Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Marco Edoardo Palma

12/May/2020

Intelligent Control System For Hybrid Power Units

Project Supervisor: Dr. Jie Zhang

Second Examiner: Dr. Mark Weal

Project Volunteer Supervisor: Dr. Ranga Dinesh Kahanda Koralage

A project report submitted for the award of
MEng Computer Science with Artificial Intelligence

Contents

1 Abstract

At the present time manufactures of hybrid vehicles implement rather diverse philosophies into the management modules of their power units. In many occasions these control units would cycle on a predefined set of pragmas: such as minimising deployment from the internal combustion engine until a certain power requirement is met; start a recharging procedure when some conditions about the battery charge level are reached; or initiate an harvesting mode on liftoffs, coasting or braking occasions. Some other hybrid control units leave to the driver the choosing of predefined deployment or harvesting programs. In most occasions such implementations seem to be making greedy optimisation decisions based on the current state of the vehicle, hence ignoring valuable information regarding the itinerary being undertaken. Motivated by today's diversity in the two engines diversity in running costs and rage, this project aims at evaluating the possibility of providing better fuel economy for hybrid vehicles, by intelligently make use of the information about the itinerary a vehicle must complete. This is extra knowledge is utilised to compose a schedule of power unit modes to be applied at various stages of the journey.

2 Acknowledgements

I would like to thank Dr. Jie Zhang in quality of project supervisor, and Dr. Ranga Dinesh Kahanda Koralage in quality of project volunteer supervisor.

3 Statement Of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

4 Nomenclature

Itinerary or I: formal description of a journey a vehicle is undertaking.

RoadUnit or *ru*: a section member of an *Itinerary* which describe road conditions which are constants for the entirety of its length.

v: Vehicle.

pu: power unit, a parallel hybrid system of a petrol engine and a three phase induction motor capable of regenerative braking.

Power Unit Configuration or *puc*: object describing the gear, fuel-flow and current to be executed by the power unit.

PUC or S: a solution to an *Itinerary* consisting of binding of *Power Unit Configuration* to each *ru* in an *Itinerary*.

Deployment Configuration: a ratio describing how much of the total power the power unit produces is coming from which motor.

ICEngine: Internal Combustion Engine.

eTorque: torque developed by the power unit.

eSpeed: angular speed of the power unit's output shaft into the gearbox.

$diffTorque$: torque developed by the power unit at the differential.

$diffSpeed$: angular speed of the power unit's at the differential.

GB: Gearbox, a vector of gear ratios.

gr: A gear ratio.

Spd: Land speed of the vehicle.

Len: Length of a road unit.

Slope: Slope of a road unit.

Rad: Corner radius.

$Temp$ or $aTemp$: Temperature of a road unit.

$HWSpd$: Head wind speed.

$TWSpd$: Tail wind speed.

$DAPress$: Dry air pressure.

$VPress$: Vapour pressure.

θ: when used in function arguments represents the set of implicit function arguments, usually inferable from previously described functions or, as arguments of inner functions.

$f_{p||e}$: generic function f of petrol (p) or current (e). Such function maps to its definition of 'f_p' and 'f_e' depending on the context invoked. These are used as template functions, to describe the invocation sequence, rather than the specific implementation.

$flow_p$: fuel-flow (mass/unit time).

$flow_e$: current.

$flow_{p||e}$: fuel-flow and current.

$tFlow_p$: Theoretical fuel-flow for some engine configuration.

$tFlow_e$: Theoretical current for some motor configuration.

$expFlow_p$: Expected fuel-flow for some engine configuration.

$expFlow_e$: Expected current for some motor configuration.

SOC: battery pack's state of charge.

5 Background

5.1 Known Approaches to Estimating Fuel Consumption

There exist two main approaches to estimating a vehicle's fuel consumption: statistical and theoretical [27]. The first focuses on generalising costs of driving patterns for generic engine configurations. This entails a reduced granularity to engine-specific parameters, which results in only representative figures over longer journeys[27]. Differently, a theoretical approach reaches higher accuracy by basing its estimation on a direct analysis of the running conditions imposed on the engine [27]. Its implementation requires the development of a simulation environment capable of deterministically compute physical requirements imposed on the vehicle.

5.2 Simulation Environment

This requires the ability of modelling vehicle dynamics in acceleration and constant speed scenarios[3], which allows for the derivation of the engine power factors torque ($eTorque$) and speed ($eSpeed$), useful in fuel economy estimation. The computation of $eTorque$ is proportional to the total road load force acting on the vehicle which is the summation of:

- Aerodynamic component [3][6, pp.97, eq.4-2].
- Road load in the form of friction [3] [6, pp.111, eq.4-12].
- Force of acceleration [3]:

In scenarios of constant power delivery, the required fuel/current can be computed by scaling the their theoretical value by the engine's efficiency rating [3]. Acceleration is modelled by sampling the engine cost at speed intervals[3] of some frequency, effectively reducing an acceleration to a sequence of constant power scenarios.

5.3 Modelling of Engines

Theoretical modelling requires the estimation of an engine's efficiency when running in certain regimes and conditions. This section presents how thermic engine and induction motor are modelled by a system to produce fine-grain estimates of the fuel-flow and current respectively.

5.3.1 Modelling of Spark Ignition Engine

IC engines operate through the combustion of fuel and air mixture. The petrol can be assumed to be octane ($C8H18$) of which ignition in air obeys to the stoichiometric combustion formula [7, pp.71]:

$$[1]C_8H_{18} + [12.5]O_2 + [12.5 \times 79/21]N_2 \rightarrow [8]CO_2 + 9H_2O + [12.5 \times 79/21]N_2$$

The quantity of the mixture defines the potential energy a single fire. It is possible to theoretically estimate the output energy by evaluating the change in heat during the engine's Otto cycle [7, pp.184]:

$$changeInTemp(cr, t1) = t3 - t2 - t4 + t1$$
$$| \; cr = \text{compression-ratio}$$
$$t1 = \text{intake}$$
$$t3 = \text{peak}$$
$$t2 = t1 \times cr^{k-1}$$
$$t4 = t3 \times \frac{1}{cr}^{k-1}$$
$$k = \frac{kp}{kv} = \text{air heat at constant pressure over constant volume}$$

The air mass required to produce some torque is:

$$airMassForTorque(\theta, eTorque) \propto \frac{eTorque}{changeInTemp(\theta) \times kv}$$

The flow is computed by scaling the mixture masses by the number of times the engine fires each second. The firing frequency is a function of *eSpeed* [7]:

$$firesPerSecond(nCyls, eSpeed) = 0.5 \times nCycls \times herz(eSpeed)$$

5.3.2 Modelling of Induction Motor

The current required by an induction motor in order to produce some $eTorque$ at some *eSpeed* is computed as[1, pp.188 eq.5.7]:

$$current(eTorque, eSpeed, v) \propto \frac{power(eTorque, eSpeed)}{1.73 \times v}$$
$$| \; v = \text{battery's constant voltage}$$

Similarly, the current harvested when operating in regenerative mode, receiving some $eTorque$ at some *eSpeed* from the driveshaft[4, pp.3239 eq.5]:

$$currentRegen(eTorque, eSpeed, v) \propto \frac{eTorque \times eSpeed}{v}$$

5.4 Methodologies for learning efficiencies

The efficiency of a motor is its degree of supplied power utilisation[3]. The models described in the previous section can be integrated in the supervised learning of efficiency ratings caused of engine parameters. There are a number of aspects impacting the efficiency of operation of both engines. However, only the subset of those predictable before the journey has begone can be considered. The efficiency curve of IC engines and induction motor (power) is largely controlled by: engine torque/load and speed, and ambient temperature. Other

constant features, such as compression ratio, bmep, injector design, etc.[7, pp.858-883], will be intrinsically scaling the efficiency rating. Finally, the efficiency of induction motors, when harvesting, is controlled also controlled by the state of charge of the battery pack (SOC)[4]. In this section follow two methodologies used for the supervised learning of engine telemetry of the features described.

5.5 K-Nearest-Neighbour Regression

KNN provides a simple interface for the binding of features and results. It consists mapping information on a multidimensional space. It can be used for both classification and regression problems, however, basing its prediction on a pattern-recognition, it is vital to consider the likelihood of neighbours's equidistance to query vectors[2]. Such consideration is addressed from a dimensionality prospective by mapping the dimension of the feature vectors used, to the volume of an N-Ball [2] $V_n(R) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)}R^n$. This formula yields in Γ a factorial scaling of the n-Ball's volume, meaning the probability of equidistance increase in a factorial fashion with number of features used[2]: the larger the number of features, the less the volume of the n-ball is compared to the entire volume of the search space. In this context, at most 3 features will be utilised for the geometric location of the neighbours, which provides a negligible probability of equidistance[2].

5.6 Multivariate Polynomial Regression

Where a KNN Regression computes an engine's efficiency by finding the telemetry points with highest affinity to the given query point; a multivariate regression is expected to compute similar results by combination of the linear regressors for each feature in the query [20, pp.67]. This technique is expected to lead to two main advantages when compared to KNN regression. The first is the ability to make reasonable predictions on query points either outside or in less dense areas of the range of values provided during the learning process. The second is a time and space complexity gain during evaluation. In fact, the model would operate in linear space and time on the number of regressors selected.
For a small number of features, as it is the case in this context, the optimal vector of regression weights can be computed by a Least Square Estimator normal function, hence avoiding the need of configuring a gradient descent routine: [20, pp.73]

$$\hat{\theta} = (X^T X + \lambda \mathbb{I})^{-1} X^T \mathbf{y} \tag{1}$$

In normal function, the scalar value λ allows for control over the level of bias of the model, by increasing the values of the diagonal of the variance-covariance matrix ($X^T X$). X is known as the design matrix, a vertical composition of vectors adhering to the regression formula[**?**, pp.71]:

$$y = \beta + \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$$

Hence, a feature vector $[x_1, x_2, ..., x_p]$ with expected value y is injected into the design matrix in the form $[1, x_1, x_2, ..., x_p]$ and its estimate into vector $\mathbf{y}$ (at the same vertical index).
Lastly, the use of polynomials as features in the input vectors can improve the expressivity of

the model, by linearly fit nonlinear models, should the nature of the telemetry not be linearly describable[20, pp.389-392].

5.7 Genetic Search Algorithms

Given the lack of overlapping subproblems **(appendix F)** which induce dynamic programming approaches to perform poorly, the use of a genetic search is evaluated.

5.7.1 Suitability to the problem

Formally, there exists no clear cut for establishing the suitability of this approach to a problem. However, there are conditions for which genetic strategies proved to perform best[18, pp116]; this context agrees with all of such points raised by Mitchell[18]:

- The problem has a large search space[18]. For an itinerary of length n and a pu with i possible modes, there exist $O(i^n)$ solutions.
- The problem space is not expected to be smooth[18]. Due to difference in running costs of the two engine, minor changes in a solution tend to translate in a significant change in the solution's cost.
- The task is to find a satisfiable solution in a preset amount of time.

5.7.2 Selection function

The selection procedure dictates the magnitude of influence each gene has on the next generation [18, pp.124]. As a natural cost based scoring system (*section* 5) would lead to a fitness-proportional selection, which encourages fast convergence to a local minima [18, pp.125], the selection function can implement a smoothing routine to the fitness values such as sigma scaling [18, pp.125]. Elitism can also be injected in the selection process to further improve the performance of the genetic search[18, pp.126]. This ensures the best genes of each generations are retained in the next generation[18, pp.126].

5.7.3 Crossover and Mutation

Crossover enables the genes of a generation to combine areas of their solutions to generate new genes, whilst mutation aids generations in not becoming stuck in one subset of genetic material. There is no definitive indication regarding which crossover technique is the most appropriate to some problem [18, pp.129]. Given the strict physical constraints marking the validity of a solution, this project should consider two crossover methods which variate in the amount of variability of a new gene: single point crossover and multiple point crossover. The former merges two genes at a random position, the second pulls genetic material from multiple genes.

5.7.4 Gene Repair

A gene repair step after crossover and mutation can be a powerful tool to improve the performance of genetic searches, when there exist the probability that new genes might carry invalid solutions [17]. The performance gain comes in a reduced number of generations required for the convergence to the same result. The repair step yields valid result when performed both by analysis of the grandparent generation or deterministically on the new gene[17]. This project will present how a solution can be repaired in a deterministic fashion.

6 The Challenge

To optimise the usage of a vehicle's hybrid power unit, it means minimising the summation of the costs of running of each portion of an itinerary.

An Itinerary I can be modelled as an ordered sequence of feature vectors called RoadUnit ru, which describe the road in terms of its power requirements they impose on the vehicle.

$$I = [ru_1, ru_2, ..., ru_{|I|}] =$$

$$\begin{bmatrix} Spd_1 & Len_1 & Slope_1 & Rad_1 & Temp_1 & HWSpd_1 & TWSpd_1 & DAPress_1 & VPress_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

To each $ru \in I$ a 'Power Unit Configuration' (puc) can be linked, which defines operation modes/regimes that the power unit should apply to meet the requirements of the RoadUnit. Individual puc can be collected into a matrix PUC which is a candidate solution for an Itinerary I, ie.

$$PUC = [puc_1, puc_2, ..., puc_{|PUC| or |I|}]$$

The cost function of a PUC is the monetary cost for executing each puc linked to every $RoadUnit$ in the $Itinerary$, which represents the cost of the vehicle completing the journey utilising the pu according to a certain program.

$$PUC\_Cost(v, I, PUC) = \sum_{i=1}^{|I|} fuelCost(v, I_i, PUC_i)$$

where

$$fuelCost(v, ru, puc) = petrolCost(flow_p, time) + electricityCost(flow_e, time)$$

($flow_p$ and $flow_e$ are feature of puc, and $time$ is the duration of such puc defined by the length and speed of ru)

with constrains:

- The power delivered by the puc is sufficient to meat the power requirements that ru imposes on v, .ie:

$$powerDelivery(flow_p, flow_e) = powerRequirement(v, ru)$$

- Both ic engine and electric motor operate in their working ranges:

$$flow_p = \begin{cases} 0, \\ flow_{p\_min} \leq flow_p \leq flow_{p\_max} \end{cases}$$

$$flow_e = \begin{cases} 0, \\ < 0, \\ flow_{e\_min} \leq flow_e \leq flow_{e\_max} \end{cases}$$

- The vehicle can provide enough energy resources for the *puc* to be sustained for the duration of the ru:

$$petrolMassRequired(flow_p, time) \leq totalPetrolMassIn(v)$$
$$currentRequired(flow_p, time) \leq totalCurrentIn(v)$$

- v is not a constant: each *puc* alters the resources available in v based on the length and magnitude of application of $flow_p$ and $flow_e$:

$$totalPetrolMassIn(v) \mathrel{-}= \textstyle\sum_{\forall puc \in PUC} petrolMassRequired(flow_{pofpuc}, time_{ru})$$

$$totalCurrentIn(v) \mathrel{-}= \sum_{\forall puc \in PUC} currentRequired(flow_{eofpuc}, time_{ru}) + harvestCurrent(puc)$$

The optimal solution is a PUC which minimises the PUC_Cost of all $PUC \in \{PUC\}$ where $\{PUC\} = allValidPUC(v, I)$ for a given vehicle v and itinerary I.

$$S(v, I, \{PUC\}) = \arg\min PUC\_Cost(PUC)$$

7 Modelling

7.1 Modelling: A Grammar For Itineraries

In this sections is a high level description of the main features of the grammar developed which enables the system to receive information about the specification of an itinerary. - *The full grammar can be found in the 'Grammar.y'. file in the development archive of this project.*

Composition of Constants. Constants are the binding of a unique name to non-mutable objects of the types: ambient conditions and road units. These are defined as:

```
<constant> ::= Const <varname> <constdefinition>;
<varname> ::= [[a-zA-Z] [0-9] _ ']*
<constdefinition> ::= <ambientcond> | <roadunit>
```

Composition of Ambient Conditions. These require the definition of values concerning: temperature, wind head and tail speed, dry air pressure and vapour pressure (no):

```
<ambientcond> ::= Ambient :
                    { <temperature>
                    ; <headwind>
                    ; <tailwind>
                    ; <dryAirPressure>
                    ; <vapourPressure>
                    }
```

Ambient conditions can also be copies of other constants of the same type:

```
<ambientcond> ::= Ambient : <varname>
```

Composition of Road Units. Like ambient conditions, *RoadUnit*s are themselves aggregation of units of measures:

```
<roadunit> ::= RoadUnit:
                 { <speed>
                 , <length>
                 , <slope>
                 , <cornerradius>
                 , <ambientcond>
                 };
             | Halt
```

Hence, a possible derivations of a constant *RoadUnit* can be:

```
Const str RoadUnit : {
        Speed: 0;
        Length: Km 5;
        Slope: 0;
        CornerRadius: 0;
        Ambient: ambConst;
    };
```

*RoadUnit*s can also be copies of constants of the same type, with one or more of their attributes remapped.

```
Const str2 RoadUnit :str(Speed: MPH 50, Slope: Deg 2, HeadWind: KPH 1);
```

Composition of Itineraries. An *Itinerary* objects can only be defined once per file, and it represents a non empty sequence of *RoadUnit* definitions, or variable names previously bounded to *RoadUnit* derivations. This is expected to be the last derivation, after a sequence of comma-separated constant definitions.

```
<itinerary> ::= <constdefinitions> Itinerary : [<roadunitseq>]
              | Itinerary : [<roadunitseq>]
<roadunitseq> ::= <roadunit>, <roadunitseq> | <roadunit>
```

An example derivation of an itinerary can be of the form:

```
Const amb1 Ambient: { Cel 21; 0; KPH 3; KPascal 101; Torr 23 };
Const ru1 RoadUnit: {0; 0; 0; 0; Ambient: amb1};

Itinerary:
  [ ru1(Speed: MPH 30, Length: Miles 3)
  , Halt
  , ru1(HeadWind: MS 1)
  , { MPH 70; Miles 2; 0; 0; Ambient: amb1 }
  ]
```

7.2 Modelling of Fuel-Flow and Current

Reminder: this document makes use of the notation '$f_{p||e}$' to refer to a generic function f of petrol or current flow. Such functions map to their definitions of 'f_p' and 'f_e' depending on the context.

