Final Year Project Report

Full Unit - Interim

Metaheuristics

Luke Sell

A report submitted in part fulfilment of the degree of

MSci (Hons) in Computer Science

Supervisor: Eduard Eiben

Department of Computer Science
Royal Holloway, University of London

December 10, 2021

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 Where other published and unpublished source materials have been used, these have been acknowledged.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: 4023

Student Name: Luke Sell

Date of Submission: December 10, 2021

Table of Contents

Abstract

Introduction and Background

A Metaheuristic is a general type of heuristic that is useful in quickly finding near ideal solutions to a range of optimisation problems, and thus does not try to use the properties of a specific problem to modify how they search for a solution, but instead follows a general strategy for all problems. This is different in comparison to standard heuristics which are designed to solve specific problems and are not usable in general outside the scope of that problem. Metaheuristics are therefore able to work efficiently even without knowing anything about the problem and is also better at dealing with local minima than standard heuristics. Metaheuristics are most useful when finding the most optimal solution through an exhaustive search would take too long comparatively, for example, the travelling salesman problem, which is NP hard and has a search space that grows exponentially with the increasing size of the problem. There are different types of Metaheuristics with varying strategies used that each have their own advantages and disadvantages relative to each other and can be grouped by classifications based on how they search for a solution to a problem. Optimisation problems are also classified by how they are structured, but generally involve an objective function to be optimised, as well as variables, variable domains and constraints, for example pseudo boolean optimisation problems are problems where the variable domain is restricted to a boolean value, that is to say zero or one.

Aims and Objectives

The overall aim of this project is to understand the different strategies used by various metaheuristics in searching for solutions to optimisation problems, to design and implement examples of each of these metaheuristics to solve a wide range of these problems and compare the efficiency of each of these different algorithms by benchmarking them against common problems. To do this I will have to research these metaheuristics, explain how they work differently to each other to find solutions and how I can create each of these to solve example problems. I will also need to be able to demonstrate my implementations to a user through the use of a GUI that allows for each algorithm to be tested on a chosen example problem, with a way of benchmarking all of them against different problems to compare the efficiency and usefulness of each. Additionally I will need to follow good software engineering principles in the design and development of my programs, such as using a version control system, using agile development, unit testing, design patterns and so on, skills which I will gain further understanding of during this project and will make me a better software engineer. Finally in writing my reports I will need to critically analyse the material I am reading and how to use this in my project to create a thorough overview of the topic.

Overview of Early Deliverables

Proof of Concept Programs

To begin I will design and implemented the tabu search metaheuristic, this will require me to first set up an initial code structure and a basic user interface. I will be writing my code in C++17 and using the SDL 2 library for GUI handling, I will also be using catch 2 as my testing framework, once the code is working I will have to thoroughly test it on example problems to get a better understanding of how it works.

Reports

I will need to write reports on computational complexity, local search and tabu search, this will involve researching these topics using the sources I have found already and describing how they relate to the overall project. Computational complexity will be important in explaining the usefulness of metaheuristics in general as the problems they will be used on are NP hard, meaning the search space grows exponentially with larger problems, finding the best solution would be infeasible, so understanding this will allow me to motivate the use of metaheuristics instead to solve the problem. I will then begin research on the first two metaheuristics and will describe how they work and design the algorithms in pseudocode so that I can implement them later, I will also note their key properties and how this modifies the way they search for a solution, which will be useful later when I need to compare the individual metaheuristics against each other.

Overview of Final Deliverables

The Program

I will continue with implementing the other metaheuristics and testing them, if I have time I will extend this by implementing even more heuristics on other problems in extension to the pseudo boolean optimisation problem. I will also implement a GUI for the metaheuristics to test them, this will involve using the SDL 2 library, I will first create draft designs of what I expect this to look like, before coding it and then testing it with users to make sure it functions properly and intuitively. I will finally benchmark and compare the different metaheuristics by running them many times on different sized problems and recording the times in a log, which I will then use for my comparison.

The Report

I will continue my research with reports on iterated local search, genetic algorithms and memetic algorithms, this will further my understanding of different types of metaheuristics and allow me to design and implement versions of them and finally benchmark them on example problems for a comparison on the efficiency of each individual metaheuristic.

