MA Draft

Ema Skottova, M21a

September 28, 2020

Contents

Chapter 1

Introduction

In the first round of the Swiss Olympiad in Informatics there is always a creativity task, which does not have an optimal solution, but there are many approaches to reach an acceptable solution. In the fall of 2019, the task was to optimize a train network. During the first round I only submitted a very simple solution, which worked for a special case of small networks. The goal of this paper is to compare the ability of two optimization methods to solve this task. The chosen methods are simulated annealing and genetic algorithms, both of which will be explained in the theory chapter. I have first encountered the former at a programming camp and the latter was suggested by my supervisor.

Chapter 2

Theoretical Background

2.1 Optimization Problems

In an optimization problem the objective is to find the best (optimal) solution. Example problems are finding the minimum/maximum of a mathematical function, the shortest train connection between two cities, a way to maximize the profit of a company, or the optimal shape for a vehicle. Mathematically a solution can be modelled using a set of parameters, which clearly define a solution. A value function is needed to compare different solutions. In applied mathematics or outside of mathematics, the term often refers to problems where it is difficult to find the optimal solution. Therefore, instead of looking for the optimal solution,the objective becomes finding a sufficiently good one. In this project that type of problem is considered.

2.2 Optimization Methods

An optimization method is a procedure to find the optimal (or a sufficiently good) solution.

2.2.1 Brute-Force

One method which is sometimes used for optimization problems is called brute-force. When brute-force is used, all possible solutions are computed and the best one is chosen in the end. The main issue with this method is that it is often too slow as it has to compute too many similar things many times.

2.2.2 Greedy

A greedy algorithm always takes the best local option. Similar to real world problems, short term thinking often does not lead to a very good long term solution. Greedy algorithms often find a solution where no similar solution is better. The issue with this approach is that it often only looks at a small number of similar solutions rather than many different ones.

2.2.3 Properties of Better Methods

Neither brute-force nor a greedy algorithm seem to be very useful for solving problems with too many possible solutions. Brute-force would take too long and a greedy algorithm would just find some solution, without considering most options. Nevertheless, both approaches have some strenghts. The goal of a good optimization method is to compare the strenghts of both methods while avoiding their weaknesses as much as possible. In practice this means using the fact that some solutions are better than similar ones just like a greedy algorithm, without computing all the bad ones like it is done with brute-force. Similarly, a diverse range of different solutions should be considered, just like it is done with brute force.

2.2.4 Simulated Annealing

Simulated annealing is the first of the two methods chosen for this project. The general idea is that at the beginning, the program will take larger steps in testing sets of parameter values and with time its step size decreases, and it only looks at options closer to options that were already good. [1]

2.2.5 Genetic Algorithms

A genetic algorithm starts with a few 'individuals' (sets of parameter values) and the individuals multiply with 'mutations' [2] changes in parameter values) and after a few mutations only the best few survive. This process continues until a sufficiently good answer is found. [3]

Chapter 3

Complete Problem Statement

As already stated in the introduction, the goal of this paper is to compare two optimization methods. In order to do this, one first needs a problem to optimize. For this paper, the creativity task SOIway of the first round of the Swiss Olympiad in Informatics 2019/2020 was chosen. The task is as follows:

The program is given constants and a list of events, then it has to print actions.

Constants:

- minimal amount of time to change a line
- maximal number of passengers allowed on a train
- maximal number of passengers allowed at a station

Events (with their time of appearance):

- Appearance of a station (coordinates of the station and the type of the station)
- Appearance of a passenger (starting station and type of end station)
- Additional train
- Additional train line

Actions:

- Setting/Changing a line (stations that the line will cover)
- Adding a train to a line (line and the station where it starts)
- Boarding a passenger

- Unboarding a passenger

The program has failed if it produces invalid output. If the program does not fail, the process ends if (a) All passengers have arrived at their final destination or (b) There are more passengers than allowed at a station

Chapter 4

Methods

A general solution, which depends on a set of parameters, is written first. Then, the two optimization methods will be used to find the best set of parameters.

4.1 General Solution

The general solution goes through each event and action and evaluates whether it should make an action. An action will be taken if the sum of the parameters which influence it is above 100.

4.1.1 Parameters

Picking up passenger

- Distance (time, stops, line changes in current train)
- Number of passengers on the train
- Number of passengers at the station
- Network capacity
- These values for the following train(s)/passenger(s)/station(s)

Letting passenger leave

- Correct station
- Number of passengers at the station
- Number of passengers on the train
- Distance (time, stops, line changes) of current line
- Distance (time, stops, line changes) of connecting lines

- Capacity of current line
- Capacity of neighboring lines
- Values for other trains/stations/passengers

Changing line for train

- Passengers per train on each line

Changing train line (time)

- Balance in current network
- Important trains on current line

New train line (route)

- Frequency of visits
- Number of passengers starting

4.2 Evaluation

The two methods have to be able to be compared. The solutions which are generated by the two methods will be compared depending on whether the system overloads and how many time steps the solutions take to either finish or overload the system. If these two are equal, both solutions are equally good. As the main objective is to distribute all passengers without overloading the system, a solution which achieves this is strictly better than a solution which does not, no matter how many time steps they run. If this goal can be achieved by both solutions, the solution which can do this fewer time steps is better. If the system becomes overloaded in both solutions, the solution which runs longer without overloading wins. The following chart shows the evaluation in a more compact way.

