COMP30050: Software Engineering Project 3

Dr. Tony Veale

Final Report

Due on 29th April 2016

Lamp
Joe Duffin - 13738019
Edwin Keville - 13718661
Niamh Kavanagh - 12495522
Gerard Fogarty - 13303911 (*the lost lamp*)

Contents

Introduction (Spec of project)

As part of our third year software engineering module, all students were required to develop a stochastic search project allocation system. This is to be completed in groups, with the initial framework designed by each individual member through weekly assignments, before amalgamating each member's solution and adding the functionality of two different solution algorithms. The end result produces an optimal allocation of data in a user-friendly format which can be saved as desired.
The project itself involves reading in a list of students and their ranked list of preferences for final year projects within computer science. Some students may have a project preassigned to them, in which case they will have no other preferences and will not be altered in the allocation process. Any students without a preassigned project will then be assigned a project using our chosen algorithms in an attempt to minimize overall student disappointment.
The proposed solution for this mapping of students to projects was developed using both a simulated annealing and a genetic algorithm. The simulated annealing approach provides a singular solution, which it adapts through a serious of changes to individual student-project mappings until a good viable solution is achieved. The genetic algorithm creates a population of random solutions, with solutions merging and being culled until a single optimal solution is reached.
To allow our software to be user-friendly, we created a Graphical User Interface and a Command Line Interface. The GUI allows users to load the data file of their choice and run our hybrid algorithm composed of a genetic algorithm populated by simulated annealing solutions, then gives the option to save these results into a spreadsheet format with the energy of the solution displayed at the bottom. The CLI uses command line arguments provided by the user to specify the desired input file, which algorithm to be used, the parameters for each algorithm and the desired format of the output file.
The following report details our software in full, discussing both algorithms used and the hybrid we created of both, a user manual for accessibility, and the entire development process undertook to create the final project.

What we produced/interfaces

One Gui and 3 command line interfaces. We wanted a fine mix of an ultimate solution, but also lots of modularity and re-usability.

The Graphical User Interface

The hybrid solution, the actual submission.