Through the computation of $flow_p$ (fuel-flow) and $flow_e$ (current), the system is enabled to theoretically estimate the energy requirements of both propulsion assemblies, and compute their efficiency.

Of each engine it is possible to compute the theoretical flow of petrol or current through a generic function $tFlow_{p||e}(\theta)$. For IC engines, this computes the required fuel-flow (*section* 4.3.1).

$$\begin{aligned} &tFlow_p(eTorque, eSpeed, aTemp; \theta) = \\ &\quad let\ af = airMassForTorque(cr, aTemp, eTorque) \\ &\qquad ff = stoichiometricFuel(af) \\ &\qquad n = firesPerSecond(nCycls, eSpeed) \\ &\quad in\ ff \times n \end{aligned}$$

Whereas, for induction motors it computes the required current output, if torque is positive (power mode), or current generated if negative torque (regenerative braking).

$$tFlow_e(eTorque, eSpeed; \theta) = \begin{cases} eTorque \geq 0 & current(eTorque, eSpeed, v) \\ eTorque < 0 & currentRegen(eTorque, eSpeed, v) \end{cases}$$

The efficiency function of an engine, $effFlow_{p||e}$, maps a power demand to a ratio of its real-world/observed $flow_{p||e}$ and theoretical supply $tFlow_{p||e}$.

$$effFlow_{p||e}(\theta) = \frac{realFlow_{p||e}(\theta)}{tFlow_{p||e}(\theta)}$$

Hence, the expected $flow_{p||e}$ of an engine is modelled by the generic function $expFlow_{p||e}$. This scales the theoretical flow by the expected efficiency of the power demand.

$$expFlow_{p||e}(\theta) = effFlow_{p||e}(\theta) \times tFlow_{p||e}(\theta)$$

7.3 Modelling of Gearbox

Gears allow the engines scale the ration of $eTorque$ and $eSpeed$ for the delivery of an amount of power. The system is provided with an interface for the selection of the gear which minimises the cost of operation.
Of vehicle v, its gearbox GB is a vector of gear ratios gr:

$$GB = [gr_0, ..., gr_{|GB|}]$$

The expected flows demanded by a gear, for the fulfilment of some $dTorque$ and $dSpeed$, is computed as higher order function of $expFlow_{p||e}$:

$$\begin{aligned} &grFlow_{p||e}(dTorque, dSpeed, gr, \theta) = \\ &\qquad expFlow_{p||e}(grTorqueInput(dTorque, gr), grSpeedInput(dSpeed, gr), \theta) \end{aligned}$$

Using $grFlow_{p||e}$ as input of the cost function $fuelCost$, enables the computation of the most efficient gear (less costly):

$$bestGear_{p||e}(dTorque,dSpeed, GB, \theta) = \arg\min_{} fuelCost(grFlow_{p||e}(dTorque, dSpeed, gr, \theta)), \forall gr \in GB$$

7.4 Modelling of Power Unit

A *pu* handles the splitting in power deployment between the IC engine and induction motor, according to some power unit mode, ie. evaluates the *puc* for some *pum*, $eTorque$ and $eSpeed$ A *pum* carries either:

- The percentage of torque delivered by the induction motor (*eg: pum=20%*), in which case, the cost of the split is computed as:

$$\begin{aligned} splitRatio(pum, eTorque, eSpeed, \theta) = \\ expFlow_e(eTorque \times \frac{pum}{100}, eSpeed, \theta) \\ +expFlow_p(eTorque \times \frac{100 - pum}{100}, eSpeed, \theta) \end{aligned}$$

- Indication that the ratio should allow the induction motor to deliver up to its maximum torque (*pum=PUMFullECompensation*):

$$\begin{aligned} splitEComp(pum, eSpeed, \theta) = \\ expFlow_e(x, \theta) \\ +expFlow_p(eTorque - x, \theta) \\ \mid x = min(eTorque, MotorMaxTorque) \end{aligned}$$

7.5 Modelling of Steady Pace

Steady pace occurs whenever the vehicle is required to maintain a constant speed and constant power output. In such scenarios, the system is given a *pum* with which to execute a $RoadUnit$, and proceeds to **Pseudocode at: B**:

1. Compute the power configuration imposed by the $RoadUnit$ using principles of vehicle dynamics.
2. Evaluate the $eTorque$ and $eSpeed$ demanded by each gear, and retain only those in the *pu*'s speed and torque range.
3. Select the configuration with the least associated cost ($bestGear_{p||e}$).

Failure to return a valid *puc* symbolises the physical impossibility of completing the $RoadUnit$.

7.6 Modelling of Accelerations

A vehicle performs an acceleration whenever its current speed is lower than the speed required by the upcoming *RoadUnit*. The system is provided with an acceleration rate target a_s and a tolerance a_{Toll}, which constrain time and space for the operation. The system must compute the optimal gearing: the sequence of gear changes which minimises the cost of acceleration. This can be seen as a schedule of the form:

$$acc(v, ru, a_s) = \begin{bmatrix} Date_1 & V_{v_1} & Gear_1 & eTorque_1 & eSpeed_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Date_p & V_{v_p} & Gear_p & eTorque_p & eSpeed_p \end{bmatrix}$$

$$\begin{aligned} |\ & eSpeed_1 = v_v \\ & eSpeed_p = v_{ru} + v_{tollerance} \\ & \sum_{i=1}^{p} Date_i \leq accTime(v, ru, as) + t_{tollerance} \\ & \sum_{i=2}^{p} (Date_i - Date_{i-1}) \times V_{v_i} \leq accDist(v, ru, as) + d_{tollerance} \\ & \forall i \in \{2..p\}\ Gear_i \leq Gear_{i-1} \end{aligned}$$

in which: at every sampled point in time $Date_i$, the vehicle is moving at speed v_i, selects gear $Gear_i$ and develops torque $eTorque_i$ at a rate $eSpeed_i$, to reach the next row vector in the schedule.

7.6.1 Sampling requirements

Pseudocode at: C
For every gear in scope, the physical requirements acting on the *pu* are sampled with time frequency *interval*. This produces a map of every gear's scheduled events.

$$sample(v, ru, a_s, a_{toll}, interval) = [(Gear, [(Speed, eTorque, eSpeed, pum)])]$$

The sampling function exit or restart on the conditions:

- if sampling has reached the final velocity, it terminates.
- If the gear over-spools the *pu*, sampling stops: the gear is exhausted.
- If the gear under-spools the pu, sampling restarts from the gear's minimum speed, if this is in the acceleration's scope.

The sampling of a schedule point follows the procedure:

1. Compute how much *eTorque* is required to cause an acceleration of a_s.
2. If the *pu* in the selected mode cannot develop the *eTorque* required, this is moved into a *pum* known to have the highest probability of complying.

3. Of the $eTorque$ the pu can produce, the system computes the acceleration it causes.

4. Sampling on this gear continues iff the pu could produce an acceleration of at least a_{toll}.

7.6.2 Scoring sample points

To enable the computation of the optimal gearing, each sample point is bounded to its propulsion cost, through a function of type:

$$\begin{aligned} &sampleEco([(Gear, [(Speed, eTorque, eSpeed, pum)])], (time, aTemp)) \\ &\quad = [(Gear, [(Speed, eTorque, eSpeed, pum, cput)])] \end{aligned}$$

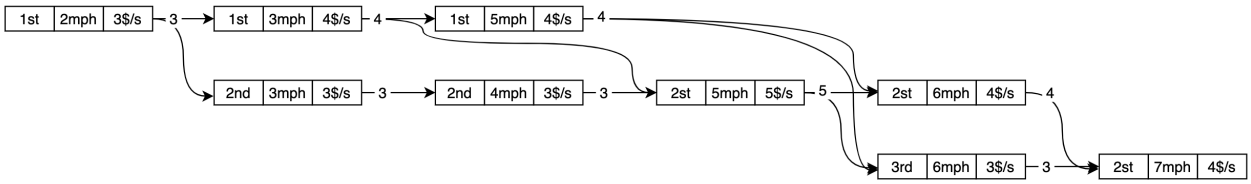
This is achieved by mapping $toSampleEcoPoint$ on each sample point.

$$\begin{aligned} &toSampleEcoPoint((Speed, eTorque, eSpeed, pum), (time, aTemp)) \\ &\quad = (Speed, eTorque, eSpeed, cput) \\ &\quad \mid cput = fuelCost(expFlow_{p||e}(eTorque, eSpeed, aTemp), time) \end{aligned}$$

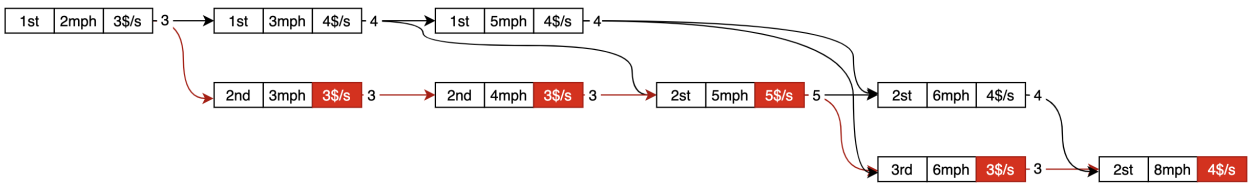
7.6.3 Optimal Gearing With Dijkstra

Motivation Of Approach

The problem can reduced to a directed acyclic graph (DAG), in which the nodes are the sample points ($(Gear, Speed, CostPerUnitTime)$, directed in order of their speed (increasing) and gear ratio (decreasing). The edges of the graph are weighted upon the $CostPerUnitTime$ of the source node. The starting node of the graph fictitious and connected to all nodes carrying the initial velocity. For example:



A greedy approach would shift to a new gear as soon as a lower $CostPerUnitTime$ is available; the resulting gearing would be:

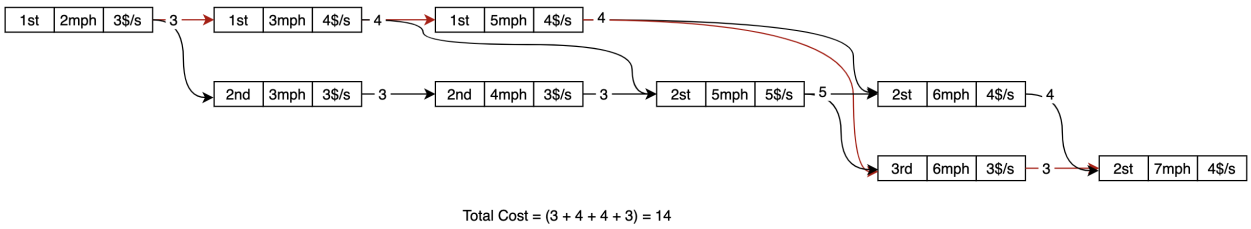


Total Cost = (3 + 3 + 3 + 5 + 3) = 17

This is a valid solution, however not optimal: shifting into $2nd$ on on the first occasion, forces the solution to take a longer path which is more economical in its units, but not in its

total cost.

Dijkstra algorithm is an optimal algorithm which can be implemented to find the shortest path (SPP) [5] between two nodes in a DAG with positive edges. The Dijkstra solution to the example is the optimal path:



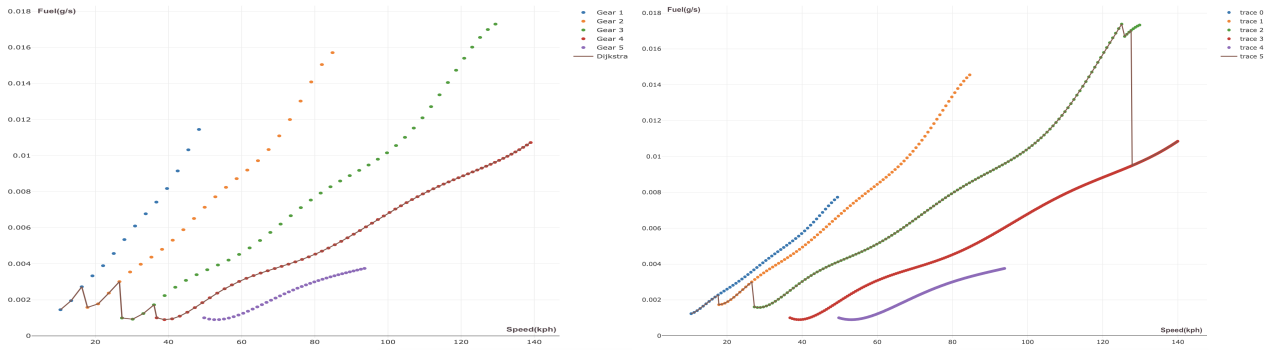
Implementation

The natural form of a fuzzy forward and backward routine is utilised [5, pp.1233], however identifying neighbouring nodes as those nodes with: an equal or higher gear, and speed non greater than its successor (on the same gear):

```
getNeighbors(sample, node)
    let # Remove all gear options with lower current speed, or higher gear.
        subSample = subSampleOnMinSpeedAndGear(node.v,node.gr)

        # Find what speed the vehicle needs to reach.
        nextSpeed = either(gearOpts[gr0][0],getMinSpeedOfSample(gearOpts))

        # Subsample again on speed smaller than the next target speed.
        nbs = subSampleOnMaxSpeed(gearOpts,nextSpeedOnGr0)

    in nbs
```

To visualise the result, it is possible to plot a graph of speed against cost per unit time of each gear during the acceleration. Plotting the Dijkstra output shows when and in what conditions then algorithm shaped the gearing:

Figure 1: Dijkstra on acceleration sample sets, for two dummy engine models moving the same vehicle from standstill to $140km/h$.



7.7 Modelling of Braking

A vehicle undergoes a braking procedure whenever its current speed is higher than the speed required by the upcoming *RoadUnit*. This is an occasion for the vehicle to harvest electrical charge by utilising its induction motor as a generator[1, pp.219]. Crucially, the amount of current harvestable is a function of the torque, angular speed incoming into the motor, and SOC[4, pp.3239 eq.5]. As the values of *eTorque* and *eSpeed* are respectively the torque and speed generated by the wheels scaled by some gear ratio in the gearbox, the task of generating a braking program also involves the searching of an optimal gearing.

The system accepts a deceleration rate d_s which constraints the final solution to a total time and space quantities. Hence, similarly to optimal acceleration gearing the solution is of the form:

$$dec(v, ru, d_s) = acc(v, ru, -d_s, -d_s)$$
with new constraints:
$$\sum_{i=2}^{p} (Date_i - Date_{i-1}) \times V_{v_i} \;=\; decDist(v, ru, d_s)$$
$$\forall i \in \{2..p\}\ Gear_i \geq Gear_{i-1}$$

7.7.1 Sampling

Compared to acceleration sample, braking drops the *pum* term as this is always full electric:

$$sample(v, ru, d_s, interval) = [(Gear, [(Speed, eTorque, eSpeed)])]$$

Sampling computes the *eTorque* and *eSpeed* as the scaling of gearbox and differential of the wheels torque and angular speed (induced by the inertia of the vehicle). This is an inductive process in which (**Pseudocode at D**):

- Exit and restart conditions inverse the logic of acceleration sampling: gear sampling stops on under -spooling gears, and restarts on over-spooling gears.
- A new sample point computes the resulting force acting on the wheels, and the wheels speed, and scales them according to the gear ratio. The sampling halts if the force acting on the wheels is zero or less: no braking is required.

7.7.2 Scoring Sample Points

With the aim of recognising the most efficient subset of sample points, the system binds to each the current energy the motor is expected to inject into the battery pack. The aim is to change the type of the sample in the fashion:

$$\begin{aligned} sampleEco([(Gear, [(Speed, eTorque, eSpeed)])], (time, aTemp, SOC)) \\ = [(Gear, [(Speed, eTorque, eSpeed, cEnergy)])] \\ \mid cEnergy = \text{expected current energy entering the battery pack} \end{aligned}$$

Which can be achieved by mapping the *toSampleEcoPoint* to each sample point.

$$
\begin{aligned}
&toSampleEcoPoint((Speed, eTorque, eSpeed), (time, aTemp, SOC)) \\
&\qquad = (Speed, eTorque, eSpeed, cEnergy) \\
&\qquad \mid cput = fuelCost(expFlow_{p||e}(eTorque, eSpeed, aTemp), time) \\
&\qquad\quad cEnergy = currentEnergy(expFlow_{e}(eTorque, eSpeed, aTemp, SOC), time)
\end{aligned}
$$

cEnergy is always a negative value due to its flow moving into the battery pack, rather than out as it is the case in power modes. Hence, the lower the value of *cEnergy*, the more energy is being harvested.