Timeline

Report on the basics of computational complexity from 08/10/2021 to 17/10/2021

Report on local search from 19/10/2021 to 27/10/2021

Report on tabu search from 30/10/2021 to 11/11/2021

Implementation of tabu search from 19/11/2021 to 30/11/2021

Interim submission on 03/12/2021

Preparation for presentation from 04/12/2021 to 05/12/2021

Report on iterated local search from 06/12/2021 to 12/12/2021

Report on genetic algorithms from 13/12/2021 to 20/12/2021

Report on memetic algorithms from 21/12/2021 to 28/12/2021

Draft GUI designs from 29/12/2021 to 30/12/2021

Implementation of GUI from 02/01/2022 to 10/01/2022

Testing of GUI from 11/01/2022 to 12/01/2022

Implementation of iterated local search from 13/01/2022 to 23/01/2022

Implementation of genetic algorithm from 25/01/2022 to 04/02/2022

Implementation of memetic algorithm from 06/02/2022 to 17/02/2022

Implementation of other metaheuristics from 19/02/2022 to 26/02/2022

Testing and benchmarking the algorithms from 01/03/2022 to 04/03/2022

Comparing the algorithms from 08/03/2022 to 16/03/2022

Proof reading the report and generating the code documentation and UML from 18/03/2022 to 23/03/2022

Final submission on 25/03/2022

Preparation for demonstration from 02/04/2022 to 18/04/2022

Risk Assessment

If the algorithms are too slow it might be difficult to benchmark and test them on large sets of example problems, this is very unlikely to be an issue as the metaheuristics should be very fast even on large sets and problems with large search spaces, but it is still important to make sure these algorithms are tested thoroughly, therefore I shall schedule plenty of time for this specific purpose.

I may have problems understanding some of the concepts needed for writing the reports, therefore I will make sure I have a wide range of different research sources that cover each report so that I can learn about each metaheuristic and cite them in my reports, I do not think this is likely to be a problem but it will still be important to have many different sources of varying types when writing the reports.

The GUI might not be intuitive enough for the user, might not display correctly or might not even function as intended, there is a high chance of this happening in some way, but it is not as important to mitigate as other potential risks as the program should hopefully still be mostly usable for the user, however I will try to mitigate this by doing sufficient user testing to observe any issues with the UX so that they can be fixed and testing the program on multiple computers to check that the resolution the GUI is displayed at is generally compatible with most modern computers that may be used by the user.

I might not be able to sufficiently and accurately benchmark the different metaheuristic implementations for comparison if I can not test them on enough problems of varying large random search spaces, this is important in proving the usefulness of the algorithms, but likely wont be a problem as I should be able to find many example problems to test the metaheuristics on, if it does happen however I will mitigate it by creating additional problems to use for testing that can help me give a good overview in comparison of where each metaheuristic succeeds or fails against the other algorithms.

It is very important that the program itself is runnable on a different computer, it is possible that due to hardware and software differences that some parts of the program may work differently or not work at all when run by another user, therefore to mitigate this risk I should thoroughly test my program works in all ways exactly as expected on different hardware and software combinations to find and fix any issues before submission of my project, the most likely issue might be not having up to date language distributions for C++, which I could solve by statically linking these into the compiled executable of the program.

Theory

Basics of Computational Complexity

Computational Complexity is used to define an algorithms or problems usage of resources, most importantly for metaheuristics, time and memory, and is expressed as a function of the size of the input for the algorithm or problem. Different complexity functions are often used, specifically big O notation, which describes the asymptotic behaviour with an input size approaching infinity, the most useful for benchmarking metaheuristics will be best, worst and average case. A complexity class can be defined as a set of similar problems, for this project the factors of significance are classes based on pseudo boolean optimisation problems, of note is the class of NP or nondeterministic polynomial time, which is the set of problems with solutions that can be verified in polynomial time. NP is the basis for several other classes, particularly NP complete and NP hard, which contain the hardest problems in NP and the problems at least as hard as the hardest problems in NP respectively. For such problems it is often better to instead search for a solution using a metaheuristic as finding the most optimal solution may be infeasible, whilst a metaheuristic may be able to find a near optimal solution in a significantly shorter time.

