part 1 – Advanced Search

You are to implement an example of A* Search. You are to analyse and compare how this algorithm works and compare A* Search with two other algorithms discussed in class (Breadth-first and Depth-first). You must ONLY edit the Jupyter Notebook as required in the A_Star_Search project and the accompanying map.py file. The Notebook must work with the original project without modification. You will submit the Jupyter Notebook files and your written answers as a Word document. To check your code runs correctly before submitting, please save your files and then "Reset and Run All" under the Kernel options. Please discuss with your lecturer if you are having any issues getting your code to run.

1. A* Search

Explain how the implementation of the A* Search algorithm works within Python, i.e., how the code functions, including the libraries it utilizes.

A. Overview

A* Search is an algorithm that is utilized to locate the most direct path between a start and end node. The algorithm is most frequently utilized for map traversal to locate the most direct path to be taken. The A* search algorithm was first produced as a graph traversal problem, to aid in building a robot that could locate its own course. A* Search remains a universally favoured algorithm for graph traversal (Ravikiran, 2023).

The algorithm searches for the most direct paths first, making the A* search algorithm a complete and optimal algorithm. A complete algorithm locates every possible outcome of a problem, whilst an optimal algorithm must locate the most cost-effective outcome to a problem (Ravikiran, 2023).

In addition, an aspect that makes the A* search algorithm so dominant is the utilization of weighted graphs in its implementation. A weighted graph utilizes numeric values to represent the cost of taking each course of action or path. As a result, the algorithm may take the most cost-effective path, and locate the best route in terms of distance and time (Ravikiran, 2023).

B. NumPy

NumPy is an essential library within Python when performing high level mathematical functions. The library allows programmers to perform mathematical functions such as calculation of square roots, and random number generation. Alongside this, the library adds support for mass, multi-dimensional matrices and arrays.

The A* Search algorithm created within Python utilizes the NumPy library to calculate the distance between nodes utilizing its proficient square root function.

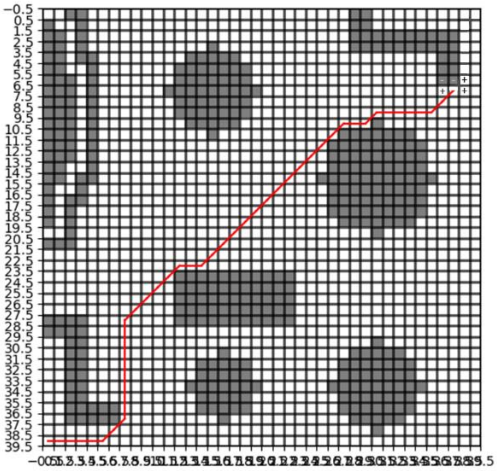
C. Get Adjacent Nodes Function

The A* Search algorithm constructed within Python utilizes the map library to perform operations in relation to the game map, such as construction of the game map, and detection of all clear nodes surrounding the currently active node.

The Get Adjacent Nodes function resides within the map library. The primary operation of this function is to get all the adjacent / clear adjoining nodes of the currently active node within a one-square-radius (as seen in Figure 1). The function performs this operation utilizing a plethora of loops and if statements, drawing a comparison between negative and positive nodes utilizing simplistic 0s and 1s.

Figure 1

Clear Adjoining Nodes Search



D. Priority Queue

A Priority Queue is a non-generic collection of objects whose size may be to set to a maximum. The collection is homogeneous to a stack, except a priority queue prioritizes the lowest values first, which means when the get "pop" function is called, the lowest values will be removed from the priority queue and returned to the programmer.

The A* Search algorithm developed within Python employs the Priority Queue to keep track of newly discovered nodes found while performing the search loop. The search algorithm does this, so it may reutilize the discovered nodes next loop while it is searching for the neighbouring best clear node.

E. Role of Heuristic

Heuristics, frequently called heuristic functions, are utilized to supply "good enough" solutions to exceptionally complex problems where locating a perfect solution may take too much time. When developers utilize heuristics, they trade correctness, accuracy, and exactness for rapid processing (Isaac Computer Science, n.d.).

One of many drawbacks with Dijkastra's greedy algorithm is that it may (and will) evaluate paths that will never supply the shortest option. Imagine attempting to locate the shortest route on a map between Tauranga and Auckland. Anyone with a reasonable grasp of the North Island's geography, would not bother to evaluate a route that went via Gisborne. The trade-off between accuracy and speed is paramount. In a handful of applications, accuracy has reduced importance to computational time. For example, in a GPS application a route that is calculated in seconds and is "short enough" is preferential to having to wait countless minutes for the perfect route (Isaac Computer Science, n.d.).

The A* algorithm utilizes a heuristic function to assist in deciding which path to follow next. The heuristic function supplies an estimation of the minimum cost between a given node and the target node. The algorithm will merge the actual cost from the start node (referred to as g(n)) with the estimated cost to the target node (referred to as h(n)) and utilizes the result to select the predominant next node to evaluate (Isaac Computer Science, n.d.).

2. Comparison

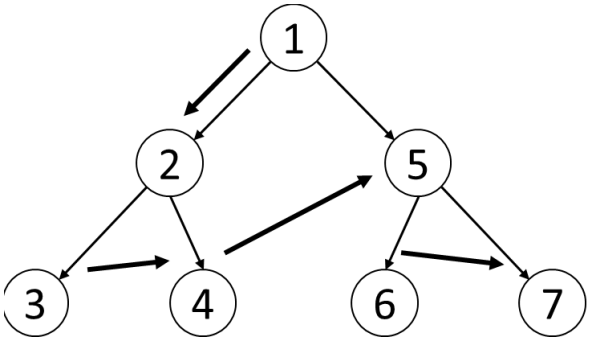
A. Depth First Search

Explain what the Depth First Search Algorithm is, and how it functions.

DFS, an abbreviation of Depth First Search, is an algorithm for searching or traversing graph or tree data structures. The search algorithm begins at the top (root) node (designating a whimsical node as the top node in the case of a graph) and explores the max extent of each branch before backtracking (as seen in Figure 2).

Figure 2

Depth First Search



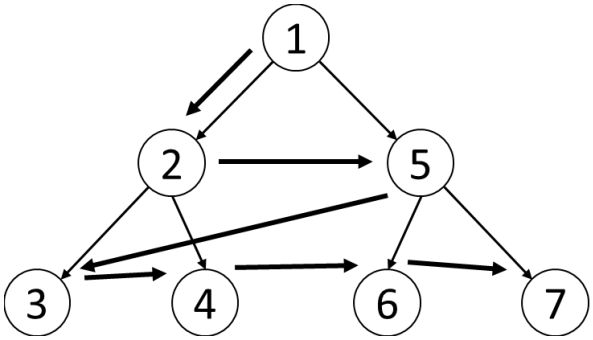
B. Breadth First Search

Explain what the Depth First Search Algorithm is, and how it functions.

BFS, Breadth First Search, is an algorithm for searching or traversing graph or tree data structures. The search algorithm starts from a selected node (source or root node) and traverses the tree/graph layer-wise, moving from left to right to each neighbouring node from layer to layer (which are connected to the starting node) in an attempt to locate the goal node (as seen in Figure 3).

Figure 3

Breadth First Search



C. Memory Efficiency

Compare the A* Search Algorithm to the Depth-First and Breadth-First Search Algorithms to find the best memory efficient search algorithm. Explain why the chosen search algorithm is the best memory efficient. (Short Paragraph)

The Breadth-First uninformed search, Depth-First uninformed search and A* informed search algorithms may outperform each other under various conditions. This is due to the algorithms operating in vastly different methods (there is no end all, be all).

I. Breadth-First Search

Breadth-First search must store a frontier of nodes to search next (where "search" simply means: see if it's the goal, if not, generate its children and add to frontier). You may visualize the memory requirements of this as a pyramid. In the initial instance, you store the root node within the frontier. However, as the search process continues and you keep heading further down, the search tree will become wider, and the frontier must hold excess nodes to a finite degree (excess memory required). The amount of memory needed in the worse case to locate a node at depth d may be expressed as O(bd), where b is the branching factor (average out degree). Expectedly, when you go one level deeper, the algorithms memory requirements multiply by a factor of b (the pyramid becomes b times wider, as a result, the branch hierarchy may swing out more rapidly than an average pyramid would) (Soemers, 2018).

II. Depth-First Search

Depth-First search must store a frontier (stack) of nodes to search next (where "search" simply means: see if it's the goal, if not, generate its children and add to frontier). However, in the case of depth-first search, the frontier usually grows less rapidly. Whenever the algorithm pops a node off a stack (visits a stack), it will then proceed to generate all its children and push them on top of the stack. However, whereas breadth-first search would proceed to continue "to the right" by visiting a proportionately older node and generating all its children again, depth-first search continues by going "down" visiting one of the children that only just got pushed onto the stack. When depth-first search has completed a portion of the search tree (you may visualize this in your head as depth-first search having finished a search on, for example, the left-hand side of the pyramid), that complete segment no longer needs to be stored within memory. Under the assumption that the search tree is of finite depth d, the worst-case scenario is that the space complexity is O(db) (the algorithm may have to search one path all the way down to a level of d, and for each of those levels have b nodes in the stack) (Soemers, 2018).