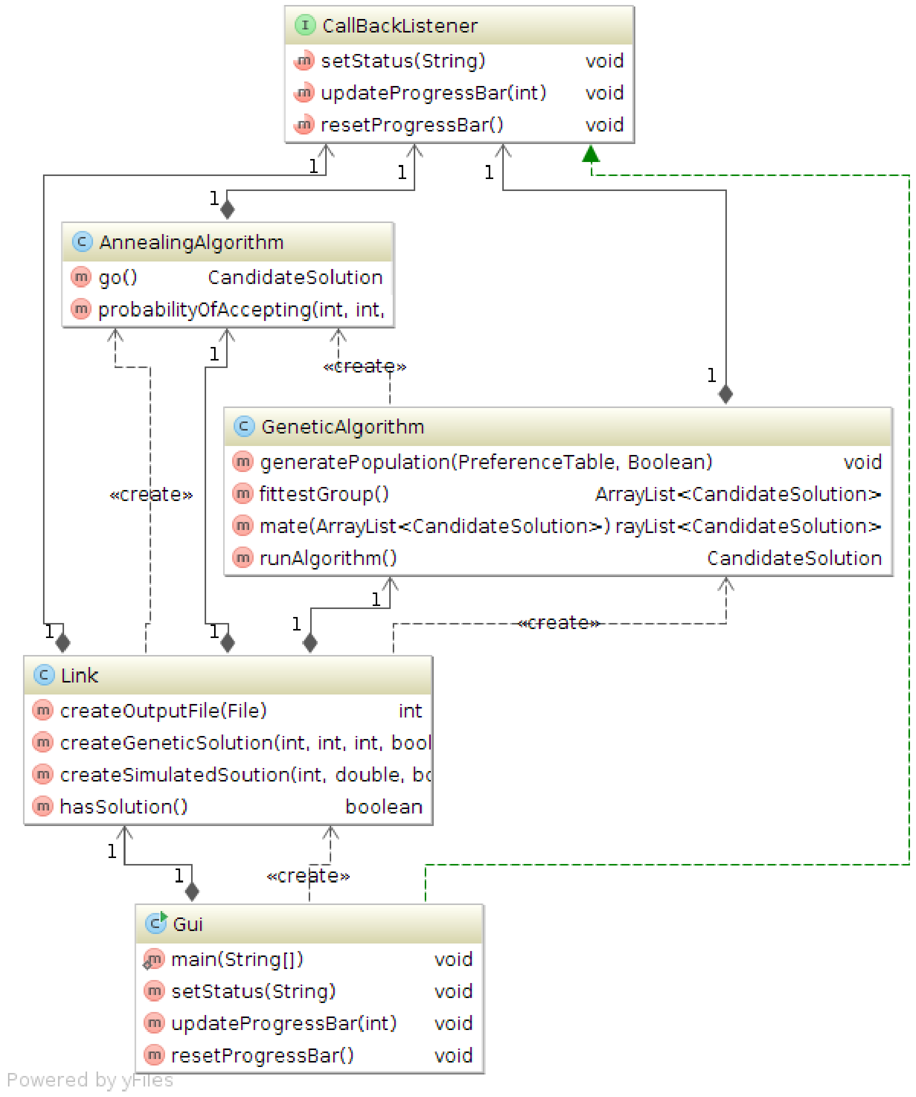
The Command Line Version

hi

Class Diagrams

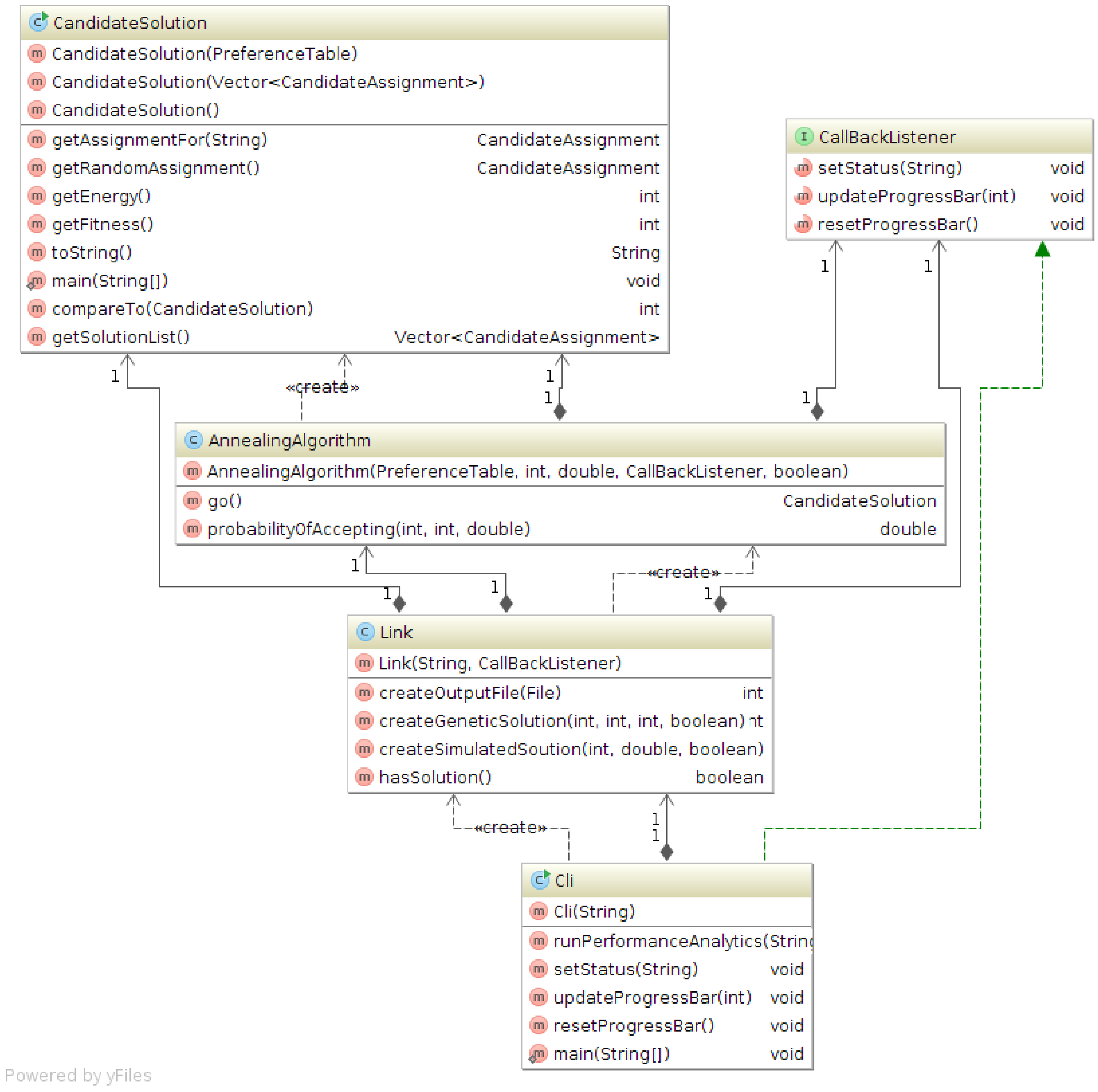
The Gui with Simulated Annealing and the Genetic Algorithm

