
Department of Electrical and Computer Engineering
ECE457A: Cooperative and Adaptive Algorithm - Spring 2015

# Project #23: Training Routine Generation for Cyclists

by

Christopher Poenaru (20409287)
Josiah Witt (20418665)
Tommy Hu (20416327)
Rahin Jegarajaratnam (20423811)
Anthony Clark (20434297)

**Summary**

Training plans are used by cyclists in order to increase their overall fitness and prepare for upcoming events. A training plan consists of a list of training activities for the cyclist to complete and a schedule of when to complete those activities. When a cyclist completes an activity they receive a corresponding increase in fitness based on the difficulty of the activity. The increase in fitness is dependent on the cyclists fitness level. Each activity consists of a distance the cyclist will travel, the time they must travel it in, as well as the elevation they will climb.

The goal of this report is to examine a few metaheuristic algorithms that attempt to obtain near-optimal values for each activity in the training plan for an athlete of a specific fitness level. The optimality of a training plan is defined as maximizing the fitness gain for a training plan while fulfilling constraints penalizing inappropriate activities, overtraining, and activity variance. The algorithms are then compared using number of iterations, CPU time, and solution optimality.

The training plan problem was split into two sub-problems: activity generation and scheduling. In order to receive the full fitness benefits of a training plan both parts must be run since the fitness gain is dependent on when the activities are completed with respect to each other. Each of the sub-problems was implemented in all five of the metaheuristic optimization algorithms examined in this report. The algorithms are as follows: Tabu Search, Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization, and Ant Colony Optimization.

The conclusions are that Particle Swarm Optimization (PSO) offer the optimal results for both training plan generation and training plan scheduling. Ant Colony Optimization offers similar results, making it a close contendor to PSO. The other metaheuristic algorithms provide mixed results, as runtime fluctuates greatly. Leveraging PSO provides the best results overall with respect to accuracy and runtime preformance.

# Table of Contents

**List of Tables** vii

**List of Figures** viii

**1 Introduction** 1

**2 Literature Review** 4

**3 Problem Formulation and Modeling** 7

3.1 Problem Formulation . . . . . 7

3.1.1 Training Plan . . . . . 7

3.1.2 Scheduling . . . . . 9

3.2 Problem Modelling . . . . . 10

3.2.1 Reduced Size Problem . . . . . 10

3.2.2 Real Sized Problem . . . . . 10

**4 Proposed Solution for Training** 12

4.1 Tabu Search . . . . . 12

4.1.1 Notation . . . . . 12

4.1.2 Population Initialization . . . . . 12

4.1.3 Frequency Tabu List . . . . . 12

4.1.4 Recency Tabu List . . . . . 13

4.1.5 Neighbour Selection . . . 13
4.1.6 Termination . . . 13
4.1.7 Hand Iterations . . . 13
4.2 Simulated Annealing . . . 16
4.2.1 Initial Value Calibration . . . 16
4.2.2 Generating Solution . . . 16
4.2.3 Cool Down . . . 16
4.2.4 Termination Criteria . . . 17
4.2.5 Adaptivity . . . 17
4.2.6 Hand Iterations . . . 17
4.3 Genetic Algorithm . . . 19
4.3.1 Data Structure . . . 19
4.3.2 Population Initialization . . . 19
4.3.3 Crossover . . . 20
4.3.4 Mutation . . . 20
4.3.5 Survival Selection . . . 20
4.3.6 Termination . . . 20
4.3.7 Hand Iterations . . . 21
4.4 Particle Swarm Optimization . . . 25
4.4.1 Data Structure . . . 25
4.4.2 Initialization . . . 25
4.4.3 Velocity and Particle Movement . . . 25
4.4.4 Termination Criteria . . . 26
4.4.5 Hand Iteration . . . 26
4.5 Ant Colony Optimization . . . 28
4.5.1 Implementation Description . . . 28
4.5.2 Hand Iterations . . . 31

**5 Proposed Solution for Scheduling** **37**
5.1 Tabu Search . . . 37
5.1.1 Notation . . . 37
5.1.2 Population Initialization . . . 37
5.1.3 Frequency Tabu List . . . 38
5.1.4 Recency Tabu List . . . 38
5.1.5 Neighbour Selection . . . 38
5.1.6 Termination . . . 38
5.1.7 Frequency Penalization . . . 38
5.2 Simulated Annealing . . . 39
5.2.1 Initial Value Calibration . . . 39
5.2.2 Generating Solution . . . 39
5.2.3 Cool Down . . . 39
5.2.4 Termination Criteria . . . 40
5.2.5 Adaptivity . . . 40
5.3 Genetic Algorithm . . . 40
5.3.1 Data Structure . . . 40
5.3.2 Population Initialization . . . 40
5.3.3 Crossover . . . 40
5.3.4 Mutation . . . 41
5.3.5 Survival Selection . . . 41
5.3.6 Termination . . . 41
5.4 Particle Swarm Optimization . . . 41
5.4.1 Data Structure . . . 41
5.4.2 Initialization . . . 41
5.4.3 Velocity and Particle Movement . . . 42
5.4.4 Termination Criteria . . . 42
5.5 Ant Colony Optimization . . . 43
5.5.1 Implementation Description . . . 43

**6 Performance Evaluation** **46**

6.1 Simulation Setup . . . . . 46

6.2 Results . . . . . 46

6.2.1 Overview . . . . . 46

6.2.2 Training Plans . . . . . 47

6.2.3 Scheduling . . . . . 49

**7 Conclusion and Recommendations** **51**

**References** **53**

# List of Tables

4.1 First Iteration Data Tabu Search . . . . . 14
4.2 Second Iteration Data Tabu Search . . . . . 15

6.1 Training Plan Performance Data . . . . . 47
6.2 Scheduling Performance Data . . . . . 47

# List of Figures

6.1 Training Plan Algorithm Scores . . . . . 48
6.2 Training Plan Algorithm CPU runtime . . . . . 48
6.3 Scheduling Algorithm Scores . . . . . 49
6.4 Scheduling Algorithm CPU runtime . . . . . 50

# Chapter 1

# Introduction

A training plan is a way to summarize activities a cyclist will complete in order to train and increase overall fitness while preparing for events. The activities are completed by a cyclist in order to increase their fitness. The cyclists overall fitness depends on the activities they complete and when they complete them. The goal of a training plan is to maximize this overall fitness gain.

Training plans are used by all sorts of cyclists, from beginners to professionals. However, creating a personalized training plan is expensive and time consuming. Being able to automatically generate a personalized training plan based on a cyclists fitness and ride preferences would be very beneficial for a large portion of cyclists who arent in a position to pay for or build their own training plans.

Training plans consist of two pieces: a list of training activities and a schedule. Each activity has three components: a distance, a time, and an elevation. These components specify a ride and how difficult it is. A schedule contains a list of start times and durations for each activity in the training plan.

The purpose of this report is to compare five metaheuristic algorithms and their ability to find an optimal training plan for a cyclist. The five algorithms examined in this report are:

- Tabu Search (TA)
- Simulated Annealing (SA)
- Genetic Algorithm (GA)

- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)

For these metaheuristics, the training plan problem simplifies to two multi-variable function optimizations.

In order to solve the problem of generating an appropriate personalized training plan for an athlete, we started with a model for measuring athlete fitness [1]. Based on the the number of points a cyclist has, a discrete level can be determined. Using simplified physical models of activities, a cyclists level then determines the difficulty of activity they are able to complete. With this knowledge, it is possible to create a training plan of activities which are all around a cyclists difficulty level by penalizing activities for being outside the specified range.

Additionally, constraints need to be set with respect to the length of activities. Based on the cyclists preferences a variance in activity lengths will be optimal. Also, the longer an activity is, the more time it takes to recover from the activity. If the recovery time for all the activities is greater than the length of the training plan, then the plan is infeasible.

A training plans score is based on the sum of the estimated effort [3] from each activity. Each of the above specified penalties are subtracted from the training plan score resulting in the effective score, which is the objective function for creating a list of training activities.

In order to solve the very complex scheduling problem we made some basic assumptions. The first is that all activities will fit in the schedule without any overlap (if they do not a shorter training plan should be generated). The second assumption is that if an activity is smaller than a free time slot then it will be scheduled at the beginning of the slot since there is no justification for scheduling it at another point. Using these assumptions, the activities are assigned to the time slots such that they each fit in their respective slot and there is only one activity per slot. The schedule is then evaluated to maximize the cyclists overall fitness gain.

The optimal point for an activity is determined based on the theory of supercompensation [2]. To maximize the cyclists overall fitness gain, the objective function is minimized. The objective function is the difference between supercompensation and the scheduled time for each subsequent activity.

The objective functions and algorithms for each of the previous optimizations are done in MATLAB. We examine the average result of many runs of each of the presented algorithms in order to provide a better idea of the general expected behaviour of the algorithms. Using the average results we examine various performance criteria such as, average run

time, number of iterations, and the optimality of the solution. Using these performance criteria, a conclusion is reached on the best algorithm for the specified problem.

# Chapter 2

# Literature Review

To our knowledge, training plan creation of this kind using optimization and metaheuristic algorithms has not been attempted or at least published. For this reason, we believe our solution to the problem to be unique in the space. The problem is, however, related to the knapsack problem but with infinitely many potential items to place in the knapsack within the search space.

Johan Oppen et al.[7] leverages Tabu Search as a metaheuristics to solve the knapsack problem. The Tabu search in this paper differs from our implementations in that a frequency list that can be used to increase diversification is not used in our implementation. Instead we favor only using a recency list. This design decision was chosen due to the unique problem of training plan generation, and the nature of similarity in two given athlete plans.

