Development of Engine Activity Cycles for the Prime-Movers of Unconventional Well Completion

Robert S. Heltzel

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
Development of Engine Activity Cycles for the Prime-Movers of Unconventional Well Completion

Robert S. Heltzel

Thesis submitted to the Benjamin M. Statler College of Engineering and Mineral Resources
At West Virginia University
In partial fulfillment of the requirements for the degree of

Master of Science
In
Mechanical Engineering

Derek Johnson, Ph.D., Chair
Andrew Nix, Ph.D.
Gregory Thompson, Ph.D.

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2016

Keywords: Engine Cycles, Markov Chain, Genetic Algorithm, Unconventional Well Completion

Copyright 2016 Robert Heltzel
Abstract

Development of Engine Activity Cycles for the Prime Movers of Unconventional Well Completion

Robert S. Heltzel

The US natural gas extraction industry has grown by 25% over the past decade. This has encouraged the well completion industry to utilize produced natural gas to meet power demands previously obtained with diesel fuel. The major “prime-movers” of diesel fuel consumption are over-the-road (OTR) trucks, high horsepower drilling, and high horsepower hydraulic fracturing engines. The US Department of Energy (DOE) recently granted West Virginia University (WVU) funding, under agreement DE-FE0013689, to “Assess Fugitive Methane Emissions Impact Using Natural Gas Engines in Unconventional Resource Development.” As part of the funding, WVU was tasked with creating engine activity cycles representative of the prime-movers’ in-use operation.

In-use data were collected from the major prime-movers in the Marcellus shale play region. Cycles were created to replicate the engine speed and load activity. Cycle creation was achieved by applying a Markov chain Monte Carlo (MCMC) technique. The Markov chain utilized second-by-second concatenation and a transition probability matrix based on the collected data distribution. This approach allowed for a large number of cycles to be created. Comparing cycles to collected data employed a Performance Value (PV), which accounted for data statistics and distribution. Populations of 10,000 cycles were created for each of the major prime-movers.

Optimization of cycles was achieved with the use of a genetic algorithm (GA). The GA used the genetic operators of crossover and mutation, had a population of 100 cycles, and lifespan of 50 generations. The GA was executed for of the major prime-mover cycles: OTR trucks, drilling, and hydraulic fracturing improving their PVs by 18%, 62%, and 65%, respectively compared to the best individual of the initial population. The final cycles were exercised on a Cummins ISLG natural gas engine mounted to an 800 horsepower dynamometer for verification of regression criteria. The drilling and hydraulic fracturing cycles met all regression criteria defined by the CFR however, the OTR cycle did not. A Savitsky-Golay (SG) smoothing technique was applied to the cycle, resulting in a 1.25% increase in the PV. The smooth OTR cycle met all regression criteria. This completed the definition of all three prime-mover cycles.
Acknowledgements
I can say with the utmost sincerity that none of my accomplishments as a Graduate Student would have been possible without my advisor and friend: Dr. Derek Johnson. Boss, your tutelage, understanding, and support have allowed me to grow both as a student and a person. I cannot thank you enough for all that you have done for me. The value of your guidance throughout my time under you cannot be quantified or measured, even by us.

I would also like to thank the other members of my committee: Drs. Andrew Nix and Gregory Thompson. Without Dr. Nix’s lead as Principal Investigator my research would not have been possible. Dr. Thompson introduced me to CAFEE as an undergraduate which ultimately led to the opportunity for me to continue my education.

A special thanks is also extended to Dr. Marc Besch, whose expertise on emissions research and data acquisition proved invaluable.

Recognition must also be given to the Department of Energy for the funding, which allowed me to complete this work.

I also have to thank Nathan Fowler for his assistance. Nate’s knowledge of engines, hands on skills, and brute strength have been valuable during my time as a student.

Lastly, I would like to thank my dad, my brother and my friends for their support as I pursued my degree. These people have always been there for me, especially, my brother, Brad Heltzel. His life perspective is always impactful and helpful. My friends and roommates also deserve a special thanks for helping me enjoy my college experience to the fullest.

This work, as is everything I do, is dedicated to my mom, Cindi Ann Scott. She is proof that not all hearts and thoughts fade, fade away. Thank you for everything and Rest in Peace.
# Table of Contents

Acknowledgements ........................................................................................................ iii

Table of Figures ........................................................................................................... vi

Table of Tables ........................................................................................................... vii

Table of Equations ...................................................................................................... viii

List of Acronyms ........................................................................................................ ix

1. Introduction ............................................................................................................. 1

2. Background/Development of Problem ................................................................... 3
   2.1 Natural Gas Well Development and Prime-Movers ............................................. 3
   2.2 Engines Used ...................................................................................................... 7
   2.3 Emissions, Testing and Regulations .................................................................. 8

3. Review of Literature ............................................................................................... 18
   3.1 Markov Chain Monte Carlo ............................................................................. 18
   3.2 Cycle Development .......................................................................................... 19
   3.3 Genetic Algorithms and Use in Cycle Development Optimization ..................... 25

4. Data Collection Methods ....................................................................................... 30
   4.1 Over-the-Road Trucks ..................................................................................... 30
   4.2 Drilling .............................................................................................................. 32
   4.3 Hydraulic Fracturing ....................................................................................... 35

5. Data Processing ...................................................................................................... 38
   5.1 Over-the-Road Trucks ..................................................................................... 38
   5.2 Drilling .............................................................................................................. 41
   5.3 Hydraulic Fracturing ....................................................................................... 43

6. Cycle Formation ...................................................................................................... 46
   6.1 Over-the-Road Trucks ..................................................................................... 46
   6.2 Drilling .............................................................................................................. 49
   6.3 Hydraulic Fracturing ....................................................................................... 51

7. Genetic Algorithm for Cycle Optimization ............................................................ 52
   7.1 Over-the-Road Trucks ..................................................................................... 53
   7.2 Drilling .............................................................................................................. 58
   7.3 Hydraulic Fracturing ....................................................................................... 59

8. Results of Genetic Algorithm ................................................................................ 60
   8.1 Over-the-Road Trucks ..................................................................................... 61
   8.2 Drilling .............................................................................................................. 63
   8.3 Hydraulic Fracturing ....................................................................................... 64
9. Cycle Denormalization and Verification .............................................................. 67
10. Smoothing and Final Verification ........................................................................... 78
11. Conclusion ............................................................................................................ 85
References.................................................................................................................. 88
Appendices.................................................................................................................. 93
    Appendix A: MATLAB® Code for Data Import .......................................................... 93
    Appendix B: MATLAB® Code for Cycle Development .............................................. 107
    Appendix C: MATLAB® Code for Genetic Algorithm ............................................. 122
    Appendix D: Denormalization and Dynamometer Calibration ................................ 145
    Appendix E: Denormalized Cycle “.cyc” File Formats ........................................... 149
    Appendix F: Smoothing ......................................................................................... 150
    Appendix G: All Cycle Points ............................................................................... 157
    Appendix H: Real World Activity Segments Similar to Cycles ............................... 188
Table of Figures

Figure 1: Major US Shale Plays [6]......................................................................................... 3
Figure 2: Truck Trips for Various Activities [8]........................................................................ 4
Figure 3: Transient FTP Test [24]........................................................................................... 11
Figure 4: Basic NTE Zone [26].................................................................................................. 13
Figure 5: Nonroad Transient Test [30]....................................................................................... 16
Figure 6: Basic Flow Chart of a Genetic Algorithm [50]............................................................ 26
Figure 7: HEM J1939 Mini Logger [53]..................................................................................... 32
Figure 8: VIA Model HDV100A1 Connected to Caterpillar 3512C Drilling Engine................. 34
Figure 9: ECU Slave Data Acquisition Software...................................................................... 34
Figure 10: ECU Packer Analyzer Software.............................................................................. 35
Figure 11: Truck Data Matrix Format....................................................................................... 40
Figure 12: Truck Data Distribution............................................................................................ 40
Figure 13: Drilling Data Distribution........................................................................................ 43
Figure 14: Fracturing Data Distribution..................................................................................... 44
Figure 15: Structure of an OTR Truck Cycle ............................................................................ 49
Figure 16: Structure of Drilling and Fracturing Cycles............................................................... 50
Figure 17: Truck Cycle with Defined Idle and Non-Idle Segments........................................... 54
Figure 18: Idle Segment Example from Cycle of Figure 17......................................................... 54
Figure 19: Non-Idle Segment Example from Cycle of Figure 17.............................................. 55
Figure 20: OTR Truck GA Improvements............................................................................... 61
Figure 21: Elite OTR Truck Cycle.............................................................................................. 62
Figure 22: Drilling GA Improvements...................................................................................... 63
Figure 23: Elite Drilling Cycle................................................................................................. 64
Figure 24: Hydraulic Fracturing GA Improvements................................................................. 65
Figure 25: Elite Hydraulic Fracturing Cycle............................................................................. 66
Figure 26: ISLG Lug Curve....................................................................................................... 68
Figure 27: Denormalized OTR Truck Cycle for Cummins ISLG............................................... 69
Figure 28: Denormalized Drilling Cycle for Cummins ISLG..................................................... 69
Figure 29: Denormalized Hydraulic Fracturing Cycle for ISLG............................................... 70
Figure 30: GE 800 HP DC Dynamometer................................................................................ 72
Figure 31: Dynamometer Side-Arm Calibration Curve.............................................................. 73
Figure 32: Cummins ISLG on 800 HP Dynamometer............................................................... 73
Figure 33: Sample of OTR Cycle Speed Smoothing ................................................................. 81
Figure 34: Sample of OTR Cycle Load Smoothing................................................................. 81
Figure 35: Normalized OTR Truck Cycle with SG13-7 Smoothing.......................................... 82
Figure 36: Denormalized OTR Truck Cycle with SG13-7 Smoothing........................................ 83
Table of Tables