The Gui relies on the Link to get access to both the Simulated Annealing and Genetic Algorithms. The 1 to 1 relationships between the CallBackListener and the algorithms allow the algorithms to update the status bar and text on the Gui.


CallBackListener
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
AnnealingAlgorithm
go() CandidateSolution
probabilityOfAccepting(int, int,
GeneticAlgorithm
generatePopulation(PreferenceTable, Boolean) void
fittestGroup() ArrayList<CandidateSolution>
mate(ArrayList<CandidateSolution>) rayList<CandidateSolution>
runAlgorithm() CandidateSolution
Link
createOutputFile(File) int
createGeneticSolution(int, int, int, bool
createSimulatedSoution(int, double, bo
hasSolution() boolean
Gui
main(String[]) void
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
«create»
1
Powered by yFiles

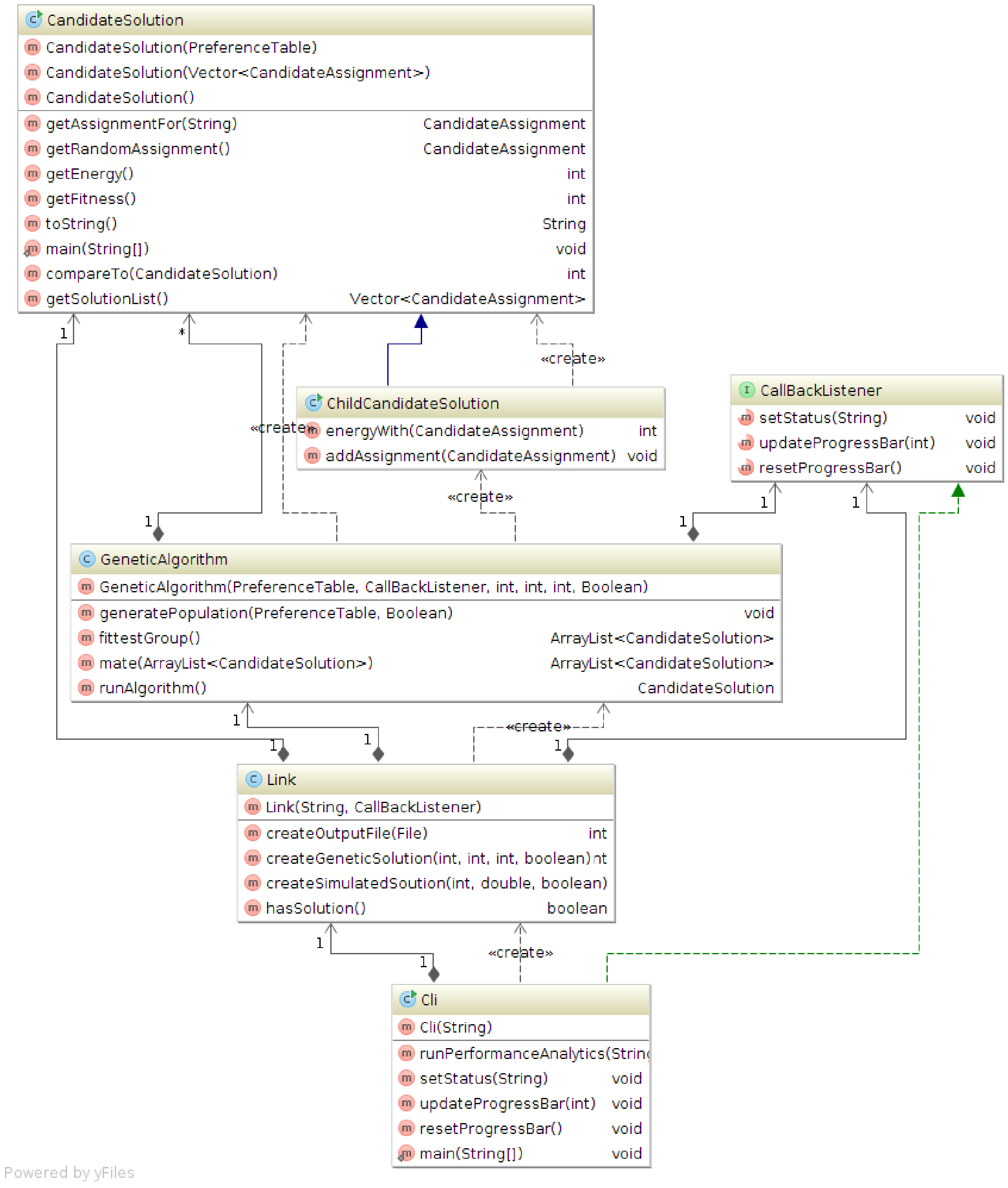
Cli for Simulated Annealing

The Cli has a 1 to 1 relationship with the Link, which here is shown to give access to the Simulated Annealing algorithm. This, in turn, has a 1 to 1 relation ship with a Candidate Solution which it permutes.


CandidateSolution
CandidateSolution(PreferenceTable)
CandidateSolution(Vector<CandidateAssignment>)
CandidateSolution()
getAssignmentFor(String) CandidateAssignment
getRandomAssignment() CandidateAssignment
getEnergy() int
getFitness() int
toString() String
main(String[]) void
compareTo(CandidateSolution) int
getSolutionList() Vector<CandidateAssignment>
CallBackListener
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
AnnealingAlgorithm
AnnealingAlgorithm(PreferenceTable, int, double, CallBackListener, boolean)
go() CandidateSolution
probabilityOfAccepting(int, int, double) double
Link
Link(String, CallBackListener)