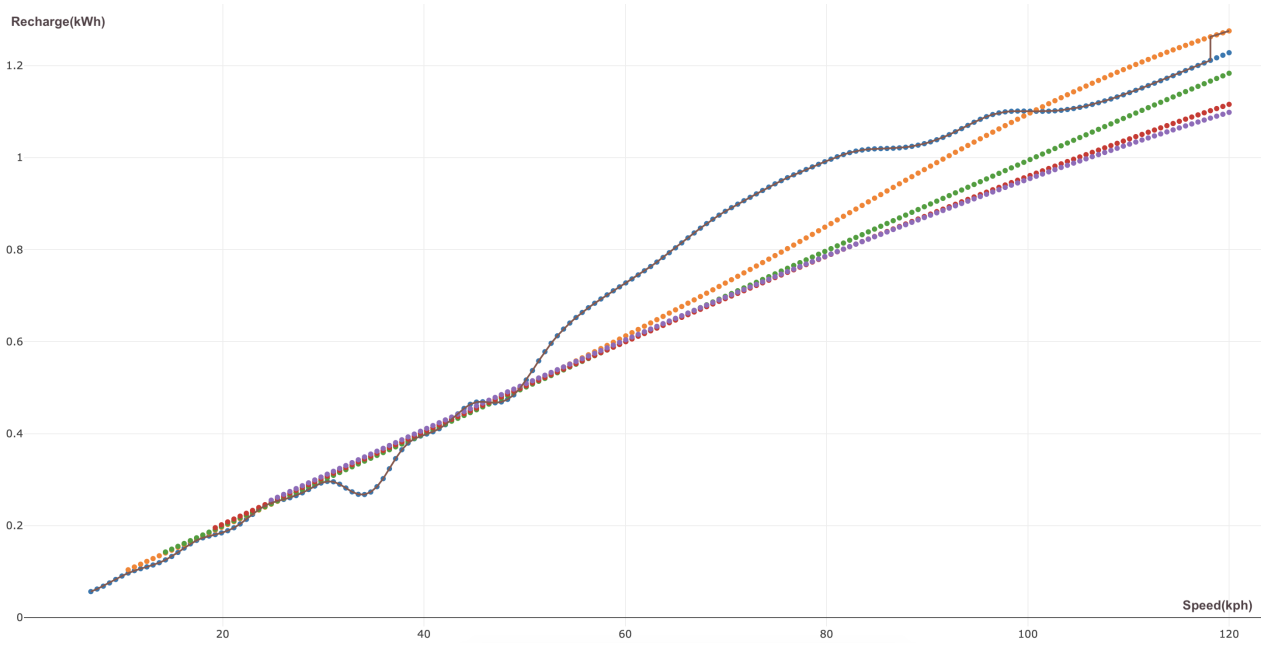
7.7.3 Optimal Gearing With Dijkstra

The sample is compiled into a DAG, of which edges are a measure of current the source node is able to induce into the battery pack. The very same implementation of Dijkstra for acceleration can be utilised in the context of deceleration/energy-harvesting: the aim is to minimise the energy leaving the battery pack. Despite the negative weighting, no negative cycles are injected into the DAG: the nodes in the graphs always require to move to neighbours with either lower velocity, meaning that at no point can one node be either directly or indirectly connected to one of its predecessors. A deceleration is reduced to an acceleration problem, by modifying the logic according to which neighbour nodes are computed. This now returns nodes with the next lower speed, and of increasing gear ratios:

```
getNeighbors(sample, node)
    let # Remove all gear options with higher current speed, or lower gear ratio.
        subSample = subSampleOnMaxSpeedAndGear(node.v,node.gr)

        # Find what speed the vehicle needs to reach.
        nextSpeed = either(gearOpts[gr0][last],getMaxSpeedOfSample(gearOpts))

        # Subsample again on speed higher than the next target speed.
        nbs = subSampleOnMinSpeed(gearOpts,nextSpeedOnGr0)

     in nbs
```

To visualise the result, it is possible to plot a graph of speed against *cEnergy* of each gear during the deceleration. Plotting the Dijkstra solution in the same plot makes the gearing evident.

Figure 2: Speed(Km/h) Vs cEnergy(Wh) of Dijkstra output on deceleration sample plots for a dummy induction motor stopping a vehicle from $120Km/h$.



From this plot one can visually recognise how Dijkstra recognised the value in starting the deceleration in third gear, to then finish the process in second gear.

8 Methodologies: Learning Engines Efficiencies.

8.1 K-Nearest-Neighbour Regression

8.1.1 Building Dataset with Points Density Control

From telemetry, engine configuration and their respective observed flow of petrol or current are organised in a vector as:

$$FV_{in} = [eTorque, eSpeed, \theta, flow_{e||p}]$$
$$| \ \theta = aTemp \text{ for power modes, or } SOC \text{ for regenerative braking.}$$

Of this vector, the $flow_{p||e}$ value are replaced with the efficiency rating, so to obtain a telemetry point $TelPoint$:

$$TelPoint(FV_{in}) = [eTorque, eSpeed, \theta, effFlow_{p||e}(FV_{in})]$$

In light of the fact that: the instruments used for the recording of the telemetry may have a limited resolution in their measurements, and that the efficiency of an engine is not expected to spike within some contained changes of configuration or ambient conditions, a $TelPoint$ can undergo a reduction phase in which each feature vector (eff excluded) is rounded to the nearest multiple of some scale. This enables the system to summarise similar $TelPoint$ into a single node, carrying the accumulated average eff value.

$$toAvgLocation(TelPoint, scales) = [eTorque_{scaled}, eSpeed_{scaled}, \theta_{scaled}, eff]$$

Features of a $TelPoint$ encapsulate values of different scales, which leads the affinity rating of KNN nodes to be saturated by the scale of one or more features only (mostly $eSpeed$). This issue is mitigated by normalising each feature to the same scale, computing the reduction:

$$toRange\{(x, (min_1, max_1, min_2, max_2)\} = min_2 + \frac{(x - min_1) * (max_2 - min_2)}{(max_1 - min_1)}$$

Hence, a normalised vector of a $TelPoint$ is computed as:

$$norm(TelPoint, \mathbf{rs}) = [toRange(eTorque, rs_1), toRange(eSpeed, rs_2), toRange(\theta, rs_3), eff]$$
$$| \ \mathbf{rs} = \text{matrix of range rescaling vectors.}$$

$TelPoint$s are stored in the dataset in $TelNode$ nodes, which carry the accumulated average at any particular normalised location of the dataset.

$$TelNode = [eTorque_n, eSpeed_n, \theta_n, eff, n_{points}]$$
Where
eff = accumulated average efficiency for all $TelPoint$ scaling to this node;
n_{points} = number of $TelPoints$ summarised in this node.

Pseudocode for the building of a KNN Dataset is included at: E.

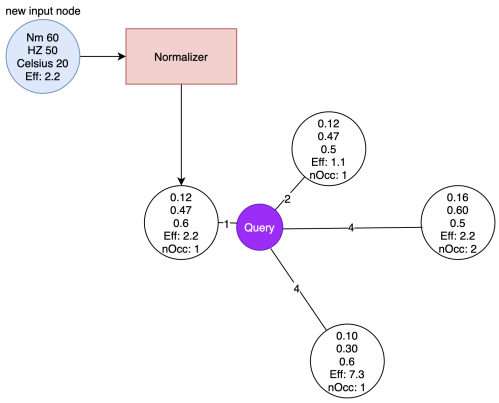
8.1.2 Evaluating Queries With Weighted Average

Aside from providing the search algorithm the ability to select the number of neighbours to consult when building a response to a query, the system will also be performing a weighted average of the eff value carried by such neighbours. This aim at allowing each neighbour to make a weighted contribution to the efficiency value, depending on their Euclidean distance to the query point. Such average can be calculated:

$$knn(query, \{node\}) = \frac{\sum_{\forall node \in \{node\}} \frac{eff_{node}}{euclidean(query, node)^n}}{\sum_{\forall node \in \{node\}} \frac{1}{euclidean(query, node)^n}}$$

n, in the above formula, scales the effect of the weight: the higher the value, the more less far away neighbours will contribute to the average.

Figure 3: Knn dataset example.



8.2 Multivariate Polynomial Regression

Design Matrix of Spark Ignition Engines and Induction Motor (Power)

The telemetry is processed into a collection relationships of feature vectors $\mathbf{x^0}$ and corresponding response vector $\mathbf{y^0}$ of the form.

$$\mathbf{x^0} = [eTorque, eSpeed, aTemp]$$
$$\mathbf{y^0} = [airFlow]$$

This represents what air-flow was demanded from the engine producing some torque at some rate in some ambient temperature. As the regression is attempting to fit an efficiency curve, the response value must be converted to an efficiency rating, hence computing:

$$effFlow_p = \frac{airFlow}{tFlow_p(eTorque,eSpeed,aTemp)}$$

to obtain a new the feature vector and response relationship:

$$\mathbf{x^0} = [eTorque, eSpeed, aTemp]$$
$$y^0 = effFlow_p$$

The input attributes are mapped to vectors of the design matrix based on the number of polynomials chosen. This come with a choice of including or not the power feature, derivative of toque and speed:

$$\mathbf{x} = toPoly([eT, eS, aT]) = \begin{bmatrix}1 & eT & eS & aT & eT & eT^2 & eS^2 & at^2 \dots\end{bmatrix}$$

$$\mathbf{x} = toPowPoly([eT, eS, aT]) = \begin{bmatrix}1 & pow(eT, eS) & eT & eS & aT & eT & pow(eT, eS)^2 & eT^2 & eS^2 & at^2 \dots\end{bmatrix}$$

As far as induction motor is concerned, the procedure is ideitical except the telemetry is processed to return the required current:

$$\mathbf{x^0} = [eTorque, eSpeed, aTemp]$$
$$y^0 = current$$

which in turn requires a different efficiency rating:

$$\mathbf{x^0} = [eTorque, eSpeed, aTemp]$$
$$y^0 = effFlow_e = \frac{current}{tFlow_e(eTorque, eSpeed)}$$

The design matrix and response vector for a set of p pairs of $\mathbf{x}$ and y are in the form:

$$\mathbf{X} = \begin{bmatrix}\mathbf{x_1}\\ \mathbf{x_2}\\ \vdots\\ \mathbf{x_p}\end{bmatrix} \quad \mathbf{y} = \begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_p\end{bmatrix} \tag{-58}$$

Design Matrix of Induction Motors (Regen)

The process is rather similar to the one presented in previous paragraph, the differences are that in this scenario, the ambient temperature is replaced by the battery pack's state of charge:

$$\mathbf{x^0} = [eTorque, eSpeed, SOC]$$
$$\mathbf{y^0} = [current]$$

The *eTorque* and *current* values are both negative as they represent torque induced into the motor, and current entering the battery. The efficiency rating is derived as:

$$effRegenFlow_e = \frac{current}{tRegenFlow_e(eTorque, eSpeed)}$$

to obtain a new the feature vector and response relationship:

$$\mathbf{x^0} = [eTorque, eSpeed, SOC]$$
$$\mathbf{y^0} = [effRegenFlow_e]$$

The input attributes are mapped to vectors of the design matrix as before:

$$\mathbf{x} = toPoly([eT, eS, SOC]) = \begin{bmatrix} 1 & eT & eS & SOC & eT & eT^2 & eS^2 & SOC^2 \dots \end{bmatrix}$$

9 Methodologies: Searching Solutions

9.1 Simulated Solution

9.1.1 Motivation

A simulated solution simulates the fashion in which a nowadays hybrid vehicle would complete an itinerary, hence providing a reference cost to which other solutions can be compared during testing.

Manufacturers of such systems do not fully disclose the precise behaviour of such engine management systems. However, reports seem to suggest these manage the IC engine and induction motor utilising a set of fuel-optimisation pragmas, which select the best known configuration for the ongoing power demands and energy state of the vehicle. In terms of the system described in this report, this translates to selecting for each *RoadUnit* the *pum* which maximises the assist of the induction motor when possible, otherwise falls back to a pure internal combustion mode.

9.1.2 Modelling And Implementation

The routine keeps track of the energy state of the vehicle (fuel and charge) and on each *RoadUnit*:

1. Chooses the most efficient available *pum* between: a full internal combustion mode, and one which maximises the use of the electric motor (*PUMFullECompensation*); or full electric if evaluating a braking zone.

2. Updates the vehicle's energy state based on the *pum* selected.

```
simulate(v,it,buff)
    # Extract the next road unit in the itinerary.
    ru = viewHead(it)

    case ru of

        # Itinerary is completed: return and exit.
        NULL -> reverse(buff)

        # The vehicle is required to brake in this section.
        BRAKING -> let brakingConfig = getBrakingConfig(v,ru)
                   in simulate(v.update(brakingConfig),tail(it),queue(brakingConfig,buff))

        # Acceleration or steady pace.
        PICK_OR_PACE ->
            let pumOpts = [PUMFullECompensation, PUMPureICEngine]
             in case evalPumInRoadUnit(v,ru,pumOpts) of
                    # No option: vehicle is out of charge or petrol.
                    [] -> error(v,ru)
                    # One or more valid option: select the most efficient.
                    validOpts ->
                        let bestConf = minCostConfig(validOpts)
                         in simulate(v.update(bestConf),tail(it),queue(bestConf,buff))
```

9.2 Genetic Search

9.2.1 Creating The First Generation

The first generation is a critical factor in the performance of a genetic search. This must have enough variability to reduce the probability of converging towards local minima, whilst the average proximity of the genes to the global minima may be a direct indicator of the number of generations required to reach a satisfiable solution.
In this section it is presented one approach to injecting a level of randomness into a solution, whilst still ensuring this is valid and purposeful.

Limitations To Randomness Levels In Genes

Firstly, it is important to present the main intuitions that discourage the composition of a solution/gene (PUC) by a random selection of *pum*s:

- There are physical limitations to this approach. In fact only a subset of all possible *pum*s allow the *pu* to develop the power demand configuration of a $RoadUnit$.
- The energy state of the vehicle also limits the available *pum*s, ie. insufficient of current and/or petrol.
- Random PUC contribute to unnecessarily high levels of variability in the generation. This idea revolves around what is understood to be a 'purposeful' solution: one which only makes the required amount of efficiency sacrifices, in one or more $RoadUnit$s. This means that such a solution is expected to always make full use of the most economical form of propulsion (nowadays electric), and only renounce to it, when it is physically not available. A fully random selection would lead to the construction of solution which

are very likely to be far from this ideology, hence unnecessarily increasing convergence times.

Design And Implementation

The first generation requires a set of PUC which make a random amount of sacrifices at random sections of an itinerary. To achieve this, it is possible to extend a greedy search *(greedy approach at F)*, to not perform a greedy step with some probability. In fact, the greedy solution is known to be the optimal PUC, which can be made invalid only by the vehicle's lack of fuel and/or charge. Allowing this routine to not select the optimal step for some levels of the problem, translates to injecting sacrifices into the invalid solution, hence increasing the probability of it becoming valid.

This task can be completed inductively in linear time to the length of the $Itinerary$. As arguments the function also accepts the probability of making a sacrifice. This is chosen to be a random value between zero and the length of the itinerary. This range enables the caller of the function to control on how many $RoadUnit$s the gene may consider not performing a greedy step.

```
generateGene(v,it,p,buff) -> Gene
```

The routine follows the following logic:

1. Terminates whenever the itinerary is empty.

```
generateGene(v,it,p,buff) -> Gene or NULL
    ru = viewHead(it)

    case ru of
        NULL -> Gene(buff)
```

2. If the $RoadUnit$ being analysed is a braking zone, the pum is pure electric.

```
        BRAKING ->
            let brakingConfig = getBrakingConfig(v,ru)
             in generateGene(v.update(brakingConfig),tail(it),p,queue(brakingConfig,buff))
```

3. If the $RoadUnit$ is a pace or acceleration zone, then the function first calculates the subset of pum which can be used to complete the $RoadUnit$. If these are empty the function exits with an error: the vehicle was brought to an energy state insufficient to move any further in the itinerary. *(Note: unlike in this pseudocode, the Haskell implementation of this project uses cached evaluations).*

```
        PICK_OR_PACE ->
            case evalPumInRoadUnit(v,ru,_PUM_ALL) of
                [] -> return null
```

4. If the pums available are only one, then the gene is forced to continue with it.

```
                [pum] ->
                    let nextConfig = getPickPaceConfig(v,ru,opt)
                    in generateGene(v.update(nextConfig),tail(it),p,queue(nextConfig,buff))
```

5. Finally, when there exist more than one *pum* options, the function identifies the most efficient one (which represents the greedy option) and chooses a second option at random.

```
pums ->
    let bestOpt = minCostConfig(pums)
        randOpt = pickAnyFrom(pums\bestOpt)
```

6. Hence picks the random choice based on p:

```
letIt = length(it) # Length of remaining portion.
weightedOptions = [(bestOpt,lenIt),(randOpt,p)]

nextPum = pickAnyFromWeightedList(weightedOptions)
nextConfig = getPickPaceConfig(v,ru,nextPum)
```

7. Before recurring on the remainder of the itinerary, the p value is decrease by a random amount of maximum 1, iff the options selected was a random one. This is to ensure the probability of making a random choice scales with the decreasing length of the itinerary. On the other hand, if in this road unit the greedy step was made, p remains unchanged, so to increase the probability of it occurring on one of the next.

```
p' = if nextPum == bestOpt
     then p
     else max(0, p-randInRange(0,1))

in generateGene(v.update(nextConfig),tail(it),p',
                                        queue(nextConfig,buff))
```

The creation of a first generation sees the invocation of *generateGene* function. In order to meet the desired population size, the routine needs to consider the possibility of a *generateGene* failing.