Qian Fubin and Ding Rui [10] presents a method to solve the knapsack problem using simulated annealing. There are two difference between the algorithm presented in this paper and ours. The first is when deciding if a value is better than the current best we use a minimum increase value whereas they accept any value greater than the current. The second is this paper uses purely geometric cooling whereas we use geometric cooling and switch to linear cooling if the score is no longer improving.

Maya Hristakeva and Dipti Shrestha [6]present a genetic algorithm to solve the knapsack problem. The key difference between our implementations is that we use a real-valued genetic algorithm which also means different cross over and mutation operators. Their algorithm also uses 1-point crossover and bit-flipping mutation. Furthermore their algorithm uses roulette-wheel selection where we use stochastic universal sampling to decrease bias.

Fangguo He [5] presents a particle swarm optimization for the knapsack problem. There is a single defining difference between Fangguo's algorithm and ours, which is the use of binary particle swarm optimization compared to our use of general particle swarm optimization.

Krzysztof Schiff [13] describes an ant colony optimization to solve the discrete knapsack problem. This algorithm differs from ours in that when updating the pheromone trails, we add extra pheromone to the most elite path to increase the chance of it being explored more.

The scheduling part of our solution is analogous to an assignment problem where we are assigning activities to time slots and implicitly assigning rest to all unused time slots.

Roberto Montemanni, Jim N. J. Moon, and Derek H. Smith [11] describe their implementation of Tabu Search as a metaheuristics to solve the Fixed-Spectrum Frequency-Assignment Problem. Their encounters with manipulating the search patterns (dynamic Tabu list size, etc) provided insight into using Tabu search in unique ways. Their implementation differed from ours in that a frequency list that can be used to increase diversification, which is not used in our implementation. Instead we favor only using a recency list, for similar reasons as described in the Tabu search implementation for training plan generation.

A team from Yuan-Ze University in Taiwan [9] explored the use of simulated annealing for solving the quadratic assignment problem. The simulated annealing algorithm used in the scheduling portion of this report is based on this report's implementation. A modification made is instead of swapping assignments to generate a new solution, our algorithm randomly assign the important events (activies) and then implicitly assign rest to the remaining slots.

Sahu and Tapadar [12] propose an attempt at a solution to the Assignment problem that utilizes both genetic algorithm and simulated annealing. The genetic algorithm used for scheduling is fairly similar to the one outlined by Sahu and Tapadar. The key difference being that instead of using the roulette-wheel selection, we utilize the stochastic universal sampling in order to decrease bias to a fit solution.

Salman, Ahmad and Al-Madani [4] discuss the use of particle swarm optimization for the task assignment problem. The main difference between thier implementation and ours is that we redefined mathematical operations according permutation PSO from the lecture slides order to satisfy the contraints of our problem.

http://people.idsia.ch/ luca/tr-idsia-4-97.pdf Gambardella, Taillard and Dorigo [8] outline an ant colony optimization for the quadratic assignment problem. They call their

implementation the HAS-QAP, which is a hybrid ant colony optimization that is combined with local search. One key feature of the HAS-QAP is that the pheromone information is used to modify candidate solutions, instead of tending to a complete solution. Our implementation of ant colony uses this as an inspiration. However, when our algorithm updates the pheromone trails, we add extra pheromone to the most elite path to increase the chance of it being explored, and thus prioritizing exploration over exploitation. We also do not use local search methods.

# Chapter 3

# Problem Formulation and Modeling

## 3.1 Problem Formulation

Our goal is to generate a training plan, including a list of training activities and a schedule, for cyclists which is personalized for their fitness level and preferences and maximizes fitness gain. This is done by finding activities [distance time elevation] which maximize the estimated cyclist effort while satisfying each of the constraints. Then a schedule is created by minimizing the distance between each activity and the supercompensation point of the previous activity.

### 3.1.1 Training Plan

The inputs to the objective function are X, p, t, and m:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$\text{where } x_n = \begin{pmatrix} x_{n_{distance}} & x_{n_{time}} & x_{n_{elevation}} \end{pmatrix}$$

Such that $x_{n_{distance}}$ is the distance of the $n^{th}$ activity, $x_{n_{time}}$ is the time of the time of the $n^{th}$ activity, and $x_{n_{elevation}}$ is the elevation of the $n^{th}$ activity. p is the fitness level of

the cyclist, t is the cyclists traits such that

$$\text{t} = \begin{pmatrix} height & weight & bike_{\text{rolling resistance coefficient}} & bike_{\text{drag resistance coefficient}} \end{pmatrix}$$

and m is the cyclists preferred activity variance such that

$$\text{m} = \begin{pmatrix} fraction_{\text{short activities}} & fraction_{\text{average activities}} & fraction_{\text{long activities}} \end{pmatrix}$$

The objective function representing the effective score of a training plan can be represented as follows:

$$f(x) = \sum (w(x_k) - q(x_k, p, t)) - v(X, m) - r(X)$$

We define the score f(s) as the sum of the effort w(x) for all activities minus the penalty for inappropriate effective activities q(x,p,t) minus the penalty for activity variance v(X,m) minus the penalty for recovery time r(X).

w(x) estimates the effort of a cyclist on an activity as follows:

$$w(x) = \begin{cases} 120 + randn \times 15 & \text{for short activities} \\ 250 + randn \times 30 & \text{for average activities} \\ 2.75 \times x_{time} & \text{for long activities} \end{cases}$$

This is based on the fact for shorter activities efforts vary significantly more based on intensity. For longer rides intensity averages out to a fairly normal distribution between training zones [2] which can be simplified down to $2.75 \times x_{time}$.

q(x,p,t) penalizes an activity for being inappropriate for a cyclist. The appropriate level for an activity is determined using physics calculations of the power needed to complete the activity in the specified amount of time. A penalty of 50 points is applied for every level an activity is above p+1 or every level an activity is below p-4.

v(X,m) penalizes a training plan for having an incorrect variance in activity length. The percentage of each activity length are calculated and compared to the user preferences. For every percent greater or less than $\pm$ 10% of the user preference a penalty of 7500 points is applied.

r(X) penalizes a training plan for having a longer recovery period than the length of the training plan. Recovery is estimated as 1 day of recovery time for every 200 points of effort

in an activity and a penalty of 500 points is assessed for every day over the length of the training plan recovery accrues. The goal is to maximize f(X) subject to the constraint that 0 <n <C, where C is the maximum number of activities that will fit in a cyclists schedule. For the average serious cyclist, the appropriate value of n is 4 activities per week, so for a standard two-week training plan, n = 8.

### 3.1.2 Scheduling

The input is X:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$\text{where } x_n = \begin{pmatrix} x_{id} & x_{duration} & x_{start} \end{pmatrix}$$

Such that $x_{id}$ is the id of the activity being scheduled, $x_{duration}$ is the duration of the activity being schedule, and $x_{start}$ is the number of the slot the activity is scheduled in. The objective function is representing the cost of a schedule can be represented as follows:

$$f(x) = \sum_{i=1}^{n-1} \left| s(w(x_{i_{time}}), \frac{w(x_{i_{time}})}{200}) - s(w(x_{i_{time}}), x_{(i+1)_{time}} - x_{i_{time}} \right|$$

We define the cost of a schedule f(x) as the sum of difference between the supercompensation s(effort,time) point for the activity and when the following activity is scheduled.

s(effort,time) is defined by the piecewise supercompensation function:

$$s(eff, t) = \begin{cases} (-\frac{3000000}{eff \times eff}) \times t^3 + (\frac{45000}{eff}) \times t^2), & t < \frac{eff}{200} \\ 8 \times t + (\frac{3}{2}) \times eff - (\frac{8}{100}) \times eff, & else \end{cases}$$

The goal is to minimize f(X) subject to the constraint that every activity is scheduled in a slot that it fits in. This constraint is satisfied by not allowing activities to be scheduled in slots that they do not fit in.

## 3.2 Problem Modelling

The search space for an 8 activity training plan is very large when generating a list of training activities, it is a 24-dimensional search space, where distance, time, and elevation for each activity is a dimension. We chose to limit the search space based on the maximum activity a cyclist has performed before. For each activity the search space for distance is $[5, U_{max_{distance}} \times \frac{5}{4}]$, for time is $[20, U_{max_{distance}} \times \frac{5}{4} \times \frac{60min}{15km}]$, and for elevation is $[0, U_{max_{climb}} \times \frac{5}{4}]$. These are chosen so that we are searching for activities which we know are in the range on possibilities for a cyclist. The training plan search space is continuous and real-valued searches are used where possible.

The goal of the training plan optimization is to maximize the increase in a users fitness over the course of the training plan. This is analogous to the knapsack problem where the volume of each activity is its time and the value of each activity is its fitness utility with additional constraints. Unlike the knapsack problem there is not a finite number of items to choose from, any activity within the continuous search space is acceptable. As well, in the training plan problem the number of activities is pre-determined and the values of the activities are altered, where in the knapsack problem the items are not altered, different items are selected.

The search space for an 8 activity schedule is 8-dimensional where the start time for each activity representing a dimension. The search space is naturally limited by the length of the training plan which we have chosen to be two weeks. The search space for each start time is [1, S] where S is the the total number of free slots available in the cyclists schedule. The scheduling of an activity into slots is a basic assignment problem using supercompensation as the variable cost for each slot.

### 3.2.1 Reduced Size Problem

To reduce the size of our problem for hand iterations we will set the number of activities in a training plan to 2. We will also set the length of the training plan to 4 days. This allows for simpler operations for iterating through the various algorithms by hand.