III. A Search*

The A* Search is similar to the breadth-first search algorithm in the method it utilizes to search a tree. The A* Search algorithm stores a frontier of nodes to visit next. However, unlike breadth-first search, A* search prioritizes the nodes based on some estimation of "goodness" rather than breadth-first order. In conjunction, this means that in the worst-case scenario memory requirements are similar to breadth-first search. In cases where the information the algorithm is utilizing (typically a combination of known/incurred costs + heuristic estimation of future costs) is of exceedingly poor quality, the algorithm may regress to the level of breadth-first search, so the worst-case space complexity is the same as that for breadth-first search. In practice, if an A* search has high-quality information (good heuristics), the memory requirements may be much better though. In the case of an ideal/perfect heuristic, the search algorithm will go directly to the end goal (target node) and hardly require any memory (Soemers, 2018).

Part 2 – Genetic Algorithms

1. Introduction

GAs (Genetic Algorithms) are flexible heuristic search algorithms that belong to the vast class of evolutionary algorithms. Genetic Algorithms are constructed on the concept of genetics. natural selection and a small probability of randomization based on chance of survival (Overide, 2023). In addition to this, Genetic Algorithms are comprised of five evolution mechanisms, populations, selection / survival, reproduction, mutation and recombination.

Genetic Algorithms imitate the process of natural selection, as a result species who are capable of adapting to changes within their environment may survive, reproduce, and move on to the next generation. Conversely, a chance of survival remains for species who are not capable of survival under such conditions. Put simply, genetic algorithms simulate "survival of the fittest" with a small chance of randomization for lesser individuals as to not cause a stagnation in developing solutions to a problem (Overide, 2023).

Each generation consists of a population of individuals and each individual portrays a point in search space and possible solution. Each individual is portrayed as a string of character/bits/float/integer. This string is comparable to the chromosome (Overide, 2023).

A scheduling problem exists for a hospital which must roster 15 and 50 nurses respectively. Due to this, the human resource management team has decided to outsource this problem to a group of students. The students must develop an optimal method to assign nurses to shifts utilizing a genetic algorithm created within the programming language Python. Each nurse has their own restrictions and wishes, as does the hospital. The optimal solution must locate a day, afternoon, and night shift schedule that both respects the constraints of the nurses and fulfils the objectives of the hospital.

In conjunction with this, students must utilize the Python library DEAP to establish the genetic algorithm. DEAP is an evolutionary algorithm framework, which places a primary focus on flexibility and manageability. The framework allows Python programmers to rapidly register operators to create algorithms with a small amount of code. Operators are available for numerous genetic mechanisms including crossover (mate), mutation (mutate), and selection (select).

In accordance with the scheduling problem, the information within this document will cover the methods that the genetic algorithm is comprised of, alongside the settings utilized to find the optimal solution, and discuss the results from the optimal solution.

2. Optimal Settings

A. Nursing Roster 15

List the Optimal Settings You Have Found Here.

Nurse Roster 15	Solve Nurses V1	Probability / Size	Solve Nurses V2	Probability / Size
Population		500		250
Selection	selTournament	2	selBest	
Cross-Over	cxTwoPoint	0.9	cxTwoPoint	0.9
Mutation	mutFlipBit	0.5	mutFlipBit	0.9
Termination	Max Generations	573	Max Generations	532
Hall of Fame		5		50
Hard Constraint Penalty		10		5
Random Seed		42		12

B. Nursing Roster 50

List the Optimal Settings You Have Found Here.

Nurse Roster 50	Solve Nurses V1	Probability / Size	Solve Nurses V2	Probability / Size
Population		500		250
Selection	selTournament	2	selBest	
Cross-Over	cxTwoPoint	0.9	cxTwoPoint	0.9
Mutation	mutFlipBit	0.5	mutFlipBit	0.9
Termination	Max Generations	900	Max Generations	1500
Hall of Fame		5		50
Hard Constraint Penalty		10		5
Random Seed		42		12

3. Methods

A. Selection Function

Introduce the concept of selection functions and how they are utilized within Genetic Algorithms. Introduce the Selection Types you used and how it impacted results.

The concept of the selection evolution mechanism is to select the fittest individuals and let them pass their genes to the next generation.

A set of individuals (parents) are chosen based upon their fitness scores. Individuals with peak fitness have a greater chance to be selected for reproduction.

The selection mechanism provides many variations of selecting the individuals (parents) to pass their genes to the next generation such as, elite preservation selection, rank-based selection, tournament selection, fitness proportionate selection and many others.

During development and testing of the genetic algorithm to solve the nurse scheduling problem, it was found that the tournament selection and best selection variations were best suited for the job of rostering 15 and 50 nurses respectively.

I. Tournament Selection

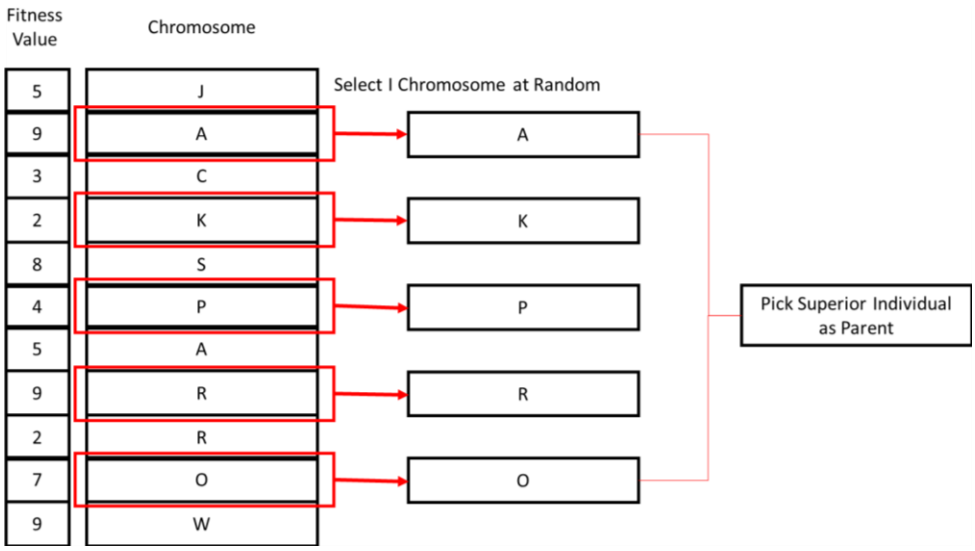
In the tournament selection approach, a “tournament, i” is run amongst multiple individuals chosen at random from the population pool and the individual with the highest fitness is selected as the winner and may carry on their genes to the next generation (Shyalika, 2019) (as seen in Figure 4).

In the assumption that i is equal to five, five entities would be selected out of the population pool, a comparison would be drawn between their fitness, and the better individual would be permitted to reproduce (Shyalika, 2019).

The selection pressure may be simplistically adjusted by modifying the tournament size (a larger I increases selection pressure) (Shyalika, 2019).

Figure 4

Tournament Selection



II. Elite Preservation Selection

The Elite Preservation or Best selection method operates by selecting I best individuals to copy to the population in the next generation. Elitism may exceedingly increase performance of genetic algorithms because elitism prevents the loss of the best-found solution. The Elite Preservation selection method is an optimist technique (Shyalika, 2019).

There are multiple factors to consider when taking an elitism stance with a genetic algorithm.

Merit – The best i chromosomes of the current generation survive until the next generation (Shyalika, 2019).

Demerit – Genes of the elites can spread in the population and diversity can be lost. In this scenario, local solutions (maxima) tend to be obtained (Shyalika, 2019).

III. Results

The tests conducted during the first stage of development – selection were astonishingly promising. The prime selection methods that stood out during the tests were selTournament (Tournament Selection) and selBest (Best Selection) of the DEAP library – evolutionary tools. As a result, the students have planned to go forth with multiple genetic search algorithm variations to locate the optimal solution.

Tournament Selection returned an impressive fitness value of 248.0 and 245.0 for the nurse roster of 15 and 50 respectively. Whilst the Best Selection method resulted in a fitness value of 251.0 and 335.0 for the nurse roster of 15 and 50 correspondingly. The results may be viewed in Figure 5 and 6.