createOutputFile(File) int
createGeneticSolution(int, int, int, boolean) nt
createSimulatedSoution(int, double, boolean)
hasSolution() boolean
Cli
Cli(String)
runPerformanceAnalytics(Strin
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
main(String[]) void
«create»
«create»
«create»
1
1
1
1
Powered by yFiles

Cli for the Genetic Algorithm

When the user elects to use the Genetic Algorithm the 1 to 1 relationship between the Link and the Genetic Algorithm is used. In the diagrams below the inheritance relationship the ChildCandidateSolution and the CandidateSolution is evident.


CandidateSolution
CandidateSolution(PreferenceTable)
CandidateSolution(Vector<CandidateAssignment>)
CandidateSolution()
getAssignmentFor(String) CandidateAssignment
getRandomAssignment() CandidateAssignment
getEnergy() int
getFitness() int
toString() String
main(String[]) void
compareTo(CandidateSolution) int
getSolutionList() Vector<CandidateAssignment>
ChildCandidateSolution
energyWith(CandidateAssignment) int
addAssignment(CandidateAssignment) void
CallBackListener
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
GeneticAlgorithm
GeneticAlgorithm(PreferenceTable, CallBackListener, int, int, int, Boolean)
generatePopulation(PreferenceTable, Boolean) void
fittestGroup() ArrayList<CandidateSolution>
mate(ArrayList<CandidateSolution>) ArrayList<CandidateSolution>
runAlgorithm() CandidateSolution
Link
Link(String, CallBackListener)
createOutputFile(File) int
createGeneticSolution(int, int, int, boolean) int
createSimulatedSoution(int, double, boolean)
hasSolution() boolean
Cli
Cli(String)
runPerformanceAnalytics(Strin
setStatus(String) void
updateProgressBar(int) void
resetProgressBar() void
main(String[]) void
«create»
1
*
Powered by yFiles