9.2.2 Evolving A Generation

Selection

The fitness function of a *PUC* (a gene) is its total cost computed by *PUC_Cost*, scaled in context of the fitness of the whole generation, though sigma scaling:

$$toSigmaScale(\mathbf{c}) = \frac{c \times \mu(\mathbf{c_0})}{2 \times \sigma(\mathbf{c_0})} \; \forall c \in \mathbf{c_0}$$

$$| \; \mathbf{c} = \text{vector of } PUC \text{ fitness costs}$$

$$\mathbf{c_0} = \text{vector of inverted } PUC \text{ fitness costs}$$

Gene Repair

Gene repair can be done deterministically in linear time by substituting any invalid *pum* in the new gene with the best option available at that stage.
The function accepts, apart from the vehicle and the itinerary, the sequence of *pum*s resulting from the crossover and mutation step.

```
geneRepair(v,it,pums,buff) -> [pum] or NULl
```

The process finishes successfully when the itinerary is exhausted.

```
    case viewHead(it) of
        NULL -> return buff
```

If the genetically selected *pum* for the next road unit allows for a valid *pum*, then this choice is retained and the repair moves to the next *RoadUnit*.

```
        ru ->
            let dpum = viewHead(pums)
             in case getConfig(v,ru,dpum) of
                    c -> geneRepair(v.update(c),tail(it),tail(pums),queue(dpum,buff))
```

On the contrary, the routine replaces the genetic *pum* with the most efficient *pum* that can be applied at this time in this *RoadUnit*.

```
                    NULL ->
                        case evalPumInRoadUnit(v,ru,PUM ALL) of
                            # No option.
                            [] -> return NULL

                            # One or more options
                            ps ->
                                let bestP = minCostConfig(ps)
                                    c = getConfig(v,ru,bestP)
                                 in geneRepair(v.update(c),tail(it),tail(pums),queue(p,buff)
```

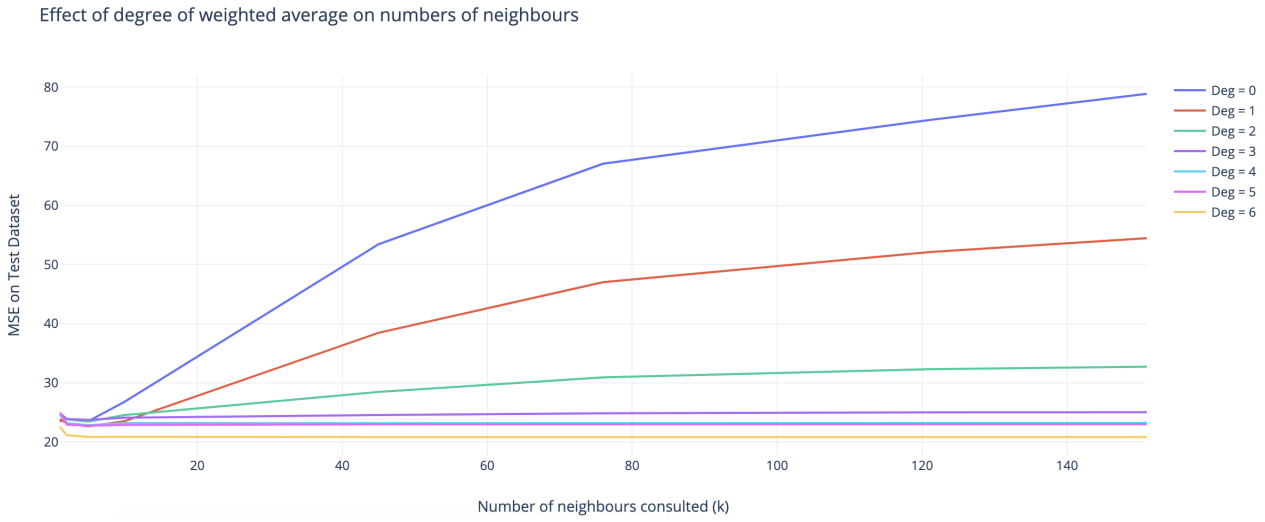
10 Testing and Results

10.1 K-Nearest-Neighbour Regression

Number of Neighbours and Weighted Average

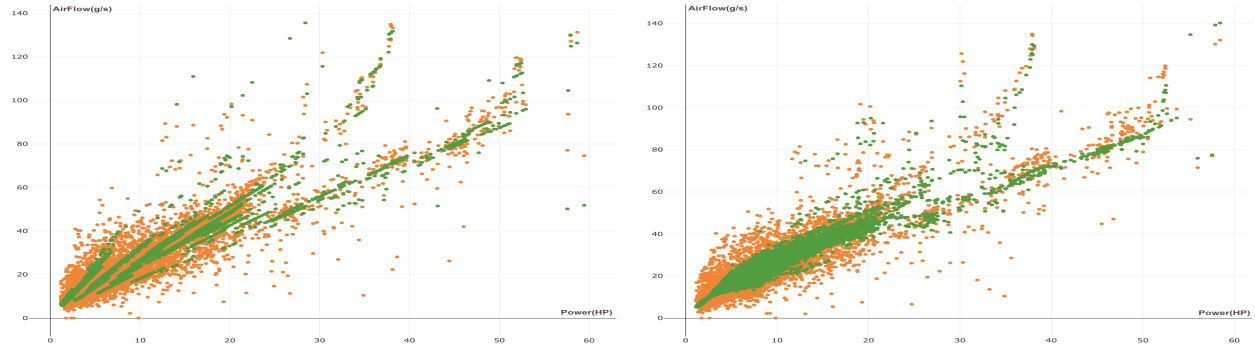
The use of a weighted average reduces the sensitivity to the number of neighbours consulted (k) as it increased the tolerance towards the inclusion of low affinity data points.

Figure 4: Effect of degree of weighted average on number of neighbours (k).



A model which selects the single closet neighbour, and one which selects multiple but with a larger average degree, although leading to similar MSE values, do resemble different prediction behaviour. As expected, a low k model tends to depend the density of the KNN dataset, in order to predict accurate values; as it is unable to weight the affinity that a query might have with telemetry points that are not only among the k selected, but also those which were excluded but otherwise still relevant.
On the other hand, a weighted average is one step closer to being able to fill the gaps in the dataset, by providing control over the shape of the affinity a query has with any number of neighbours based on their distance.
This point is easily portrait into a plot of some unseen queries, about their power (here function fo $eTorque$ and $eSpeed$ in hp) and the required air-flow (g/s):

Figure 5: Effects of weighted average on predicting unseen query points. In orange the real data point, in green the predictions.

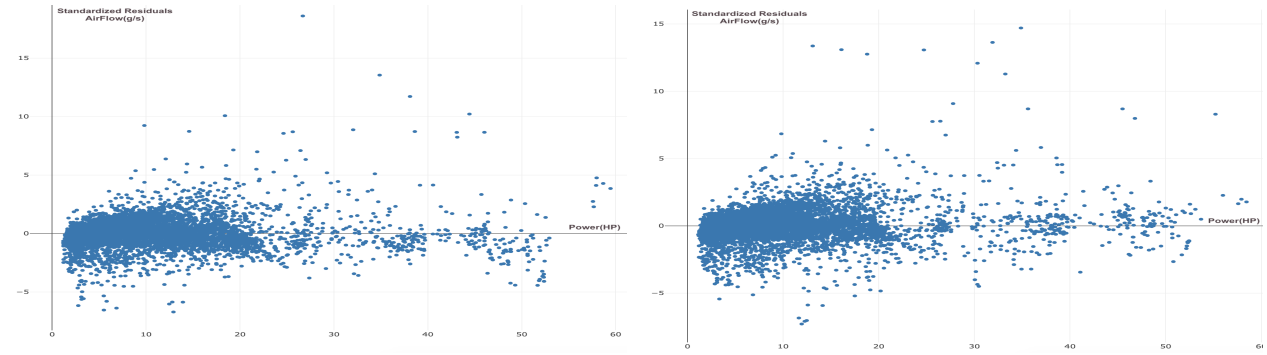


(a) Power Vs Air-Flow: prediction using low k value and no weighted average.

(b) Power Vs Air-Flow: prediction using high k and average degree value.

The latter approach can also provide some level of control over overfit, as having the ability to select a larger number or k and a suitable degree of weighted average, can help drawing the final result away from an overfitting telemetry point.
Shifting the attention towards the scaled residuals of the two models, the use of a weighted average seems to be able to reduce the downward pattern of the model utilising a low k. This is mostly due to the fact that the original dataset is less dense around such engine configurations, leading the natural average averaging with points far away from the query, which tend to have higher efficiency.

Figure 6: Effects of weighted average: standardised residuals.



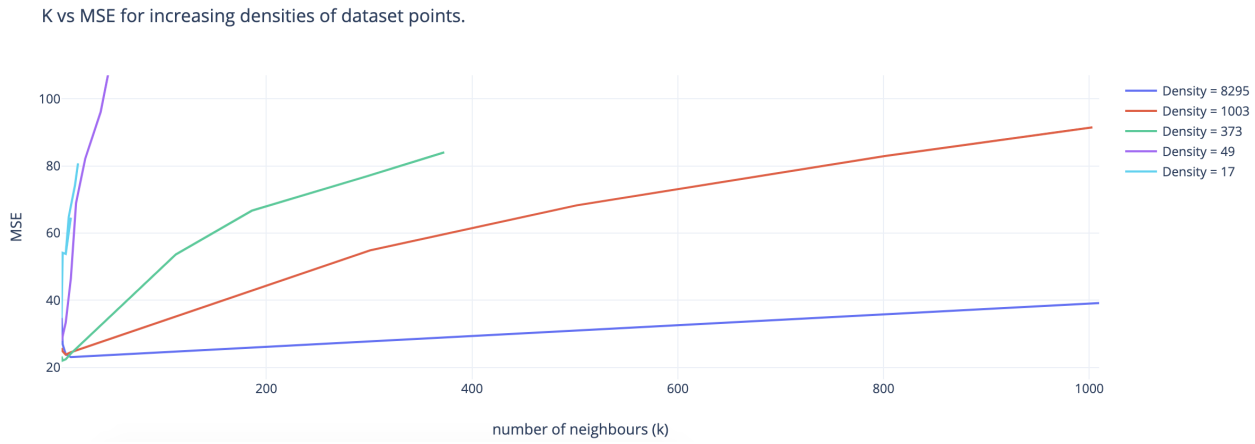
(a) Power Vs Air-Flow standardised residuals: prediction using low k value (5) and no weighted average.

(b) Power Vs Air-Flow standardised residuals: prediction using k of 10 and average degree value of 3.

Effects of Points Density Control

Density Control is the feature of this implementation of KNN Regression, which allows the dataset to accumulate the average of similar vectors into a single node. As deducible, choosing the number of neighbours where the weighted average is not utilised, becomes less critical with higher density the datasets: higher nodes density leads to lower probability of the closest k neighbours to include one or more neighbours with low affinity to the query.

Figure 7: Number of consulted neighbours with natural average vs MSE of unseen data queries. The higher the dataset density, the slower increasing values of k reduce the average model's accuracy.



Increasing the density of the dataset did not however always translate to higher model accuracy. This proved to be the first symptom indulging the density assumptions made earlier: averaging similar feature vectors to a single vector does not necessarily lead to a less accurate model, as engine efficiency is not expected to spike. On the other hand, the inability to decrease the MSE with higher density datasets, may also suggest that more features might be required in order to capture minor changes in the engines' efficiencies; yet for the purpose of this system, no further features can be utilised.

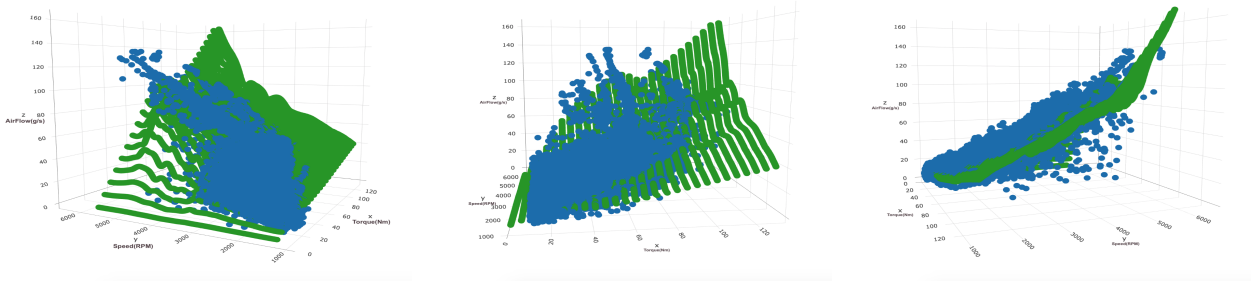
Adequacy To Telemetry Data.

This approach is capable of computing accurate estimates to queries lying within the range of values used during the learning phase. However, if the telemetry provided happens to either be significantly scarce, or lacking of entries for large sections of allowed engine configurations, this can easily lead the system querying such model to dramatic changes in behaviour.
One such telemetry might be one only consisting of engine configurations with low engine speeds, which from literature review tend to be more efficient. This would lead a model to classify configurations with higher engine speeds to more efficient estimates, hence possibly inducing the system to prefer higher gear ratios.
It is therefore important to note that a model for the prediction of $expFlow_{p||e}$ operating on this implementation of KNN Regression, requires a minimum level of completeness of the telemetry supplied; with completeness it is intended am homogeneous sequence of values in range of each of the features which lead to highest variance of the residuals (such as $eTorque$ and $eSpeed$).

Internal Combustion Engines

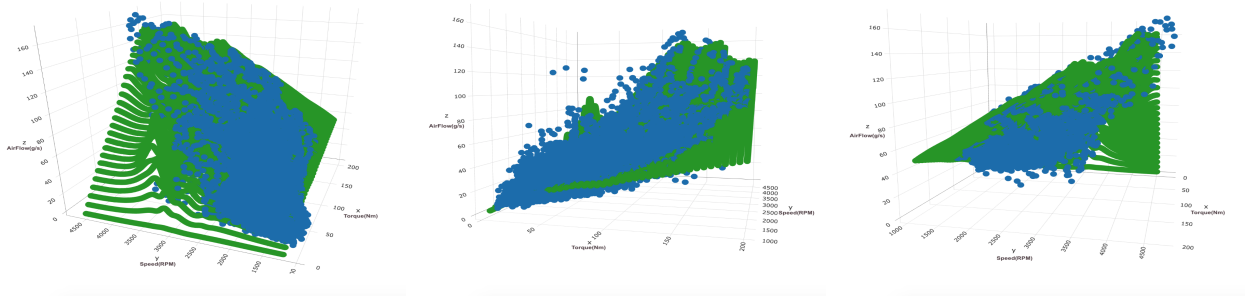
This implementation of KNN Regression was tested on the vehicle telemetry of a Fiat 500 and a Ford Fusion. The models produced responded positively to the claims made. The learning of engine efficiencies means that not only the model can predict air and fuel-flow for seen configurations, but also produce reliable predictions of unseen configurations by generalising the engine's efficiency curve.

Figure 8: Fiat 500, KNN Regression.
In blue training data points, in green the prediction of all engine's configurations at fixed ambient temperature



Models obtained from the Ford's telemetry, better address the claim made earlier about dataset completeness, as these plots highlight, the predicted air-flow(z *axis*) for configurations with low torque and high engine speed, diverges considerably from the general curve of the engine, towards a downward trend; meaning this area of configurations will appear as more efficient, that they would be.

Figure 9: Ford Fusion V6, KNN Regression.
In blue training data points, in green the prediction of all engine's configurations at fixed ambient temperature.



Induction Motors: Power Mode

The approach reviled suitable also for the regression on induction motors tested: a Nissan Leaf and a BMW i3. As for the IC engines' models, the features mapped during the learning process deliver a high level of confidence for the estimation of the motor's efficiency even in low density control settings.

Figure 10: Nissan Leaf, Power Mode, KNN Regression.
In blue training data points, in green the prediction of all engine's configuration at fixed ambient temperature.

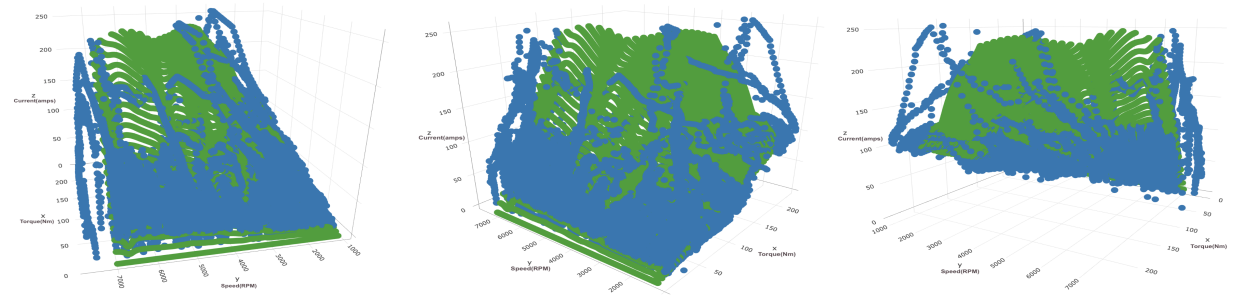
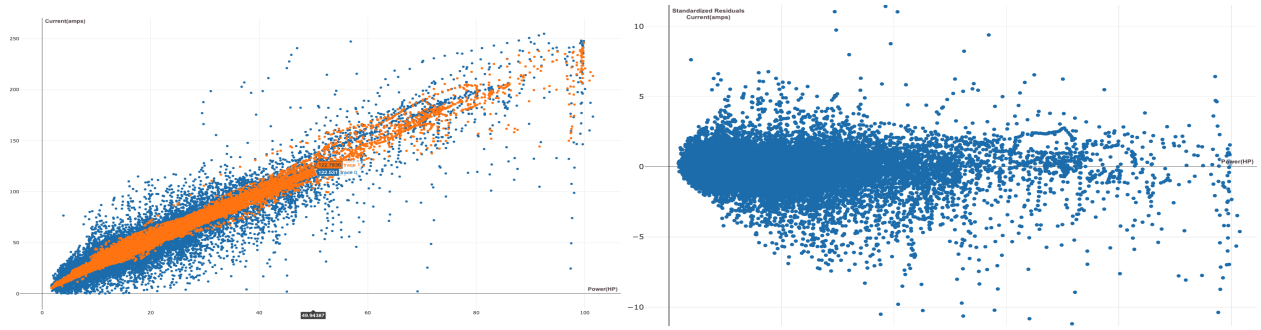
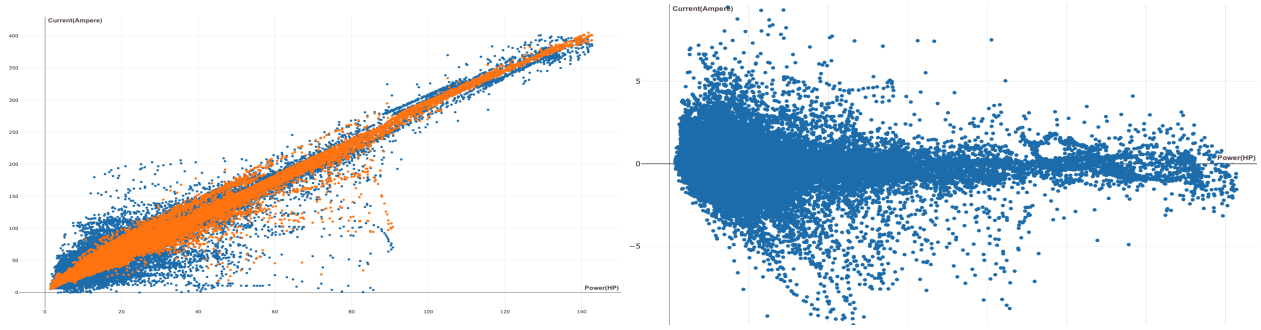


Figure 11: Nissan Leaf, Power Mode, KNN Regression



(a) Power(HP) vs. Current(Ampere) (b) Standardised residuals.