Figure 5

Tournament Selection Graphs

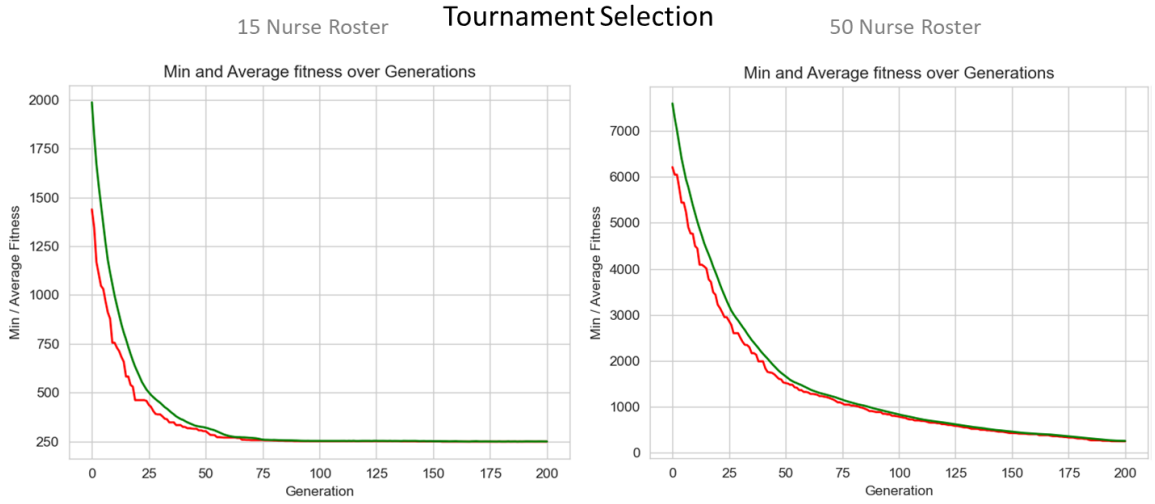
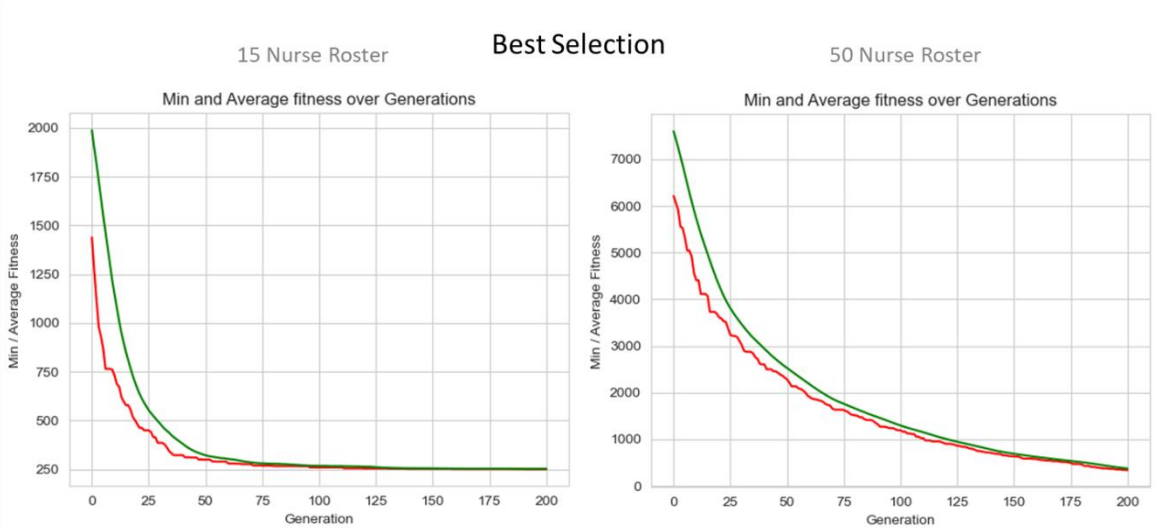


Figure 6

Best Selection Graphs



B. Hall of Fame

Introduce the concept of a Hall of Fame and how it is used within GA. What settings you have changed on the Hall of Fame and how it impacted results.

Hall of Fame is a mechanism within the library DEAP for developers to utilize for genetic algorithms. The Hall of Fame mechanism acts as an elitism collection bucket. The collection bucket is lexicographically organized at all times so that the initial element of the Hall of Fame is the individual that has had the best initial fitness value ever observed. The collection bucket size may be set by the developer as an attribute of the Hall of Fame method call.

The Hall of Fame function operates by collecting the superior individual from each generation. During the mechanism's operation, it will draw a comparison between the individuals within the Hall of Fame to the superior individual of each generation to distinguish if the finer individual has a higher fitness value than an individual within the collection bucket. Succeeding this, the Hall of Fame mechanism will remove the lowest grade individual from the collection before compressing the collection and attaching the superior individual from the latest generation to the collection bucket. Once the algorithm has concluded, the Hall of Fame collection bucket may be formatted and displayed to the end-user.

I. Results

The tests performed during phase two of development – Hall of Fame were interestingly erratic between nurse roster 15 and 50. On one hand, nurse roster 15 displayed incremental results for the fitness value from 244.0 to 251.0 for search one with six tests performed (as shown in Figure 7). Whilst search two exhibited similar incremental results for the fitness value from 249.0 to 257.0 (as shown in Figure 8).

In comparison, nurse roster 50 displayed a substantial change in fitness value from 201.0 to 245.0 for search one (as shown in Figure 9). Alongside, search two obtaining an even more considerable variation in fitness values from 276.0 to 396.0 (as shown in Figure 10). A similar change was observed for each of the nurse rosters in relation to violations, alongside numerous different variations.

The search one algorithm saw a slight improvement for nurse roster 15 with an initial fitness value of 248.0 improving to 244.0. In conjunction, nurse roster 15 for the search two algorithm saw a similar incremental change in improvement for the fitness value, changing from 252.0 to 249.0. Alongside this, comparable incremental changes were also observed for the violations.

In contrast, nurse roster 50 had an impressive enhancement for the search one algorithm, with a beginning fitness value of 245.0 changing to 201.0 in later iterations of the Hall of Fame tests. Whilst nurse roster 50 displayed parallel enhancements for the search two algorithm, transposing from 295.0 to 276.0. In conjunction, similar changes were noticed with the violations.