Table 1: Parameters of Well Development Truck Traffic [10] ............................................................ 5
Table 2: Diesel Fuel Consumed for Drilling and Pad Construction [10] ............................................... 6
Table 3: Aspects of Fracturing Jobs in Various Plays [13] ................................................................. 6
Table 4: Onroad Heavy Duty Diesel Emissions (g/bhp-hr) [22] .......................................................... 9
Table 5: Heavy-Duty Vehicle GHG Emissions [23] ................................................................. 10
Table 6: Heavy-Duty Engine GHG Emissions [23] ............................................................................. 10
Table 7: Ramped Mode Cycle SET [25] ........................................................................................ 12
Table 8: Tier 1-3 Nonroad Emissions Standards g/kWh (g/bhp-hr) [28] ................................................ 14
Table 9: Tier 4 Nonroad Emissions Standards g/kWh (g/bhp-hr) [28] ................................................ 15
Table 10: Tier 4 Emissions Standards for Nonroad Engines Greater than 560 kW [28] ..................... 15
Table 11: ISO 8178 Test Cycle and Weighting Factors [29] ............................................................ 16
Table 12: Well Service Vehicles Data Collection Information .......................................................... 31
Table 13: Drilling Activity Breakdown ............................................................................................. 33
Table 14: Hydraulic Fracturing Data Collection Information ............................................................... 37
Table 15: Engine Speed Information ............................................................................................... 39
Table 16: Trucking Data Statistics .................................................................................................... 41
Table 17: Drilling and Fracturing Data Matrix Format ....................................................................... 42
Table 18: Drilling Data Statistics ........................................................................................................ 43
Table 19: Fracturing Data Statistics .................................................................................................. 45
Table 20: Genetic Algorithm Iteration Results .................................................................................. 60
Table 21: OTR Truck Cycle and Data Statistics .................................................................................. 62
Table 22: Drilling Cycle and Data Statistics ....................................................................................... 64
Table 23: Hydraulic Fracturing Cycle and Data Statistics ................................................................. 66
Table 24: Cummins ISLG Engine Specifications [63] ........................................................................ 67
Table 25: Cycle Regression Validation Criteria for Cummins ISLG ................................................... 71
Table 26: Dynamometer Side-Arm Calibration Points ....................................................................... 72
Table 27: Parameters of the Engine Configuration File .................................................................... 74
Table 28: OTR Truck Cycle Regression Results ............................................................................... 75
Table 29: Drilling Cycle Regression Results ..................................................................................... 76
Table 30: Hydraulic Fracturing Cycle Regression Results ............................................................... 77
Table 31: Smoothing Techniques and Their Effects on PVs ............................................................. 79
Table 32: SG Effect on PV for All Types of Cycles .......................................................................... 80
Table 33: Smooth OTR Truck Cycle and Data Statistics ................................................................. 82
Table 34: Smooth OTR Truck Cycle Regression Results ............................................................... 83
# Table of Equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1</td>
<td>38</td>
</tr>
<tr>
<td>Equation 2</td>
<td>48</td>
</tr>
<tr>
<td>Equation 3</td>
<td>48</td>
</tr>
<tr>
<td>Equation 4</td>
<td>48</td>
</tr>
<tr>
<td>Equation 5</td>
<td>51</td>
</tr>
<tr>
<td>Equation 6</td>
<td>57</td>
</tr>
<tr>
<td>Equation 7</td>
<td>57</td>
</tr>
<tr>
<td>Equation 8</td>
<td>57</td>
</tr>
<tr>
<td>Equation 9</td>
<td>57</td>
</tr>
<tr>
<td>Equation 10</td>
<td>58</td>
</tr>
<tr>
<td>Equation 11</td>
<td>58</td>
</tr>
<tr>
<td>Equation 12</td>
<td>58</td>
</tr>
<tr>
<td>Equation 13</td>
<td>67</td>
</tr>
</tbody>
</table>
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Word or Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL</td>
<td>Argonne National Laboratory</td>
</tr>
<tr>
<td>CAA</td>
<td>Clean Air Act</td>
</tr>
<tr>
<td>CAFEE</td>
<td>Center for Alternative Fuels, Engines and Emissions</td>
</tr>
<tr>
<td>CAN</td>
<td>Communication Area Network</td>
</tr>
<tr>
<td>CARB</td>
<td>California Air Resources Board</td>
</tr>
<tr>
<td>CFR</td>
<td>Code of Federal Regulations</td>
</tr>
<tr>
<td>CH₄</td>
<td>Methane</td>
</tr>
<tr>
<td>CITT</td>
<td>Curb Idle Transmission Torque</td>
</tr>
<tr>
<td>CO₂</td>
<td>Carbon Dioxide</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DGB</td>
<td>Dynamic Gas Blending</td>
</tr>
<tr>
<td>DOE</td>
<td>Department of Energy</td>
</tr>
<tr>
<td>DOT</td>
<td>Department of Transportation</td>
</tr>
<tr>
<td>DPF</td>
<td>Diesel Particulate Filter</td>
</tr>
<tr>
<td>DV</td>
<td>Distribution Value</td>
</tr>
<tr>
<td>E&amp;P</td>
<td>Exploration and Production</td>
</tr>
<tr>
<td>EIA</td>
<td>Energy Information Administration</td>
</tr>
<tr>
<td>EF</td>
<td>Evaluation Function</td>
</tr>
<tr>
<td>EPA</td>
<td>Environmental Protection Agency</td>
</tr>
<tr>
<td>EERC</td>
<td>Engines and Emissions Research Center</td>
</tr>
<tr>
<td>lb-ft</td>
<td>Foot Pounds</td>
</tr>
<tr>
<td>FTP</td>
<td>Federal Test Procedure</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>gal</td>
<td>Gallons</td>
</tr>
<tr>
<td>GE</td>
<td>General Electric</td>
</tr>
<tr>
<td>genset</td>
<td>Electric Generator Set</td>
</tr>
<tr>
<td>GHG</td>
<td>Greenhouse Gas</td>
</tr>
<tr>
<td>GVWR</td>
<td>Gross Vehicle Weight Rating</td>
</tr>
<tr>
<td>GWP</td>
<td>Global Warming Potential</td>
</tr>
<tr>
<td>HP</td>
<td>Horsepower</td>
</tr>
<tr>
<td>HPDI</td>
<td>High Pressure Direct Injection</td>
</tr>
<tr>
<td>in</td>
<td>Inches</td>
</tr>
<tr>
<td>kW</td>
<td>Kilowatt</td>
</tr>
<tr>
<td>MCMC</td>
<td>Markov Chain Monte Carlo</td>
</tr>
<tr>
<td>MLE</td>
<td>Maximum Likelihood Estimation</td>
</tr>
<tr>
<td>NHTSA</td>
<td>National Highway Traffic Safety Administration</td>
</tr>
<tr>
<td>NOₓ</td>
<td>Oxides of Nitrogen</td>
</tr>
<tr>
<td>NRTC</td>
<td>Non-road Transient Cycle</td>
</tr>
<tr>
<td>NTE</td>
<td>Not-to-Exceed</td>
</tr>
<tr>
<td>OTR</td>
<td>Over-the-Road</td>
</tr>
<tr>
<td>PCA</td>
<td>Principle Component Analysis</td>
</tr>
<tr>
<td>PHEV</td>
<td>Plug-in Hybrid Electric Vehicle</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivative</td>
</tr>
<tr>
<td>PT</td>
<td>Pipe Tripping</td>
</tr>
<tr>
<td>PV</td>
<td>Performance Value</td>
</tr>
<tr>
<td>qts</td>
<td>Quarts</td>
</tr>
</tbody>
</table>
R²  Coefficient of Determination
RPM  Revolutions Per Minute
SAE  International Society of Automotive Engineers
SAFD  Speed Acceleration Frequency Distribution
SCR  Selective Catalytic Reduction
SEE  Standard Estimate of Error
SET  Supplemental Emissions Test
SG  Savitsky-Golay
SSD  Steady State Drilling
SV  Statistic Value
TCEQ  Texas Commission for Environmental Quality
Tcf  Trillion Cubic Feet
TF  Total Fitness
US  United States
WVU  West Virginia University
V  Volts
1. Introduction
The United States (US) has experienced a shale gas “boom” over the past decade. The increase in available gas reserves has led to exploration and production (E&P) companies attempting to utilize new technologies for extraction. These new technologies include horizontal drilling and hydraulic fracturing. With the use of these new technologies, strategies are being implemented to reduce fuel costs and emissions [1] [2]. These strategies often involve utilizing natural gas as fuel to displace diesel fuel in the prime movers associated with well development either via dual fuel or dedicated natural gas applications. These prime-movers, or major diesel consumers, include heavy-duty over-the-road (OTR) trucks and high horsepower drilling and fracturing engines.

Government agencies including the US Environmental Protection Agency (EPA), California Air Resources Board (CARB), the US Department of Transportation (DOT), and the US Department of Energy (DOE) are tasked with determining the environmental and health effects of engine emissions. The DOE recently granted West Virginia University (WVU) funding under agreement DE-FE0013689 in order to “Assess Fugitive Methane Emissions Impact Using Natural Gas Engines in Unconventional Resource Development.” As a part of this funding opportunity, WVU was tasked with the collection of operating data of the representative engine models and processing the data to develop emissions cycles representing the operating features of respective models.