Figure 12: BMW i3, Power Mode, KNN Regression

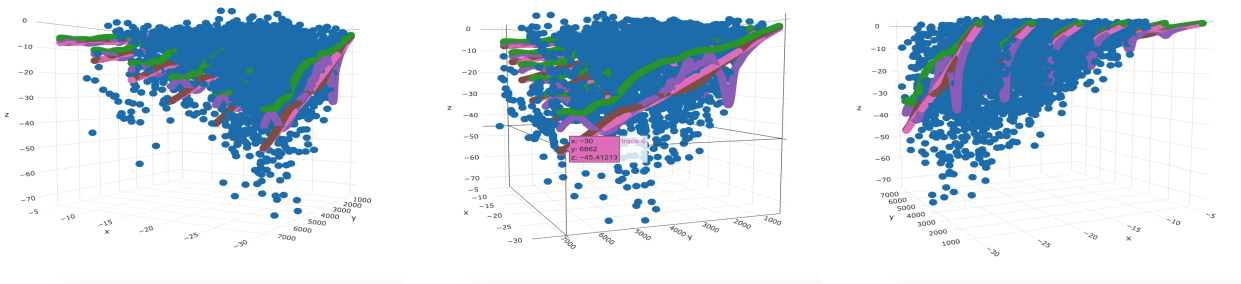


(a) Power(HP) vs. Current(Ampere) (b) Standardised residuals.

Induction Motors: Harvest Mode

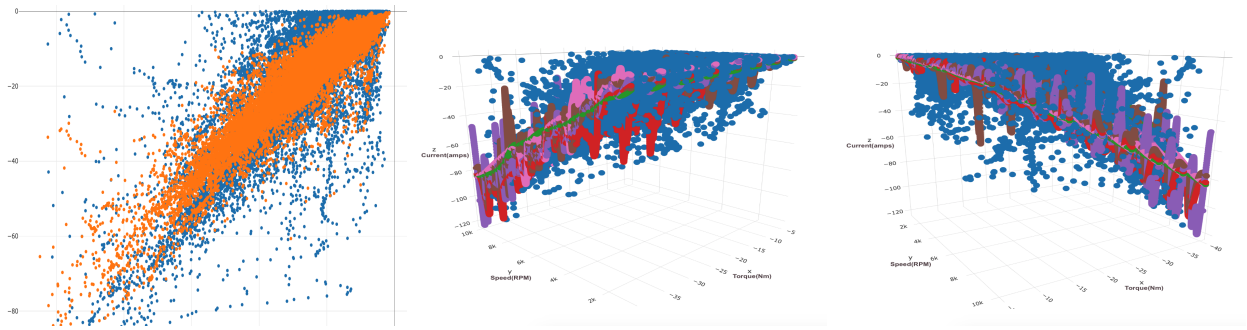
Testing on the regenerative telemetry of the Nissan Leaf and BMWi3, yield reasonable results. The inclusion of SOC in the knn nodes proved to be a valuable addition to the torque and motor speed as can be seen in the harvesting telemetry of the Nissan Leaf:

Figure 13: Nissan Leaf, Regenerative Braking, KNN Regression.
In blue training data points, in colours the harvest current at varying levels of SOC.



With regards to the BMW i3 data, the motor seemed to be a lot less consistent, presenting more variance in the current harvested for similar configurations.

Figure 14: BMWi3, Regenerative Braking, KNN Regression.
In blue training data points, in colours the harvest current at varying levels of SOC.



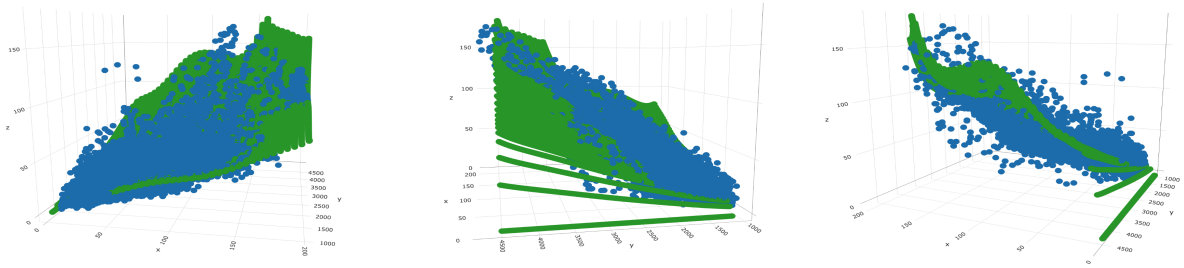
10.2 Multivariate Polynomial Regression

Multivariate polynomial regression proved to be a competent approach for the modelling of engine's efficiency curve. Compared to KNN Regression, it is a high maintenance model, which can be sensitive to the completeness of the telemetry provided; meaning the data must be rich enough to provide to each regressor insight of the linearity of each feature with the result (which is to be expected).
This section includes a brief account of what regression models can be achieved on the same telemetry/vehicles previously presented in the section about KNN Regression.

Internal Combustion Engines

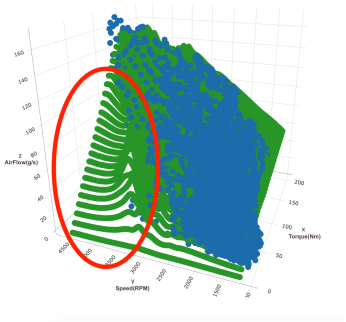
The IC engines analysed proved to have significant correlation of their efficiency with regards to the power being produced: the addition of the power feature (function of torque and speed) lead to dramatic improvements. An example of the results are the Ford Fusion achieved with a polynomial degree of 4:

Figure 15: Ford Fusion V6, Multivariate Polynomial Regression.
In green the prediction of all engine's configurations at fixed ambient temperature.

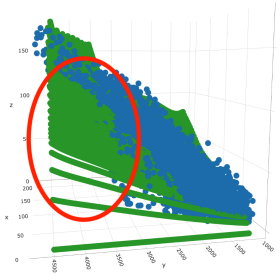


The fitting is similar to the result obtained with KNN Regression, but with a clear improvement about unseen areas of the telemetry. In fact, the main week point of knn regression about engine configurations outside the learning rage, here is being addressed:

Figure 16: Ford Fusion V6, KNN vs Multivariate Polynomial Regression.



(a) KNN Regression

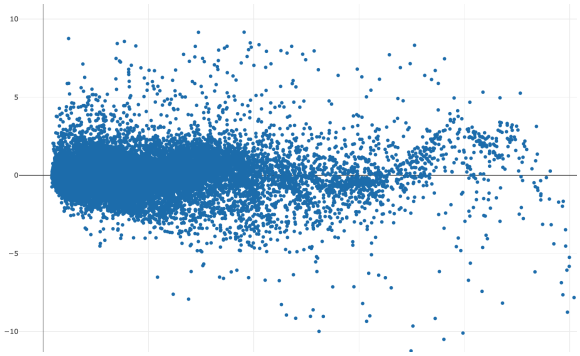


(b) Multivariate Polynomial Regression

The unseen engine configurations are now set to scale as the general curve of the engine efficiency, which, from background literature, seems the most appropriate assumption.

The telemetry dataset being scarce in some areas of engine configuration does lead to the model generalising excessively trends presented in other more dense areas. For example, renouncing to include high power modes in the telemetry for the Ford, leads to the following scaled residuals plot:

Figure 17: Ford Fusion V6, Multivariate Polynomial Regression, Residual Plots.



For the most part, the residuals are normally distributed, which confirms the engine configuration being linear to the engine efficiency. However, towards the end of the plot, where lay configurations with higher output power, the residuals present a mild trend.

Induction Motors: Power Mode

The regression is also capable to produce competitive models for the modelling of energy efficiency of induction motors. These, like IC engines, benefitted of the introduction of the power feature.

Figure 18: BMW i3, Multivariate Polynomial Regression.
In blue training data points, in green the prediction of all engine's configuration at fixed ambient temperature

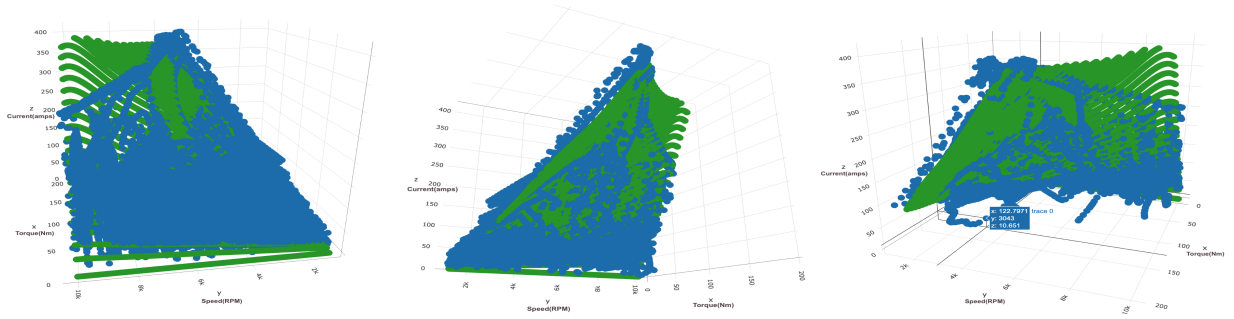
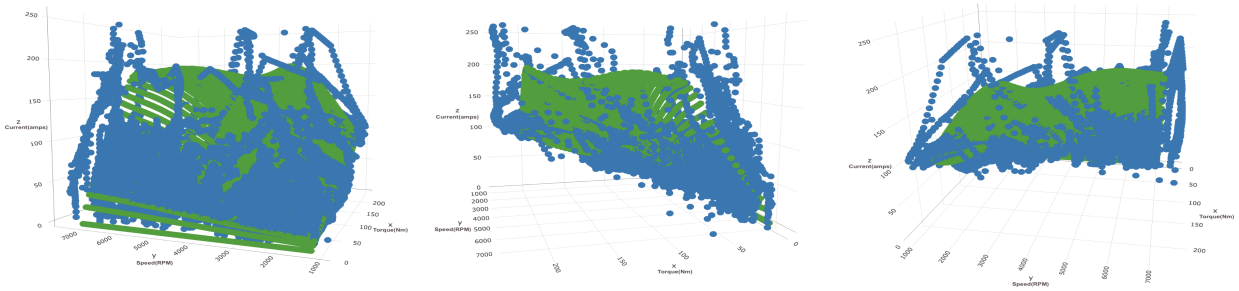
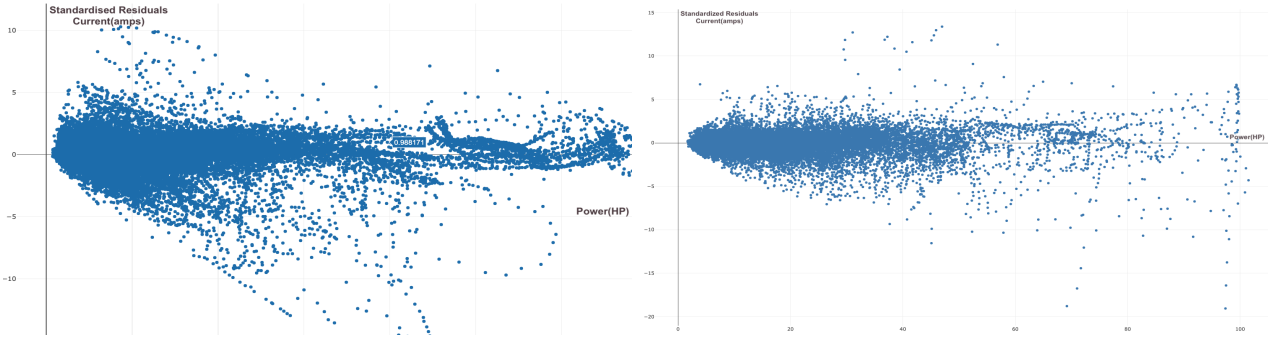


Figure 19: NissanLeaf, Multivariate Polynomial Regression.
In blue training data points, in green the prediction of all engine's configuration at fixed ambient temperature



The models obtained did not present significant trends in their residuals, especially in the case of the Nissan Leaf.



(a) BMW i3, Multivariate Polynomial Regression, Residual Plots.

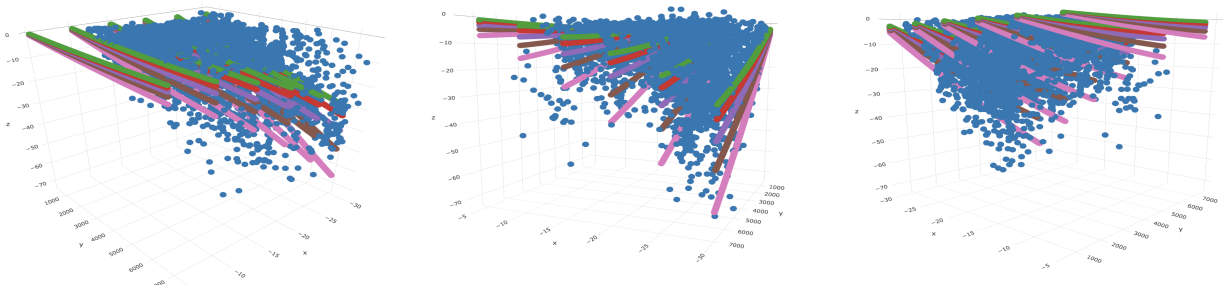
(b) Nissan Leaf, Multivariate Polynomial Regression, Residual Plots.

Induction Motors: Harvest Mode

Regression models for the learning of regenerative currents can be used although these do highlight some, although negligible in this context, increase in variance for higher harvesting configurations.

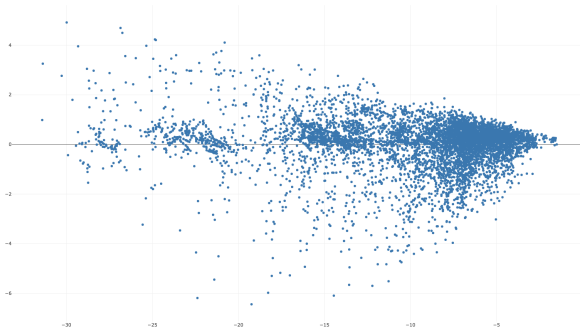
Here follows a model based of one degree polynomial for the Nissan Leaf:

Figure 21: NissanLeaf, Regenerative Braking, Multivariate Regression.
In blue training data points, in colours the harvest current at varying levels of SOC.



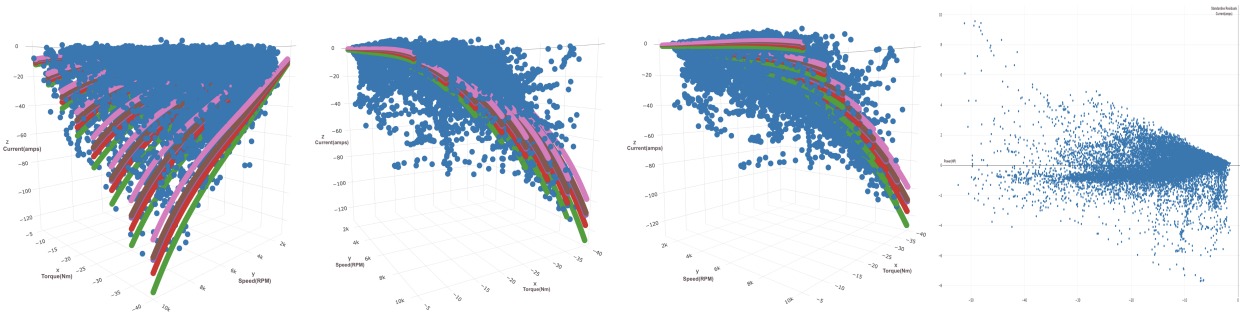
The resulting model at first sight appears to be a smoothened predictor of its Knn Regression equivalent. However, plotting the standardises residuals highlights a minor increase in variance for a smaller number of entries. This suggests that those entries cannot in fact be explained by linear correlation to the expected values. That being said this trend is well contained in the 95^{th} percentile of the distribution, and in smaller density compared to those lying on the mean.

Figure 22: Nissan Leaf, Regenerative Braking, Multivariate Regression, Residual Plots.



A similar argument can be made about the regression model for the BMW i3:

Figure 23: BMW i3, Regenerative Braking, Multivariate Regression.



10.3 Genetic Search

Problems with trivial corrections.

These are problems which, although cannot be solved trivially by the greedy algorithm, still require a solution characterised by an even distribution of minor sacrifices. Hence, an itinerary which could be executed entirely by the greedy solution, if not for a small number of road units.

*Problem*1 is one of such problems, in which a low powered electric motor cannot contribute to the most efficient regimes for the entirety of the itinerary, however it could be if a few minor sacrifices were made. To this problem, a simulated solution amounts to $1.73. Each genetic configuration was given 30 seconds to find a better solution; the best solution found amounted to $0.62.
The triviality of the problem promoted the convergence of most configurations towards a good solutions, however:

- Higher generation size means faster convergence and, on average, better results.
- Zero mutation limits the solution to the variety of genes in the first generation.
- High mutation leads to slower convergence and no gain in result quality.
- Multiple crossover can work in such problems, however being a lot less consistent than its single point variant.
- Multiple crossover tends to maintain a lower standard deviation in fitness of its genes, which prejudices its performance it less trivial problem.

Figure 24: *Problem*1: Single and Multi point crossover for generation of 30% and 100% size: generation number vs generation fitness.