Method 1		Method 2		~~Result~~
Reason for end	Steps	Reason for end	Steps	
(b)	x	(b)	$< x$	Method 1 wins
(b)	x	(b)	x	Draw
(b)	x	(a)	y	Method 2 wins
(a)	x	(a)	$> x$	Method 1 wins
(a)	x	(a)	x	Draw

Chapter 5

Process

5.1 Input Generator

At first a program which generates random test data was written. This program's parameters were set to match a test data sample from the original competition.

5.2 Input/Output

Next, the first part of the main program was created. This part is responsible for reading the input and printing the output.

5.3 General Solution V1

After this, the general program which calculates a solution will be written. This version will only consider very simple parameters, which can be calculated easily. The goal of this version is to have a running program so that tests can be done on the program. In particular, this should make it possible to test first versions of the two methods.

5.4 Simulated Annealing

This algorithm will be implemented and tested using the first version of the general solution.

5.5 General Solution V2

After first tests with simulated annealing, improvements will be added to the general solution.

5.6 Genetic Algorithm

This algorithm will be implemented and tested. First comparisons between the algorithms can be made.

5.7 Improvements General Solution

In subsequent versions, missing parameters will be added and other necessary improvements will be made.

Chapter 6

Issues

6.1 General Solution

While implementing the general solution, larger issues arose for the first time. The main problem was that it is difficult to test parts of the general solution before a primitive version of it is running. This lead to over 400 lines of code being written, where the only testing was compiling it a few times.

6.1.1 Circular References

Due to some circular references the program had to be restructured a few times. At first, solving some of the references worked by just changing the order of the definition of some classes. Later, it became necessary to make a forward declaration of all the classes. This wasn't enough either, and after a lot of googling I found out how to do a so-called inline definition, which means one can write a method outside the class. This enabled first defining the classes and their methods and only later actually writing out the methods.

6.1.2 Class hierarchy, Pointers and Priority Sorting

In order to fix an issue with calling methods of derived classes I was forced to use pointers (which I hadn't done very often before that). With this I ran into a new issue as I needed to sort the events the pointers were pointing at by time. As it is impossible to overwrite comparison operators for pointers in C++ a lambda function had to be used.

Chapter 7

Program

In this chapter, the most important methods and classes will be explained to show the essence of the program while excluding uninteresting details.

7.1 General Information

The program was written in C++. Currently its length is greater than 650 lines, including comments and blank lines.

7.2 Structure

7.2.1 Object-oriented programming

The program is object-oriented. The main idea behind object-oriented programming (OOP) is to create independent classes, each with its own properites and methods (functions). Every class is only responsible for a part of the program. This has some advantages: One of the main advantages of OOP is that similar pieces of code can be written only once and then reused easily. Another advantage is that using OOP, a program is easier to debug as it is always clear which part does what. In addition to helping maintain a large project it also simplifies collaboration, as every person only has to worry about the class they are working on and some of the interactions with other classes. This eliminates the need to know anything about the rest of the project and avoids naming conflicts (i.e. naming two different variables which do something completely different in different parts of the program x).

7.2.2 *Solution*

Solution is the class for a general solution. When initialized, it takes a set of parameters automatically computes the number of time steps and whether it succeeds in getting all passengers to their target station.

Pseudocode for calculation of general solution:

```
events = input_events
overfilled = false
while(!overfilled)
{
     e = next_event
     events.remove(next_event)
     overfilled = e.run(events)
}
```

The most important part here happens in the *run*() methond, which is different for each event. This will be described separately for each type of event:

NewObject

The appearance of an object (train, station, passenger, line) is an event. Here, *run* mainly initializes the object and adds it to necessary lists. For lines and trains, the line or train is added to the system immediately after this.

TrainArrival

When a train arrives at a station, multiple things happen. Firstly the train is registered at the station. Next, it is decided which passengers should leave the train and which passengers should board. It never makes sense for a passenger to stay on a train if it has the option to leave the train to a station of the correct type (i.e. the type of station where the passenger was headed). Therefore, this will be done automatically, no matter what the parameters for that action say. The same thing happens with passengers who should board or leave a train if that action leads to an overfilled train or station.

TrainDeparture

The most important thing that happens when a train departs is that a new train arrival and departure are scheduled, i.e. added to the list of events.

7.2.3 $Station, Passenger, Train, Line$

The clear objects in the problem of course also get their own classes. All these classes have properties such as their id and time of appearance.

7.2.4 Optimization methods

The optimization methods have been implemented as described in the theory chapter, where their pseudocode can be found.

7.3 New Line and Line Changes

Creating a new line or changing a line comes with many challenges. Firstly, one has to decide on the stations which the new line should cover. After that the order in which the stations on the line are visited has to be determined. In many cases, it makes sense to just connect the stations in an order which minimizes the total time it takes for a train to make a round on the line. This is an optimization problem of its own. The total number of possibilities for the order to visit n stations is $n!$. This can be reduced to $(n-1)!$ by chosing the first station, which does not matter asymptotically and is still too slow in many cases.

Chapter 8

Results

Bibliography

[1] Frank Liang. *Optimization Techniques — Simulated Annealing.* URL: `https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7`. (last accessed: 27.09.2020).

[2] Vijini Mallawaarachchi. *Introduction to Genetic Algorithms — Including Example Code.* URL: `https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3`. (last accessed: 23.09.2020).

[3] Niranjan Pramanik. *Genetic Algorithm — explained step by step with example.* URL: `https://towardsdatascience.com/genetic-algorithm-explained-step-by-step-65358abe2bf`. (last accessed: 27.09.2020).