### 3.2.2 Real Sized Problem

For the real size problem, we will use n = 8 activities in a training plan over a 14-day or 2-week period. This is a significantly bigger search space than the the reduced size problem

but is a very realistic representation of the real problem that cyclists face when creating a training plan. We use a fairly simple calendar of the average work-week schedule. Everyday before 7am is busy and everyday after 9pm is busy. As well every weekday from 9am to 5pm is busy, but everything else is free. This gives 12 free slots per week, 2 per weekday and 2 per weekend, or 24 free slots in a 14-day period.

# Chapter 4

# Proposed Solution for Training

## 4.1 Tabu Search

### 4.1.1 Notation

For notation we represent the configuration

$$x_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} x_2 = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} x_3 = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$$

as

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

### 4.1.2 Population Initialization

Initial solution can be anything so starting values are randomly generated across the entire search space.

### 4.1.3 Frequency Tabu List

It is not feasible to consider dimensionality based on the number of activities in the training plan. Instead we will only use a recency list for Tabu search

### 4.1.4 Recency Tabu List

In the Tabu implementation being evaluated both a recency and frequency list are used. A receny list prevents the algorithm from re-evaluating recent configurations. The recency list contains the 100 most recently visited training plans. Training plans are the same if two plans are an exact match.

### 4.1.5 Neighbour Selection

This Tabu implementation generates 50 neighbors. Neighboring configurations are generated from the parent by adding a different value to each field. The random numbers generated for each field are chosen to be large enough to avoid being in the recency list while still being small enough to be a neighbour. For configuration x neighbour x' is generated as follows:

$$x' = x + \begin{pmatrix} normrnd\,(0,1) & normrnd\,(0,4) & normrnd\,(0,10) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

### 4.1.6 Termination

In our Tabu implementation the algorithm terminates after current iterations reaches maximum iterations.

### 4.1.7 Hand Iterations

#### Initialization

We pick a configuration randomly to start:

$$x = \begin{bmatrix} 49 & 102 & 182 \\ 60 & 118 & 213 \end{bmatrix}$$

which has a fitness of 98

**Iteration 1**

We generate 50 neighbors of the configuration using the described method. The below table provides a selection of the generated neighbors with their fitness.

Table 4.1: First Iteration Data Tabu Search

| Distance | Time | Elevation | Fitness |
|---|---|---|---|
| 48 | 99 | 184 | |
| 60 | 117 | 215 | 108 |
| 49 | 100 | 180 | |
| 61 | 115 | 213 | 0 |
| 47 | 104 | 184 | |
| 60 | 118 | 205 | 136 |
| 50 | 98 | 175 | |
| 60 | 116 | 214 | 0 |
| 50 | 104 | 180 | |
| 60 | 118 | 216 | 125 |
| 50 | 104 | 182 | |
| 60 | 120 | 206 | 0 |
| 48 | 99 | 181 | |
| 61 | 120 | 210 | 45 |

In the first iteration the recency list is empty, therefore none of these neighbors will be eliminated by being contained on the recency list. Since we aren't using a frequency list we do not penalize any solutions according to their frequency and we pick the neighbor with the highest fitness which is:

$$x_{new} = \begin{bmatrix} 47 & 104 & 184 \\ 60 & 118 & 205 \end{bmatrix}$$

Before we update the selected value, the initial configuration:

$$x = \begin{bmatrix} 49 & 102 & 182 \\ 60 & 118 & 213 \end{bmatrix}$$

is added to the recency list which now contains:

$$\left[ \begin{bmatrix} 49 & 102 & 182 \\ 60 & 118 & 213 \end{bmatrix} \right]$$

**Iteration 2**

We start with the configuration

$$x = \begin{bmatrix} 47 & 104 & 184 \\ 60 & 118 & 205 \end{bmatrix}$$

We generate 50 neighboring solutions as in iteration 1. Below are a sampling of these solutions:

Table 4.2: Second Iteration Data Tabu Search

| Distance | Time | Elevation | Fitness |
|---|---|---|---|
| 46 | 103 | 186 | |
| 58 | 120 | 213 | 367 |
| 48 | 103 | 186 | |
| 58 | 121 | 208 | 0 |
| 47 | 102 | 186 | |
| 59 | 117 | 203 | 157 |
| 46 | 102 | 184 | |
| 58 | 119 | 204 | 291 |
| 47 | 106 | 183 | |
| 61 | 119 | 209 | 41 |
| 47 | 102 | 180 | |
| 60 | 119 | 204 | 188 |
| 47 | 103 | 181 | |
| 60 | 117 | 205 | 191 |

We remove any solutions from the table that are an exact match to any solutions in the recency list. The recency list contains:

$$\left[ \begin{bmatrix} 49 & 102 & 182 \\ 60 & 118 & 213 \end{bmatrix} \right]$$

Which matches none of the neighbors generated therefore all solutions are considered feasible. The neighbor with the highest fitness is:

$$x_{new} = \begin{bmatrix} 46 & 102 & 184 \\ 58 & 119 & 204 \end{bmatrix}$$

We add

$$\begin{bmatrix} 47 & 104 & 184 \\ 60 & 118 & 205 \end{bmatrix}$$

to the recency list after this iteration.

## 4.2 Simulated Annealing

### 4.2.1 Initial Value Calibration

$$T_0 \simeq -\frac{max\left(\Delta\right)}{ln\left(p_0\right)}$$

To calibrate the initial temperature a heuristic approach was used. Evaluations started at a high temperature and the temperature was rapidly decreased until 50% to 60% of the unacceptable solutions were accepted and that temperature was selected as the initial temperate. In our calibration we found this temperature to consistently be the value of 10.

### 4.2.2 Generating Solution

Solutions are generated by generating normally distributed random numbers and then added to each field in the activity. The standard deviation used for distance, time and elevation 1, 4 and 10 respectively with a mean of 0.

$$x' = x + \begin{pmatrix} normrnd\left(0,1\right) & normrnd\left(0,4\right) & normrnd\left(0,10\right) \\ . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$$

### 4.2.3 Cool Down

The accept threshold determines the cool down schedule. The temperature is lowered by a previously initialized alpha whenever accepted solutions exceed the threshold. The temperature is also used to determine the probability that a worse solution will be accepted.

$$T' = a \times T$$

alpha = 0.95

### 4.2.4 Termination Criteria

The 4 termination conditions for the algorithms are as follows: The first condition occurs when the number of rejected solutions exceed the max permitted number of rejected solutions. The second occurs when the current greatest fitness exceed the acceptable fitness threshold. The third occurs when the number of iterations is greater than the maximum allotted number of iterations. The fourth and final termination condition occurs if the current temperatures becomes lower than the previously initialized final temperature.

### 4.2.5 Adaptivity

If the best fitness isnt improved by more than 0.1% in the last 100 generations over the 100 generations previous use linear cooling, otherwise use geometric cooling.

### 4.2.6 Hand Iterations

**Initialization**

$$Temperature_{initial} = 10.0$$

$$Temperature_{min} = 1e - 10$$

$$obj_{max} = 3000$$

$$Max_{rejections} = 2500$$

$$Max_{runs} = 500$$

$$Max_{accept} = 15$$

$$k = 1$$

$$alpha = 0.95$$

$$\text{Energy Norm} = 1$$

$$\text{Number Of Activities} = 2$$

An initial guess is made using our heuristic initialization:

$$Guess = \begin{pmatrix} 1 & 99 & 178 \\ 1 & 117 & 217 \end{pmatrix}$$

Adjusting distance to meet level

$$Guess = \begin{pmatrix} 49.5 & 99 & 178 \\ 58.5 & 117 & 217 \end{pmatrix}$$

Fitness = 54.8836

**Iteration 1**

A new solution is generated by generating normally distributed random numbers and adding them to the current solution

$$Random = \begin{pmatrix} normrnd(0,1) & normrnd(0,4) & normrnd(0,10) \\ normrnd(0,1) & normrnd(0,4) & normrnd(0,10) \end{pmatrix}$$

$$= \begin{pmatrix} -0.1241 & 2.9794 & 4.4558 \\ 1.4172 & 1.3430 & -3.8184 \end{pmatrix}$$

$$Solution + Random = \begin{pmatrix} 49.3759 & 101.9794 & 182.4558 \\ 59.9172 & 118.3430 & 213.1816 \end{pmatrix}$$

New fitness = 134.7427

The new fitness is greater than the current fitness the solution is accepted and the accepted count is incremented

$$\Delta_{fitness} = 134.7427 - 54.8836 = 79.8591$$
$$\text{Accepted Count } = \text{Accepted Count} + 1$$

**Iteration 2**

A new solution is generated for this iteration

$$Random = \begin{pmatrix} normrnd(0,1) & normrnd(0,4) & normrnd(0,10) \\ normrnd(0,1) & normrnd(0,4) & normrnd(0,10) \end{pmatrix} = \begin{pmatrix} -0.1977 & -2.4157 & 9.1959 \\ 0.8252 & 2.7579 & -3.3463 \end{pmatrix}$$

$$Solution + Random = \begin{pmatrix} 49.1782 & 99.5637 & 191.6517 \\ 60.7424 & 121.1009 & 209.8353 \end{pmatrix}$$

New fitness = 0

The new fitness is lower than the previous, so we use the accept probability to determine if the solution is accepted or not

$$\Delta_{fitness} = 0 - 134.7427 = -134.7427$$

$$Probability = e^{\Delta/T} = e^{-134.7427/10} = 1.4089 \times 10^{-6}$$

$$\text{Random Number} = rand() = 0.3786$$

RandomNumber is not less than Probability the solution is not accepted

## 4.3 Genetic Algorithm

### 4.3.1 Data Structure

$$Individual = \big(\big[Distance \quad Time \quad Elevation\big] \quad fitness\big)$$

### 4.3.2 Population Initialization

The population is set to have 20 individuals. The population is maintained at 20 individuals during the algorithm run time. Each individual will carry its genes and calculated fitness. During initialization each individuals will first have its time and elevation genes randomized based on what kind of activity it is. The distance gene is then raised until the activity meets target athlete fitness levels. The fitness of each individual is then calculated and saved.