Emissions cycles, activity cycles, or engine test cycles allow researchers to quantify engine emissions in a laboratory setting from similar engines to those that operate in the field. Representation of a specific industry was achieved by developing unique test cycles. Determining the true impact of engines being used in the field could theoretically be accomplished with test cycles that were representative of the actual operation. In-use data were collected from the prime-movers of unconventional well completion, so that representative cycles could be created. The technique chosen to build cycles was a second-by-second Markov chain construction approach. A Monte Carlo simulation created a large population of possible cycles for comparison to the data collected. This Markov chain Monte Carlo (MCMC) approach allowed for a nearly infinite number of potential cycles to be created. It was important to create a cycle that best represented in-use engine operation in a condensed amount of time to allow for in-laboratory engine testing. Comparing created cycles to collected data was completed using a Performance Value (PV). This value accounted for individual cycle statistics and distribution compared to those found based on data collected from in-use activity. Optimization of a representative cycle was achieved with the use of genetic algorithm (GA). The GA allowed for the best possible cycle to be selected based on the prescribed PV metric. Once the optimal cycle was defined for all three types of engine activity, they were verified with a
Cummins ISLG natural gas engine on a GE 800 HP dynamometer. In order for a cycle to be considered acceptable for use, it was required to pass the regression criteria defined by the Code of Federal Regulations (CFR) in Part 1065 Subpart F Section 514. Cycles were checked against these criteria to ensure they were suitable for emissions testing.
2. Background/Development of Problem

2.1 Natural Gas Well Development and Prime-Movers

The US has experienced growth in the natural gas industry over the past decade. Annual gross withdrawals of natural gas in the US have increased 25% since 2005 from 23 million to over 31 million cubic feet [3]. The US Energy Information Administration (EIA) also estimates there are about 2.276 trillion cubic feet (Tcf) of recoverable resources of dry natural gas in the US. At the current rate of consumption, is enough to last about 84 years [4]. This indicates that natural gas could serve as a source of reliable energy until around 2100. The extraction of this natural gas has been made possible by the development of technologies such as horizontal/directional drilling and hydraulic fracturing. The latest available EIA data from 2014 estimates that there are 514,786 gas producing wells in the US [5]. Many of these wells are found within regions of major US “shale plays.” Shale plays are areas in which pockets of natural gas are available deep below the surface. Major shale plays in the US include the Marcellus, Bakken, Barnett, Fayetteville, Eagle Ford, Haynesville, and Utica. A map of the lower 48 major US shale plays is shown in Figure 1.

Figure 1: Major US Shale Plays [6]

Large amounts of natural gas can be obtained from these plays provided that they can be horizontally or directionally drilled and hydraulically fractured. The processes of constructing a well-pad, drilling wells, and hydraulically fracturing wells are energy intensive. The entire process of developing one unconventional natural gas well can result in the consumption of hundreds of thousands of gallons of petroleum fuels. This fuel, usually diesel fuel, is consumed by what are known as “prime-movers.” The
prime-movers, or major fuel consumers, associated with the process of unconventional well completion include OTR trucks, high-horsepower drilling, and high-horsepower hydraulic fracturing engines. The process of unconventional well completion consists of four major steps: planning and preparation of the well pad site, drilling, casing and cementing, and completion and stimulation [7]. The primary consumer of diesel fuel during planning and preparation of the site is OTR trucks. Heavy-duty diesel trucks are necessary for delivery and removal of gravel, dirt, and other materials. The drilling step involves the use of high-horsepower engines used to power drilling rigs. Heavy-duty trucks are also used for the delivery of water, drilling fluids, and other supplies. Casing and cementing is usually performed by the drilling company and involves the same engines as drilling along with an increase in truck traffic. Completion and stimulation of the well with hydraulic fracturing is the most energy intensive step in the process and involves the use of high-horsepower engines and large amounts of truck traffic for the delivery of water, sand, and fracturing chemicals.

OTR trucks are required for all stages of the well completion process. A study of a Marcellus shale well pad with four wells produced data on the amount of truck traffic to and from the well. The study, performed by Deloitte, estimates that 20,000 to 30,000 total truck trips occurred during completion of the four wells. The total number of truck trips in the Marcellus shale play per year could be as high as four million according to the study [8]. Figure 2 shows the number of truck trips required during each phase of well completion.

**Figure 2: Truck Trips for Various Activities** [8]
Another study from un-naturalgas.org estimated the total number of truck loads needed for the completion of a well which was hydraulically fractured. This study estimated that nearly 320 truck trips are required for fracturing a well with two million gallons of water, which was considered small to average by those conducting the study. It stated that 1440 truck trips were needed for a nine million gallon well, which they considered large. These numbers are based on an individual well and most pads contain between four and twenty wells. This can result in 38,000 to 172,000 truck trips over the life of a well [9]. Other publications have estimated the amount of diesel fuel consumption in different shale plays from OTR trucks. A study from Argonne National Laboratory (ANL) estimated different truck traffic parameters involved in the transportation of fracturing fluids. Table 1 shows the conclusions of the study based on four major shale plays [10]. The amount of traffic varied by shale play due to terrain, average well depth or length, and other regulations and limitations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Barnett</th>
<th>Marcellus</th>
<th>Fayetteville</th>
<th>Haynesville</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water volume per truck (gal/truck)</td>
<td>5500</td>
<td>3400</td>
<td>5500</td>
<td>5500</td>
</tr>
<tr>
<td>Distance to transport freshwater to sit (miles)</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Distance to transport recycled fluid site (miles)</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Distance to transport fluid to disposal well (miles)</td>
<td>10</td>
<td>50</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Flowback fraction</td>
<td>2.75</td>
<td>0.2</td>
<td>0.25</td>
<td>0.9</td>
</tr>
<tr>
<td>Recycled fraction</td>
<td>0.2</td>
<td>0.95</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>Diesel Fuel Consumption (gal/well)</td>
<td>4771</td>
<td>4444</td>
<td>4063</td>
<td>5689</td>
</tr>
</tbody>
</table>

Vertical and horizontal drilling rigs utilize high-horsepower engines to power their draw works, drills, mud pumps, and other equipment. Most current drilling rigs are electrical. These types of rigs use two to three stationary engines which power electric generators. The generated electricity can then be distributed to components as needed. These engines operate continuously from the time that rig setup begins until drilling of the well is completed and it is cemented and cased. The Texas Commission on Environmental Quality (TCEQ), in a survey of 1261 wells, found that the average drilling duration for a well was between 8 and 27 days depending on the depth of the vertical section and length of the horizontal section. On average these drilling rigs consisted of 2.15 engines, with an average size of 1381 horsepower per engine. These engines operate 62.6 hours per 1000 feet drilled and a load of 48.5% on average [11]. A report by Global Hunter Securities in collaboration with Prometheus Energy states that the average drilling rig uses between 62.5 and 83.3 gallons of diesel fuel per hour, resulting in 1500 to 2200 gallons of diesel fuel per day [12]. In a study by ANL, it was assumed that a rig ran on one 2000 horsepower engine at 45% load and consumed 0.06 gal/hp/hr. The study focused on four major shale plays and the
fuel consumed in these plays depended mostly on the depth of the well and the amount of time required for drilling. The estimated diesel fuel consumption levels of several different plays are shown in Table 2 [10].

Table 2: Diesel Fuel Consumed for Drilling and Pad Construction [10]

<table>
<thead>
<tr>
<th>Shale Play</th>
<th>Barnett</th>
<th>Marcellus</th>
<th>Fayetteville</th>
<th>Haynesville</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diesel Fuel (gallons)</td>
<td>61,768</td>
<td>55,080</td>
<td>43,041</td>
<td>85,845</td>
</tr>
</tbody>
</table>

Hydraulic fracturing pumps are also usually powered by high-horsepower diesel engines. Some fracturing fleets are now using dual-fuel diesel-natural gas engines in an attempt to reduce cost. A typical fracturing fleet features engines combined capacities over 20,000 horsepower. On average, 8 to 12 pumps are used per site, but sites can require up to two dozen fracturing pumps, each of which is rated between 1500 and 2500 horsepower. Fracturing a well can consume between 2000 and 8000 gallons of diesel fuel per day. A more in depth study by the University of Michigan performed site surveys in the Marcellus and Eagle Ford shale plays. The Eagle Ford and Marcellus fracturing fleets were made up of 14 and 16 engines, respectively. The engines at both sites were Caterpillar 3512C models rated at 2250 horsepower. The data collected during the survey is presented in Table 3 and gives an estimate of fuel consumed per fracturing job [13].

Table 3: Aspects of Fracturing Jobs in Various Plays [13]

<table>
<thead>
<tr>
<th></th>
<th>Eagle Ford</th>
<th>Marcellus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fracturing Stages per Job</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>Pumping hours per job (hrs)</td>
<td>27</td>
<td>31</td>
</tr>
<tr>
<td>Time at &quot;Reserve power&quot; mode-between stages (hrs)</td>
<td>51</td>
<td>26</td>
</tr>
<tr>
<td>Standby Time</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>Total Hours per Job (hrs)</td>
<td>98</td>
<td>82</td>
</tr>
<tr>
<td>Number of Fracturing Pumps per Job</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>Number of Active Pumps per Job</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Number of Pumps at &quot;Reserve Power&quot; per Job</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of back-up Pumps per Job</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Pump Horsepower</td>
<td>2250</td>
<td>2250</td>
</tr>
<tr>
<td>RPM @ Pumping Mode</td>
<td>1800</td>
<td>1800</td>
</tr>
<tr>
<td>RPM @ &quot;reserve power&quot; mode</td>
<td>700</td>
<td>700</td>
</tr>
<tr>
<td>Engine Load @ pumping mode</td>
<td>60%</td>
<td>50%</td>
</tr>
<tr>
<td>Engine Load @ &quot;reserve power&quot; mode</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>Fuel Usage per Job (gal)</td>
<td>22,100</td>
<td>20,800</td>
</tr>
</tbody>
</table>
Another study by Apache Corporation states that a typical fracturing job in the Granite Wash play of Texas and Oklahoma uses about 36,000 gallons of diesel fuel. According to Apache, the oil and gas industry used more than 700 million gallons of diesel fuel for hydraulic fracturing in 2012 [2]. Hydraulic fracturing is a large consumer of diesel fuel due to the extensive operation of the high-horsepower engines involved. A variety of different engines are used by the industry and many are being converted to dual-fuel or replaced with dedicated natural gas engines in order to displace the diesel fuel costs by these prime-movers.