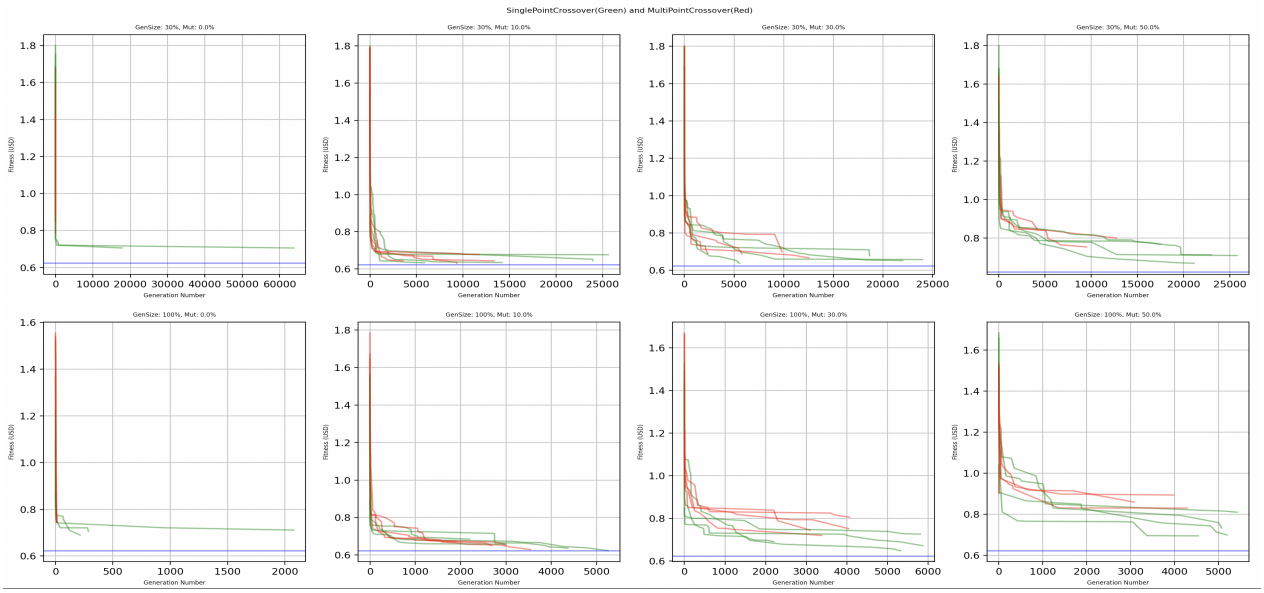
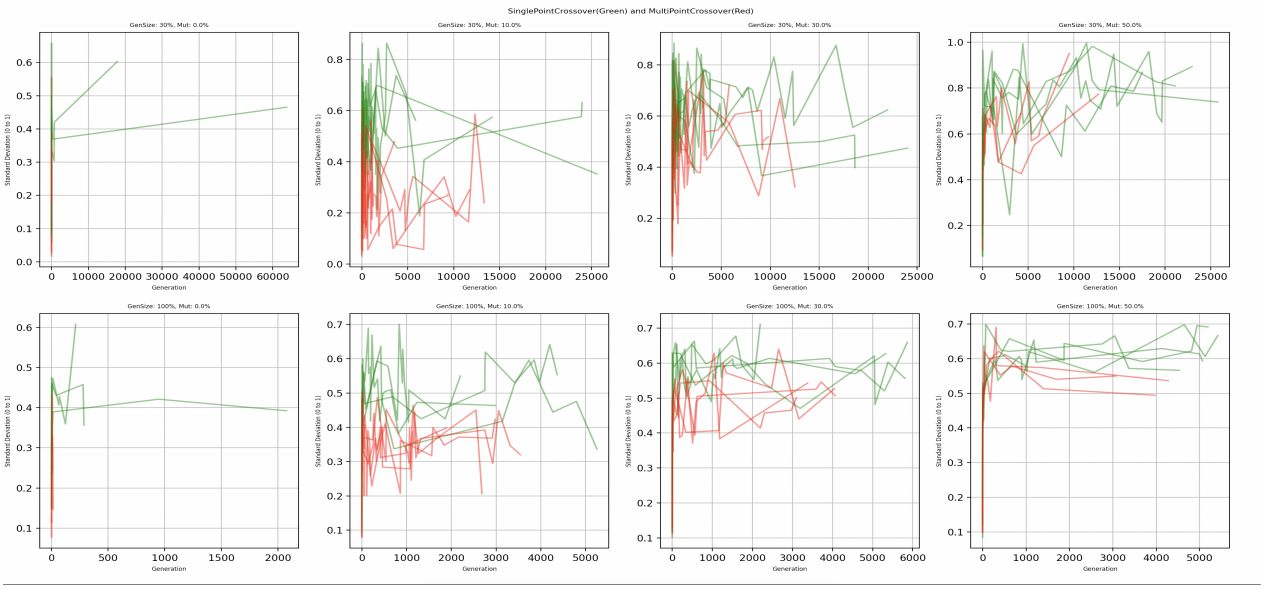
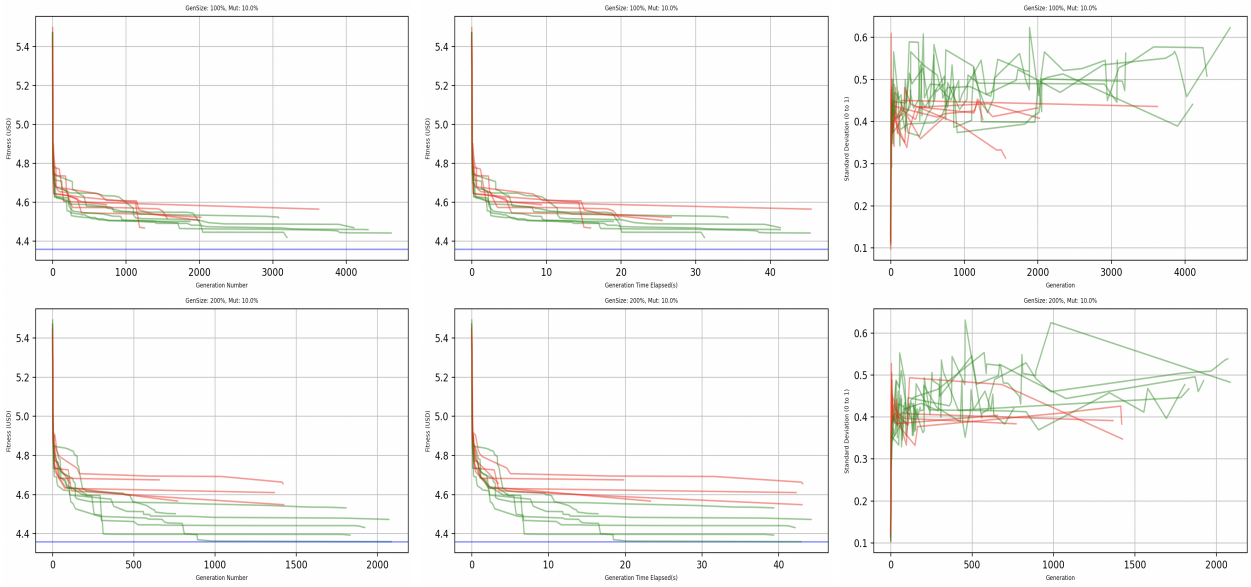


Figure 25: *Problem*1: Single and Multi point crossover for generation of 30% and 100% size: generation number vs standard deviation in generation's genes (0-1).



*Problem*2 is extends *Problem*1 with a longer itinerary. This means more subtle sacrifices are required to compose a valid solution. For this problem, the simulated cost is of $5.25; genetic solutions were consistently below $4.46 after 10 seconds of evaluation, with a lowest of $4.35.

Figure 26: *Problem*2 results, Single and Multi point crossover



(a) Generation number vs fitness (USD).

(b) Time elapsed (seconds) vs fitness (USD).

(c) Generation number vs generation's standard deviation (0-1).

These results confirm the observations of *Problem*1, however strengthening the superiority

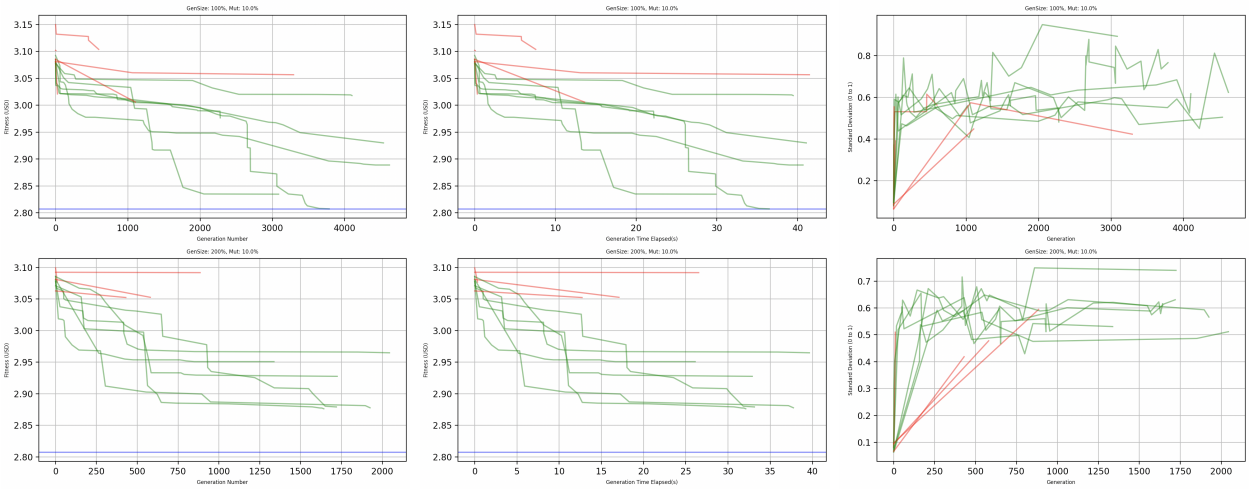
of single point crossover, no only in average quality of the results, but also in average variance of the genes across generations, smoother convergence curves, and better ability of utilising larger generation sizes. The reason why multiple crossover is able to perform comparably to single point crossover is mainly due to the nature of this problem instances. In fact, these require minor and evenly distributed sacrifices, which means the probability of all gene sharing similar solution patterns is higher than in other cases.

Problems with non trivial corrections.

Non trivial solutions can be induced in a problem through two main scenarios: high performance induction motors, and highly insufficient electrical charge.
A vehicle featuring an induction motor capable of substituting the IC engine in most scenarios, leads to a problem formulation presenting more opportunity of cost minimisation, as the thermic engine can be expected to perform less work. In these scenarios, the fitness cost of the genes can spike as a number of power unit configurations are moved to full electric: a critical test for the GA search, as it is required to maintain a high generation variance to avoid converging too quickly towards a local minima.

*Problem*3 is an instance of such problems: it modifies the problem statement of *Problem*2 to have an induction motor as powerful as the IC engine, and a higher initial charge. The simulated solution now drops to $3.22, the genetics consistently lead to values below $3.00, with a lowest of $2.81.

Figure 27: *Problem*3 results, Single and Multi point crossover



(a) Generation number vs fitness (USD).

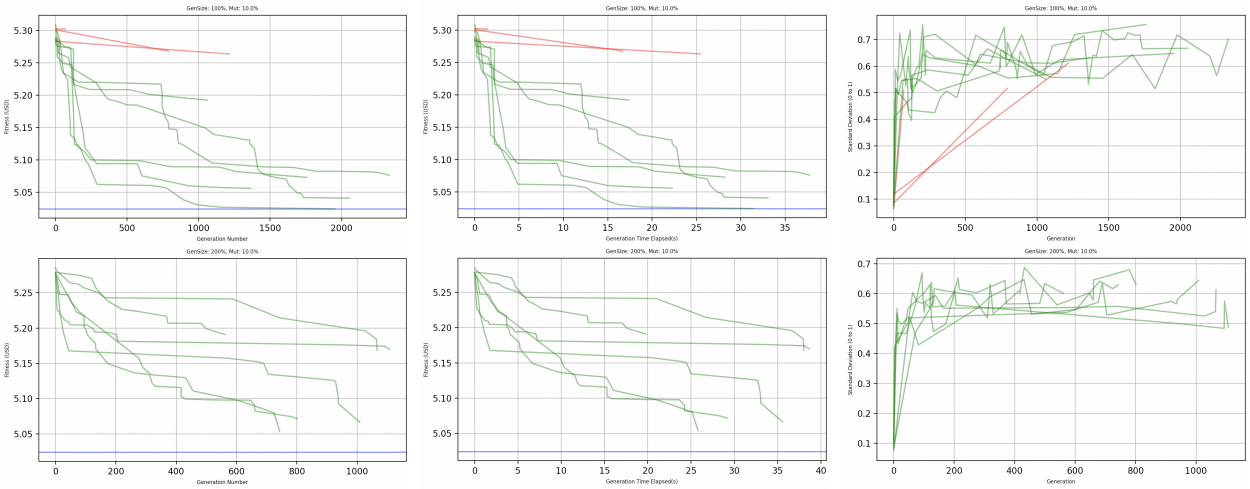
(b) Time elapsed (seconds) vs fitness (USD).

(c) Generation number vs generation's standard deviation (0-1).

The results confirm the points raised earlier: compared to single point crossover, the multi point variant becomes stuck around local minima, as it fails in maintaining a high generation variance.

*Problem*4 modifies *Problem*3 to include a longer itinerary. As margins of gains are slim, due to the increase in total energy demand, it is a good representation of the system optimising problem instances in which require larger amount of rework of the solution. The simulated solution for this problem is $5.37; GAs found a best solution of $5.02 (max 65 seconds):

Figure 28: *Problem*4 results, Single and Multi point crossover



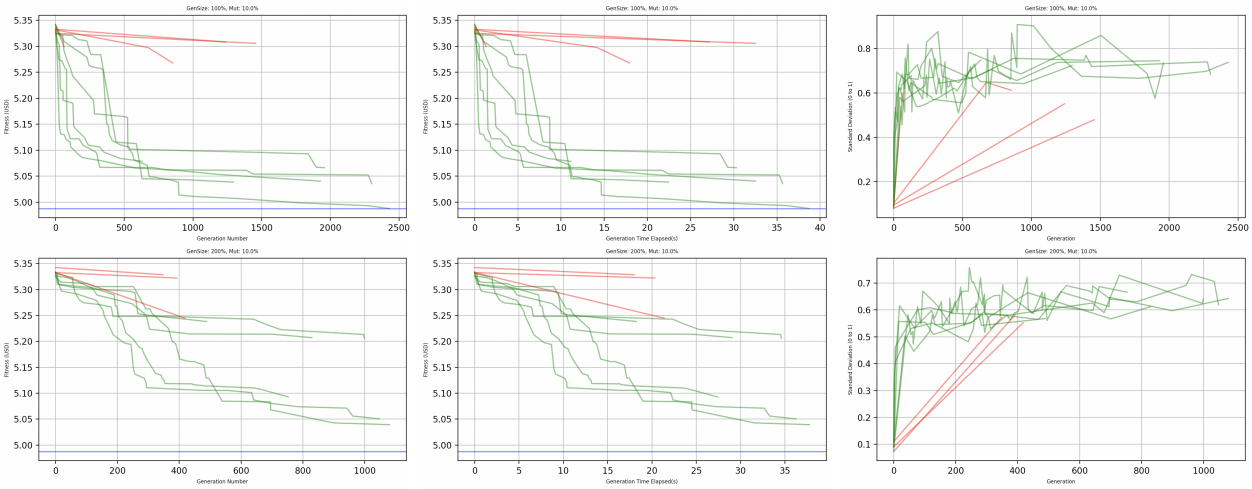
(a) Generation number vs fitness (USD).

(b) Time elapsed (seconds) vs fitness (USD).

(c) Generation number vs generation's standard deviation (0-1).

The triviality of the solution can also be decreased by enlarging the length of the itinerary, whilst maintaining a low SOC. *Problem*5 extends the itinerary length of *Problem*2, retaining the low initial charge and the induction motor as support role in the pu. Hence, the simulated solution raises to $6.30. During the allowed 65 seconds, GAs found a best solution of $4.99:

Figure 29: *Problem*5 results, Single and Multi point crossover



(a) Generation number vs fitness (USD).

(b) Time elapsed (seconds) vs fitness (USD).

(c) Generation number vs generation's standard deviation (0-1).

10.4 Average Effectiveness

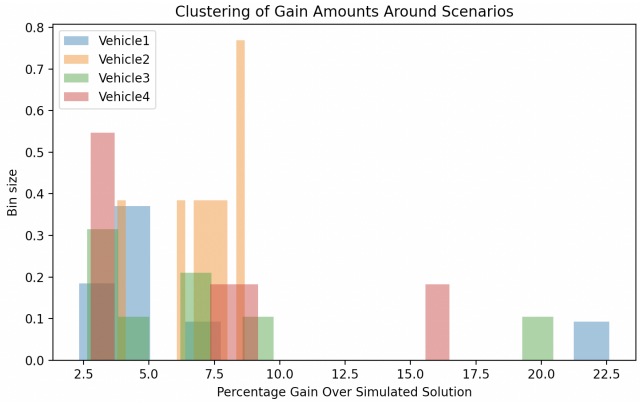
Scenario testing was conducted in order to estimate the average effectiveness of the system in a variety of

- Itineraries: long and short sessions of city driving (frequent decelerations, low top speed, short stints), backroads (continuous pace, average national speed), combined (moving from city driving to backroads and vice-versa), mountain sections follow and/or anticipated by city of combined driving.
- Vehicle Specifications: combinations of fictitious vehicles modelled by telemetry of: Fiat500, FordFusion V6, Nissan Leaf and BMW i3; with the electric engines capped at different max power and torque output (to model varying types of assist).
- Initial charge available.

The solutions were obtained by computing genetic search for a maximum of 2 minutes (on the same machine), configured to: construct an initial population of twice the number of *RoadUnit*s in the *Itinerary*, single point crossover, and a mutation chance of 1%; and compared to the *Simulated Solution* and the vehicle operating with no electrical assistance. The cost of fuel and electricity were kept to constant values of $1.55\frac{USD}{l}$ and $0.13\frac{USD}{kWh}$.