Optimisation Problems

Optimisation problems involve finding the most optimal solution to a problem from a range of possible solutions. Initially I will be focusing on the travelling salesman problem as this is a well known NP hard problem that can be expressed as a pseudo boolean optimisation problem and is also a good example of the advantages of using metaheuristics to find a near optimal solution in a relatively short time. To express it as an optimisation problem, we must have a set of possible moves from the currently selected solution, this neighbourhood can be constructed in a variety of ways by defining what a neighbour solution is and how limited this definition is. For the travelling salesman problem we can move to a neighbour by changing the order in which the cities are visited, this can be done by swapping cities adjacent in the ordering or those that are further apart in the order, these definitions are inversion and transposition respectively and produce neighbourhoods of size roughly of the order of n and n squared respectively[1]. Transposition is suggested to have the best results[1] so I am using that as the basis for my implementation of tabu search, it is also suggested that evaluating the quality of solution be done using the previous solution and the modifications made moving to the current solution, this will significantly reduce the computational complexity from linear time to constant time[1], for example the quality of a travelling salesman problem solution can be found by subtracting the removed edges and adding the edges that replaced them where the edges refer to the route taken between cities and how this is changed. Finally, it is not necessary to evaluate the quality of all neighbours before making a move to the next solution, instead a limited number can be evaluated to reduce the time taken by the algorithm, this does not reduce the average quality of chosen solutions by a significant amount so proves to be very efficient and it is suggested that for this neighbourhood a few dozen would be a sufficient number of neighbours to evaluate, for a large n.

Local Search

Local Search is a simplistic metaheuristic, it involves making small 'local' changes to a found solution to try and improve it until a most optimal solution is found, by reaching a 'peak' or 'hill', where all possible neighbour solutions are less optimal than the current solution. This specific implementation of local search is known as hill climbing and is the most basic type of local search, as well as being the most limited in its success, it will often get stuck at local best solutions and stop without attempting to find an overall optimal solution if that meant first trying less optimal solutions to reach it.

Other methods of deciding the next solution or 'move strategy' exist, such as first improving and least improving[4], but the most significant adaption is simulated annealing. Simulated annealing differs in that it chooses its next move with some randomness included, whilst it will still be more likely to choose moves that improve, this means it will be able to choose worsening moves which allows it to escape local optima. The basis for simulated annealing is the metropolis algorithm which in use as the acceptance rule always accepts improving moves, otherwise they are only accepted with probability exp(-(e' - e) / T) where e refers to the energy of the solution and T refers to the current temperature[1]. As the temperature of the algorithm decreases over time, it reduces the probability that a worsening move will be accepted until at zero only improving moves are accepted, this means the algorithm initially starts as a random walk of the search space and eventually transitions to simple hill climbing by the time it finishes[2].

Tabu Search

Tabu Search betters the local search metaheuristic by attempting to fix its main flaw, getting stuck at local optima, it does this by allowing the algorithm to select less optimal neighbours if necessary to continue moving, and avoiding going back to already visited solutions if possible, this introduces the usage of memory to make the metaheuristic more 'intelligent'. More specifically, solutions previously visited are remembered as such for an arbitrary amount of time to stop the algorithm considering them as potential moves, when these solutions are removed from the tabu list it allows the algorithm to revisit them and try other neighbours that were not selected previously. Intensification and diversification rules are also used to help move the search into more promising regions and into entirely new regions respectively.

Eric Taillard suggests the importance of using some form of intensification and diversification rules[1], however I will not be implementing them in my code as a basic tabu search is sufficient to provide good results and a more complicated algorithm would be outside the scope of this project as I have other metaheuristics to cover, but a brief overview will allow me to understand the rules advantages. The search can be intensified by exploring the neighbourhood of the best solution found so far, using shorter tabu list durations this allows for better solutions to be found by covering this region more deeply than others. Diversification can be achieved by jumping to other solutions in the search space far away from the current region being explored, memory can be used to keep track of which regions have not been explored in a long time or not deeply enough, this can help prevent the algorithm getting stuck searching a specific region or cycling. Aspiration rules can also be used to allow moves to solutions on the tabu list if certain conditions are met, such as the solution being of a more optimal value, perhaps allowing this neighbourhood to be better searched, which allows the search to continue even if many solutions are on the tabu list.

Iterated Local Search

Iterated local search is another attempt at fixing the main flaw of local search, this time instead of changing the actual algorithm, the way in which it is used is changed instead, this involves starting new searches after each preceding search ends using different starting points each time, this will result in finding a sequence of the locally optimal solutions. By using knowledge learnt from previous searches better start points can be chosen, it is often the case that these locally optimal solutions are near each other, so starting the next search in the region of a local optima, but outside of its direct influence on the algorithm, will speed up the process of finding other local optima.