### 4.3.3 Crossover

Whole Arithmetic Crossover is selected for use. This algorithm is specified for Real-Valued Genetic Algorithms. In Whole Arithmetic Crossover, every child gene will be generated as a weighted average of the two parent genes with alpha set to be the weight. For example assuming GeneA belongs to ParentA and GeneB belongs to ParentB the child genes will be:

$$Child_A = \begin{pmatrix} \alpha * Gene1_A + (1-\alpha) * Gene1_B \\ \alpha * Gene2_A + (1-\alpha) * Gene2_B \\ \alpha * Gene3_A + (1-\alpha) * Gene3_B \end{pmatrix}$$

### 4.3.4 Mutation

The mutation algorithm used is to add Gaussian noise to each individual. For each gene a Gaussian noise is generated and added to that gene. Gaussian noise is different depending on if the gene is distance, time or elevation. Gaussian noise is added as follows using sigma=1

$$\text{Gaussian Noise} = \begin{bmatrix} randn * sigma & randn * 2 * sigma & randn * 4 * sigma \\ randn * sigma & randn * 2 * sigma & randn * 4 * sigma \end{bmatrix}$$

### 4.3.5 Survival Selection

Stocastic universal sampling is used to select survivors. Stochastic universal sampling builds on Roulette wheel selection where each slice in the wheel is proportional to the normalized selection probability of each individual. In Stochastic Universal Sampling an outer roulette wheel is also place around the pie with N equally spaced pointers, resulting in a single spin simultaneously selecting all the N individuals for reproduction.

### 4.3.6 Termination

There are two termination criteria for the algorithm. The first termination criteria is if the best hasn't improved by more than 0.1% in the last 100 generations previous to the current. The second criteria for termination is if the current iterations exceed the maximum alloted iterations.

### 4.3.7 Hand Iterations

#### Notation

For hand iteration notation to represent individuals is:

$$\text{Initial Population} = \left( \begin{bmatrix} Distance & Time & Elevation \\ Distance & Time & Elevation \end{bmatrix} \quad fitness \right) \left( \begin{bmatrix} Distance & Time & Elevation \\ Distance & Time & Elevation \end{bmatrix} \quad fitness \right)$$

#### Population Initialization

Initial population is initialized as two average length activities with mean time of 90 and a standard deviation of 10 and mean elevation of 200 and a standard deviation of 50.

$$\text{Initial Population} = \left( \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} \quad 319.8663 \right) \left( \begin{bmatrix} 46.0000 & 93.0706 & 206.7587 \\ 47.0000 & 95.1525 & 213.0703 \end{bmatrix} \quad 129.7279 \right)$$

#### Iteration 1

First select pairs of parents from the population

$$Parent_A = \left( \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} \quad 319.8663 \right)$$

$$Parent_B = \left( \begin{bmatrix} 46.0000 & 93.0706 & 206.7587 \\ 47.0000 & 95.1525 & 213.0703 \end{bmatrix} \quad 129.7279 \right)$$

Whole Arithmetic Crossover is applied to generate two children using a randomly selected alpha.

$$alpha = 0.4$$

$$Child_A = \begin{bmatrix} 0.4 \times (47) + 0.6 \times (46) & 0.4 \times (96.2) + 0.6 \times (93.0) & 0.4 \times (209.1) + 0.6 \times (206.7) \\ 0.4 \times (39) + 0.6 \times (47) & 0.4 \times (79.7) + 0.6 \times (95.1) & 0.4 \times (247.4) + 0.6 \times (213.0) \end{bmatrix}$$

$$Child_A = \begin{bmatrix} 46.4 & 94.3431 & 207.72 \\ 43.8 & 88.9724 & 226.827 \end{bmatrix}$$

$$Child_B = \begin{bmatrix} 0.6 \times (47) + 0.4 \times (46) & 0.6 \times (96.2) + 0.4 \times (93.0) & 0.6 \times (209.1) + 0.4 \times (206.7) \\ 0.6 \times (39) + 0.4 \times (47) & 0.6 \times (79.7) + 0.4 \times (95.1) & 0.6 \times (247.4) + 0.4 \times (213.0) \end{bmatrix}$$

$$Child_B = \begin{bmatrix} 46.6 & 94.9794 & 208.2 \\ 42.2 & 85.8824 & 233.705 \end{bmatrix}$$

Add a Gaussian Noise to each gene sigma=1

$$\text{Gaussian Noise} = \begin{bmatrix} randn * sigma & randn * 2 * sigma & randn * 4 * sigma \\ randn * sigma & randn * 2 * sigma & randn * 4 * sigma \end{bmatrix}$$

$$Child_A = Child_A + \text{Gaussian Noise} = \begin{bmatrix} 46.1259 & 97.4032 & 206.7239 \\ 42.7358 & 92.1793 & 231.8087 \end{bmatrix}$$

$$Child_B = Child_B + \text{Gaussian Noise} = \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix}$$

These children's fitness is then calculated and added to the current population pool

$$\text{Current Population} = \left( \begin{array}{cc} \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} & 319.8663 \\ \begin{bmatrix} 46.0000 & 93.0706 & 206.7587 \\ 47.0000 & 95.1525 & 213.0703 \end{bmatrix} & 129.7279 \\ \begin{bmatrix} 46.1259 & 97.4032 & 206.7239 \\ 42.7358 & 92.1793 & 231.8087 \end{bmatrix} & 447.9264 \\ \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} & 500.4471 \end{array} \right)$$

For survival selection Stochastic Universal Sampling is used. A roulette is rolled and with 2 equally spaced pointers on the outer wheel.

To emulate a roulette roll here a random number is generated between 0 and the total fitness of all individuals. The first survivor is chosen based on where that random number lands and the second survivor is directly opposite the first on the roulette

$$Survivor_1 = rand() * 1300] = 288$$

$$Survivor_1 = \left( \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} \quad 319.8663 \right)$$

$$Survivor_2 = 288 + 650 = 938$$

$$Survivor_2 = \left( \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

The new population is now:

$$NewPopulation = \left( \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} \quad 319.8663 \right.$$
$$\left. \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

**Iteration 2**

$$Population = \left( \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} \quad 319.8663 \right.$$
$$\left. \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

First select pairs of parents from the population

$$Parent_A = \left( \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

$$Parent_B = \left( \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

Whole Arithmetic Crossover is applied to generate two children using a randomly selected alpha.

$$alpha = 0.75$$

$$Child_A = \begin{bmatrix} ccc0.75\times(46.3)+0.25\times(46.3) & 0.75\times(98.0)+0.25\times(98.0) & 0.75\times(207.2)+0.25\times(207.2) \\ 0.75\times(41.1)+0.25\times(41.1) & 0.75\times(89.0)+0.25\times(89.0) & 0.75\times(238.6)+0.25\times(238.6) \end{bmatrix}$$

$$Child_A = \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix}$$

$$Child_B = \begin{bmatrix} 0.25\times(46.3)+0.75\times(46.3) & 0.25\times(98.0)+0.75\times(98.0) & 0.25\times(207.2)+0.75\times(207.2) \\ 0.25\times(41.13)+0.75\times(41.1) & 0.25\times(89.0)+0.75\times(89.0) & 0.25\times(238.6)+0.75\times(238.6) \end{bmatrix}$$

$$Child_B = \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix}$$

Add a Gaussian Noise to each gene sigma=1

$$GaussianNoise = \begin{bmatrix} randn * sigma & randn * 2 * sigma & randn * 4 * sigma \\ randn * sigma & randn * 2 * sigma & randn * 4 * sigma \end{bmatrix}$$

$$Child_A = Child_A + GaussianNoise = \begin{bmatrix} 46.0519 & 101.0997 & 206.2078 \\ 40.0716 & 92.2962 & 243.5824 \end{bmatrix}$$

$$Child_B = Child_B + GaussianNoise = \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix}$$

These children's fitness is then calculated and added to the current population pool

$$CurrentPopulation = \left( \begin{array}{cc} \begin{bmatrix} 47 & 96.2519 & 209.1614 \\ 39.0000 & 79.7023 & 247.4611 \end{bmatrix} & 319.8663 \\ \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} & 500.4471 \\ \begin{bmatrix} 46.0519 & 101.0997 & 206.2078 \\ 40.0716 & 92.2962 & 243.5824 \end{bmatrix} & 349.6562 \\ \begin{bmatrix} 45.3780 & 96.5573 & 205.1726 \\ 40.8152 & 89.1143 & 226.5270 \end{bmatrix} & 551.0688 \end{array} \right)$$

Using same survival selection process as in iteration 1

$$Survivor_1 = rand() * 1719] = 769$$

$$Survivor_1 = \left( \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right)$$

$$Survivor_2 = 769 + 860 = 1628$$

$$Survivor_2 = \left( \begin{bmatrix} 45.3780 & 96.5573 & 205.1726 \\ 40.8152 & 89.1143 & 226.5270 \end{bmatrix} \quad 551.0688 \right)$$

The new population is now:

$$NewPopulation = \left( \begin{bmatrix} 46.3259 & 98.0395 & 207.2039 \\ 41.1358 & 89.0893 & 238.6437 \end{bmatrix} \quad 500.4471 \right.$$
$$\left. \begin{bmatrix} 45.3780 & 96.5573 & 205.1726 \\ 40.8152 & 89.1143 & 226.5270 \end{bmatrix} \quad 551.0688 \right)$$

## 4.4 Particle Swarm Optimization

### 4.4.1 Data Structure

Each particle in the system contains tuples representing its own position in the search space and velocity. Each particle also contains its best found fitness and best found position.