Figure 7

Search One – 15 Nure Roster

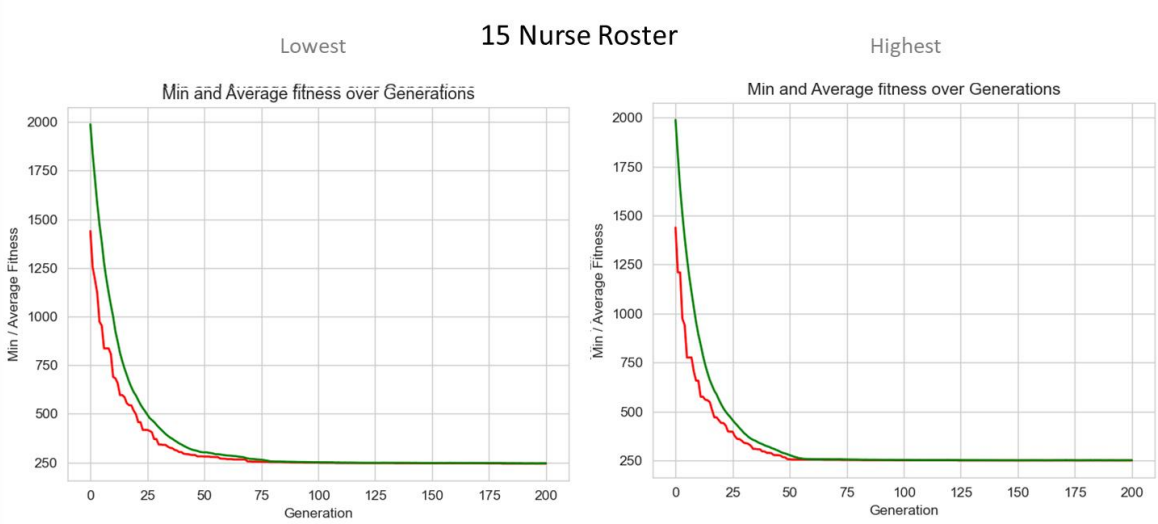


Figure 8

Search Two – 15 Nurse Roster

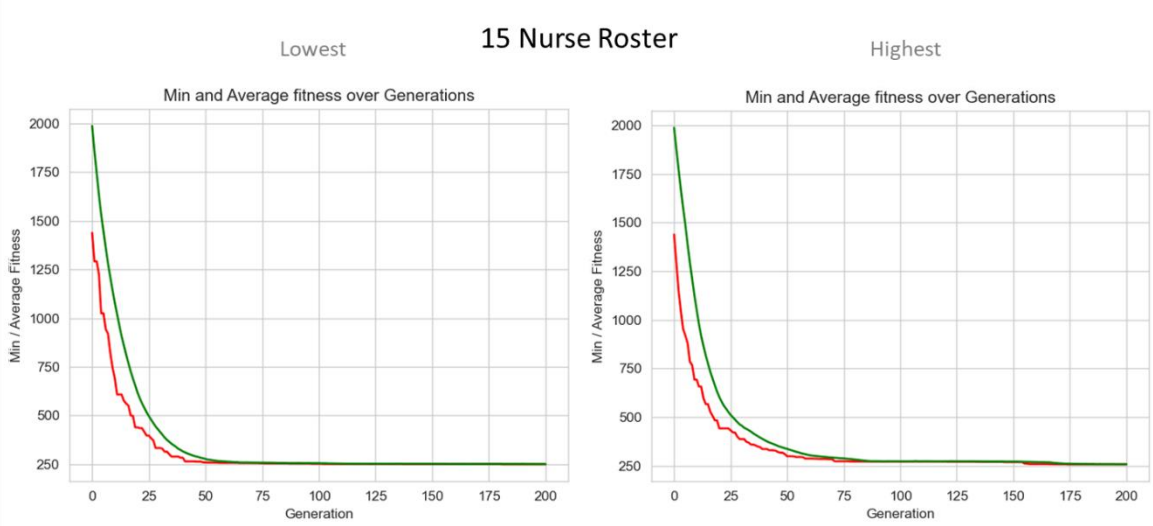


Figure 9

Search One – 50 Nurse Roster

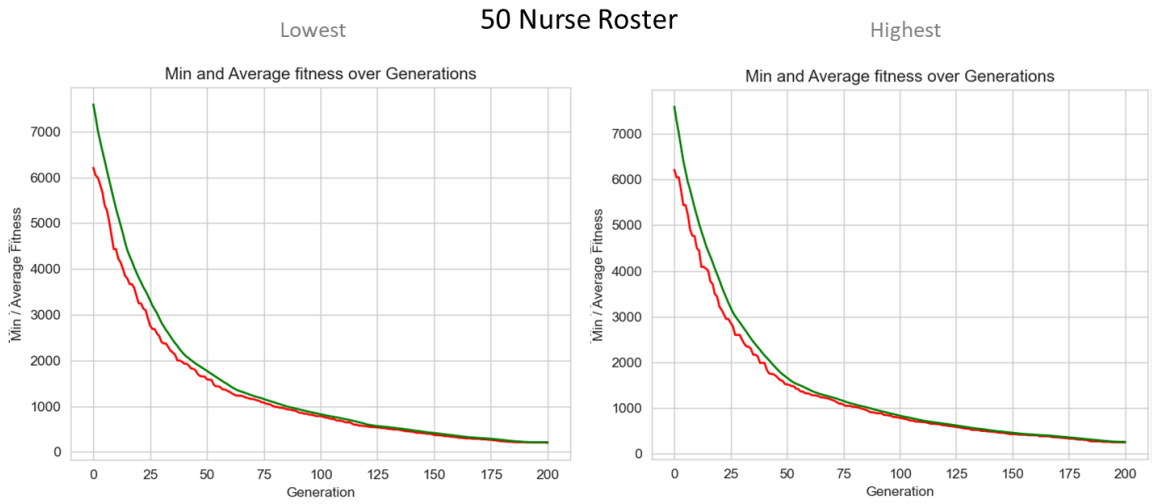
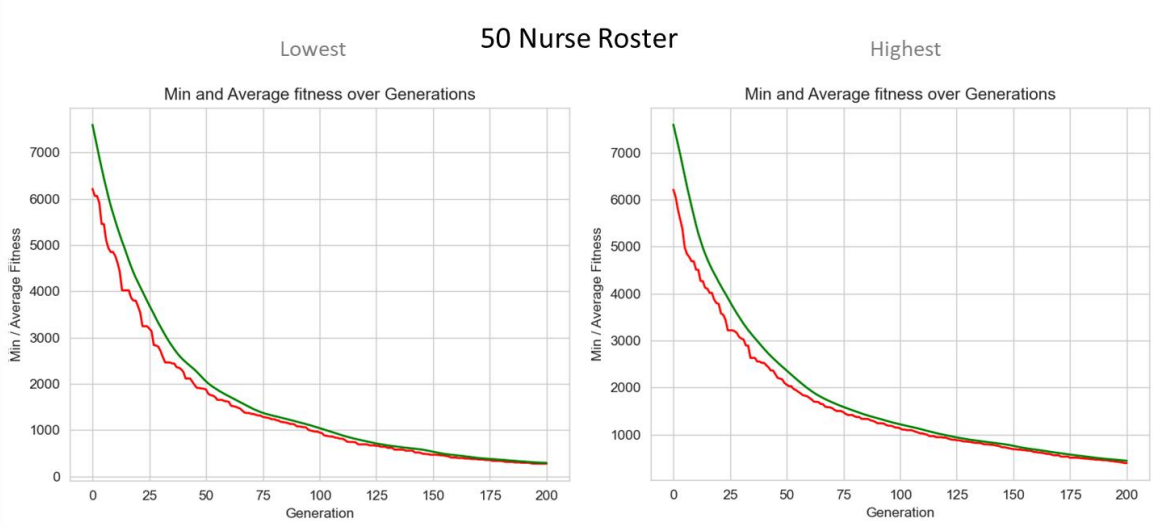


Figure 10

Search Two – 50 Nurse Roster



C. Cross-Over Function

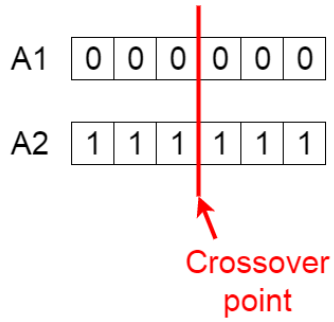
Introduce the concept of Cross-over and how it is used within GA. What settings you have changed and how it impacted results. How one of the cross-over functions you chose works, why you chose it, and how you configured it in the code.

Crossover is the most crucial segment within a genetic algorithm. For each set of parents to be mated, a crossover point must be established at random from within the gene pool (Mallawaarachchi, 2017).

For example, observe the crossover point to be 3 as shown in Figure 11.

Figure 11

Crossover Point

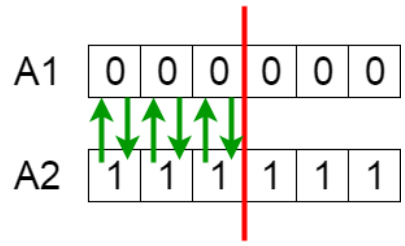


(Mallawaarachchi, 2017)

Offspring are developed by transferring the genes of the parents among each other until the crossover point is struck (as seen in Figure 12) (Mallawaarachchi, 2017).

Figure 12

Offspring Creation



(Mallawaarachchi, 2017)

The developed individuals (offspring) are consequently added to the population pool for the next generation (Mallawaarachchi, 2017).

I. Results

The tests performed during stage three of testing – Crossover were astoundingly unpredictable between nurse roster 15 and 50. Nurse Roster 15 showed incremental results for the fitness value from 244.0 to 272.0 for search one with six tests executed (as shown in Figure 13). Whilst search two manifested similar cumulative results for the fitness value from 249.0 to 258.0 (as shown in Figure 14).

Conversely, nurse roster 50 exhibited a notable change in fitness value from 201.0 to 1116.0 for search one (as shown in Figure 15). Alongside this, search two obtained an even more substantial variation in fitness values from 276.0 to 1949.0 (as shown in Figure 16). A similar result was observed for each of the nurse rosters in regards to violations, alongside countless different variations.

Unfortunately, the search one algorithm saw no noticeable difference in fitness value for nurse roster 15. In addition, disappointingly, the search two algorithm also saw no drastic change in fitness value for nurse roster 15. Alongside this, no improved changes were observed for the violations.

In comparison, nurse roster 50 experienced no substantial enhancement for the search one algorithm with a beginning value of 201.0 changing to 218.0 in later iterations of the Crossover tests and getting worse thereafter. Whilst nurse roster 50 exhibited similar results for the search two algorithm, changing from 276.0 to 331.0. In conjunction, similar changes were observed with the violations.