### 2.2 Engines Used

The increase in the availability of natural gas has led companies to look for new ways to utilize this resource. Engines used to service the industry are typically heavy-duty diesel engines, however, many companies are now looking to displace diesel fuel with natural gas, because of the potential cost savings. Industry is accomplishing this by switching to dual-fuel or dedicated natural gas engines either with aftermarket kits or by purchasing new dedicated engines. This is more common in off-road engines than those used in OTR trucks. One example of producing natural gas engines for OTR trucks is by Cummins. Cummins-Westport has developed both dedicated natural gas engines as well as dual-fuel high pressure direct injection (HPDI) engines. Cummins dedicated OTR engines include the ISLG and the ISX12G. The latter is the more commonly used for heavy-duty hauler applications such as those used by the natural gas service industry. The ISX12G has a displacement of 11.9 liters and is rated up to 400 horsepower with 1450 lb-ft of torque [14]. The HPDI 2.0 is a dual fuel engine which allows companies to use natural gas as the primary fuel and diesel fuel as a pilot for compression ignition. This engine allows for over 90% diesel fuel substitution. Westport HPDI 2.0 also claims efficiency of 44% compared to 37% of conventional spark ignited dedicated natural gas systems [15]. These engines are not often seen in industry because most E&P companies rely on third parties for their hauling needs. These companies tend to use lower cost, more conventional options. The Cummins ISX15 was observed to be the most common model of heavy-duty truck engine servicing the industry. These engines are conventional diesel engines with horsepower options ranging from 400 to 600 hp [16]. A plethora of other conventional heavy-duty diesel engines are also found in trucks servicing the industry. Conventional diesel engine manufacturers Caterpillar, Mack, Volvo and Detroit Diesel were observed in vehicles servicing the industry.

There are fewer types of engines being used by drilling and hydraulic fracturing companies. Dual-fuel and dedicated natural gas engines are more commonly seen in the off-road sector of the industry compared to OTR. Many engines used in the industry are manufactured by Caterpillar, Cummins, and Waukesha. Caterpillar produces several models used in the industry. The Caterpillar 3512 and 3516 are two engines which are common for drilling and hydraulic fracturing applications. These engines have a variety of
power ratings and fueling options available. The 3512 is commonly used to power drill rigs and is available as an industrial engine or as an electric generator set (genset) package outfitted with Caterpillar’s Dynamic Gas Blending (DGB) Kit. These engines are rated at 1500 horsepower at 1800 rpm [17]. The Caterpillar 3516 engine is also used in high-horsepower drilling and hydraulic fracturing applications. It is rated at 2000 horsepower at 1800 rpm [18]. These engines can also be outfitted with the Caterpillar DGB kits. The DGB kit offers the option for the engine to run in diesel-natural gas dual-fuel mode with diesel fuel substitution up to 60% [19]. Caterpillar also has a gas model of the 3516 which can run exclusively on natural gas. Waukesha is another producer of dedicated natural gas engines which are being used in the unconventional well completion industry. These engines have power ratings between 265 and 1175 horsepower [20]. Another engine used by hydraulic fracturing fleets is the Cummins QSK50. This engine is rated at 1820 horsepower at 1800 rpm. Altronic GTI has developed an aftermarket dual-fuel kit, referred to as the Bi-Fuel System. This kit can be used with a variety of engines and offers substitution rates up to 70% [21]. There are other, less common engines, both diesel and dedicated natural gas available, which allow companies to utilize the gas available at well sites.

The use of these technologies changes the emissions profiles of engines being used in the field. In order to properly quantify the impact of these emissions and ultimately ensure compliance, the in-field activity of these engines should be replicated. Replicable activity could be performed in an engine test cell to observe differences in emissions.

2.3 Emissions, Testing and Regulations
Current emissions standards and engine testing procedures are outlined by the US EPA and CARB. These regulations are outlined in several parts of the CFR, including: 86, 89, 1036, 1039, and 1065. The emissions standards of highway heavy-duty compression ignition engines were introduced in 1974 based on the introduction of the Clean Air Act (CAA) of 1970, with more stringent standards being introduced over the years. A major push for more stringent emissions standards came in 1998 as part of the consent decrees between the EPA, CARB, the US Department of Justice and settling US heavy-duty engine manufacturers. The primary focus of the consent decrees was high NOx emissions from OTR heavy-duty diesel engines. The emissions standards outlined by the CFR Title 40 Chapter I Subchapter C Part 86 Subpart A are shown in Table 4 [22].
Table 4: Onroad Heavy Duty Diesel Emissions (g/bhp-hr) [22]

<table>
<thead>
<tr>
<th>Year</th>
<th>CO</th>
<th>HC</th>
<th>HC + NOx</th>
<th>NOx</th>
<th>PM General</th>
<th>PM Urban Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1974</td>
<td>40</td>
<td>-</td>
<td>16</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1979</td>
<td>25</td>
<td>1.5</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1985</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>10.7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1987</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>10.7</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1988</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>10.7</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>6</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1991</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>5</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>1993</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>5</td>
<td>0.25</td>
<td>0.1</td>
</tr>
<tr>
<td>1994</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>5</td>
<td>0.1</td>
<td>0.07</td>
</tr>
<tr>
<td>1996</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>5</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>1998</td>
<td>15.5</td>
<td>1.3</td>
<td>-</td>
<td>4</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2002</td>
<td>15.5</td>
<td>-</td>
<td>2.4</td>
<td>-</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2007-10</td>
<td>15.5</td>
<td>0.14</td>
<td>-</td>
<td>0.2</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>2015*</td>
<td>15.5</td>
<td>0.14</td>
<td>-</td>
<td>0.02</td>
<td>0.01</td>
<td></td>
</tr>
</tbody>
</table>

*California Optional Low NOx Standards [22]

In addition to standard emission regulations, the US has recently implemented greenhouse gas (GHG) and fuel consumption standards for heavy- and medium-duty vehicles. These new regulations were developed by the EPA, National Highway Traffic Safety Administration (NHTSA) and the DOT and apply to vehicle model years 2014-2018. The standards apply to all onroad vehicles with a gross vehicle weight rating (GVWR) above 8500 pounds. Different GHG standards apply to different types of heavy-duty vehicles. Onroad heavy-duty vehicles are subdivided into three categories for the purpose of applying these fuel economy and GHG standards: combination tractors, heavy-duty pickup trucks, and vans and vocational vehicles. Typically only combination tractors and vocational vehicles are seen in unconventional well completion. Vehicles with a model year of 2014 or newer must be certified on the chassis dynamometers and their respective engines must be certified outside of the vehicle on an engine dynamometer. The vehicle emissions standards outlined in the CFR Title 40 Chapter I Subchapter U Part 1036 Subpart B are presented in grams per ton-mile and shown in Table 5 [23]. The engine emissions standards are presented in grams per brake-horsepower-hour and are presented in Table 6 [23].
### Table 5: Heavy-Duty Vehicle GHG Emissions [23]

<table>
<thead>
<tr>
<th>Category</th>
<th>Vehicle Type</th>
<th>EPA CO₂ Emissions</th>
<th>NHTSA Fuel Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>g/ton-mile</td>
<td>gal/1,000 ton-mile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low Roof</td>
<td>Mid Roof</td>
</tr>
<tr>
<td>Combination Tractors</td>
<td>Day Cab Class 7</td>
<td>104</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>Day Cab Class 8</td>
<td>80</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Sleeper Cab Class 8</td>
<td>66</td>
<td>73</td>
</tr>
<tr>
<td>Vocational Vehicles</td>
<td>Light Heavy Class 2b-5</td>
<td>373</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium Heavy Class 6-7</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Heavy Heavy Class 8</td>
<td>222</td>
<td></td>
</tr>
</tbody>
</table>

These emissions standards apply to heavy-duty engines dependent on their model year. Engines must comply with the most recent emissions standards relative to their time of manufacturing. Generally, California has more stringent standards than the rest of the country and often requires compliance several years ahead of the rest of the US. The emissions standards of the CFR are not based on heavy-duty vehicles, but instead are focused on the engines used in the vehicles. This requires engines to be tested and certified on engine dynamometers outside of the vehicle, rather than in the vehicles on chassis dynamometers. The engines must meet brake-specific emissions standards during several different test procedures. The primary engine test cycle over which onroad heavy-duty diesel engines are certified is the transient Federal Test Procedure (FTP) engine dynamometer test cycle defined in CFR Title 40 Subchapter C Part 86 Appendix I. The FTP cycle is also used for certification of the vocational engines.

### Table 6: Heavy-Duty Engine GHG Emissions [23]

<table>
<thead>
<tr>
<th></th>
<th>Year</th>
<th>CO₂ Emissions</th>
<th>Fuel Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>g/bhp-hr</td>
<td>gal/100 bhp-hr</td>
</tr>
<tr>
<td>Combination Tractors</td>
<td>2014</td>
<td>502</td>
<td>4.93</td>
</tr>
<tr>
<td></td>
<td>2017</td>
<td>487</td>
<td>4.78</td>
</tr>
<tr>
<td></td>
<td>2014</td>
<td>475</td>
<td>4.67</td>
</tr>
<tr>
<td></td>
<td>2017</td>
<td>460</td>
<td>4.52</td>
</tr>
<tr>
<td>Vocational Vehicles</td>
<td>2014</td>
<td>600</td>
<td>5.89</td>
</tr>
<tr>
<td></td>
<td>2017</td>
<td>576</td>
<td>5.66</td>
</tr>
<tr>
<td></td>
<td>2014</td>
<td>600</td>
<td>5.89</td>
</tr>
<tr>
<td></td>
<td>2017</td>
<td>576</td>
<td>5.66</td>
</tr>
<tr>
<td></td>
<td>2014</td>
<td>567</td>
<td>5.57</td>
</tr>
<tr>
<td></td>
<td>2017</td>
<td>555</td>
<td>5.45</td>
</tr>
<tr>
<td>HH Gasoline Engines</td>
<td>2016</td>
<td>627</td>
<td>7.06</td>
</tr>
</tbody>
</table>
under the new GHG standards. This cycle is a series of speed and torque points normalized to a percentage so that it can be applied to all engines. The normalized FTP transient cycle is shown in Figure 3 [24].