$$Particle_1 = [\,[\,^{distance}_{\cdot}\ ^{time}_{\cdot}\ ^{elevation}_{\cdot}\,][\,^{velocity_{distance}}_{\cdot}\ ^{velocity_{time}}_{\cdot}\ ^{velocity_{elevation}}_{\cdot}\,][\,^{best_{distance}}_{\cdot}\ ^{best_{time}}_{\cdot}\ ^{best_{elevation}}_{\cdot}\,]\ ^{fitness}_{\cdot}\,]$$

### 4.4.2 Initialization

The inertia weight w and acceleration coefficient c1 and c2 need to be initializaed for the algorithm to run.

$$w = 0.792$$

$$c_1, c2 = 1.4944$$

### 4.4.3 Velocity and Particle Movement

Before generating a new solution and moving the particle, the algorithm calculates the velocities of the particles. This velocity calculation depends on the best global position, best particle position, current particle position, various constants and uniformly generated random numbers. The following equations are used:

$$r1 = rand\,(popsize, npar)\,; r1 = rand\,(popsize, npar)\,;$$

Update Velocity

$$vel = (w * vel + c1 * r1\,(localbestpar - par) + c2 * r2 * (globalbestpar - par))\,;$$

Update particle positions

$$par = par + vel;$$

### 4.4.4 Termination Criteria

The algorithm is terminated when best fitness isn't improved by more than 0.1% in the last 100 generations over the 100 previous generations. The second termination criteria is when max iterations is exceeded by current iterations.

### 4.4.5 Hand Iteration

#### Initialization

$$c_1 = c_2 = 1.4944$$

$$w = 0.792$$

$$Particle_1 = [\,[\begin{smallmatrix}46.32 & 98.03 & 207.20\\ 41.13 & 89.08 & 238.64\end{smallmatrix}][\begin{smallmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{smallmatrix}][\begin{smallmatrix}46.32 & 98.03 & 207.20\\ 41.13 & 89.08 & 238.64\end{smallmatrix}]\ 500.44\,]$$

$$Particle_2 = [\,[\begin{smallmatrix}45.37 & 96.55 & 205.17\\ 40.81 & 89.11 & 226.52\end{smallmatrix}][\begin{smallmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{smallmatrix}][\begin{smallmatrix}45.37 & 96.55 & 205.17\\ 40.81 & 89.11 & 226.52\end{smallmatrix}]\ 551.06\,]$$

#### Iteration 1

Update all best positions from the last iteration. In the first iteration each particle's best position and highest fitness should remain the same

$$Particle_1 = [\,[\begin{smallmatrix}46.32 & 98.03 & 207.20\\ 41.13 & 89.08 & 238.64\end{smallmatrix}][\begin{smallmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{smallmatrix}][\begin{smallmatrix}46.32 & 98.03 & 207.20\\ 41.13 & 89.08 & 238.64\end{smallmatrix}]\ 500.4471\,]$$

$$Particle_2 = [\,[\begin{smallmatrix}45.37 & 96.55 & 205.17\\ 40.81 & 89.11 & 226.52\end{smallmatrix}][\begin{smallmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{smallmatrix}][\begin{smallmatrix}45.37 & 96.55 & 205.17\\ 40.81 & 89.11 & 226.52\end{smallmatrix}]\ 551.0688\,]$$

The best fitness and position of all the particles are saved into a global best fitness and best position.

$$GlobalBest = \begin{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} & 551.0688 \end{bmatrix}$$

New velocities are calculated for each particle and saved. Velocities are calculated using the previously described function

$$Velocity_1 = \begin{bmatrix} 0.792\times0.0+1.4944\times0.49\times(45.37-46.32)+1.4944\times0.62\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.68\times(96.55-98.03)+1.4944\times0.40\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.38\times(205.17-207.20)+1.4944\times0.99\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.04\times(40.81-41.13)+1.4944\times0.89\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.91\times(89.11-89.08)+1.4944\times0.80\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.10\times(226.52-238.64)+1.4944\times0.26\times(46.32-46.32) \end{bmatrix}$$

$$= \begin{bmatrix} -0.696 & -1.50 & -1.15 \\ -0.02 & 0.04 & -1.81 \end{bmatrix}$$

$$Velocity_2 = \begin{bmatrix} 0.792\times0.0+1.4944\times0.51\times(45.37-45.37)+1.4944\times0.23\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.45\times(96.55-96.55)+1.4944\times0.32\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.86\times(205.17-205.17)+1.4944\times0.15\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.39\times(40.81-40.81)+1.4944\times0.03\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.76\times(89.11-89.11)+1.4944\times0.43\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.16\times(226.52-226.52)+1.4944\times0.22\times(46.32-46.32) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Iteration 2**

$$Particle_1 = \begin{bmatrix} \begin{bmatrix} 45.62 & 96.53 & 206.05 \\ 41.11 & 89.12 & 236.83 \end{bmatrix} \begin{bmatrix} -0.696 & -1.50 & -1.15 \\ -0.02 & 0.04 & -1.81 \end{bmatrix} \begin{bmatrix} 46.32 & 98.03 & 207.20 \\ 41.13 & 89.08 & 238.64 \end{bmatrix} & 504.3395 \end{bmatrix}$$

$$Particle_2 = \begin{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} & 551.0688 \end{bmatrix}$$

$$GlobalBest = \begin{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} & 551.0688 \end{bmatrix}$$

$$Velocity_1 = \begin{bmatrix} 0.792\times0.0+1.4944\times0.49\times(45.37-45.62)+1.4944\times0.62\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.68\times(96.55-96.53)+1.4944\times0.40\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.38\times(205.17-206.05)+1.4944\times0.99\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.04\times(40.81-41.11)+1.4944\times0.89\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.91\times(89.11-89.12)+1.4944\times0.80\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.10\times(226.52-236.83)+1.4944\times0.26\times(46.32-46.32) \end{bmatrix}$$

$$= \begin{bmatrix} -0.18 & 0.02 & -0.50 \\ -0.2 & -0.01 & -1.54 \end{bmatrix}$$

$$Velocity_2 = \begin{bmatrix} 0.792\times0.0+1.4944\times0.51\times(45.37-45.37)+1.4944\times0.23\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.45\times(96.55-96.55)+1.4944\times0.32\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.86\times(205.17-205.17)+1.4944\times0.15\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.39\times(40.81-40.81)+1.4944\times0.03\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.76\times(89.11-89.11)+1.4944\times0.43\times(46.32-46.32) \\ 0.792\times0.0+1.4944\times0.16\times(226.52-226.52)+1.4944\times0.22\times(46.32-46.32) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Particle Values at end of Iteration 2

$$Particles_1 = \begin{bmatrix} \begin{bmatrix} 45.62 & 96.53 & 206.05 \\ 41.11 & 89.12 & 236.83 \end{bmatrix} \begin{bmatrix} -0.18 & 0.02 & -0.50 \\ -0.2 & -0.01 & -1.54 \end{bmatrix} \begin{bmatrix} 46.32 & 98.03 & 207.20 \\ 41.13 & 89.08 & 238.64 \end{bmatrix} & 504.3395 \end{bmatrix}$$

$$Particles_2 = \begin{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 45.37 & 96.55 & 205.17 \\ 40.81 & 89.11 & 226.52 \end{bmatrix} & 551.0688 \end{bmatrix}$$

## 4.5 Ant Colony Optimization

### 4.5.1 Implementation Description

#### Data Structure

The three main data structures that need to be represented are the pheromone graph, the ant structure, and distance matrix. The pheromone graph is an array that represents which path it takes from node to node. Each intersection in this matrix represents a path. The ants are represented as the path that they walk to reach the goal.

$$Ant = \begin{bmatrix} 3 & 9 & 4 \end{bmatrix}$$

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

**Distance Matrix**

The distance matrix is used to find the fitness of ants. To calculate the distance matrix, first take an initial feasible training plan for the cyclist and calculate its score. Then for each path replace its respective value in the training plan and recalculate the score. The distance for the path is based on the difference between the initial benchmark score and the path score. For each set of paths between two nodes we take the minimum value and subtract it from all the values to normalize the scores to a minimum of 0. Then we calculate the distance as 1/(1+x) where x is the normalized score.

**Move Ants**

Each ant has its path to the goal determined by the pheromone levels at each path. When an ant needs to determine which path to take to reach the next node, the ant will calculate the probability for each path. The probability will depend on the pheromone level in the graph.

$$Probability\,(i) = \frac{Pheromone\,(NextNode)^a \times \frac{1}{Distance(i)}^b}{TotalPheromone\,(NextNode)^a \times \frac{1}{TotalDistance}^b}$$

where:
a = 0.5
b = 0.5
Distance(i) = distance of the path i

TotalDistance = sum of all distances i = a possible path
NextNode = the node to traverse
Pheromone(NextNode, i) = pheromone level in the graph at index (NextNode, i)
TotalPheromone(NextNode) = Sum of pheromones in the graph corresponding to column NextNode

The ant generates a uniformly random number and picks on of the paths based on their probability once the probabilities are determined. This path is saved and traversed and continued until the ant reaches the goal. The fitness is then calculated.

**Evaporate**

The pheromone levels in the pheromone graph decreaases by a factor called the evaporation rate. This evaporation rate is multiplied to the pheomone table over time.

$$EvaporationRate = 0.5$$

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \cdot EvaporationRate = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

**Pheromone Deposit**

Ants deposit more pheromone onto the specific path that they took once they have a valid path. The pheromone is deposited onto the pheromone graph. The amount of pheromone deposited is determined by the following function:

$$AddPheromone = \frac{LowestDistance}{CurrentAntDistance}$$

LowestDistance = the distance of a path that an ant found
CurrentAntDistance = the distance of the current ant This function allows us to deposit more pheromone to the path that has the lowest distance. This steers the ants towards the best path which the ants should eventually converge to.