Performance Analysis

Each algorithm has its own set of parameters which are needed and these have a significant effect on the overall energy of the given solution. We wanted our Gui to be a simple no frills program to give the ultimate solution so we removed the option of user input parameters. They are set as constant. In order to determine the best constants we ran overnight tests, subjecting each algorithm to a barrage of parameters. These results can be reproduced using the '-analyse' tag with in Cli version of our code.

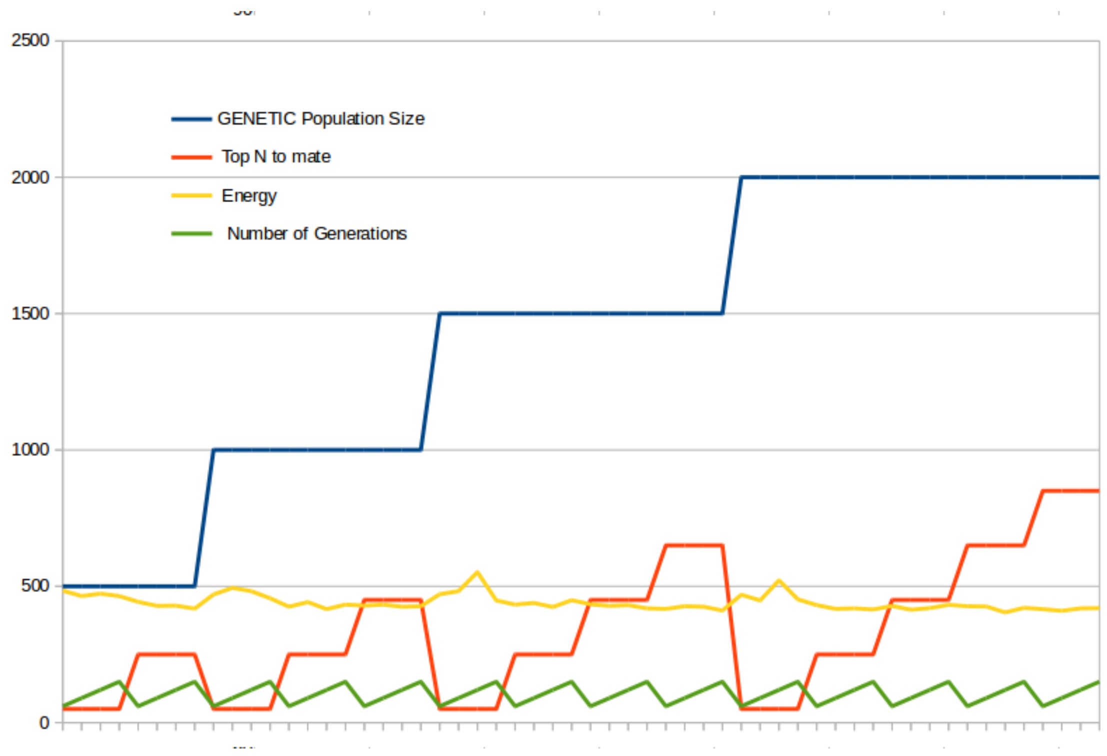
If an advanced user wishes to specify their own parameters, the Cli version of our program will facilitate that.

The Genetic Alogrithm

The Genetic Algorithm has 3 parameters:

- The initial population size
- How may of the population to choose for mating (The top N)
- The number of generations the population should evolve through

The overnight test yielded the following graph.



The algorithm was ran with a variety of population sizes, number to mate and number of generation and the best parameters which gave the lowest energy were extracted.