![Figure 3: Transient FTP Test][24]

The FTP cycle was developed simultaneously with the EPA Urban Dynamometer Driving Schedule (UDDS) chassis dynamometer drive cycle which was based on chase car data from New York and Los Angeles in the 1970s using a Markov chain approach. In addition to the FTP transient cycle, the Supplemental Emission Test (SET) and Not-to-Exceed (NTE) standards were implemented as part of the 1998 consent decrees [22]. The SET test is a steady state and ramped mode test procedure used to ensure that diesel engines could maintain emissions standards in real world situations, such as long haul freeway activity. This test is applicable to all engines manufactured in 2007 or newer and to tractor engines under the new GHG standards. The SET consists of 13 modes of steady-state engine dynamometer testing. The SET test is defined in the CFR Title 40 Subchapter C Part 86 Subpart N Section 1360 and outlined in Table 7 [25].
<table>
<thead>
<tr>
<th>RMC Mode</th>
<th>2007-2009</th>
<th>2010 &amp; later</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time, s</td>
<td>Speed</td>
</tr>
<tr>
<td>1a</td>
<td>Steady-state</td>
<td>170</td>
</tr>
<tr>
<td>1b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>2a</td>
<td>Steady-state</td>
<td>170</td>
</tr>
<tr>
<td>2b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>3a</td>
<td>Steady-state</td>
<td>102</td>
</tr>
<tr>
<td>3b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>4a</td>
<td>Steady-state</td>
<td>100</td>
</tr>
<tr>
<td>4b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>5a</td>
<td>Steady-state</td>
<td>103</td>
</tr>
<tr>
<td>5b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>6a</td>
<td>Steady-state</td>
<td>194</td>
</tr>
<tr>
<td>6b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>7a</td>
<td>Steady-state</td>
<td>219</td>
</tr>
<tr>
<td>7b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>8a</td>
<td>Steady-state</td>
<td>220</td>
</tr>
<tr>
<td>8b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>9a</td>
<td>Steady-state</td>
<td>219</td>
</tr>
<tr>
<td>9b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>10a</td>
<td>Steady-state</td>
<td>171</td>
</tr>
<tr>
<td>10b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>11a</td>
<td>Steady-state</td>
<td>102</td>
</tr>
<tr>
<td>11b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>12a</td>
<td>Steady-state</td>
<td>100</td>
</tr>
<tr>
<td>12b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>13a</td>
<td>Steady-state</td>
<td>102</td>
</tr>
<tr>
<td>13b</td>
<td>Transition</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Steady-state</td>
<td>168</td>
</tr>
</tbody>
</table>
NTE standards are applied during FTP or SET tests and were implemented to ensure emissions standards were upheld during a wide range of speed and load combinations common to real world activity. These were introduced with the consent decrees of 1998 and apply to all 2004 and newer engine models. The NTE “zone” is defined in the CFR Title 40 Subchapter C Part 86 Subpart N Section 1370 and is based on common heavy-duty diesel operation as is shown in Figure 4 [26]. The implementation of a plethora of diesel emission control strategies came in response to these new regulations, including exhaust gas recirculation (EGR), diesel oxidation catalysts (DOCs), diesel particulate filters (DPFs), and selective catalytic reduction (SCR).

![Figure 4: Basic NTE Zone](image)

In addition to onroad engines, the CFR also outlines emissions standards and test regulations for nonroad engines, such as those used in drilling and hydraulic fracturing. These engines are considered stationary engines by the EPA and distinguished from “nonroad” engines by the fact that they are not used to propel a motor vehicle of any kind [27]. Although stationary engines are differentiated from nonroad engines based on definition, they must meet many of the same emissions standards. Emissions regulations for these stationary diesel engines are outlined in the CFR Title 40 Chapter I Subchapter C Part 60 Subpart III. Stationary engines with a displacement less than ten liters per cylinder, a horsepower rating less than 3000 and a model year newer than 2007 must comply with the nonroad emissions standards of Tiers 2/3/4 [28]. Tier 1-3 emissions standards of nonroad engines are outlined in the CFR Title 40 Subchapter C Part
89 Subpart B Section 112 and shown in Table 8 [28]. The initial Tier 1 standards were phased in from 1996 to 2000 after being implemented in 1994.

Table 8: Tier 1-3 Nonroad Emissions Standards g/kWh (g/bhp-hr) [28]

<table>
<thead>
<tr>
<th>Engine Power</th>
<th>Tier</th>
<th>Year</th>
<th>CO (g/kWh)</th>
<th>HC (g/kWh)</th>
<th>NMHC+NOx (g/kWh)</th>
<th>NOx (g/kWh)</th>
<th>PM (g/kWh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kW &lt; 8</td>
<td>Tier 1</td>
<td>2000</td>
<td>8.0 (6.0)</td>
<td>-</td>
<td>10.5 (7.8)</td>
<td>-</td>
<td>1.0 (0.75)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2005</td>
<td>8.0 (6.0)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.8 (0.6)</td>
</tr>
<tr>
<td>8 ≤ kW &lt; 19</td>
<td>Tier 1</td>
<td>2000</td>
<td>6.6 (4.9)</td>
<td>-</td>
<td>9.5 (7.1)</td>
<td>-</td>
<td>0.8 (0.6)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2005</td>
<td>6.6 (4.9)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.8 (0.6)</td>
</tr>
<tr>
<td>19 ≤ kW &lt; 37</td>
<td>Tier 1</td>
<td>1999</td>
<td>5.5 (4.1)</td>
<td>-</td>
<td>9.5 (7.1)</td>
<td>-</td>
<td>0.8 (0.6)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2004</td>
<td>5.5 (4.1)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.6 (0.45)</td>
</tr>
<tr>
<td>37 ≤ kW &lt; 75</td>
<td>Tier 1</td>
<td>1998</td>
<td>-</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2004</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.4 (0.3)</td>
</tr>
<tr>
<td></td>
<td>Tier 3</td>
<td>2008</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>4.7 (3.5)</td>
<td>-</td>
<td>-†</td>
</tr>
<tr>
<td>75 ≤ kW &lt; 130</td>
<td>Tier 1</td>
<td>1997</td>
<td>-</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2003</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>6.6 (4.9)</td>
<td>-</td>
<td>0.3 (0.22)</td>
</tr>
<tr>
<td></td>
<td>Tier 3</td>
<td>2007</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>4.0 (3.0)</td>
<td>-</td>
<td>-†</td>
</tr>
<tr>
<td>130 ≤ kW &lt; 225</td>
<td>Tier 1</td>
<td>1996</td>
<td>11.4 (8.5)</td>
<td>1.3 (1.0)</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>0.54 (0.4)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2003</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>6.6 (4.9)</td>
<td>-</td>
<td>0.2 (0.15)</td>
</tr>
<tr>
<td></td>
<td>Tier 3</td>
<td>2006</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>4.0 (3.0)</td>
<td>-</td>
<td>-†</td>
</tr>
<tr>
<td>225 ≤ kW &lt; 450</td>
<td>Tier 1</td>
<td>1996</td>
<td>11.4 (8.5)</td>
<td>1.3 (1.0)</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>0.54 (0.4)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2001</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>6.4 (4.8)</td>
<td>-</td>
<td>0.2 (0.15)</td>
</tr>
<tr>
<td></td>
<td>Tier 3</td>
<td>2006</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>4.0 (3.0)</td>
<td>-</td>
<td>-†</td>
</tr>
<tr>
<td>450 ≤ kW &lt; 560</td>
<td>Tier 1</td>
<td>1996</td>
<td>11.4 (8.5)</td>
<td>1.3 (1.0)</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>0.54 (0.4)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2002</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>6.4 (4.8)</td>
<td>-</td>
<td>0.2 (0.15)</td>
</tr>
<tr>
<td></td>
<td>Tier 3</td>
<td>2006</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>4.0 (3.0)</td>
<td>-</td>
<td>-†</td>
</tr>
<tr>
<td>kW ≥ 560</td>
<td>Tier 1</td>
<td>2000</td>
<td>11.4 (8.5)</td>
<td>1.3 (1.0)</td>
<td>-</td>
<td>9.2 (6.9)</td>
<td>0.54 (0.4)</td>
</tr>
<tr>
<td></td>
<td>Tier 2</td>
<td>2006</td>
<td>3.5 (2.6)</td>
<td>-</td>
<td>6.4 (4.8)</td>
<td>-</td>
<td>0.2 (0.15)</td>
</tr>
</tbody>
</table>

† Not adopted, engines must meet Tier 2 PM standard.