### 4.5.2 Hand Iterations

#### Initialize Graph and Ants

The search space has a lower and upper bound of 0.1 to 1.0. We have 3 variables to optimize and we discretize the search space by a precision of 0.1. The algorithm is simplified to only have 2 ants. The two ants are created and initialized to empty array paths. The evaporation rate is set to 0.5.

$$EvaporationRate = 0.5$$
$$Ant(1) = []$$
$$Ant(2) = []$$

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

#### Iteration 1

The ants each attempt to find a path that would reach the goal. When moving from one node to another, the ants must first calcualte the pheremone probability. The ants then select a path given the probabilities. The ants select a path, and the fitness is then calculated.

Pheromone Probabilties after applying the previously described method:

$$Prob_1(1) = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} \quad Prob_1(2) = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} \quad Prob_1(3) = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

$$Path_1^1 = random() = 0.78$$
$$Path_2^1 = random() = 0.47$$
$$Path_3^1 = random() = 0.03$$
$$Ant_1 = \begin{bmatrix} 8 & 5 & 1 \end{bmatrix}$$
$$Distance_1 = 0.056$$

$$Path_1^2 = random() = 0.18$$
$$Path_2^2 = random() = 0.72$$
$$Path_3^2 = random() = 0.47$$
$$Ant_2 = \begin{bmatrix} 2 & 8 & 5 \end{bmatrix}$$
$$Distance_2 = 0.042$$

The evaporation is then applied to the pheremone graph by multiplying the evaporation rate to all entries in the graph.

Evaporation rate = 0.5

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \cdot EvaporationRate = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

The ants then deposit pheremones to the paths they each select after determining how much pheremone they are going to deposit in the proper row.

$$Ant_1 = \frac{LowestDistance}{Distance_1} = \frac{0.042}{0.056} = 0.75$$

$$Ant_2 = \frac{LowestDistance}{Distance_2} = \frac{0.042}{0.042} = 1$$

$$AddPheremone = \begin{bmatrix} 0 & 0 & 0.75 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.75 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.75 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$PheremoneMatrix = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0.75 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.75 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.75 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 1.25 \\ 1.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 1.25 & 1.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 1.25 & 1.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

**Iteration 2**

Ants Find a path: Pheremone Probabilities:

$$Prob_1(1) = \begin{bmatrix} 0.074 \\ 0.222 \\ 0.074 \\ 0.074 \\ 0.074 \\ 0.074 \\ 0.074 \\ 0.185 \\ 0.074 \\ 0.074 \end{bmatrix} \; Prob_1(2) = \begin{bmatrix} 0.083 \\ 0.083 \\ 0.083 \\ 0.083 \\ 0.083 \\ 0.208 \\ 0.083 \\ 0.240 \\ 0.083 \\ 0.083 \end{bmatrix} \; Prob_1(3) = \begin{bmatrix} 0.208 \\ 0.05 \\ 0.078 \\ 0.078 \\ 0.145 \\ 0.078 \\ 0.078 \\ 0.078 \\ 0.078 \\ 0.078 \end{bmatrix}$$

$$Path_1^1 = random() = 0.76$$
$$Path_2^1 = random() = 0.30$$
$$Path_3^1 = random() = 0.13$$
$$Ant_1 = \begin{bmatrix} 8 & 4 & 2 \end{bmatrix}$$
$$Distance_1 = 0.037$$

$$Path_1^2 = random() = 0.99$$

$$Path_2^2 = random() = 0.30$$
$$Path_3^2 = random() = 0.03$$
$$Ant_2 = \begin{bmatrix} 10 & 4 & 1 \end{bmatrix}$$
$$Distance_2 = \inf$$

$$PheremoneMatrix = \begin{bmatrix} 0.5 & 0.5 & 1.25 \\ 1.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 1.25 & 1.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 1.25 & 1.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \cdot EvaporationRate = \begin{bmatrix} 0.25 & 0.25 & 0.625 \\ 0.75 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.625 & 0.75 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 0.625 & 0.75 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$$

Deposit Pheremones:

$$Ant_1 = \frac{LowestError}{Distance_1} = \frac{0.037}{0.037} = 1$$

$$Ant_2 = \frac{LowestError}{Distance_2} = \frac{0.037}{\inf} = 0.0$$

$$AddPheremone = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$PheremoneMatrix = \begin{bmatrix} 0.5 & 0.5 & 1.25 \\ 1.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 1.25 & 1.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 1.25 & 1.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.25 & 0.625 \\ 0.75 & 0.25 & 1.25 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 1.25 & 0.25 \\ 0.25 & 0.625 & 0.75 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 1.625 & 0.75 & 0.25 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$$

# Chapter 5

# Proposed Solution for Scheduling

This problem was added in addition to training plan problem to satisfy the constraint of the training plan fitting into the cyclist's schedule. The two hand iterations were completed in the previous chapter for the training plan problem, and thus will not be conducted in this chapter.

## 5.1 Tabu Search

### 5.1.1 Notation

For notation we represent the configuration

$$x_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} x_2 = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} x_3 = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$$

as

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

### 5.1.2 Population Initialization

Initial solution can be anything so starting values are randomly generated across the entire search space.

### 5.1.3 Frequency Tabu List

It is not feasible to consider dimensionality based on the number of activities in the training plan. Instead we will only use a recency list for Tabu search

### 5.1.4 Recency Tabu List

In the Tabu implementation being evaluated, a recency list is used. A receny list prevents the algorithm from re-evaluating recent configurations. The recency list contains the 100 most recently visited training schedules. Training schedules are the same if two schedules if the root-mean-square distance between two training schedules is less than a determined threshold

### 5.1.5 Neighbour Selection

This Tabu implementation generates 50 neighbors. Neighboring configurations are generated from the parent by adding a different value to each field. The random numbers generated for each field are chosen to be large enough to avoid being in the recency list while still being small enough to be a neighbour. For configuration x neighbour x' is generated as follows:

$$x' = x + \begin{pmatrix} 0 & 0 & normrnd\,(0, 10) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

### 5.1.6 Termination

In our Tabu implementation, the algorithm terminates after current iterations reaches maximum iterations.

### 5.1.7 Frequency Penalization

Due to the addition of penalization, the formula for Effective fitness is:

$$\text{Effective Fitness} = Fitness - \sqrt{freq}$$

## 5.2 Simulated Annealing

### 5.2.1 Initial Value Calibration

$$T_0 \simeq -\frac{max\,(\Delta)}{ln\,(p_0)}$$

To calibrate the initial temperature a heuristic approach was used. Evaluations started at a high temperature and the temperature was rapidly decreased until 50% to 60% of the unacceptable solutions were accepted and that temperature was selected as the initial temperate. In our calibration we found this temperature to consistently be the value of 1.

### 5.2.2 Generating Solution

Solutions are generated by generating normally distributed random numbers and then added to each field in the activity. The normal variable has a mean of 0 and a standard deviation of 10.

$$x' = x + \begin{pmatrix} 0 & 0 & normrnd\,(0,10) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

### 5.2.3 Cool Down

The accept threshold determines the cool down schedule. The temperature is lowered by a previously initialized alpha whenever accepted solutions exceed the threshold. The temperature is also used to determine the probability that a worse solution will be accepted.

$$T' = a \times T$$

alpha = 0.95

### 5.2.4 Termination Criteria

The 4 termination conditions for the algorithms are as follows: The first condition occurs when the number of rejected solutions exceed the max permitted number of rejected solutions. The second occurs when the current greatest fitness exceed the acceptable fitness threshold. The third occurs when the number of iterations is greater than the maximum allotted number of iterations. The fourth and final termination condition occurs if the current temperatures becomes lower than the previously initialized final temperature.

### 5.2.5 Adaptivity

If the best fitness isnt improved by more than 0.1% in the last 100 generations over the 100 generations previous use linear cooling, otherwise use geometric cooling.

## 5.3 Genetic Algorithm

### 5.3.1 Data Structure

$$Individual = \big(\big[\text{Activity ID} \quad Duration \quad \text{Start time}\big] \quad fitness\big)$$

### 5.3.2 Population Initialization

The population is set to have 20 individuals. The population is maintained at 20 individuals during the algorithm run time. Each individual will carry its genes and calculated fitness.

### 5.3.3 Crossover

Uniform crossover of Start Time bits is used for the crossover phase. In Uniform crossover, two parent solutions are taken and for each gene, based on a Uniform random variable, a crossover may or may not occur.

### 5.3.4 Mutation

Bit-flipping mutation is used for the mutation phase. Specifically, for each solution, for each gene, a random number is generated. If the random number is greater than 0.5, the bit is flipped.

### 5.3.5 Survival Selection

Stocastic universal sampling is used to select survivors. Stochastic universal sampling builds on Roulette wheel selection where each slice in the wheel is proportional to the normalized selection probability of each individual. In Stochastic Universal Sampling an outer roulette wheel is also place around the pie with N equally spaced pointers, resulting in a single spin simultaneously selecting all the N individuals for reproduction.

### 5.3.6 Termination

There are two termination criteria for the algorithm. The first termination criteria is if the best hasn't improved by more than 1% in the last 10 generations previous to the current. The second criteria for termination is if the current iterations exceed the maximum allotted iterations.