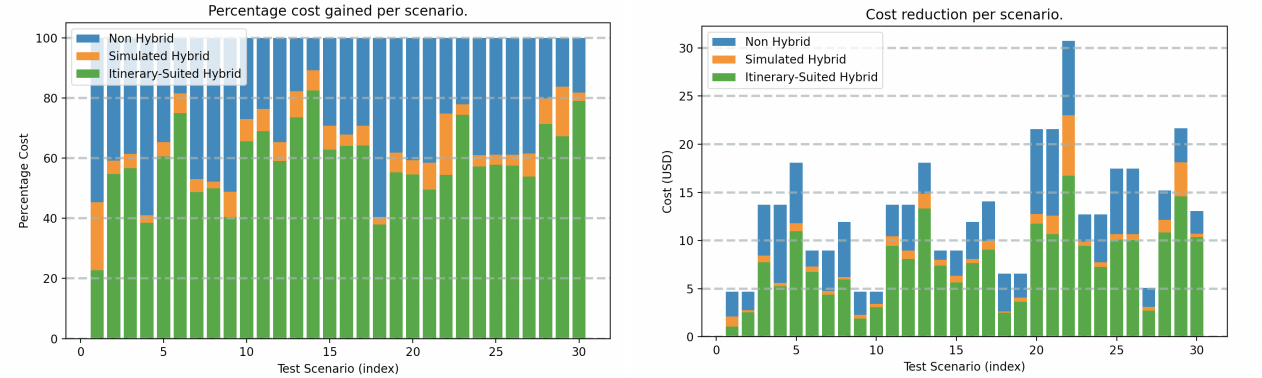
As expected, the margin of cost reduction proved to be mostly dependent on the level of uniformity of the power demands presented by the itinerary. In fact, although the average cost reduction amounted to 7%, with a maximum of 22%, the system performed best in combined conditions. This phenomenon can be viewed in an histogram, highlighting the clustering of smaller cost reductions across all 4 vehicle tested:

Figure 30: Itinerary specification causes average cost gains to cluster across vehicles.



This is due to the fact that, in uniform conditions it is more likely for the two engines to be performing similar amount of work for similar durations, hinting to a cyclical operation of the power unit. Hence the ability to allocate electrical assist for different areas of the itinerary is less effective: the system shifts to optimising the average consumption of the journey, as there are no spikes in operational costs; hence why it is still possible to reduce the costs in such circumstances.

Figure 31: Scenario testing, comparison per scenario of cost of completion through: no electrical assist, simulated hybrid, and scenario optimised hybrid.



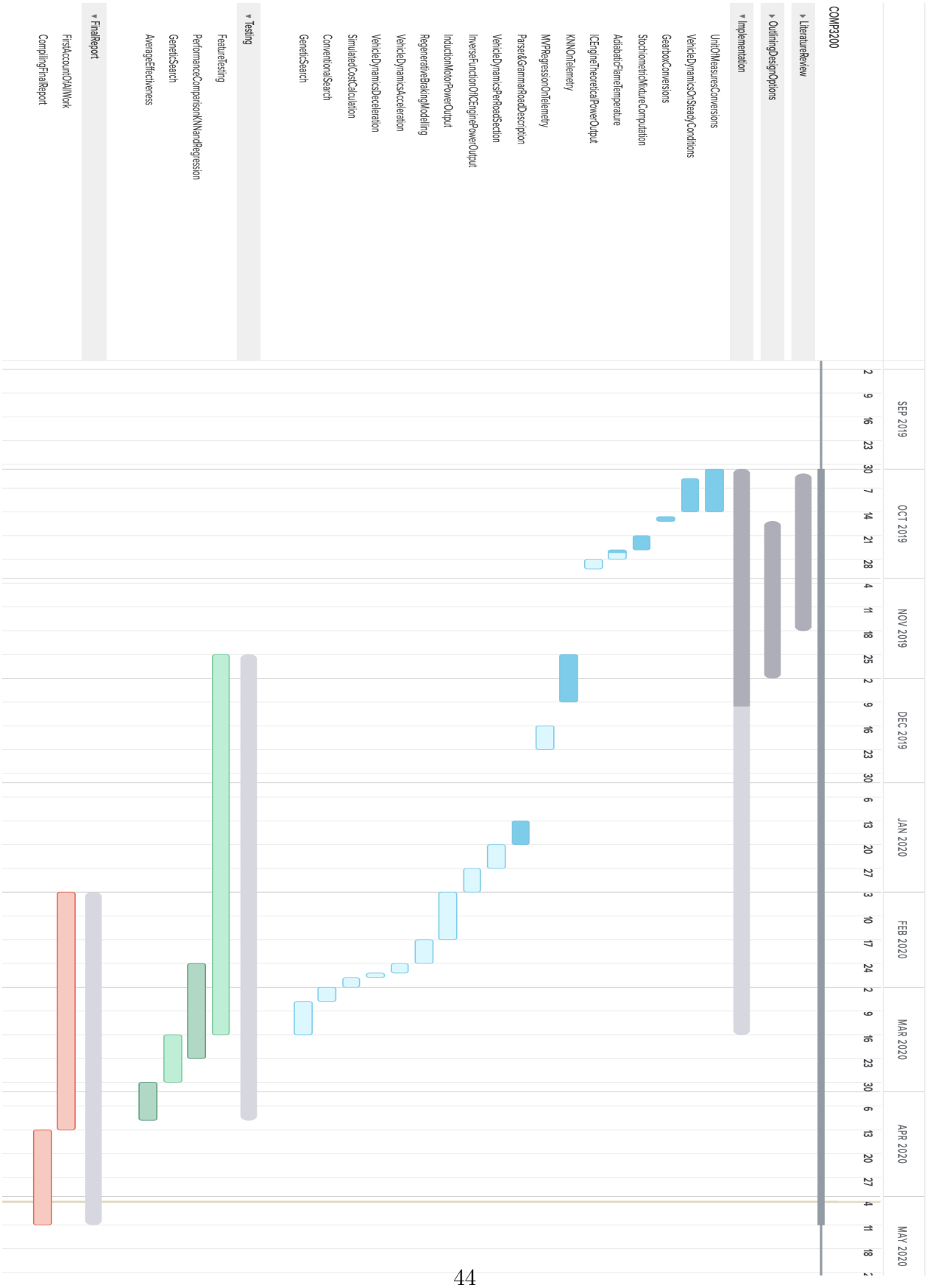
- It may be argued that the margin of gain could be larger, as the simulated solution is already performing a respectable level of optimisation in the operation of the power unit, especially regarding the acceleration and braking routines (optimal gearing).

11 Conclusions And Future Work

A system capable of improving fuel economy of hybrid power units by utilising information of the journey about to be undertaken was presented. The power factors imposed by the journey on the vehicle were modelled with a theoretical approach, which enabled for fine-grain estimation of the power unit's requirements. Two supervised machine learning of the engine's efficiency curves. The first, a KNN-Regression with density point control and weighed average; the second a multivariate polynomial regression. Both approaches were tested on dynamometer telemetry of a: Fiat 500, Ford Fusion V6, Nissan Leaf and BMW i3, created by Argonne National Laboratory. The use of dynamic programming for the task of computing the optimal sequence of power unit modes for a given itinerary, was discussed to be inefficient given the lack of overlapping subproblems. The computation of the problem search space was formalised by the introduction of a producer-consumer implementation of Depth First Search. It was however a genetic search to be deeply evaluated in this setting. Single and multiple point crossover were compared in a number of scenarios, as was generation size and mutation chance. A procedure for the generation of random but on-target first generation genes was motivated and formally described, as was a deterministic gene repair function. Finally, the cost effectiveness of the system was compared to the cost of a solution which mimics what it is understood to be today's management logics of hybrid vehicles. A grammar for the composition of itineraries was also introduced.
Results confirmed it is possible to improve the fuel economy of hybrid drivetrains by adapting to a journey the utilisation of thermic and induction engines. Although in the scenarios tested, an average gain of 7% was recorded, it is important to note that the cost effectiveness of the system can be larger. In fact, this varies upon the uniformity of the power requirement of the journey, hence preferring itineraries with combined conditions.
As this approach is capable (on a simulation level) to obtain better fuel economy whilst not scarifying a journey's desired velocities, accelerations rates, travel times or average speed, it seems appropriate to consider how its effectiveness could be further consolidated. One area of future work may be the ability to recalculate the solution whenever the expected efficiency is breached during the execution of the itinerary. The effectiveness of such feature however cannot be guaranteed at this stage. In fact, an engine's efficiency is not expected to deviate largely compared a substantial amount of its telemetry. The system could also be adapted to evaluate solutions in which refuelling/recharging is allowed at certain stages of the journey. Lastly, the simulation of the environment could be enriched to account for power unit operations which might want to be avoided for reasons such to components' wear and limits of operation.

Body's word count: 9996, copying from Apple Preview through clipboard to `www.wordcounter.io`.

12 Gantt Chart



A Development Environment

The development of the system was carried out in Haskell, and organised by the Haskell Stack build tool. External libraries utilised in the project are listed under every '$build - depends$:' tag in the '*.cabal*' file attached to this report. No library was utilised for the any modelling aspects, nor machine learning, nor searching aspects of the project; rather these consisted of: generic data structures, such as HashMap, Heap, HashSet, Matrix, Array, Text, Vector; 2D and 3D graph plotting tools; CSV parser interface; monadic random generators and selectors (utilised in the selection of genetic searching routine); shared resources management; descendants of Unix's Lec and YACC, Alex and Happy, were utilised for the definition of token and grammar of the *Itinerary* files (these are part of the nowadays Haskell Platform). Python and libraries '*matplotlib*' and '*pandas*' were used for the plotting of all search related graphs of the report only.
The general purpose libraries utilised in the Haskell implementation were pulled from the standard open source package manager for Haskell project Hackage (`https://hackage.haskell.org`). As of 11/May/2020 the resources utilised are found at:

- Generic Data Structures (hashing and non hashing based)
  - `https://hackage.haskell.org/package/array`
  - `https://hackage.haskell.org/package/vector`
  - `https://hackage.haskell.org/package/containers`
  - `https://hackage.haskell.org/package/unordered-containers`
  - `https://hackage.haskell.org/package/heap`
  - `https://hackage.haskell.org/package/matrix`
  - `https://hackage.haskell.org/package/hashable`
  - `https://hackage.haskell.org/package/scientific`
- Random Generators and Shufflers
  - `https://hackage.haskell.org/package/random`
  - `https://hackage.haskell.org/package/random-shuffle`
  - `https://hackage.haskell.org/package/MonadRandom`
- Text formats manipulation and CSV parser interface
  - `https://hackage.haskell.org/package/text`
  - `https://hackage.haskell.org/package/bytestring`
  - `https://hackage.haskell.org/package/utf8-string`
  - `https://hackage.haskell.org/package/cassava`
- Graph plotting support
  - `https://hackage.haskell.org/package/plotlyhs`
  - `https://hackage.haskell.org/package/aeson`
  - `https://hackage.haskell.org/package/lucid`
  - `https://hackage.haskell.org/package/microlens`
- Generic multithreading support

  - https://hackage.haskell.org/package/parallel
  - https://hackage.haskell.org/package/rio
  - https://hackage.haskell.org/package/deepseq

- Others
  - https://hackage.haskell.org/package/time
  - https://hackage.haskell.org/package/ansi-terminal
  - https://hackage.haskell.org/package/directory

B Appendix: Steady Pace Pseudocode

```
    let # Force for constant speed.
        fReq = roadFriction(v,ru) + aeroDrag(v,ru)
        #
        wTorque = fReq * v.wheelRadius
        wSpeed = ru.speed / (2*pi*v.wheelRadius)

        # pu eequirements per gear.
        allFeasibleGearReq =
            [ (g, t, s)
            | g <- v.GB
            #
            # Compute required engine speed.
            , let s = engineSpeedOfWheelSpeed(wSpeed,g,v.diff)
            # Verify this is feasible engine speed.
            , isValidGearForOutputSpeed(v,pum,g,ru.speed)
            #
            # Compute required engine torque.
            , let t = engineTorqueOfWheelTorque(wTorque,g,v.diff)
            # Verify this is feasible torque output.
            , isValidGearForOutputTorque(v,pum,s,t)
            ]

        # Most efficient gear.
        (g, t, s) = minimumBy(fuelCost, allFeasibleGearReq)
        #
        # return thePower Unit Configuration
        # of best gear requirements.
     in puConfigOf(v,ru,t,s,g)
```

C Appendix: Acceleration Sampling Requirements

```
sampleGear(v,ru,vNow,vMax,gear,timeNow,timeSample,as,aToll)
    | vNow >= vTarget -> exit
    # IFF the gear is too LOW for current speed: stop, gear is exhausted.
    | isGearTooLow(gear, vNow) -> exit
    # IFF the gear is too HIGH for current speed: consider restarting
    # with gear's minimum speed
    | isGearTooHigh(gear, vNow) ->
        let gearMinSpd = gearMinSpeed(v, pum, gear)
         in # If the min speed is in the acceleration range.
            if vNow < gearMinSpd <= vMax
            # Then restart sampling from this point.
            then sampleGear(v,ru,gearMinSpd,vMax,gear,timeNow,timeSample,as,aToll)
            # Otherwise stop, the gear is out of range.
            else exit
    # Else if no exit condition applies, compute a sample from vNow.
```

```
    | otherwise ->
        let # Calculate the total force opposing to the vehicle motion.
            fOpp = totalRoadLoad(v, ru{roadUnitSpeed := vNow})

            # Calculate the force of acceleration required.
            fAcc = v.mass * as

            # Hence the total force required.
            fReq = fOpp + fAcc

            # Translate this force to wheel torque, and engine torque.
            reqeTorque = let tWheels = fReq * v.wheelRadius
                         in engineTorqueOfWheelTorque(tWheels,gear,v.diff)

            # Calculate the engine speed for vNow.
            eSpeed = engineSpeedOfWheelSpeed(vNow, gear, v.diff, v.wheelRadius)

            # Pull the power unit, and verify if the desired eTorque can be delivered
            # with the preferred power unit mode. If not get valid torque and $pu$ mode.
            (eTorque, pum) = biasedPuModeForTorque(v, pum, eSpeed, reqeTorque)

            # Calculate what wheel torque eTorque can deliver.
            tWheelApplicable =
                      wheelTorqueOfEngineTorque(eTorque,gear,v.diff,v.wheelRadius)

            # Hence the pushing force on the vehicle.
            fResultantAcc = (tWheelApplicable / v.wheelRadius) - fOpp

            # Vehicle's acceleration.
            acc = fResultantAcc / v.mass

            # Vehicle's final velocity in this sample unit.
            vNew = vNow + acc * timeSample

         in # Verify enough acceleration has occurred.
            if acc >= aToll then
            queue((vNew,eTorque,eSpeed,pum)
                  ,sampleGear(v,ru,vNew,vMax,gear,(timeNow+timeSample),
                                                  timeSample,as,aToll))

            # Not enough acceleration: gear is exhausted, or too high.
            | otherwise exit
```

D Appendix: Deceleration Sampling Requirements

```
sampleGear(v,ru,vNow,vMin,gear,timeNow,timeSample,ds)
    # IFF target speed is reached: exit.
    | vMin <= vTarget -> exit

    # IFF the gear is too HIGH for current speed: stop, gear is exhausted.
    | isGearTooHigh(gear, vNow) -> exit

    # IFF the gear is too LOW for current speed: consider restarting
    # with gear's maximum speed
    | isGearTooLow(gear, vNow) ->
        let gearMaxSpd = gearMaxSpeed(v, pum, gear)
         in # If the min speed is in the acceleration range.
            if vNow > gearMaxSpd >= vMin
            # Then restart sampling from this point.
            then sampleGear(v,ru,gearMaxSpd,vMin,gear,timeNow,timeSample,ds)
            # Otherwise stop, the gear is out of range.
            else exit

    | otherwise =
```

```
        let # Calculate the engine speed.
            eSpeed = engineSpeedOfWheelSpeed(vNow, gear, v.diff, v.wheelRadius)

            # Calculate the engine torque.
            wTorque = wheelBrakingTorque(v, ru, d_s)
            eTorque = engineTorqueOfWheelTorque(wTorque, gear, v.diff, v.wheelRadius)

         in # Check braking force is required
            if eTorque <= 0 then exit
            else let # Create the telemetry point.
                     samplePoint = (vNow, eTorque, eSpeed)

                     # Compute the vehicle new speed.
                     vNew = vNow + d_s*sampleTime

                 in queue(samplePoint,
                          sampleGear(v, ru, vNew, vMin, gear, timeNow+timeSample, timeSample, ds))
```

E Appendix: KNN Point Density Control

```
add(map :: HashMap TelNode, newp:: TelPoint) -> HashMap TelNode
    let # Rescale with multiple of: Torque: 5Nm, Speed: 100RPM, Temp: 5 Celsius.
        scaledPoint :: TelPoint = rescalePoint(point=newp, t=5, r=100, at=5)
        # Normalize the point.
        normScaledPoint :: TelPoint = normalize(scaledPoint)
        # Compute its index value.
        key = hash(newScaledPoint)
        #
    in case HashMap.lookup(key, map) of
        # In the map there exists a TelNode at this index: update its average.
        tnode :: TelNode ->
                let newAvg = updateAvg(tnode.eff, tnode.nPoints, normScaledPoint.eff)
                    newNpoints = tnode.nPoints + 1
                    updNode = tnode{eff=newAvg, nPoints=newNpoints}
                 in HashMap.insert(key, updNode, HashMap.delete(key, map))
        #
        # New entry in the map: insert new node.
        Void  ->
                let newNode :: TelNode = toNode(normScaledPoint)
                 in HashMap.insert(key, newNode, map)
```

F Appendix: The Need of an Exhaustive Search

Performing an exhaustive search entails the selection of the solution with minimum associated cost from the set of all valid solutions ie. $x \mid \forall y \in \{PUC\}, PUC\_Cost(x) \leq PUC\_Cost(y)$.

F.1 Motivation

A greedy solution, which for each road unit selects, regardless of the vehicle's energy state, the best known *puc*, is the optimal solution iff the vehicle can supply enough energy throughout the itinerary.

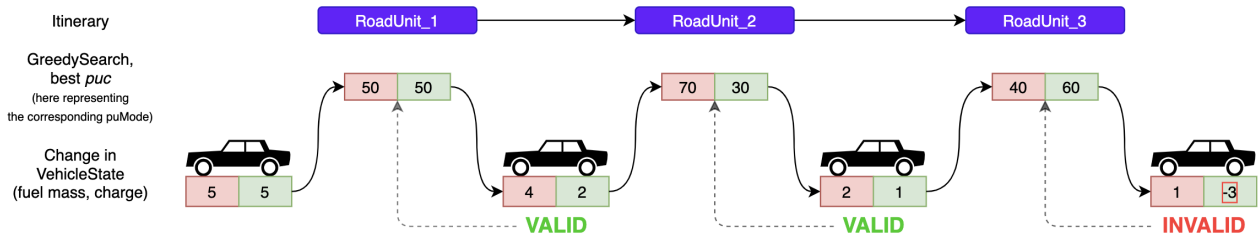
```
greedySolution(v:Vehicle, i:Itinerary) -> PUC or NULL
    # Solution buffer.
    acc = empty PUC
    # State of the vehicle leaving the previous RoadUnit.
```

```
    currState : VehicleState = vehicleState(v)

    forAll (ru : RoadUnit) in i

        allOpts = getAllPowerUnitConfigsOf(v, ru)
        bestOpt = argmin pucCost(allOpts)

        case bestOpt of
            NULL:
                acc = NULL
                break

            otherwise:
                queue(acc, bestOpt)
                currState = vehicleStateAfter(vState, bestOpt)

    return acc
```

Failure of the greedy search signals the need for a number of 'sacrifices' (ie. not selecting the best *puc* at each given stage) are required by the optimal solution.