Population Size: 2000

Top N: 650

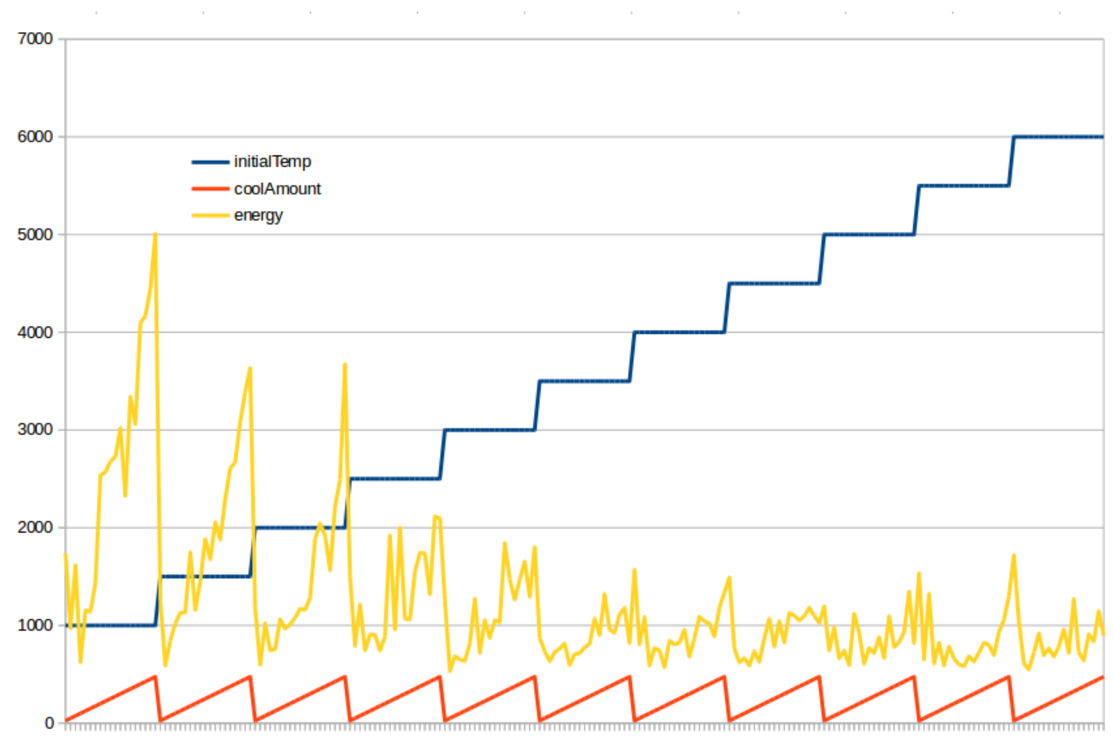
Number of Generations: 120

The Simulated Annealing alogrithms

The Simulated Annealing Algorithm has 2 parameters:

- The initial temperature of the solution
- The amount the solution cools on each iteration.

The overnight test yielded the following graph.



The algorithm was ran with a variety of cooling amounts for a variety of initial temperatures. The cool amount has been scaled by a factor of 100 for clarity. The best parameters are shown below.

Initial Temperature 3000

Cooling Amount 0.5

Development Phases/Issues

The Genetic Algorithm

mate was the biggun
initially when mating 2 solutions we choose the 'happiest' corresponding assignments from each and created a new assignment with these. Each assignment was independently better, the solution as a whole was pants as there were lots of penalties incurred.
We move to considering a complete child solution and used the notion of 'happier with' to determine which parent's assignment should be used.

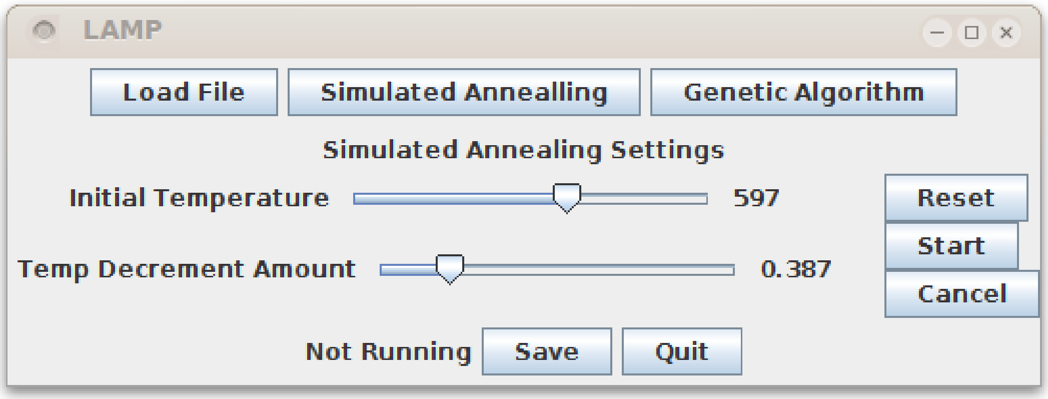
extras
checking for duplicates manually (winning) - bad
mutations, several attempts - bad