## 5.4 Particle Swarm Optimization

### 5.4.1 Data Structure

Each particle in the system contains tuples representing its own position in the search space and velocity. Each particle also contains its best found fitness and best found position.

$$Particle_1 = [\,[\overset{activity_{id}}{\cdot}\ \overset{duration}{\cdot}\ \overset{starttime}{\cdot}][\overset{starttimevelocity}{\cdot}][\overset{beststarttime}{\cdot}]\ \overset{fitness}{\cdot}\,]$$

### 5.4.2 Initialization

The inertia weight w and acceleration coefficient c1 and c2 need to be initialized for the algorithm to run.

$$w = 0.792$$

$$c_1, c2 = 1.4944$$

### 5.4.3 Velocity and Particle Movement

Before generating a new solution and moving the particle, the algorithm calculates the velocities of the particles. This velocity calculation depends on the best global position, best particle position, current particle position, various constants and uniformly generated random numbers. The following equations are used:

$$r1 = rand\,(popsize, npar)\,; r1 = rand\,(popsize, npar)\,;$$

Update Velocity

$$vel = (w * vel + c1 * r1\,(localbestpar - par) + c2 * r2 * (globalbestpar - par))\,;$$

Update particle positions

$$par = par + vel;$$

Since the values of the start times are represented as free slots it makes the most sense to do permutation PSO. Thus the operations in the velocity and position calculations were redefined and velocity is defined as a sequence of position swaps. The definitions of the operations are as follows. Addition of a position and a velocity performs the swaps in the velocity on the position resulting in another position. Subtraction of two positions creates swaps for each index which differentiates between the two positions resulting in a velocity. Finally, multiplication of a velocity and a constant truncates the velocity by one swap if the constant is less than one and augments the velocity with the first swap if the constant is greater than one.

### 5.4.4 Termination Criteria

The algorithm terminates when the maximum number of iterations is exceeded by the current iteration count.

## 5.5 Ant Colony Optimization

### 5.5.1 Implementation Description

#### Data Structure

The three main data structures that need to be represented are the pheromone graph, the ant structure, and distance matrix. The pheromone graph is an array that represents which path it takes from node to node. Each intersection in this matrix represents a path. The ants are represented as the path that they walk to reach the goal.

$$Ant = \begin{bmatrix} 3 & 9 & 4 \end{bmatrix}$$

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

#### Distance Matrix

The distance matrix is used to find the fitness of ants. To calculate the distance matrix, first take an initial feasible schedule for the cyclist and calculate its score. Then for each path replace its respective value in the schedule and recalculate the score. The distance for the path is based on the difference between the initial benchmark score and the path score. For each set of paths between two nodes we take the minimum value and subtract it from all the values to normalize the scores to a minimum of 0. Then we calculate the distance as 1/(1+x) where x is the normalized score.

### Move Ants

Each ant has its path to the goal determined by the pheromone levels at each path. When an ant needs to determine which path to take to reach the next node, the ant will calculate the probability for each path. The probability will depend on the pheromone level in the graph.

$$Probability\,(i) = \frac{Pheromone\,(NextNode)^a \times \frac{1}{Distance(i)}^b}{TotalPheromone\,(NextNode)^a \times \frac{1}{TotalDistance}^b}$$

where:

i = a possible path
NextNode = the node to traverse
Pheromone(NextNode, i) = pheromone level in the graph at index (NextNode, i)
TotalPheromone(NextNode) = Sm of pheromones in the graph corresponding to column NextNode

The ant generates a uniformly random number and picks on of the paths based on their probabilty once the probabilities are determined. This path is saved and traversed and continued until the ant reaches the goal. The fitness is then calculated.

### Evaporate

The pheromone levels in the pheromone graph decreaases by a factor called the evaporation rate. This evaporation rate is multiplied to the pheomone table over time.

$$EvaporationRate = 0.5$$

$$PheromoneMatrix = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \cdot EvaporationRate = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

**Pheromone Deposit**

Ants deposit more pheromone onto the specific path that they took once they have a valid path. The pheromone is deposited onto the pheromone graph. The amount of pheromone deposited is determined by the following function:

$$AddPheromone = \frac{LowestDistance}{CurrentAntDistance}$$

LowestDistance = the distance of a path that an ant found
CurrentAntDistance = the distance of the current ant This function allows us to deposit more pheromone to the path that has the lowest distance. This steers the ants towards the best path which the ants should eventually converge to.

# Chapter 6

# Performance Evaluation

## 6.1 Simulation Setup

In order to evaluate the performance of the metaheuristic algorithms on the two problems being considered, each algorithm is run ten times on the two problems. Each algorithm is implemented in MATLAB, and run on a Core i5 (I5-5287U) at 2.9 GHz. The simulation system has 16GB of RAM and runs OSX 10.10.4 with MATLAB 2014.

## 6.2 Results

### 6.2.1 Overview

The results for each of the five algorithms being run for both of the problems, training plan generation and scheduling, are summarized below. Each algorithm was executed ten times and the mean and standard deviation are calculated from the results. The values recorded are the score, CPU time in seconds, and the number of iterations.

The algorithms will be compared as they perform on each problem using the data in the above two tables. The iteration values will not be used in the analysis as it does not provide additional insight into the performance of the alogorithms as opposed to the CPU time used.

Table 6.1: Training Plan Performance Data

| Algorithm | Score | | Time | | Iterations | |
|---|---|---|---|---|---|---|
| | Mean | St. Dev | Mean | St. Dev | Mean | St. Dev |
| Tabu Search | 2407.4 | 124.4 | 67.796 | 12.2993 | 1000 | 0 |
| Simulated Annealing | 2370 | 130.8 | 1.149 | 0.2512 | 4304.4 | 501.1 |
| Genetic Algorithms | 1827.5 | 167.5 | 7.982 | 1.4294 | **233.3** | 43.8534 |
| Particle Swarm | **2525.3** | 126.3 | **1.072** | 0.3367 | 363.4 | 133.9653 |
| Ant Colony | 59.3739 | 187.7569 | 127.717 | 5.0487 | 500 | 0 |

Table 6.2: Scheduling Performance Data

| Algorithm | Score | | Time | | Iterations | |
|---|---|---|---|---|---|---|
| | Mean | St. Dev | Mean | St. Dev | Mean | St. Dev |
| Tabu Search | 1087 | 130 | 61.084 | 0.8454 | 1000 | 0 |
| Simulated Annealing | 1034 | 87 | 8.57 | 1.0412 | 10000 | 0 |
| Genetic Algorithms | 1106 | 1042 | 402.017 | 63.9225 | **100** | 0 |
| Particle Swarm | **1219** | 112 | **2.144** | 0.124 | 500 | 0 |
| Ant Colony | 1110 | 117 | 5.663 | 0.2475 | 500 | 0 |

### 6.2.2 Training Plans

With regards to the problem of generating training plans, it is observed from the simulations results that particle swarm (PSO), tabu search (TS) and simulated annealing (SA) have the best performance when looking at the score. The standard deviations of the three algorithms are also very similar. When looking at the runtime of the algorithms, tabu search (TS) and ant colony (ACO) have extremely long run times, whereas simulated annealing (SA) and particle swarm (PS) have very low runtimes. Both SA and PSO perform very well in both metrics, however PSO slightly surpasses SA in both score and runtime. It is concluded that PSO provides the most accurate results in the least runtime.

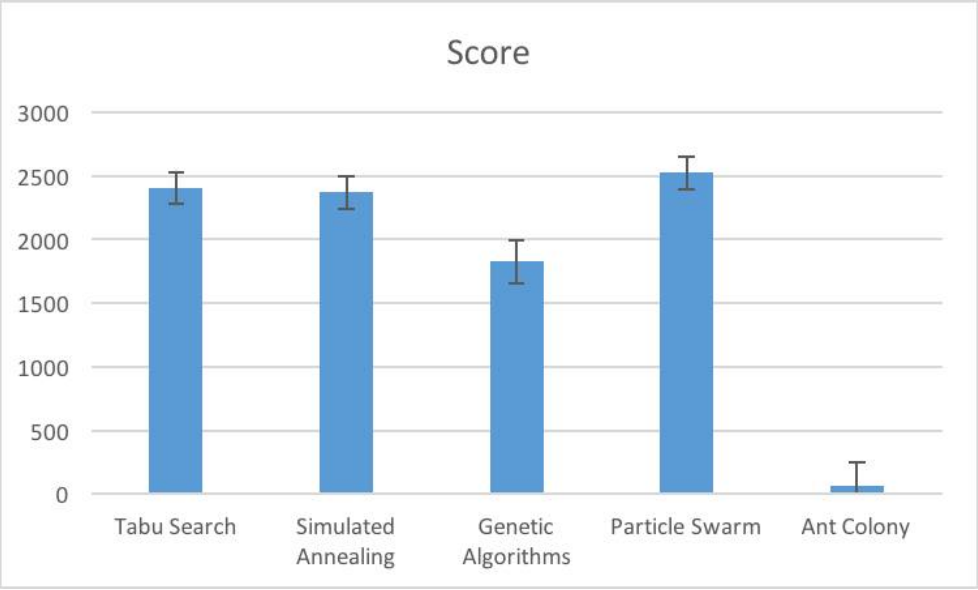


Figure 6.1: Training Plan Algorithm Scores

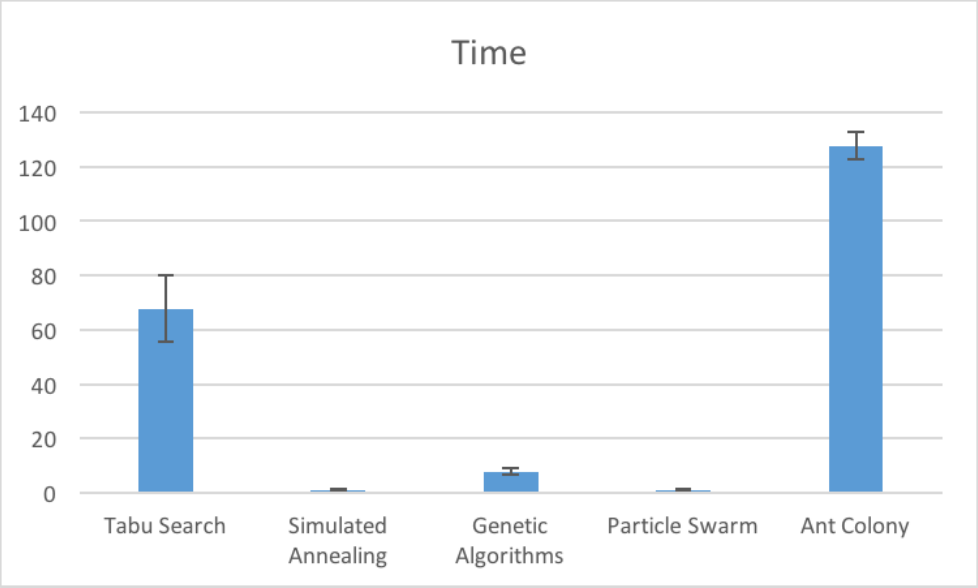


Figure 6.2: Training Plan Algorithm CPU runtime

The remaining algorithms did not perform as well as PSO and SA. Ant Colony (ACO)

performed particularly badly in both the score and runtime metrics. GA performs relatively poorly on both score and runtime.