Figure 13

Search One – 15 Nurse Roster

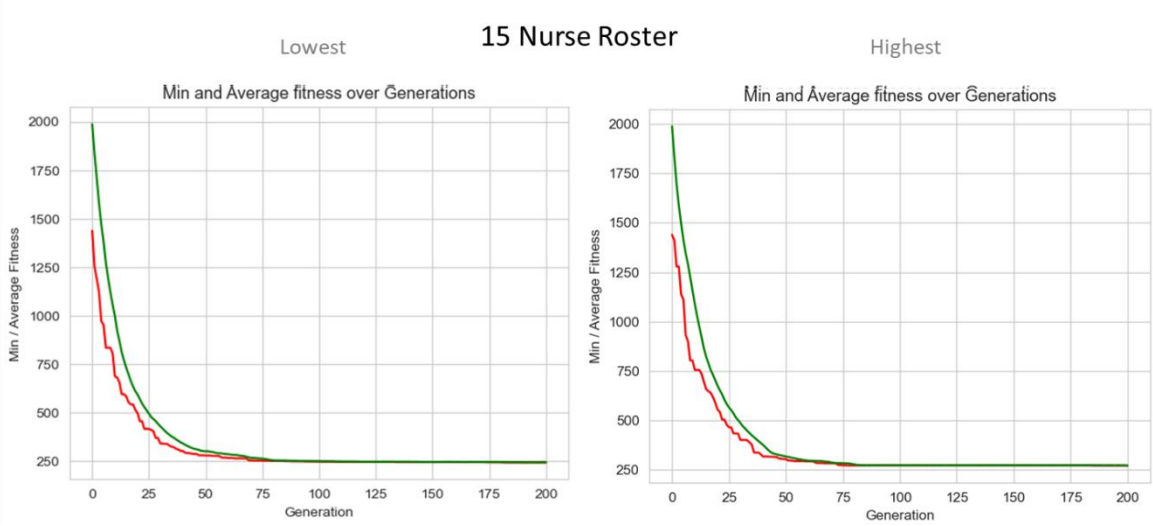


Figure 14

Search Two – 15 Nurse Roster

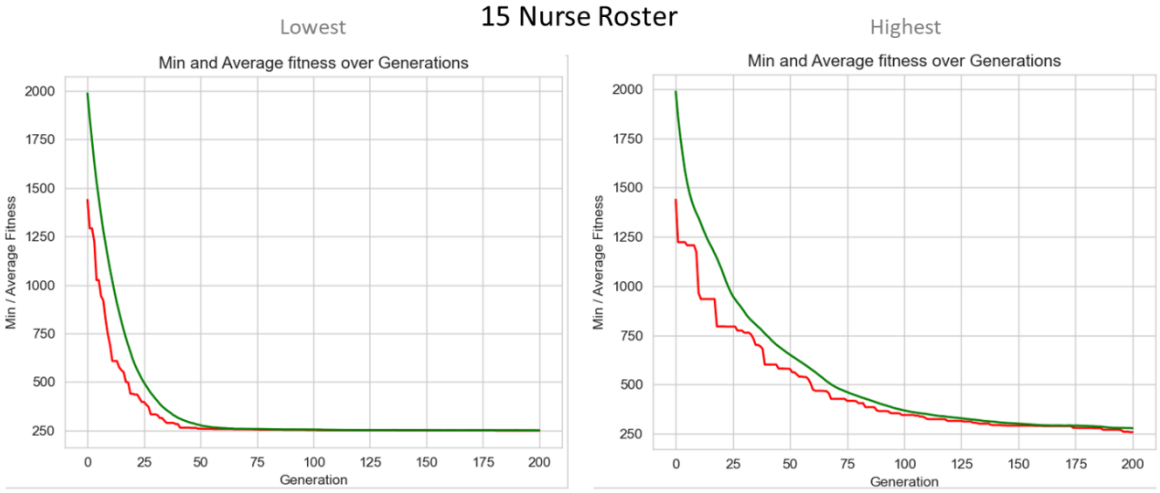


Figure 15

Search One – 50 Nurse Roster

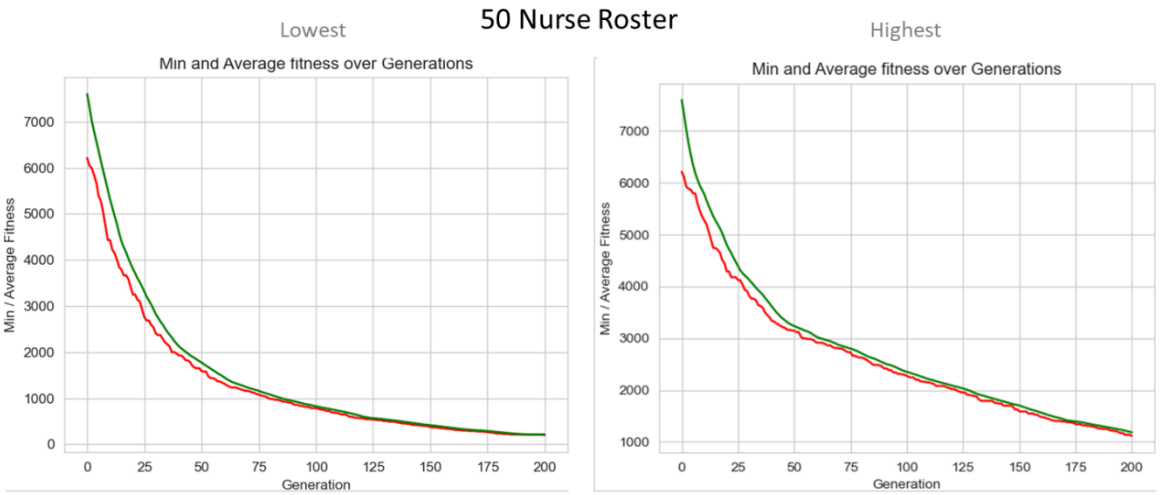


Figure 16

Search Two – 50 Nurse Roster

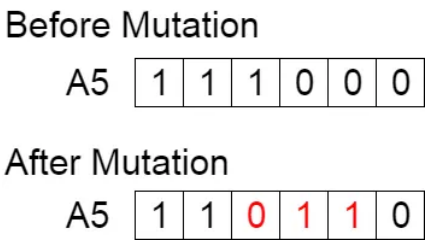
D. Mutation Function

Introduce the concept of Mutation and how it is used within GA. What settings you have changed and how it impacted results. How one of the mutation functions you chose works, why you chose it, and how you configured it in the code.

In specific developed offspring, a random chance exists for an individual's genes to be subjected to a mutation with a low probability. If a certain condition is met, some of the bits in the bit string of an individual may be flipped (as shown in Figure 17) (Mallawaarachchi, 2017). The condition, mutation probability (P_MUTATION) is a property which must be set by a developer to determine the probability or chance of a mutation occurring. The mutation probability may be set on a float scale from 0.1 to 0.9.

Figure 17

Mutation



(Mallawaarachchi, 2017)

Mutation occurs to preserve diversity within the population and avert premature convergence (Mallawaarachchi, 2017).

I. Results

The tests performed during stage four of development – Mutation were surprisingly uniform between nurse roster 15 and 50. Nurse Roster 15 displayed incremental improvements for search one and two. The search one algorithm displayed incremental results for the fitness value from 237.0 to 248.0. The lowest fitness value incremental improved upon the base fitness value of 244.0. In addition, the search two algorithm showed incremental results for the fitness value from 236.0 to 249.0 (as shown in Figure 18). This cumulatively improved upon the base fitness value of 249.0. A similar change was observed for the violations in nurse roster 15 for search one and two.

Nurse Roster 50 showed drastic improvement for search one and two. The search one algorithm displayed distinctive results for the fitness value from 77.0 to 218.0 (as shown in Figure 19). The smallest fitness value drastically improved upon the base fitness value of 201.0. In conjunction with this, the search two algorithm displayed similar substantial results for the fitness value from 112.0 to 276.0. The lowest fitness value made a drastic improvement upon the base fitness value of 276.0. A comparable change was noticed for the violations in nurse roster 50 for search one and two.

Figure 18

Search Two – 15 Nurse Roster

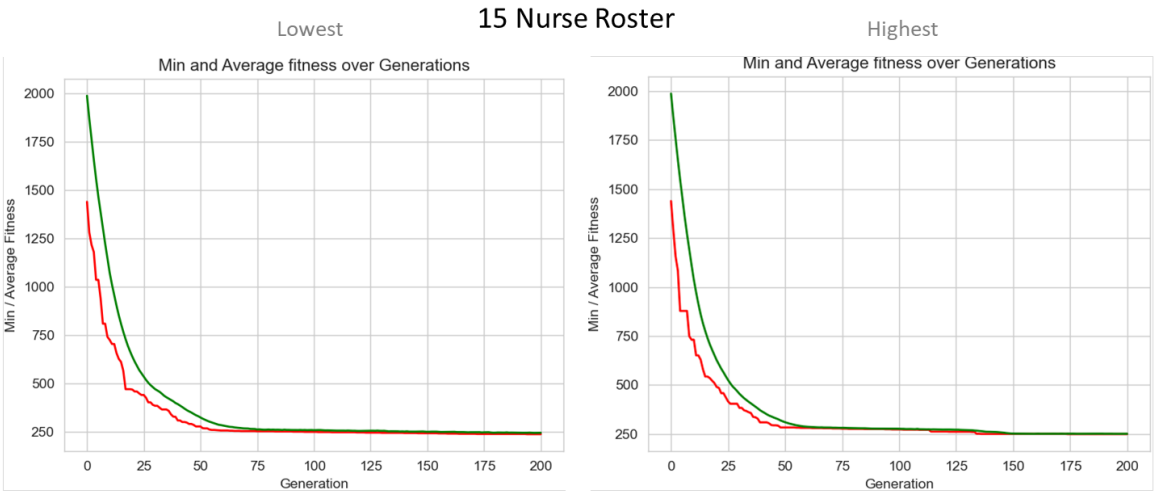


Figure 19

Search One – 50 Nurse Roster

E. Other Configuration Settings

Provide information on the configured settings you have changed here. You should tell the reader what these are and what they do i.e., what does "Hard Constraint Penalty" mean and how does changing it affect what happens during the algorithm?

I. Hard Constraint Penalty

In genetic algorithms, constraints are primarily managed by utilizing the concept of penalty functions, which penalize futile solutions via a reduction in an individual's fitness values in proportion to the degrees of constraint violation. In many penalty schemes, a few constraints or coefficients must be described at the inception of the calculation. Since coefficients ordinarily have no clear physical definition, it is almost impossible to estimate suitable values of these coefficients even via experience. Furthermore, countless schemes employ constant coefficients throughout the whole calculation. This could result in too strong or too weak a penalty during various phases of the evolution process (Nanakorn & Meesomklin, 2001).

The Hard Constraint Penalty scheme for the Nurse Scheduling Problem placed a primary focus on the predominant values of consecutive shift violations, shift per week violations, nurses per shift violations, and shift preference violations. In turn, if an individual (solution) was found to be exceedingly unfeasible, that individual would be ruled out to not procreate for the next generation.

i. Results

The tests executed during phase six of development - Hard Constraint Penalty were astoundingly dissimilar. Nurse Roster 15 displayed enormously differing results for both search one and two. The search one algorithm displayed enormously different results for the fitness value from 122.0 to 2080.0. The smallest fitness value greatly improved upon the base fitness value of 237.0. In conjunction with this, the search two algorithm displayed a similar variety in results for the fitness value from 122.0 to 2083.0 (as shown in Figure 20). The lowest fitness value drastically improved upon the base fitness value of 236.0.

Nurse Roster 50 displayed slight changes for search one and two. Unfortunately, the search one algorithm showed no improvement for nurse roster 50. The highest fitness value recorded was 182.0. However, the search two algorithm displayed slight improvements, with results for the fitness value being 87.0 to 276.0 (as shown in Figure 21). The smallest fitness value made a slight improvement upon the base fitness value of 112.0. In addition, a comparable change was noticed for the violations in nurse roster 50 for search one and two.