Tier 4 standards were implemented in 2004 and phased in from 2008-2015. These emissions standards are defined in the CFR Title 40 Chapter I Subchapter U Part 1039 Subpart B Section 101 and are shown in Table 9 [28].
Table 9: Tier 4 Nonroad Emissions Standards g/kWh (g/bhp-hr) [28]

<table>
<thead>
<tr>
<th>Engine Power</th>
<th>Year</th>
<th>CO</th>
<th>NMHC</th>
<th>NMHC+NOx</th>
<th>NOx</th>
<th>PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>kW &lt; 8</td>
<td>2008</td>
<td>8.0 (6.0)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.4a (0.3)</td>
</tr>
<tr>
<td>8 ≤ kW &lt; 19</td>
<td>2008</td>
<td>6.6 (4.9)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.4 (0.3)</td>
</tr>
<tr>
<td>19 ≤ kW &lt; 37</td>
<td>2008</td>
<td>5.5 (4.1)</td>
<td>-</td>
<td>7.5 (5.6)</td>
<td>-</td>
<td>0.3 (0.22)</td>
</tr>
<tr>
<td></td>
<td>2013</td>
<td>5.5 (4.1)</td>
<td>-</td>
<td>4.7 (3.5)</td>
<td>-</td>
<td>0.03 (0.022)</td>
</tr>
<tr>
<td>37 ≤ kW &lt; 56</td>
<td>2008</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>4.7 (3.5)</td>
<td>-</td>
<td>0.3b (0.22)</td>
</tr>
<tr>
<td></td>
<td>2013</td>
<td>5.0 (3.7)</td>
<td>-</td>
<td>4.7 (3.5)</td>
<td>-</td>
<td>0.03 (0.022)</td>
</tr>
<tr>
<td>56 ≤ kW &lt; 130</td>
<td>2012-2013</td>
<td>5.0 (3.7)</td>
<td>0.19 (0.14)</td>
<td>-</td>
<td>0.40 (0.30)</td>
<td>0.02 (0.015)</td>
</tr>
<tr>
<td>130 ≤ kW ≤ 560</td>
<td>2011-2014</td>
<td>3.5 (2.6)</td>
<td>0.19 (0.14)</td>
<td>-</td>
<td>0.40 (0.30)</td>
<td>0.02 (0.015)</td>
</tr>
</tbody>
</table>

Engines above 560 kW are grouped into two categories based on whether or not they act as part of a genset. The standards for engines of this size are shown in Table 10 [28].

Table 10: Tier 4 Emissions Standards for Nonroad Engines Greater than 560 kW [28]

<table>
<thead>
<tr>
<th>Year</th>
<th>Category</th>
<th>CO</th>
<th>NMHC</th>
<th>NOx</th>
<th>PM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generator sets &gt; 900 kW</td>
<td>3.5</td>
<td>0.40</td>
<td>0.67</td>
<td>0.10 (0.075)</td>
</tr>
<tr>
<td>2011</td>
<td>All engines except gensets &gt; 900 kW</td>
<td>3.5</td>
<td>0.40</td>
<td>3.5</td>
<td>0.10 (0.075)</td>
</tr>
<tr>
<td></td>
<td>Generator sets</td>
<td>3.5</td>
<td>0.19</td>
<td>0.67</td>
<td>0.03 (0.022)</td>
</tr>
<tr>
<td>2015</td>
<td>All engines except gensets</td>
<td>3.5</td>
<td>0.19</td>
<td>3.5</td>
<td>0.04 (0.03)</td>
</tr>
</tbody>
</table>

Nonroad engines must also be certified based on the emissions standards outlined by the CFR. These engines are tested on engine dynamometers and certified based on the procedures of CFR Title 40 Chapter I Subchapter U Part 1039. Tier 1-3 engines must be certified over steady state tests. The ISO 8178 is the standard for nonroad engine emissions testing. This test, also known as the Non-Road Steady Cycle (NRSC), consists of eight modes of steady state operation. The test, applicable to most engines including those used in drilling and hydraulic fracturing, is outlined in Table 11 [29]. The test sequence applicable to high-horsepower drilling and hydraulic fracturing engines is known as D2.
Table 11: ISO 8178 Test Cycle and Weighting Factors [29]

<table>
<thead>
<tr>
<th>Mode number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Torque, %</td>
<td>100</td>
<td>75</td>
<td>50</td>
<td>25</td>
<td>10</td>
<td>100</td>
<td>75</td>
<td>50</td>
<td>25</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rated speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intermediate speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Off-road vehicles | Type C1 | 0.15 | 0.15 | 0.15 | -    | 0.1  | 0.1  | 0.1  | -    | -    | 0.15 |
|                   | Type C2 | -    | -    | -    | 0.06 | -    | 0.02 | 0.05 | 0.3  | 0.3  | 0.1  |

| Constant speed   | Type D1 | 0.3  | 0.5  | 0.2  | -    | -    | -    | -    | -    | -    | -    |
|                   | Type D2 | 0.05 | 0.25 | 0.3  | 0.3  | 0.1  | -    | -    | -    | -    | -    |

Some Tier 4 engines must also undergo transient testing over the nonroad transient cycle (NRTC). This testing was applied starting in 2011 and was phased in based on engine horsepower ratings. The normalized NRTC cycle is shown in Figure 5 [30].

![Figure 5: Nonroad Transient Test](image)

Engines above 560 kW are not subject to transient testing, so it is generally not applied to engines servicing unconventional well completion. Tier 4 engines need to meet NTE standards regardless of the test cycle over which they are applied. The NTE standards were implemented in 2011 for engines above 130 kW. The primary reason for applying NTE standards was to prevent the use of defeat devices.
The cycles defined in the CFR are the ones used for certification of both onroad and nonroad engines. The results obtained from these cycles determine whether or not an engine can operate with acceptable emissions levels while being used for its intended application in the real world. However, these cycles are not representative of all types of engine operation. This fact has led researchers to attempt to create test cycles which better represent real world engine activity. Engine cycles developed by third party researchers often have the benefit of using the activity of the specific industry being studied. This allows for more representative emissions of an industry to be determined. A review of literature was conducted to investigate the findings of research in this area.
3. Review of Literature

3.1 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) simulations are used in a wide variety of sciences as a way to estimate probabilities and perform numerical integration. MCMC is defined by Geyer as “a general method for the simulation of stochastic processes having probability densities known up to a constant proportionality” [31]. The first usage of such Monte Carlo techniques appeared in 1953 article by Metropolis et al [32]. Monte Carlo methods came to widespread use much later in the mid-to-late 1990s. The reason that MCMC techniques were not developed further between these time periods was largely due to the lack of computing power of the time. A history of the development and use of MCMC is available from Robert and Casella [33].

The theory behind MCMC can be subdivided into Markov chain techniques and Monte Carlo simulations. A Markov chain is a stochastic process that has stationary transition probabilities. The Markov chain is a string of concatenated values from a defined “state space” or data being sampled [34]. A Markov chain is often described by saying that there exists some state, or event, and that state contains a transitional probability matrix which gives the likelihood of the next state. The next state in the chain depends only on the current state and its transition probability matrix [35]. The Markov chain is considered “stationary” if any state in the chain has a transitional probability distribution that is independent of the location of the state [35]. This means that the probability of reaching the next state is the same for all states in the chain and likely dependent of the initial distribution. If a Markov chain is not stationary than the transition probabilities will change as the chain moves further from the initial state. As the length of the chain increases, then the distribution of the states becomes less dependent on the initial state and more dependent on the transition probabilities [34]. If an overall distribution defines each states transition probability matrix and if the chain is allowed to become “long” the distribution of the chain itself will approach the overall distribution [34]. The most effective way to reach a desired distribution is by allowing the Markov chain to grow to an array length that is considered large. The actual size necessary to properly represent a distribution will be dependent on the size of the data set and the problem at hand. Markov chains by themselves have been used extensively in cycle development, as shown in the next section.

MCMC simulations expand on the idea of Markov chains, but flip the objective of the problem. Rather than attempting to find the distribution of a specific Markov chain, MCMC starts with a defined distribution and attempts to create a Markov chain that has this same distribution [34]. When this is the case, the distribution of interest is known and the Markov chain is created in the same stochastic manner until it has an array length which is long enough to adequately represent the distribution attempted to be
found. Some issues that are encountered in implementing MCMC include: number of iterations, whether to build one long chain or several smaller chains, determining a starting point, parameterization of variables, and how to use auxiliary variables to affect the current distribution [36]. The problem at hand often helps to clarify some of these issues. When the Markov chain must have a specified length, or must start in a specific state, for example, the answer to these questions is apparent. Often computation or time limitations play a role in determining number of iterations. There are also numerous techniques that can be used to address these issues if they still arise. For example, to eliminate the issue of choosing a starting point, one can use what Geyer calls “burn-in.” This technique involves throwing away a section of the chain in order to attempt to eliminate initial state bias [31]. MCMC techniques have been applied to genomics, variable selection in regression, spatial statistics, longitudinal studies, mathematical chemistry, statistical physics or when more simplistic techniques cannot be used [33]. Techniques have been used for estimating vehicle speeds and could be used to develop cycles [37] [38].

3.2 Cycle Development
The most relevant example of engine cycle development using MCMC techniques was one of the first studies of its kind. Smith of the EPA [38] focused on developing universal drive and engine test cycles for OTR heavy-duty vehicles. The study focused on 55 trucks and 5 buses in New York and Los Angeles. An MCMC technique was used to develop the cycles involved in the study. Both a drive cycle and an engine cycle were developed using these techniques. The drive cycle was built by initially creating a speed frequency occurrence matrix, populated with the initial speeds and first derivatives of speed segments. Initial speeds of segments defined the rows and derivatives of speed defined the columns of the matrix. This matrix was normalized so that it contained empirical conditional probabilities. The speed matrix was transformed further into a transition probability matrix by converting a density function of each row into a distribution function. Chassis cycles were generated by using this transition-probability matrix while using random number generation to first determine a change in speed and ultimately the next speed point in the cycle. This process continued until the desired cycle length was reached.

A similar approach was used in developing an engine dynamometer cycle. The engine matrix was populated with columns of points of initial engine speed and power versus rows of final engine speed and power of segments. Percent engine speed were split into 87 classes ranging from -22% to 150% in increments of 2%. The percentage of rated power ranged from motoring, any value less than 0%, to 100% in increments of 10% resulting in a total of 12 classes. Combining these created a matrix that was again converted into a distribution function in order to develop the transition probability matrix. Each cycle generated started with an idle condition. Random number generation selected the next speed and power combination based on the matrix. This process continued until the minimum desired duration was
reached, then continued until another idle condition was selected. The cycle ended when a non-idle point was selected and the point preceding this selection marked the end of the cycle. If idle times were not appropriate based on the data distribution then accelerations and decelerations were forced. As the cycle was built, a cycle matrix was created for comparison with the input data set.