Genetic Algorithms

Genetic algorithms are a different type of metaheuristic to those covered previously, they instead implement rules based on natural selection processes, these rules are selection, crossover, and mutation. Starting with multiple solutions to the problem, the algorithm moves towards more optimal solutions by selecting the best of this 'generation' and 'evolving' these solutions to form a new generation from which the algorithm will repeat this iterative process.

Memetic Algorithms

Memetic algorithms extend upon the processes used in genetic algorithms by combining them with a form of local search,

Appendices and Diary

After the plan submission, I first started work on setting up the libraries I needed for the first program, I used some test code to make sure everything compiled and ran as expected.

I then moved onto writing my reports, I decided to first write a short introduction to each report to motivate my research and summarise the contents of each report.

Upon completion of the initial introductions, I read through all my sources fully to give myself a good understanding and overview of the topics before I started writing my draft reports.

I also used this time to look at some pseudocode for the tabu search metaheuristic and examples for the travelling salesman problem so that I could begin designing my implementation with pseudocode and uml, as well as noting what programming language aspects I needed to learn to implement these.

I am currently in the process of writing my report on local search after having finished my report on computational complexity, and once finished will move onto tabu search.

After receiving feedback on my draft report I have added a new section discussing optimisation problems, I also plan to further detail technical and pseudocode parts in each individual report section, as well as moving some sections to appendices.

Having researched tabu search and updated my report, I have now created pseudocode and uml in note form for the code implementation, this is inspired by algorithm 15 in source 2, and I have implemented significant changes that I discussed in its relevant theory section to improve the algorithm.

I started working on the gui using the sdl library and have tested simple gui actions, I plan to add a better way of seeing the results of using a metaheuristic on a problem, but for now will implement a logging method to write the results to a text file, I also plan to allow random generation of problems for the travelling salesman problem and allow input files that store these in text or xml format as well.

Upon finishing the gui I learnt about lambda functions and used them to implement a better way of handling actions, this replaced the enum and switch method I was using previously and has resulted in much simpler and readable code for these classes.

While implementing the tabu search method I decided it would be better to have the quality and neighbour methods in a separate optimisation problem class object, but due to time constraints I will currently be implementing this as lambdas since I have not created the uml for this class yet, but want to allow for easier refactoring of this code later.

This new optimisation problem class will simplify the code by using the visitor design pattern, rather than having the methods repeated in each metaheuristic I will have a class to the represent the problem and its constraints.

In addition to this I have already implemented the strategy design pattern with an abstract class to provide an interface for the metaheuristics which will allow the logging method to take an instance of these objects and use them.

I have refactored my code to use type aliases as the containers I was using had long declarations that made the code difficult to read, after this I was able to finish my implementation of tabu search and will now test the program ready for submission, as well as updating the user manual and compiling all necessary files to run the program.

Bibliography

[1] Metaheuristics First Edition 2016 by Patrick Siarry

This book provides a detailed overview of all the metaheuristics I will writing about in my reports and explains how they are useful, this will be helpful in gaining a basic understanding of each individual metaheuristic before I do any further research specific to the report I am writing and will most likely be the main source I am using for this project.

[2] Essentials of Metaheuristics Second Edition 2013 by Sean Luke

This book covers many of the metaheuristics I will be implementing for the project, explaining them and giving good examples, it should be useful in understanding how to design the algorithms I will be writing for the project.

[3] Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison 2001 by Christian Blum and Andrea Roli and published in the research journal ACM Computing Surveys Volume 35 Number 3 2003 pages 268 to 308

This research paper compares the metaheuristics I will be using in my project and will be useful in explaining the differences between them.

[4] Metaheuristics 2015 by Fred Glover and Kenneth Sorensen and published on `www.scholarpedia.org/article/Metaheuristics`

This web page provides a brief introduction to metaheuristics and gives an overview and comparison of them which will be useful in first understanding them before progressing my research further onto specific classifications of metaheuristics.

[5] Metaheuristic Optimization 2011 by Xin She Yang and published on `www.scholarpedia.org/article/Metaheuristic_Optimization`

This web page also provides a brief overview of each metaheuristic which will also be useful for gaining a base understanding before any research on specific algorithms.