Figure 20

Search Two – Nurse Roster 15

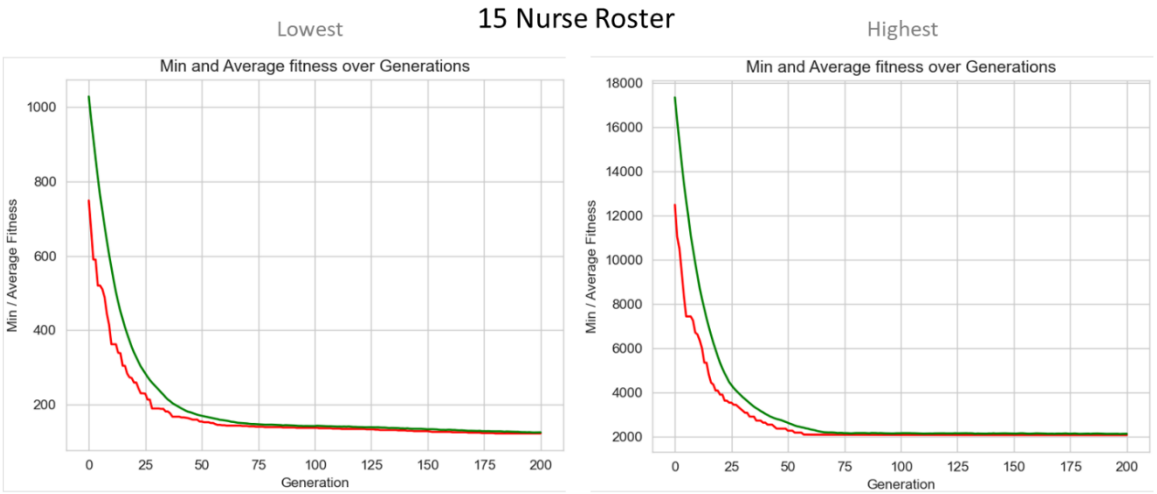
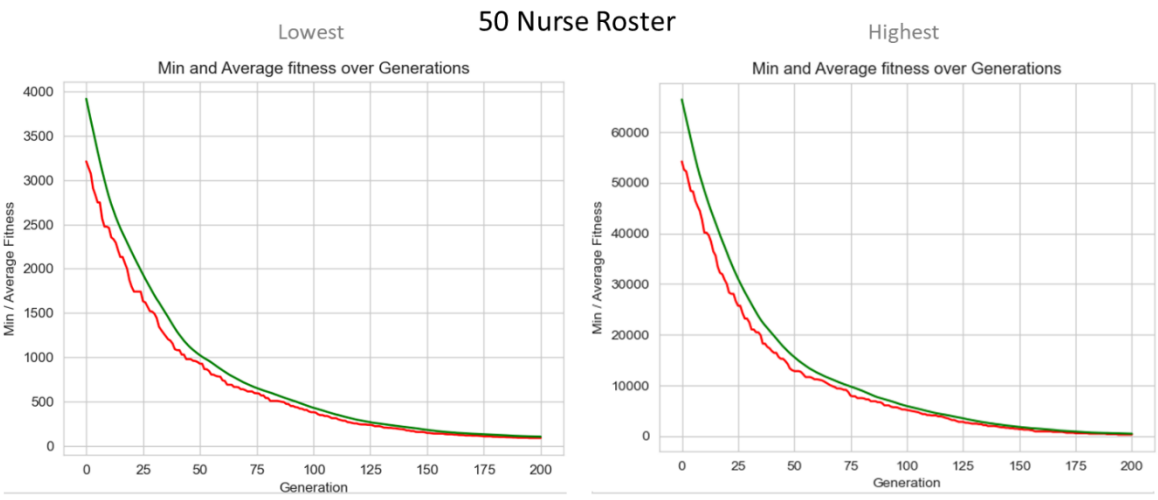


Figure 21

Search Two – Nurse Foster 50



II. Random Seed

The seed or random seed is a property which is at the forefront of every genetic algorithm. The seed is a starting point for the genetic algorithm which is, set by a random number generator (RNG), or manually by the developer through a property which will be referred to as RANDOM_SEED.

In the case of genetic algorithms, the algorithms are comprised of a population with a determined size. The seed is a starting point which influences the sequence of random numbers each-individual within the population is composed of (Generative Design Primer, 2022).

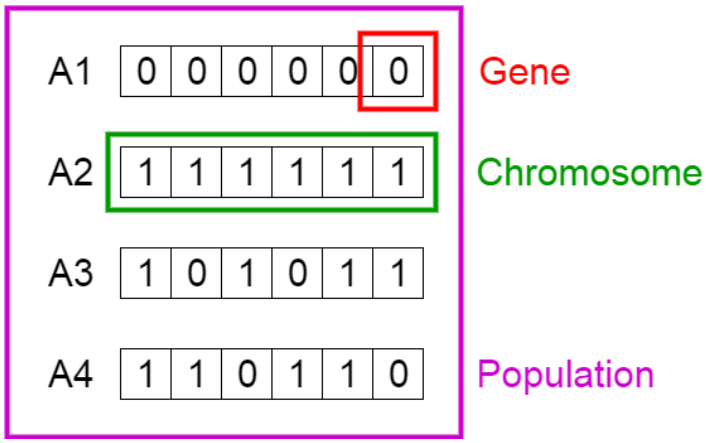
III. Population Size

The genetic algorithm process initiates with a set of individuals which is referred to as a population. Each individual is a solution to a problem the genetic algorithm was developed to solve. An individual is made distinctive via a set of parameters (variables) known as Genes. Genes are fused together into a string to form a Chromosome (solution) (Mallawaarachchi, 2017).

In a genetic algorithm, the set of genes of an individual is represented using a string of binary values (1s and 0s). It may be said that genetic algorithms encode genes within a chromosome (Mallawaarachchi, 2017).

Figure 22

Development of Chromosome



(Mallawaarachchi, 2017)

F. Termination

Introduce the concept of termination and how it is used within GA. What settings you have changed for termination and how it impacted results.

The primary concept behind the termination evolution mechanism is to determine when to stop a genetic algorithm. The termination of a genetic algorithm may be performed under four alternative conditions, satisfactory solution, maximum generations, no improvement, or time limit.

Whilst undergoing development and testing of the genetic algorithm to locate the optimal solution to the nurse scheduling problem, it was established that the maximum generations termination condition was the leading termination condition for the job of rostering 15 and 50 nurses correspondingly.

The maximum generations condition states a preset limit on the number of generations may serve as a stopping basis. The algorithm may terminate once it reaches this limit (Sugandhi, 2023). For example, a programmer may set a preset limit of fifty generations, this would result in the genetic algorithm running for fifty generations before returning the optimal solution.

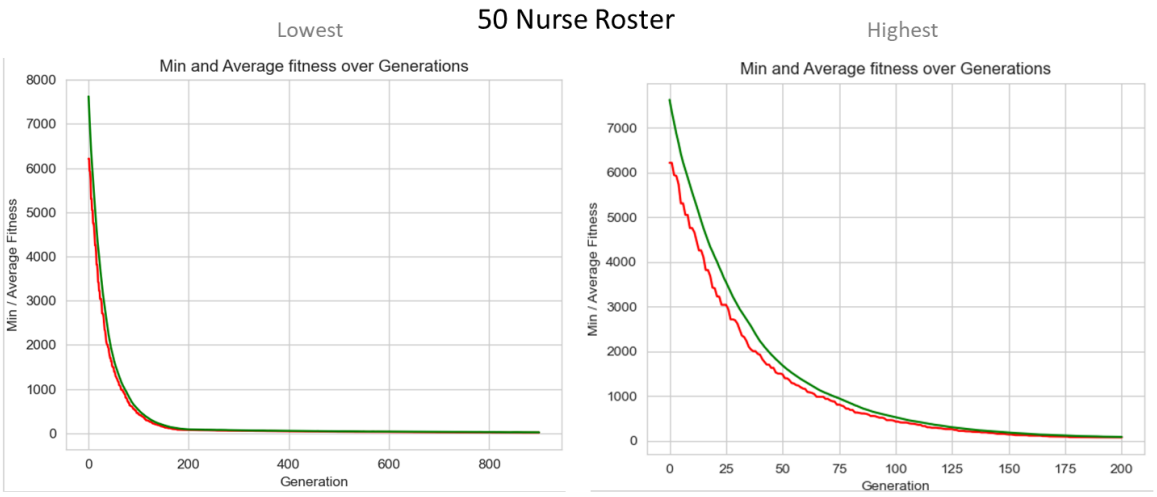
I. Results

The tests executed during phase eight of development – Termination were astonishingly fluctuating. Nurse Roster 15 displayed relatively static results for search one and two. The search one algorithm showed static results for the fitness value, with four tests performed, three tests resulted in the fitness value of 130.0, whilst the fourth resulted in a fitness value of 136.0. In conjunction, the violations saw a similar change of variation. The search two algorithm displayed similar static results to those found within search one. The search algorithm performed eight tests, six tests resulted in a fitness value of 115.0, whilst the seventh and eighth tests resulted in fitness values of 120.0 and 124.0 respectively. However, numerous variations were observed in the violations.

Nurse Roster 50 displayed drastic improvements for search one and two. The search one algorithm showed distinguishable results for the fitness value from 14.0 to 75.0 (as shown in Figure x). The lowest fitness value exceedingly improved upon the base fitness value of 75.0. In addition to this, the search two algorithm showed similar considerable results for the fitness value from 15.0 to 101.0. The smallest value made an extreme improvement upon the base fitness value of 101.0. A similar change was observed for the violations in nurse roster 50 for search one and two.

Figure 23

Search One – Nurse Roster 50



4. Results

The results section of the report displays all the relevant results from the above-described methodologies. If you have multiple figures that are very similar located in your appendix, place one example in the text and refer the reader to the appendix where they can find the rest. All alternative solutions should also be presented here to ensure that the best result is presented.

A. Settings

List the Settings You Have Utilized for the Testing of All Functions / Mechanisms Here.

I. Generic Settings

List the Settings You Have Utilized for the Testing of Generic Settings (Settings that Belong to No Particular Search Solution) Here.