### 6.2.3 Scheduling

With regards to scheduling of a training plan into a calendar, it is observed that particle swarm (PSO) and ant colony (ACO) seem to provide the best results with respect to score. It should be noted that genetic (GA) also provided very similar results to ACO. However, the runtimes for these algorithms fluctuate greatly. In order to run 10 complete iterations of the algorithms, PSO performed the most efficiently clocking in at an average of just over 2 seconds. On a similar magnitude, ACO took approximately 5 seconds. GA however took substantially longer, at over 400 seconds. Given these results, and the respective standard deviations, it's concluded that PSO provides the most accurate, consistent results.

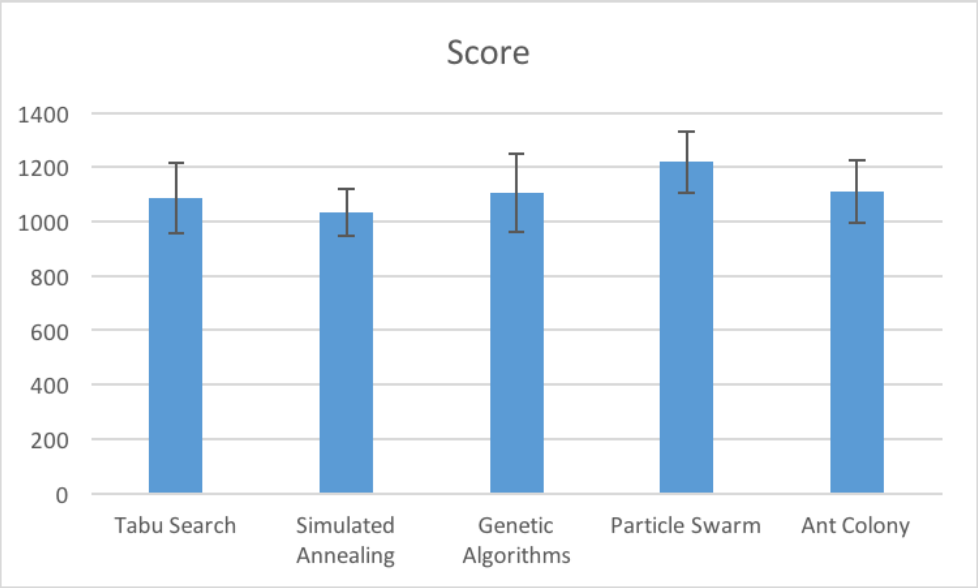


Figure 6.3: Scheduling Algorithm Scores

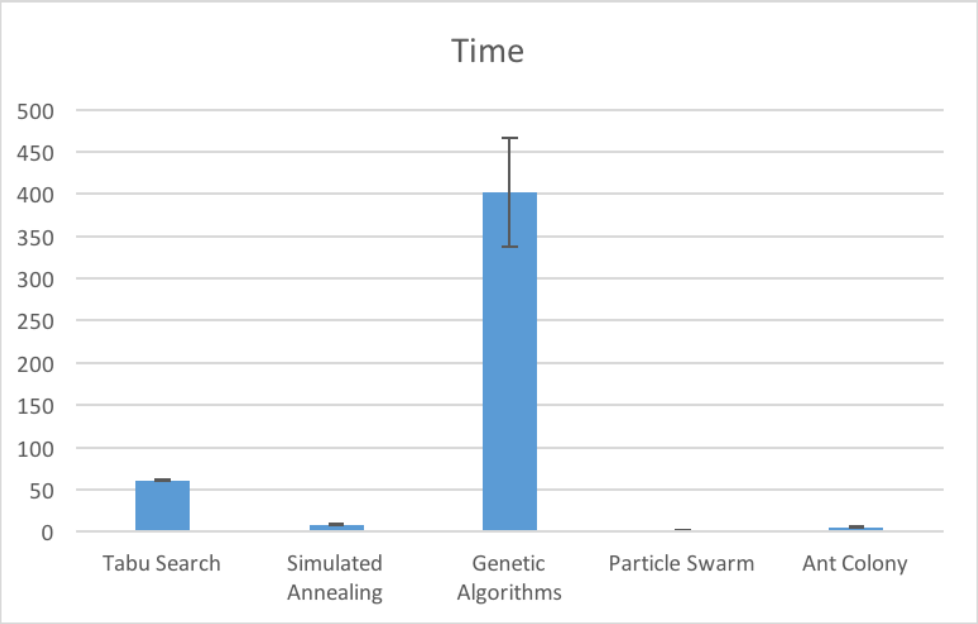


Figure 6.4: Scheduling Algorithm CPU runtime

The remaining algorithms, Tabu Search (TS) and Simulated Annealing (SA) both offered results with less accuracy than the other three previously examined. Tabu Search offered slightly better results than SA, but at the cost of runtime in which it was substantially less efficient.

# Chapter 7

# Conclusion and Recommendations

Restating as outlined before, this report aims to examine and draw conclusions from five metaheuristic algorithms - Tabu Search, Simulated Annealing, Particle Swarm Optimization, Genetic Algorithm, and Ant Colony Optimization - and their performance and accuracy when leveraged to generate training plans and schedule.

The mentioned algorithms were designed and implemented in MATLAB. Performance metrics were taken in terms of accuracy of results, runtime, and number of iterations preformed. The resulting metrics provided insight into the effectiveness of leveraging that metaheuristic to solve the problem.

The resulting information lead to the conclusion that the Particle Swarm Optimization (PSO) metahueristic algorithm provided the optimal results for both training plan generation and scheduling. Ant Colony Optimization (ACO) was a close contendor in many scenarios, while Genetic Algorithm (GA) required too many iterations (and thus computation time) in order to be feasible.

From the collection and analysis of literature and publications, our findings favouring PSO seem to align with the majority of publically available research documents. As training plan generation is a rather esoteric problem that was conceived by our group, there were no publications to compare our results with. However, our implementation of the training plan generation problem is analagous to the knapsack problem with an infinite number of possible items. Therefore, the epicenter of our literature is focus on this known problem.

Based on the results acquired from our testing, we recommend particle swarm optimization for the most optimal and consistent results in both training plans and scheduling. For training plans Simulated annealing is the second best algorithm followed by genetic

algorithms which lie in the middle for optimality an runtime. It is not recommended to use Tabu Search or Ant Colony Optimization as both algorithms have very high run times and in the case of ACO the optimality score is also low. For scheduling, ACO and SA algorithms closely follow PSO in runtimes and optimality. Tabu search is also viable but has a higher runtime and genetic algorithms are not recommended since they have very high runtimes.

# References


[1] Modeling human performance. http://fellrnr.com/wiki/Modeling_Human_Performance. [Online; accessed 3-August-2015].

[2] Supercomposition. http://fellrnr.com/wiki/Supercompensation. [Online; accessed 3-August-2015].

[3] Trimp. http://fellrnr.com/wiki/TRIMP. [Online; accessed 3-August-2015].

[4] Imtiaz Ahmad Ayed Salman and Sabah Al-Madani. *Particle swarm optimization for task assignment problem.* 2002.

[5] Fangguo He. *An improved particle swarm optimization for knapsack problem.* 2009.

[6] Maya Hristakeva and Dipti Shrestha. *Solving the 0-1 Knapsack Problem with Genetic Algorithms.* 2004.

[7] Solveig Irene Grüner Johan Oppen and Arne Løkketangen. *A tabu search based heuristic for the 0/1 Multiconstrained Knapsack Problem.* 2003.

[8] M.Dorigo L.M. Gambardella, É.D. Taillard. *Ant Colonies for the Quadratic Assignment Problem.* 1999.

[9] Department of Industrial Engineering and Taiwan R.O.C. Management, Yuan-Ze University. *The Simulated Annealing for Solving the Quadratic Assignment Problem and Continuous Problem.* 2005.

[10] Fubin Qian and Rui Ding. *Simulated Annealing for the 0/1 Multidimensional Knapsack Problem.* 2007.

[11] Jim N. J. Moon Roberto Montemanni and Derek H. Smith. *An Improved Tabu Search Algorithm for the Fixed-Spectrum Frequency-Assignment Problem.* 2003.

[12] Anshuman Sahu and Rudrajit Tapadar. *Solving the Assignment problem using Genetic Algorithm and Simulated Annealing*. 2007.

[13] Krzysztof Schiff. *Ant Colony Optimization Algorithm For the 0-1 Knapsack Problem*. 2013.