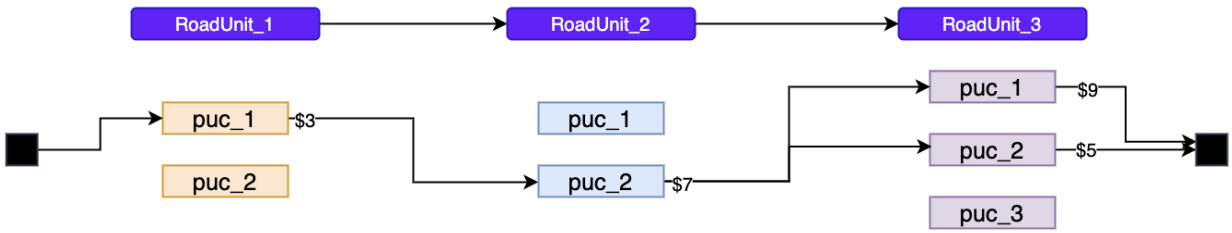
Nowadays the most likely cause of failure of the greedy search is insufficient electric charge. The figure following is an example of such scenario:

Figure 32: Example of greedy solution being invalid for a vehicle's energy state.

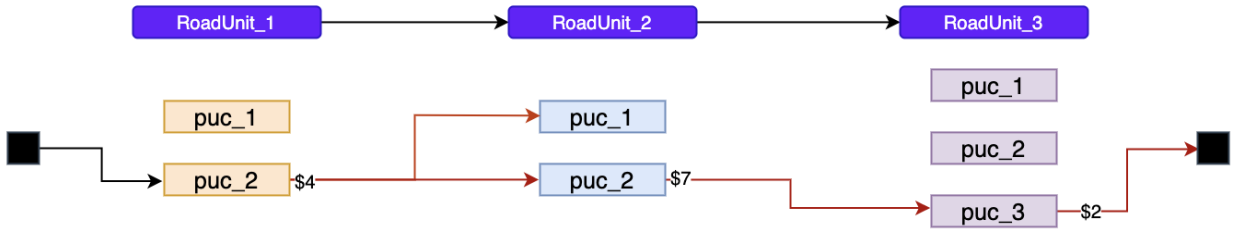


Finding the optimal sequence of compromises is an indeterministic optimisation problem, to which dynamic programming cannot be successfully implemented to solve. In fact, choosing a *puc* from any *RoadUnit* can result in a change of available *puc* later in the Itineraries; which induces DP into producing invalid solutions. Consider the following scenario in which

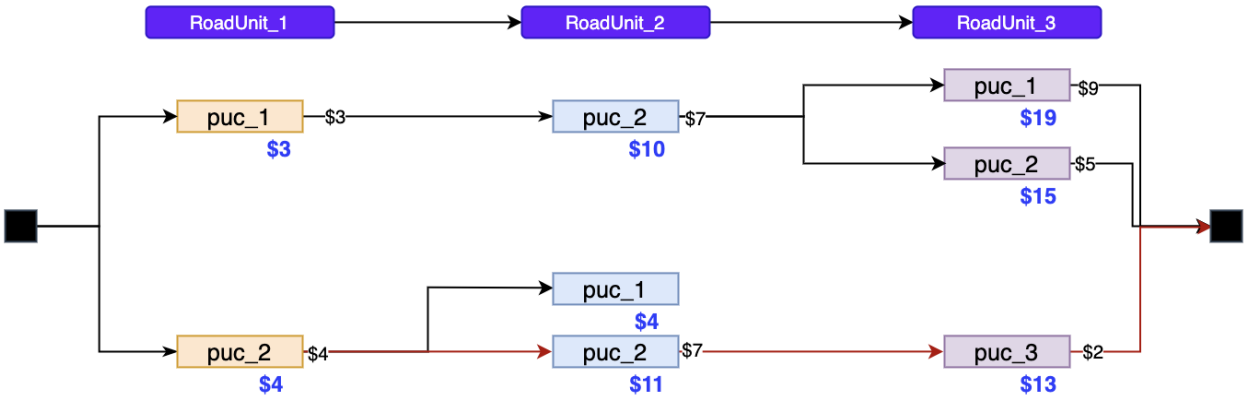
- Choosing the first option available in *RoadUnit*_1, allows for only the selection of *puc*_2 in the following section, and a choice of *puc*_1 and *puc*_2 in the last:



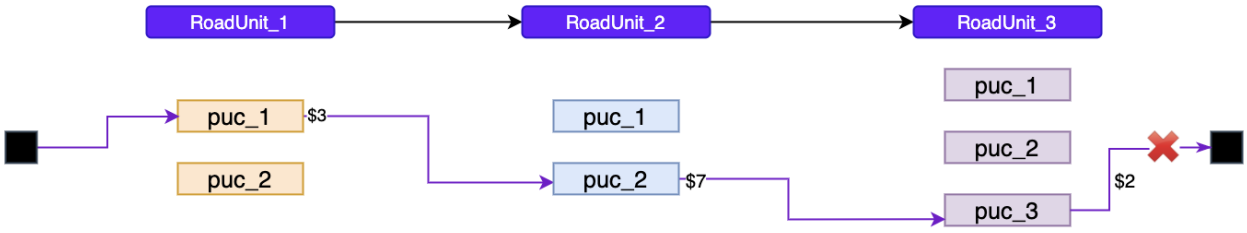
- Choosing of the second option in *RoadUnit*_1 allows for *puc*_1 and *puc*_2 in *RoadUnit*_2. However, only the latter allows for the finishing of the itinerary through the selection of *puc*_3 in *RoadUnit*_3:



To compute the optimal solution with a dynamic programming strategy, the problem graph needs to be adjusted to remove all non overlapping subproblems. This passage ensure that, after the selection of an option, the costs of the remaining paths are recomputed according to the subpaths the initial selection allowed. Crucially this means exploring the entire search space, questioning the usefulness of dynamic programming.



On the other hand, not performing such transformation step is likely to lead, for the very same reasons already mentioned, to invalid results:



F.2 Modelling And Implementation

In choosing a search method for this problem, it must be considered that:

- Unknown goal state.
- Inability to score the fitness of partial solutions.

These points motivate the avoiding of an A* search, as this is based not only on the ability to recognise the goal state, but also to estimate the fitness of the partial solutions. On the

other hand, Breath First Search (BFS) would find the optimal solution, but, in this scenario, lead to larger space complexity with no time complexity gains when compared to a Depth First Search (DFS).

Therefore, DFS seems like the most appropriate options, as it would compose the entire search tree, whilst leaving the smallest unit memory footprint. Furthermore, it can be implemented in a producer-consumer format, which would allow the system to return a stream of decreasing cost solutions, before having evaluated all of the options.
In these terms, a consumer would take the current best solution, a new solution, and output the better of the two:

```
bestSolutionFilter(chan:SolutionChannel, currBest:Solution) -> IO

  newSol <- readChan chan

  if PUC_cost(newSol) < PUC_cost(currBest)
        output(newSol)
        bestSolutionFilter(chan, newSol)
  else
        bestSolutionFilter(chan, currBest)
```

The producer, performs a DFS search and output every next generated solution to the consumer. Each level of the search requires the evaluation of the modes the power unit can be asked to run, given the *RoadUnit* and the energy state of the vehicle. The function attaches to each *pu* mode the corresponding *pu* configuration, or null if the mode isn't physically available:

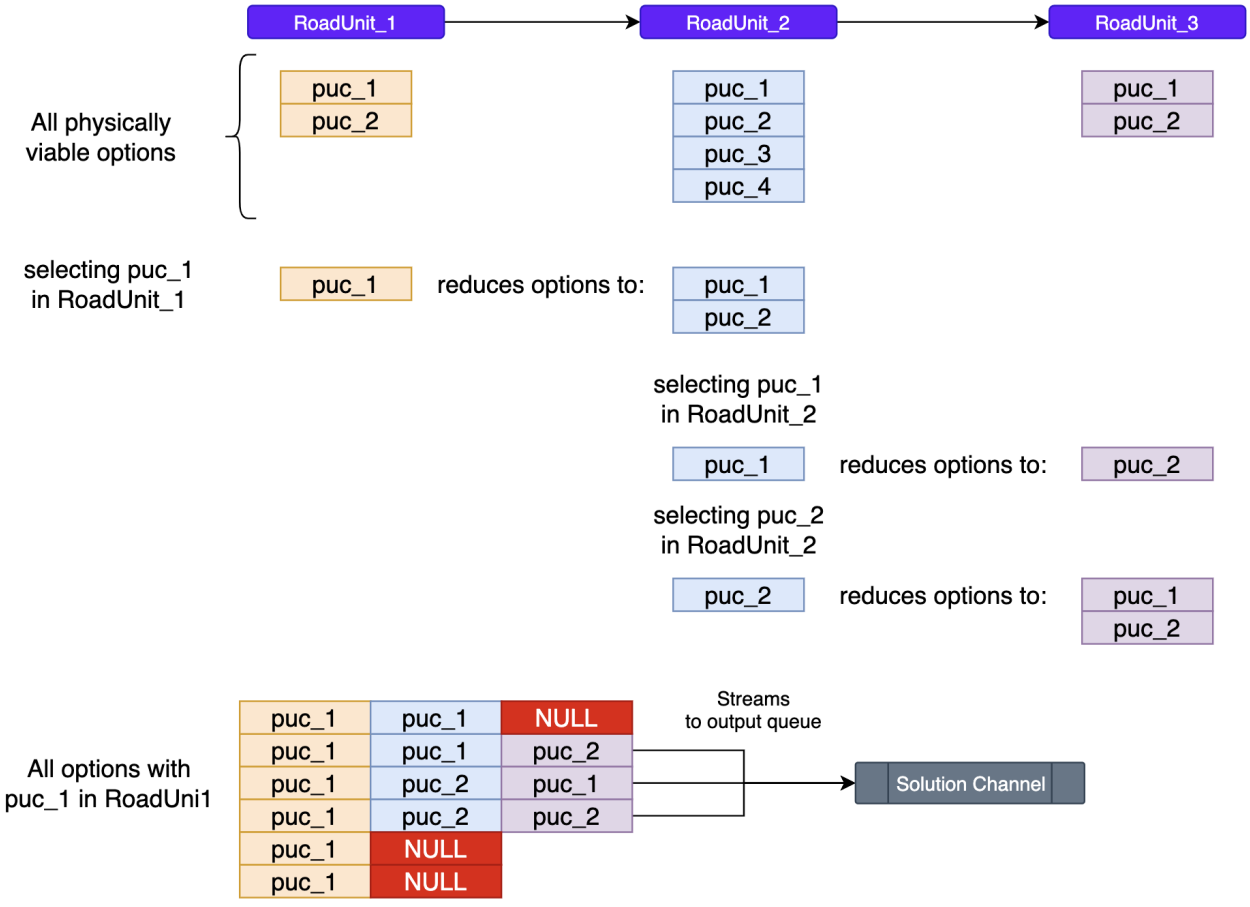
```
expandRoadUnit(v:Vehicle, ru:RoadUnit, puModes:listOf(PowerUnitMode)) -> listOf(puc or NULL)

    returnVal = listOf(puc or NULL)

    for (puMode:PowerUnitMode) in puModes
        configRes:(puc or NULL) = NULL

        if isSteadyPace(ru)
            configRes = steadyPaceConfig(v, ru, puMode)

        else if isAcceleration(ru)
            configRes = accelerationConfig(v, ru, puMode)

        else if isDeceleration(ri)
            configRes = decelerationConfig(v, ru, puMode)

        queue(returnVal, configRes)

    return returnVal
```

Hence, this function can be implemented in a DFS search:

```
search(v:Vehicle, it:Itinerary, exps:PartialSolution, chan:SolutionChannel) -> IO
    # Finished itinerary: output the solution to the channel.
    if isEmpty(it)
        writeToChan(chan, reverse(exps))

    # Need to expand this level and for every option on this level
    # fully expand the chain.
    else
        # Take the next RoadUnit in the itinerary.
        ru:RoadUnit = head(it)

        # Compose the vehicle state on this level.
```

```
        vState:VehicleState = isEmpty(exps) ?
                                vehicleState(v) # Vehicle setoff state.
                              : vehicleState(head(exps)) # State of vehicle after
                                                         # leaving the previous RU.


        # Update the vehicle's state
        newV = v{vehicleState = vState}

        # Get the all the options on this level (lazily).
        opts = expandRoadUnit(newV, ru, _PUM_ALL)

        # For each option fully extend the solution chain.
        for opt in opts
            if opt /= NULL
                search(newV, tail(it), headqueue(opt exps), chan)

            # If null this partial solution is truncated.
```

Figure 33: DFS Produce-Consumer example.



References

[1] Hughes Austin. *Electric Motors and Drives Fundamentals, Types and Applications*. George Newnes Ltd, Third edition, 2006.

[2] Jonathan Bac and Andrei Zinovyev. Local intrinsic dimensionality estimators based on concentration of measure. *arXiv preprint arXiv:2001.11739, Cornell University*, 2020.

[3] Michael Ben-Chaim, Efraim Shmerling, and Alon Kuperman. Analytic modeling of vehicle fuel consumption. *Energies*, 6(1):117–127, 2013.

[4] L. Chu, F. Zhou, J. Guo, and M. Shang. Investigation of determining of regenerative braking torque based on associated efficiency optimization of electric motor and power battery using ga. In *Proceedings of 2011 International Conference on Electronic Mechanical Engineering and Information Technology*, volume 6, pages 3238–3241, Aug 2011.

[5] Yong Deng, Yuxin Chen, Yajuan Zhang, and Sankaran Mahadevan. Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, 12(3):1231–1237, 2012.

[6] Thomas D Gillespie. *Fundamentals of vehicle dynamics*, volume 400. Society of automotive engineers Warrendale, PA, 1992.

[7] John B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill, 1988.

[8] J. Hou, H. Gao, Q. Xia, and N. Qi. Feature combination and the knn framework in object classification. *IEEE Transactions on Neural Networks and Learning Systems*, 27(6):1368–1378, 2016.

[9] Argonne National Laboratory. Energy systems d3 2012 fiat 500 sport. `https://www.anl.gov/es/energy-systems-d3-2012-fiat-500-sport`, Last Accessed on 03-05-2020.

[10] Argonne National Laboratory. Energy systems d3 2012 ford fusion v6. `https://www.anl.gov/es/energy-systems-d3-2012-ford-fusion-v6`, Last Accessed on 03-05-2020.

[11] Argonne National Laboratory. Energy systems d3 2013 nissan leaf sv. `https://www.anl.gov/es/energy-systems-d3-2013-nissan-leaf-sv`, Last Accessed on 03-05-2020.

[12] Argonne National Laboratory. Energy systems d3 2014 bmw i3-rex. `https://www.anl.gov/es/energy-systems-d3-2014-bmw-i3rex`, Last Accessed on 03-05-2020.

[13] Warne D. J. Laughton M. A. *Electrical Engineer's ReferenceBook*. Elsevier Ltd, Sixteenth edition, 2003.

[14] Chao Li, Qifang Liu, Lulu Guo, and Hong Chen. Fuel economy optimization of hybrid electric vehicles. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 810–815. IEEE, 2015.

[15] Massachusetts Institute of Technology. 15.5 adiabatic flame temperature. `https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node111.html`, Last accessed on 15-11-2019.

[16] Massachusetts Institute of Technology. 15.6 muddies points on chapter 15. https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node112.html, Last accessed on 15-11-2019.

[17] George G Mitchell, Diarmuid O'Donoghue, David Barnes, and Mark McCarville. Generepair-a repair operator for genetic algorithms. 2003.

[18] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[19] Morteza Montazeri-Gh, Amir Poursamad, and Babak Ghalichi. Application of genetic algorithm for optimization of control strategy in parallel hybrid electric vehicles. *Journal of the Franklin Institute*, 343(4-5):420–435, 2006.

[20] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*, volume 821. Hoboken, New Jersey : John Wiley & Sons Ltd, 2012. ©2012, fifth edition, 2012.

[21] Harald Naunheimer, Bernd Bertsche, Joachim Ryborz, and Wolfgang Novak. *Automotive transmissions: fundamentals, selection, design and application*. Springer Science & Business Media, second edition, 2010.

[22] Nissan. Nissan leaf prices & specifications. https://www.nissan.co.uk/vehicles/new-vehicles/leaf/prices-specifications.html#grade-LEAFZE1A-0|specs, Last accessed on 20-11-2019.

[23] Dalby Paul Sherwood Dennis. *Modern Thermodynamics for Chemists and Biochemists*. Oxford Scholarship Online, 2018.

[24] CM Silva, GA Gonçalves, TL Farias, and JMC Mendes-Lopes. A tank-to-wheel analysis tool for energy and emissions studies in road vehicles. *Science of the Total Environment*, 367(1):441–447, 2006.

[25] Tesla. Model s owner's manual. https://www.tesla.com/sites/default/files/model_s_owners_manual_north_america_en_us.pdf, Last accessed on 20-11-2019.

[26] Toyota. Toyota prius technical specifications. https://media.toyota.co.uk/wp-content/files_mf/1329489972120216MTOYOTAPRIUSTECHNICALSPECIFICATIONS.pdf, Last accessed on 20-11-2019.

[27] Qi Zhao, Qi Chen, and Li Wang. Real-time prediction of fuel consumption based on digital map api. *Applied Sciences*, 9(7):1369, 2019.