Generic Settings								
Ver	Population	Selection	Cross-Over	Mutation	Termination	Hall of Fame	Hard Constraint Penalty	Random Seed
Selection Function								

1	300	selNSGA2	Two Point, 0.9	mutFlipBit, 0.1	200	30	10	42
2	300	selWorst	Two Point, 0.9	mutFlipBit, 0.1	200	30	10	42
3	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	30	10	42
4	300	Tournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	30	10	42

II. Search One - Settings

List the Settings You Have Utilized for the Testing of the solve-nurses-v1.py Solution Here.

Settings								
Ver	Population	Selection	Cross-Over	Mutation	Termination	Hall of Fame	Hard Constraint Penalty	Random Seed
Hall of Fame								
1	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	30	10	42
2	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	20	10	42
3	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	50	10	42
4	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	45	10	42
5	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	12	10	42
6	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	5	10	42
Cross-Over Function								
1	300	selTournament, 2	Two Point, 0.2	mutFlipBit, 0.1	200	12	10	42
2	300	selTournament, 2	Two Point, 0.5	mutFlipBit, 0.1	200	12	10	42
3	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	12	10	42

4	300	selTournament, 2	Two Point, 0.2	mutFlipBit, 0.1	200	5	10	42
5	300	selTournament, 2	Two Point, 0.5	mutFlipBit, 0.1	200	5	10	42
6	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	5	10	42
Mutation Function								
1	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	12	10	42
2	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	12	10	42
3	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	12	10	42
4	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.1	200	5	10	42
5	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
6	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	10	42
Hard Constraint Penalty								
1	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
2	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	90	42
3	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	50	42
4	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	20	42
5	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	5	42
6	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	10	42
7	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	90	42

8	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	50	42
9	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	20	42
10	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.9	200	5	5	42
Random Seed								
1	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	5	42
2	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	5	70
3	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	5	12
4	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
5	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	70
6	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	12
Population Size								
1	300	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
2	250	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
3	500	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
Termination								
1	500	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	200	5	10	42
2	500	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	1000	5	10	42
3	500	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	573	5	10	42
4	500	selTournament, 2	Two Point, 0.9	mutFlipBit, 0.5	900	5	10	42

III. Search Two - Settings

List the Settings You Have Utilized for the Testing of the solve-nurses-v2.py Solution Here.

Settings								
Ver	Population	Selection	Cross-Over	Mutation	Termination	Hall of Fame	Hard Constraint Penalty	Random Seed
Search Two								
Hall of Fame								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	70	10	42
2	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	60	10	42
3	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	50	10	42
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	40	10	42
5	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	45	10	42
Cross-Over Function								
1	300	selBest	Two Point, 0.2	mutFlipBit, 0.1	200	45	10	42
2	300	selBest	Two Point, 0.5	mutFlipBit, 0.1	200	45	10	42
3	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	45	10	42
4	300	selBest	Two Point, 0.2	mutFlipBit, 0.1	200	50	10	42
5	300	selBest	Two Point, 0.5	mutFlipBit, 0.1	200	50	10	42
6	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	50	10	42
Mutation Function								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	45	10	42
2	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	10	42

3	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	45	10	42
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.1	200	50	10	42
5	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	50	10	42
6	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	10	42
Hard Constraint Penalty								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	10	42
2	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	90	42
3	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	50	42
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	20	42
5	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	5	42
6	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	10	42
7	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	90	42
8	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	50	42
9	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	20	42
10	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	42
Random Seed								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	10	42
2	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	10	70

3	300	selBest	Two Point, 0.9	mutFlipBit, 0.5	200	45	10	12
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	42
5	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	70
6	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	12
Population Size								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	70
2	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	70
3	500	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	70
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	12
5	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	12
6	500	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	12
Termination								
1	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	70
2	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	1000	50	5	70
3	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	361	50	5	70
4	300	selBest	Two Point, 0.9	mutFlipBit, 0.9	1500	50	5	70
5	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	200	50	5	12
6	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	1000	50	5	12

7	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	532	50	5	12
8	250	selBest	Two Point, 0.9	mutFlipBit, 0.9	1500	50	5	12

B. Results

List the Results from Testing all the Functions / Mechanisms Here.

I. 15 Nurse Roster - Results

List the Results from Testing all the Functions / Mechanisms for the 15 Nurse Roster Here.

Figure 24

15 Nurse Roster - Results

15 Nurse Roster - Results					
Version	Fitness Value	Consecutive Shift Violations	Shifts Per Week Violations	Nurses Per Shift Violations	Shift Preference Violations
Selection Function					
1	254.0	0	2	21	24
2	495.0	13	18	15	35
3	251.0	0	2	21	21
4	248.0	0	2	21	18
Hall of Fame					
Search One					
1	248.0	0	2	21	18
2	247.0	0	3	20	17
3	251.0	0	2	21	21
4	249.0	0	0	23	19
5	244.0	0	1	22	14
6	248.0	0	2	21	18
Search Two					
1	252.0	0	2	21	22
2	250.0	0	2	21	20
3	249.0	0	2	21	19
4	257.0	0	2	21	27
5	249.0	0	2	21	19
Cross-Over Function					
Search One					
1	251.0	0	1	22	21
2	272.0	1	3	20	32
3	244.0	0	1	22	14
4	260.0	1	3	20	20

5	249.0	0	1	22	19
6	248.0	0	2	21	18
Search Two					
1	250.0	0	2	21	20
2	251.0	0	2	21	21
3	249.0	0	2	21	19
4	258.0	0	4	19	28
5	251.0	0	2	21	21
6	249.0	0	2	21	19
Mutation Function					
Search One					
1	244.0	0	1	22	14
2	239.0	0	7	16	9
3	239.0	0	5	18	9
4	248.0	0	2	21	18
5	237.0	0	1	22	7
6	237.0	0	6	17	7
Search Two					
1	249.0	0	2	21	19
2	236.0	0	1	22	6
3	238.0	0	7	16	8
4	249.0	0	2	21	19
5	240.0	0	4	19	10
6	239.0	0	3	20	9
Hard Constraint Penalty					
Search One					
1	237.0	0	1	22	7
2	2080.0	0	3	20	10
3	1160.0	0	3	20	10
4	470.0	0	3	20	10
5	122.0	0	3	20	7
6	237.0	0	6	17	7
7	2078.0	0	8	15	8
8	1158.0	0	8	15	8
9	468.0	0	8	15	8
10	122.0	0	3	20	7
Search Two					
1	236.0	0	1	22	6
2	2083.0	0	1	22	13

3	1163.0	0	1	22	13
4	471.0	0	4	19	11
5	122.0	0	2	21	7
6	239.0	0	3	20	9
7	2076.0	0	3	20	6
8	1156.0	0	3	20	6
9	468.0	0	2	21	8
10	126.0	0	1	22	11
Random Seed					
Search One					
1	122.0	0	3	20	7
2	126.0	0	6	17	11
3	123.0	0	4	19	8
4	237.0	0	1	22	7
5	243.0	0	2	21	13
6	241.0	0	4	19	11
Search Two					
1	236.0	0	1	22	6
2	238.0	0	1	22	8
3	240.0	0	2	21	10
4	126.0	0	1	22	11
5	120.0	0	1	22	5
6	121.0	0	2	21	6
Population Size					
Search One					
1	237.0	0	1	22	7
2	245.0	0	4	19	15
3	236.0	0	2	21	6
Search Two					
1	120.0	0	1	22	5
2	125.0	0	3	20	10
3	128.0	0	3	20	13
4	121.0	0	2	21	6
5	124.0	0	2	21	9
6	122.0	0	3	20	7
Termination					
Search One					
1	236.0	0	2	21	6
2	230.0	0	5	18	0

3	230.0	0	5	18	0
4	230.0	0	5	18	0
Search Two					
1	120.0	0	1	22	5
2	115.0	0	0	23	0
3	115.0	0	0	23	0
4	115.0	0	0	23	0
5	124.0	0	2	21	9
6	115.0	0	6	17	0
7	115.0	0	6	17	0
8	115.0	0	6	17	0

II. 50 Nurse Roster - Results

List the Results from Testing all the Functions / Mechanisms for the 50 Nurse Roster Here.