The Simulated Annealing Algorithm

The key turning point being understanding the relationship between initial tmeperature and cooling amount. Number iterations is a function of these two.

The Gui

The Gui was initially overcomplicated,,, after performance analysis we determined none of this was necessary. We elected for a simple LOAD GO SAVE option, with an epic title bar. It allows the user to create the ultimate solution, (or very close too) insert percentage.



Initial Temperature 597
Temp Decrement Amount 0.387

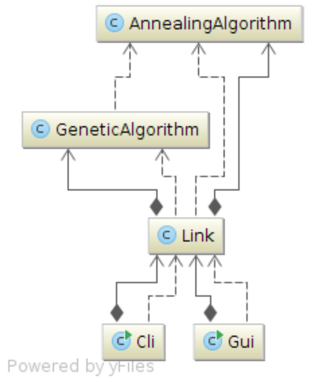
Our Development Model

Not sure what needs to go in here

Technical Details Of Note

Our Design Pattern

In keeping with our core values of simplicity and re-usability we tried to use a very modular design pattern. We identified two distinct types of classes that we created, worker classes and interfaces (the algorithms and the Cli/Gui respectively). An interface invokes methods on a worker. In order to modularise this we created a Link class. This is a pivotal class in our design. Every method call from an interface must go through the link. This gives the ability to pair any interface with any algorithm with ease. Once an interface has access to a Link it can call upon either or both of the algorithms as is necessary. The linear flow of our design pattern is very evident below.



Callback listeners

Independent communication between classes is typically one directional. Usually a class invokes a method on one of the classes that it has references to and a single value is returned upon completion. Sometimes the need for interaction with the calling class is needed during execution of the called method; this is where callback listeners come in. They provide the ability for a method to invoke a method on the calling class.

Use of callback listeners:

1. The listener is ultimately an interface. It declares which methods in the calling class are available to class containing the called method.
2. The calling class implements each of the methods declared in this interface.
3. When creating an instance of a class, whose methods will be called at some point, a reference to the calling class is passed using the ”this” keyword to the classes constructor.
4. That class, which needs the reverse communication path, then instantiates an object of type callback listener with the passed reference.
5. This allows method calls back to the calling class during execution and also keeps encapsulation tight as it does not provide the full public interface of the calling class.
6. Callback listeners can also be used with threads to alert the class which created the thread of completion or another significant event.

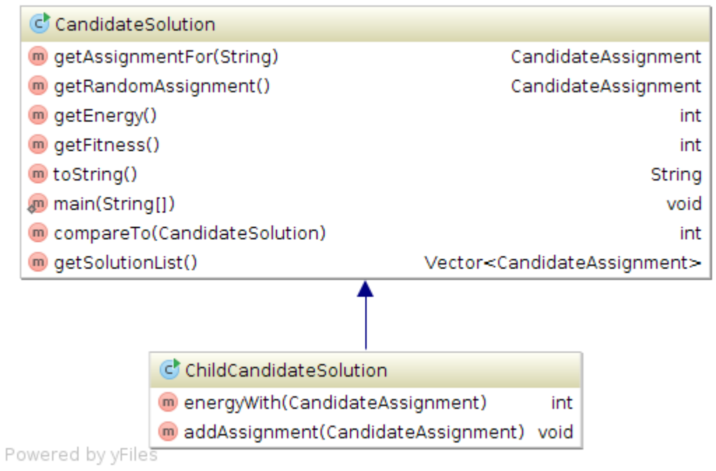
In our project methods in the algorithm classes are invoked by either of the interface classes (via the Link). We used a callback listener to allow both the Genetic and Simulated Annealing algorithms to invoke methods

on which ever interface invoked the algorithm. The callback listener is a direct connection straight back to the calling class.
We used the listener to update the progress bar every time 10 percent of the total number of iterations have been completed and also when the status bar needed updating with new information. Our two interfaces, the Cli and Gui, both implement the callback listener and have their own respective versions of the status bar and progress bar.