Once cycles were completed they were sent through a statistical filter, which was a method of comparing the cycle to the input chase data. Some of the statistics that were used to compare the input data to the cycles included: average percent power, percent time motoring, percent time at zero power, percent time cruising, percent time accelerating, percent time decelerating, average engine speed percentage and percent time at zero engine speed. The primary evaluation technique was a K-S one sample test. This required distribution functions of the input and cycle data matrices and three submatrices of cruise, acceleration, and deceleration. Criteria were developed to determine if the cycles passed the test. If more than one cycle met the criteria desired, then a manual inspection was completed to select the best cycle. This focused first on the results of the K-S test, second on the average engine speed and power, and third on the differences in percentages. The work performed by Smith [38] is still the most relevant example of engine test cycle development using a MCMC method.

The EPA performed a Federal Test Procedure Review Project in May of 1993 [39]. The preliminary technical report contained a review of “Test Cycle Development Methods and Approach.” This review defined a driving cycle as “a time series of vehicle speeds occurring at successive time points which attempts to synthesize real driving conditions in terms of parameters such as speed, acceleration, specific power, trip patterns, road grade, and temperature.” The cycle must also be a reasonable length, drivable, and must be able to be performed by a dynamometer. These aspects apply not only to drive cycles, but also to engine test cycles. The review project examined three types of cycle generation: segment splicing, Monte Carlo simulation, and the “engineering” approach. Segment splicing is the most widely accepted use of cycle generation and consists of piecing together segments of actual speed-time data collected in the field. The best feature of this type of cycle generation, according to the review, is it ensures that real world type activity is mimicked on laboratory dynamometers. Questions raised in the use of segment splicing include: selecting segments that most adequately represent the entire set of data and meet cycle requirements. Other issues are encountered if the segments are not micro-trips, but are kinematic sequences or modal events. These types of segments can produce problems during concatenation.

The Monte Carlo simulation involves developing a model for describing how actual driving occurs. This type of model relies on deterministic probabilities and stochastic elements to account for known physical properties of driving. Generally, when creating drive cycles, random number generation is incorporated along with a probability analysis. The primary challenge when using this technique is developing a model
that creates cycles which are similar to in-use data. This is accomplished by the EPA’s heavy-duty test cycle with the second-by-second Markov process, in which the probability of the next speed depends only on the current speed. This model uses in-use frequency distributions. The fear of these models is that they are not sufficiently complex to capture all important emissions related elements. The Monte Carlo simulation approach offers considerable flexibility in modeling, but results in “artificially” created cycles. They are considered artificial because the sequences may not have occurred in real world use.

The final type of cycle development reviewed was the “engineering” construction of cycles which consists of building cycles not based on frequency or in-use data. These cycles are essentially designed by the user to meet certain conditions. Test cycle validation is generally performed by matching various characteristics of the cycle to those of in-use driving. A cycle is considered valid by the EPA if it has statistical properties that are reasonably close to those of the in-use data used for generating it. This is generally at the discretion of the creator, but often involves a set of arbitrary statistical filters. This procedure is generally followed by a more in-depth analysis to select the most realistic cycle. Most of the MCMC approaches in literature focus on the development of drive cycles rather than engine cycles such as those created by the EPA [39].

Gong et al. [40] of Ohio State University’s Center for Automotive Research generated driving cycles of Plug-in Hybrid Electric Vehicle (PHEV) using a Markov chain technique. Researchers collected data from nine PHEV vehicles consisting of over 70,000 miles of data that were segmented into 530 “sessions” defined from key-on to key-off. These sessions are described using principle component analysis (PCA) which transforms a large number of related variables into a small number of uncorrelated variables known as principle components. Eighteen metrics were initially defined, but were reduced to five principle components that were ultimately used to describe each speed vector or session. A Markov chain model was used to define the next state of velocity and acceleration based on the current velocity and acceleration. In order to use the Markov chain, transition probabilities were determined using “nearest-neighbor quantization,” which involved mapping the sequence of observations into a sequence of quantized states. The Markov chain approach uses the frequency of each of the states based on the current state. Each trip was made up of an arbitrary number of segments pieced together based on the Markov chain and followed by a period of idle before the end of the cycle [40].

Dembski et al. [41] also of Ohio State University, used similar PCA techniques and calculations of sequence occurrence probabilities to analyze drive cycles. Kinematic sequences defined by velocity and acceleration were developed and linked together based on the probability of each type of sequence occurring, essentially utilizing a Markov chain to create drive cycles [41].
Similar Markov chain techniques were used by Ivanco et al [43]. This worked defined states of activity as a function of speed and acceleration. A transition matrix was again used in this work with a matrix defined by acceleration and velocity, discretized into “states.” One difference in the transition matrix, compared to other works, was the use of interpolation if an empty “state” was selected by the chain so that the next state would be selected from a state populated with data. A mean speed correction function was also used to decrease calculation time and increase likelihood of constructing a cycle with the desired characteristics. This function calculated mean speed periodically as the cycle was constructed and if it showed a major difference from the reference cycle modified the acceleration departure state in the current iteration of transition [42].

Three other uses of Markov chain techniques to produce drive cycles were co-authored by Niemeier et al [44] [45] [41], of the University of California-Davis. These works focus on reviews of cycle construction methodologies that have been used and new techniques to develop drive cycles. The first work published in 2002 was authored by Lin et al. This publication used a “speed-acceleration frequency distribution (SAFD)” as the primary matrix for cycle generation. The work did not employ micro-trips due to concern that they did not adequately replicate actual frequency, duration, and intensity of modal events. The modal event bins were selected via a stochastic probability. Once a bin was selected the modal event within the bin that most improves the SAFD is selected. A performance value (PV) was used for final cycle comparison. This PV compared statistics of the cycle to those from the in-use data. The metrics used for comparison included: average speed, average acceleration rate, maximum speed, minimum speed, the speed of the 95th percentile, the maximum acceleration rate, the minimum acceleration rate, the acceleration rate of the 95th percentile, percent idle time, average road power, and the road power at the 95th percentile. Road power represented the effects of rolling resistance, aerodynamic resistance, and acceleration resistance. Any number of cycles can be created and the cycle with the lowest PV was chosen as the top candidate. One noted limitation of this method was that the PV value was somewhat arbitrary in definition [43].

The second work, published in 2003, by Lin and Neimeier employed a four part Markov process theory to build drive cycles [44]. In a review of current drive cycle construction the paper outlined that longer driving cycles are often more representative, but are more time and cost intensive to perform, therefore, it is important to limit the length of cycles. The paper also states that using the Markov process requires more data than some other development techniques. The four step Markov chain process consisted of data partitioning, transition matrix estimation, cycle construction, and cycle selection. Data partitioning consisted of segmenting collected speed-time traces into modal events and binned. This involved maximum likelihood estimation (MLE), which allowed for clustering of modal events. Transition
probabilities were used to govern the selection of one modal event based on the current modal event. The bins were filled with modal events of similar speed and acceleration characteristics. Weighted random number generation was used to determine the next bin based on the transition probability matrix containing the modal events. These modal events are also referred to as “snippets.” The snippet that most improved the cycle was selected from the modal event bin. This methodology allowed for a large number of potential cycles to be generated due to the pseudo-randomness. The same metrics were used for analysis as in the previous work. The publication notes that there was a degree of judgement associated with selecting the metric, weights, and the final cycle [44].

The final publication by Niemeier et al. [45] also defined cycle development as a four step process including: collecting real world data, segmenting the driving data, constructing cycles, and evaluation and selection of a final cycle. A review of cycle construction methodologies was presented analyzing the differences between micro-trip, segment-based, pattern classification, and modal types. Some limitations are presented such as the fact that micro-trip based methods do not differentiate between roadway type and level of service, segment-based methods can make concatenation difficult, and pattern classification may not be directly related to emissions. All of these limitations led to the use of modal type cycle construction. This method sorted snippets of activity into categories of acceleration, deceleration, cruise, and idle. These snippets were binned and a transition matrix of succession possibilities was formed which allowed for the Markov chain construction technique, used in the previous two works, to again be utilized. In this work similar logic was used in terms of the chain including the random selection of snippets based on the transition matrix probability. Two differences that were applied included dynamic statistical analysis and cycle synthesis for specific driving patterns. These techniques were applied to drive cycle construction based on modal events. Cycle selection in this work was based on two things: the sum of absolute differences between the candidate cycle and the data of the speed acceleration frequency distribution (SAFDdiff) and the sum of the absolute differences in time that the cycle and the data spent in each mode (MODEdiff). These two sums of differences were combined into a final metric which is a weighted combination of SAFDdiff and MODEdiff [45].

Clearly, there are numerous examples of Markov chain techniques being used in cycle development. While most of the examples were based on the construction of drive cycles, techniques were applied to second-by-second engine cycle development by Smith [38]. Engine cycle development, rather than drive cycle development, was necessary to develop cycles from OTR data which did not include vehicle speed and off-road data. Engine cycles were required to operate an engine on an engine dynamometer rather than a chassis dynamometer. Therefore, it was important to review engine cycle development techniques for both OTR and off-road engines.
An example of OTR engine cycle development was performed by West Virginia University in the *ACES-1 Final Report* by Clark et al [46]. For this work, data were collected from 84 heavy-duty diesel trucks and grouped into four modes of operation: creep, transient, cruise, and high-speed cruise. The engine cycles developed were based on drive cycles formed from the same data. Modes were developed based on engine percent speed and engine percent torque. It was observed during data conversion that not all engines from which data were collected broadcasted engine percent torque accurately, but all engines did broadcast engine percent load. Data comparison of several cases showed that percent load often corresponded closely to percent torque, which resulted in no need for translation between the two. In other cases, there were slight differences at lower loads. This may have been a result of a lack of correction for idling. At high loads, however, engine percent load was almost always equivalent to engine percent torque. Therefore, engine percent torque was scaled from 0 to 100% on the same scale as engine percent load, ranged from idle to 100%. This work used micro-trips to divide data. It defined a micro-trip as two periods of operation separated by at least five seconds of idle operation. A micro-trip matrix was formed based on four different types of engine operation. Instead of creating complete engine cycles the work focused on creating “modes” of operation. Transient, creep, cruise, and high-speed cruise modes were created from micro-trips of similar activity. The combination of these micro-trips was not based on probability, but was accomplished with an engineering approach involving concatenation of four micro-trips and comparing to data based on an “S value.” This value was based on the average percent speed, average percent torque, average squared percent speed, and average squared percent torque. Each candidate mode was compared to the database with the square root of the sum of the differences of these metrics. Another noteworthy application cited in this work was a smoothing technique used when combining micro-trips to ensure that the rate of change of engine speed did not exceed eight revolutions per minute per second [46].