Figure 25

50 Nurse Roster - Results

50 Nurse Roster - Results					
Version	Fitness Value	Consecutive Shift Violations	Shifts Per Week Violations	Nurses Per Shift Violations	Shift Preference Violations
Selection Function					
1	434.0	14	8	10	114
2	3739.0	120	146	90	179
3	335.0	8	10	4	115
4	245.0	10	3	2	95
Hall of Fame					
Search One					
1	245.0	10	3	2	95
2	237.0	12	2	0	97
3	233.0	10	1	2	103
4	203.0	7	3	0	103
5	218.0	11	1	0	98
6	201.0	9	1	1	91
Search Two					
1	295.0	12	5	1	115
2	276.0	14	2	1	106
3	276.0	12	3	3	96
4	396.0	15	10	4	106
5	331.0	13	4	5	111
Cross-Over Function					
Search One					

1	999.0	44	34	10	119
2	513.0	23	14	4	103
3	218.0	11	1	0	98
4	1116.0	49	39	13	106
5	668.0	25	25	5	118
6	201.0	9	1	1	91
Search Two					
1	1811.0	79	65	23	141
2	675.0	31	22	4	105
3	331.0	13	4	5	111
4	1949.0	73	77	31	139
5	599.0	27	19	3	109
6	276.0	12	3	3	96
Mutation Function					
Search One					
1	218.0	11	1	0	98
2	101.0	1	0	0	91
3	84.0	0	0	0	84
4	201.0	9	1	1	91
5	77.0	0	0	0	77
6	96.0	1	0	0	86
Search Two					
1	249.0	0	2	21	19
2	236.0	0	1	22	6
3	238.0	0	7	16	8
4	276.0	12	3	3	96
5	118.0	2	0	0	98
6	112.0	1	1	0	92
Hard Constraint Penalty					
Search One					
1	77.0	0	0	0	77
2	182.0	1	0	0	92
3	142.0	1	0	0	92
4	92.0	0	0	0	92
5	90.0	1	0	0	85
6	96.0	1	0	0	86
7	94.0	0	0	0	94
8	94.0	0	0	0	94
9	82.0	0	0	0	82

10	91.0	1	0	0	86
Search Two					
1	123.0	2	0	0	103
2	276.0	2	0	0	96
3	196.0	2	0	0	96
4	119.0	1	0	0	99
5	121.0	5	1	0	91
6	112.0	1	1	0	92
7	91.0	0	0	0	91
8	91.0	0	0	0	91
9	104.0	0	0	0	104
10	87.0	0	0	0	87
Random Seed					
Search One					
1	90.0	1	0	0	85
2	92.0	0	0	0	92
3	93.0	1	0	0	88
4	77.0	0	0	0	77
5	124.0	3	0	0	94
6	77.0	0	0	0	77
Search Two					
1	123.0	2	0	0	103
2	192.0	6	2	3	82
3	132.0	3	1	0	92
4	87.0	0	0	0	87
5	101.0	1	0	1	91
6	87.0	1	0	0	82
Population Size					
Search One					
1	77.0	0	0	0	77
2	97.0	0	0	0	97
3	75.0	0	0	0	75
Search Two					
1	101.0	1	0	1	91
2	98.0	1	0	0	93
3	93.0	0	0	0	93
4	93.0	0	0	0	93
5	92.0	2	0	0	82
6	88.0	1	0	0	83

Termination					
Search One					
1	75.0	0	0	0	75
2	14.0	0	0	0	14
3	31.0	0	0	0	31
4	14.0	0	0	0	14
Search Two					
1	101.0	1	0	1	91
2	26.0	0	0	0	26
3	61.0	0	0	0	61
4	25.0	0	0	0	25
5	92.0	2	0	0	82
6	23.0	0	0	0	23
7	41.0	0	0	0	41
8	15.0	0	0	0	15

C. Completion

Explain table contents and best results. Explain why one algorithm was better than the other. Explain why you think the settings utilized resulted in one algorithm outperforming the other.

The results tables shown in Figure 24 and 25 above, reveal a few key aspects of the testing methodology utilized to optimize the search one and two algorithms. Optimization was performed in eight stages, following the genetic algorithm methods (functions) linearly, selection, hall of fame, cross-over, mutation, hard constraint penalty, random seed, population size, and termination. The 15 and 50 Nurse Roster tables are each comprised of ninety-two tests throughout the eight test stages.

The optimal solutions found in search one and two for nurse rosters 15 and 50 are relatively similar. This is the consequence of Tournament Selection and Best Selection behaving in a similar manner. Whilst Tournament selection selects I individuals from the current generation to compare them against one another to find the superior individual from the selection to be a parent for the next generation. Best Selection selects the best individuals from the entire population to be parents for the next generation.

Nurse Roster 15 displayed satisfactory results for search one and two. The search one algorithm exhibited adequate results for nurse roster 15 with a fitness value of 230.0, in addition to 0 consecutive shift violations and shift preference violations, alongside shift per week violations and nurses per shift violations of 5 and 18 respectively (as seen in Figure 27). As a result, the total violations encountered were 23. In conjunction with this, the search two algorithm displayed sufficient results for nurse roster 15 with a fitness value of 115.0, additionally, 0 consecutive shift violations and shift preference violations were recorded, alongside this, shift per week violations and nurses per shift violations recorded values of 6 and 17 correspondingly (as seen in Figure 27). Strangely, when testing, nurse roster 15 seemed to remain at a minimum fitness value of 230.0 to 115.0 which caused nurse roster 50 which did not have this issue to skyrocket above nurse roster 15.

Nurse Roster 50 showed remarkable results for search one and two. The search one algorithm displayed impressive results for nurse roster 50 with a fitness value of 14.0, in addition to 0 consecutive shift violations, shift per week violations, and nurses per shift violations, alongside 14 shift preference violations (as seen in Figure 29). Search one for nurse roster 50, far surpassed the results for nurse roster 15. However, in conjunction with this, search two for nurse roster 50 displayed similar phenomenal results for nurse roster 50 with a fitness value of 15.0, additionally, consecutive shift violations, shift per week violations, and nurses per shift violations saw 0 faults, alongside 15 shift preference violations (as seen in Figure 29).

Comprehensible, it is clear to say that the results for nurse roster 15 and 50 in relation to search one and two are profoundly similar. However, this does not translate to performance. Nurse Roster 15 would perform best with search two. This is the consequence of search one having a more substantial termination rate at 573, compared to search two's low termination rate of 532 which is 39 extra generations that could be skipped. This would result in a similar violation rate whilst not having to create as many generations. However, this does not hold true for nurse roster 50. Nurse Roster 50 would perform exceedingly better with search one as opposed to search two. This is the result of search two having a considerably higher termination rate at 1500, in comparison to search one's low termination rate of 900. In this scenario, search two has an exceedingly higher termination rate than search one with a further 600 generations that could be removed from the equation. This would result in a lower violation rate whilst not having to develop as many generations.

Figure 26

Nurse Roster 15 – Settings

Nurse Roster 15 - Settings				
	Solve Nurses V1	Probability / Size	Solve Nurses V2	Probability / Size
Population		500		250
Selection	selTournament	2	selBest	
Cross-Over	cxTwoPoint	0.9	cxTwoPoint	0.9
Mutation	mutFlipBit	0.5	mutFlipBit	0.9
Termination	Max Generations	573	Max Generations	532
Hall of Fame		5		50
Hard Constraint Penalty		10		5
Random Seed		42		12

Figure 27

Nurse Roster 15 – Results

Nurse Roster 15 - Results		
	Solve Nurses V1	Solve Nurses V2
Fitness Value	230.0	115.0
Consecutive Shift Violations	0	0
Shift Per Week Violations	5	6
Nurses Per Shift Violations	18	17
Shift Preference Violations	0	0

Figure 28

Nurse Roster 50 – Settings

Nurse Roster 50 - Settings				
	Solve Nurses V1	Probability / Size	Solve Nurses V2	Probability / Size
Population		500		250
Selection	selTournament	2	selBest	
Cross-Over	cxTwoPoint	0.9	cxTwoPoint	0.9
Mutation	mutFlipBit	0.5	mutFlipBit	0.9

Termination	Max Generations	900	Max Generations	1500
Hall of Fame		5		50
Hard Constraint Penalty		10		5
Random Seed		42		12

Figure 29

Nurse Roster 50 – Results

Nurse Roster 50 - Results		
	Solve Nurses V1	Solve Nurses V2
Fitness Value	14.0	15.0
Consecutive Shift Violations	0	0
Shift Per Week Violations	0	0
Nurses Per Shift Violations	0	0
Shift Preference Violations	14	15

5. Discussion

Provide some discussion of your results here. A clear explanation of why one algorithm was better than the other. Explanation of why you think the settings used results in one algorithm outperforming the other. Describe something interesting learned as part of this project and your opinion on the usefulness of generic algorithms.

Unfortunately, due to time constraints, I was not able to complete this portion of the assignment.

6. Conclusions

This section summarizes your report and provides closure. No new information should be present but expand on what your recommendations are for the nurse scheduling problem, given the results of your testing.

Unfortunately, due to time constraints, I was not able to complete this portion of the assignment.

References

Generative Design Primer. (2022). *Genetic Algorithm Q&A.* https://www.generativedesign.org/02-deeper-dive/02-06_faq-under-the-hood

Isaac Computer Science. (n.d.). *A* Search Algorithm*. https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all

Mallawaarachchi, V. (2017, July 8). *Introduction to Genetic Algorithms – Including Example Code*. Towards Data Science. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Nanakorn, P; Meesomklin, K. (2001). *An Adaptive Penalty Function in Genetic Algorithms for Structural Design Optimization.* Science Direct. https://www.sciencedirect.com/science/article/abs/pii/S0045794901001377#:~:text=In%20genetic%20algorithms%2C%20constraints%20are,the%20degrees%20of%20constraint%20violation.

Overide. (2023, April 19). *Genetic Algorithms*. Geeks for Geeks. https://www.geeksforgeeks.org/genetic-algorithms/

Ravikiran, S. (2022, August 9). *A* Algorithm Concepts and Implementation.* https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

Shyalika, C. (2019, January 30). *Genetic Algorithms - Selection*. Medium. https://medium.datadriveninvestor.com/genetic-algorithms-selection-5634cfc45d78

Soemers, D. (2018, July 17). *Uniformed and Informed Search Techniques*. Stack Exchange. https://ai.stackexchange.com/questions/7179/which-are-more-memory-efficient-uninformed-or-informed-search-algorithms

Sugandhi, A. (2023, September 5). Genetic Algorithms in Machine Learning: an Introduction. *Knowledge Hut*. https://www.knowledgehut.com/blog/data-science/genetic-algorithm-in-machine-learning