Inheritance

When building software, in order to efficiently utilise the classes at one's disposal. it is always important to be mindful of potential inheritance relationships. An inheritance relationship gives a subclass access to the methods and fields of the superclass. Inheritance is used when needing to add case specific functionality to a class, i.e. extra functionality that would otherwise bloat other clients of the class.

We identified a need to subclass the CandidateSolution class when creating the Genetic Algorithm. The Genetic Algorithm has to build a CandidateSolution one project at a time and also query the potential fitness of that solution during the building phase. This led to the creation of the ChildCandidateSolution class, which inherits from CandidateSolution, with two extra methods, evident below.


CandidateSolution
getAssignmentFor(String) CandidateAssignment
getRandomAssignment() CandidateAssignment
getEnergy() int
getFitness() int
toString() String
main(String[]) void
compareTo(CandidateSolution) int
getSolutionList() Vector<CandidateAssignment>
ChildCandidateSolution
energyWith(CandidateAssignment) int
addAssignment(CandidateAssignment) void
Powered by yFiles

Threads

By default programs run on one main thread. They have one time line, one stream of execution, on which the programs instructions are carried out consecutively. Sometimes, when expensive routines are needed to be called it's advantageous to put them on their own thread, their stream of execution. When the routine is complete is typically not known, but it will complete at some point, (assuming no runtime/logic errors). Callback listeners can be used to signal the main thread of completion or some other significant event.

In our project, the Cli runs in only one thread, and procedurally runs the algorithm and prints the the console to indicate its progress. During execution the user has no interactivity with the program as it's main thread is blocked by the expensive task of computing a solution.
When we designed the Gui it became evident that a separate execution thread was needed as there are two tasks to complete. The Jframe that is the Gui needs redrawing with the progress bar updates and the algorithm has to be executed. We created a separate thread on which the algorithm would run, allowing these two tasks to done concurrently.

Team Development

what we changed
what we liked/didnt,
stuckto/deviated from interim

What went well

the night out and the burgers

What didn't go so well

everything else

Who did what

everything else

The Team Night Out

Burgers and Drama

The development of a large software project involves both individual and group contributions. Many large tasks were broken up into smaller subsections which were divided between the team members. These were researched and brainstormed individually before meeting as a group and developing solutions together.
To keep morale high while we worked through the assignment, we organized a team bonding night between the interim report and the submission of the final report. This took place after a long group coding session in college where the bones of both algorithms were written. We felt that an evening of discussion on topics other than the software project would help keep us connected as a group, which would improve overall work ethic and quality of code produced.
The team night out involved travelling together into town for a large dinner, before returning to UCD campus for a production of The Nightman Cometh (an episode of the popular TV show It's Always Sunny in Philadelphia) by UCD's DramaSoc. A night off from code development and report writing was well deserved and earned, and allowed for the team to work harder upon our return to college the next day.

A Sad Day For Lamp

The loss of a team member

As a young adult, college can often be a very stressful and time consuming environment to be in. A large amount of students are dropping out of all courses, with 23% of students entering the computer science program in UCD not progressing past their first year. Unfortunately for Team Lamp, we experienced a loss of a member partway through the development process.
Team Lamp started off well, producing strong weekly assignments by each team member. Sadly, we found that as time passed, one member was slowly fading off the radar and was no longer contributing towards the module in both individual assessments and team projects. Gerard Fogarty, our lost Lamp, has unfortunately not been a presence within the team for several weeks.
Team Lamp has persevered through this loss, providing what we feel is still a marketable software solution to this module's brief. Although this module heavily suggested teams of four to produce a solid working solution, we feel that we overcame the obstacle of having only three team members and still maintained a high quality software project and solid written report.