Steven [47] also used the technique of converting drive cycles into engine duty cycles. The goal was to create a worldwide test cycle which was universally representative of all vehicles. The data used were collected from different vehicle classes, road categories, and regions of the world. These variables were selected because they all influence engine speed and torque. A basic four step approach was used for engine cycle creation. The steps consisted of: creation of a reference database of driving patterns which included real-life situations, derivation of a transient vehicle cycle in terms of vehicle speed and normalized power, transformation of the transient vehicle cycle into a transient engine cycle, and finally selection of a reference transient test cycle that best approximated a drive train model. Micro-trips were again used in this work. A cycle time of 30 minutes was selected as a reasonable length. A series of micro-trips and stops were concatenated based on an engineering approach [47].
Another example of off-road engine cycle generation is presented by Ullman et al. [48]. This study created two types of cycles, a micro-trip based duty cycle and a “day in the life” duty cycle. The cycles were created based on off-road vehicles including a tractor, a backhoe loader, and crawler/dozer. Micro-trip based cycles were not based on idle-to-idle periods, but instead focused on the performance of an activity specific to the piece of equipment. “Day in the life” cycles were created based on 1 Hz data collected during operation of the equipment while performing actual work tasks in which operators were instructed to complete a job at normal pace. Micro-trip cycle creation required the use of several metrics including: rate of change of speed, rate of change of power, speed, torque, rate of change in speed, and rate of change of torque. Creation of micro-trip cycles involved counting the occurrences of power and derivatives of speed, torque, and power called “Relative Frequency Occurrences.” Micro-trips were weighted based on these occurrences and normalized speed and load were used to generate a final set of activity for comparison with actual data. “Day in the life” cycles were generated differently. The data collected were sorted into bins sized at 100 rpm for speed and 200 lb-ft for torque. These bins contained the second-by-second data which were collected from each piece of equipment. A frequency table was created to indicate the amount of time spent in each bin. Each data file collected was 20 minutes in duration. A one minute section of activity was selected and compared with a chi-squared statistic to the other 19 minutes of activity. This was repeated until all 20 minutes were used as the one minute section. Four iterations were completed for each piece of equipment and the window of activity that best represented the full body of data was selected as the most typical duty cycle. This ensured that the segment for each cycle was actual in use activity [48].

Most examples of cycle development focus on drive cycle construction. This is due to the fact that industry is often concerned with creating representative activity of a specific type of vehicle or task. Off-road engine cycle development often involves equipment that moves and uses the engine only for its own operation. Creation of cycles for off-road stationary engines is rare, because these engines are often certified using steady state testing. OTR vehicles, on the other hand, are often certified based on transient testing. Regardless of how the engine cycle is created, a critical step is to determine whether the engine cycle is representative of the activity attempting to be replicated. Creating the most optimal cycle is critical to ensure that in-use data are most adequately represented. There are a myriad of optimization techniques that can be applied to engineering problems, however, the one that was investigated for the purpose of this problem was the genetic algorithm (GA).

### 3.3 Genetic Algorithms and Use in Cycle Development Optimization

The concept of the GA stems from Darwin’s theory of evolution, which was developed in 1859. The governing principle of the theory is survival of the fittest. This theory proposes that stronger, more fit,
individuals are more likely to survive and reproduce passing their superior genetics onto offspring. Less capable individuals will not survive as long. They will not produce offspring and their inferior genetic traits will become extinct. So, better genetic material will be passed on more frequently to the next generation. This results in the entire population evolving so that it contains, on average, more fit individuals than the population of the generation prior to it. The key aspect that separates GAs from more traditional algorithms is the use of the population rather than analyzing one individual at a time [49].

A GA can be applied to problems in which the form of all possible solutions is known, but the best possible solution is unknown. This includes problems in which the population of potential solutions could be infinite. The purpose of a GA is to find the best of these solutions within reasonable limits on computational time. The basic flow of a GA is shown in Figure 6.

![Figure 6: Basic Flow Chart of a Genetic Algorithm][50]

GAs start with a population of possible solutions. Each solution is represented by a “chromosome,” an abstract representation that is not always straightforward. A GA is also defined by the genetic operators it uses. The most commonly used genetic operators are crossover and mutation. The genetic operators are
applied to the chromosomes within each generation of the GA. One of the difficulties encountered when constructing a GA is the definition of what constitutes a chromosome and how to apply genetic operators. These are critical factors, as the performance of the GA is extremely dependent on these definitions. The finding of the solution is ultimately governed by selection which depends on the fitness function. This fitness function is a method of comparing each individual in the population. Each chromosome is associated with a fitness function, which is indicative of the representativeness of a “good” solution. The goal of the GA is to maximize or minimize this fitness function, depending on the problem [49].

The definitions of reproduction and the fitness function are problem specific and critical to an effective GA. If these aspects are defined effectively, the flow of the GA should be similar across a variety of problems. The first population of a GA should offer a diverse array of genetic material. The gene pool should be as large as possible so that any solution available will have genes in the search space. In general, an initial population is generated randomly. Each iteration of the GA should consist of four steps: selection, reproduction, evaluation, and replacement. Selection is the first step and involves choosing the individuals which will reproduce in some manner. This is usually completed randomly with a probability depending on the relative fitness function. The second step, reproduction, is performed by using the genetic operators on the selected individuals to create new individuals. The third step, evaluation, involves the calculation of a fitness function for each individual, old and new. This is followed by the final step, replacement, which leads to the extinction of lesser individuals from the previous generation by replacing them with newly generated ones [51].

Breeding governs GAs and allows them to move toward a solution. There are three steps in breeding: selecting parents, crossing the parents to create new individuals, and replacing old individuals with new ones. Selection is choosing the parents which will breed in order to produce new individuals. Selection chooses individuals from populations with a probability related to their fitness function. Individuals with a higher fitness function have a higher chance of advancing to the next generation and therefore have a higher probability of producing offspring in current or future generations. Common selection techniques include: roulette wheel, random, rank, and tournament. Roulette-wheel is one available technique and involves using a ratio of each individual’s fitness function to the sum of all the fitness functions. Once individuals are selected genetic operators can be employed [49]. Crossover can be implemented between two individuals by swapping sections of their genes with one another. This is accomplished by randomly selecting two individuals, randomly selecting a cross point along their string length, and then swapping the sections of their gene strings to create two new individuals. Mutation is the second common genetic operator and involves changing a single bit of chromosome in the traditional binary sense in which each
individual is a string of 1’s and 0’s. This type of operator prevents the algorithm from becoming trapped at a local minimum and is aimed at maintaining the genetic diversity of the population [49].

GAs are advantageous in solving complex problems for several reasons. These advantages include their ability to discover global optimum solutions, their ability to handle large and poorly understood search spaces, their performance with large-scale optimization problems, and their ability to be applied to a large variety of problems [49]. These advantages make GAs a choice for difficult problems such as cycle development. Several works from WVU have used GAs for problems involving cycle development [50] [52].

Perhinschi et al. [52] used a GA to create a new cycle for which emissions could be estimated. This was possible because second-by-second emissions data already existed for previously tested drive cycles. A micro-trip database was created using 12 standard cycles. Individuals or chromosomes were defined as a set of micro-trips combined to make a cycle. The genes of the individuals are the micro-trips from the twelve already existing cycles. Individuals were allowed to have a variable number of genes, which changed the lengths of the individuals. The initial population was generated by randomly selecting micro-trips and aligning them into individuals which could vary in length with a maximum limit.

Independent variables were used to model the fitness function, which was referred to as the performance index. The variables used in this study included: average speed with idle, standard deviation of speed with idle, stops per mile, percentage idle time, and kinematic intensity. These variables of each created cycle were compared to the desired values of the variables to obtain a relative error. The relative error was then used to calculate the performance index of each individual as a ratio of each individual sum of relative errors to the total sum of relative errors of the entire population. Roulette-wheel and elitist selection were used as the methods to govern reproduction. The probability of each individual advancing to the next population was a ratio of the performance index to the sum of all performance indexes of the population, known as the total fitness.

Three genetic operators were used: mutation, crossover, and karyotype, at rates defined by the user of the algorithm. The subjects to experience genetic operation were selected at random. Mutation was applied first, which consisted of replacing a micro-trip, or gene, with another one from a different individual or the overall database. Crossover was applied by randomly selecting two individuals and selecting a random point between two genes of the larger individual. The genes from the two sides of the cross point were traded between the selected individuals. Karyotype operation was done by randomly adding or removing a gene from an individual. This type of alteration was applicable because the length of the individual was not fixed.
A new generation was defined once the genetic operators were applied at the dictated rates. The next generation was subject to the same criteria as the previous one and the process continued until the prescribed number of iterations was reached. The GA was demonstrated to produce cycles that were statistically similar to design parameters defined by the user. This allowed users to estimate emissions of certain types of driving with data that had already been collected [52]. The same GA was utilized by co-author, Marlowe, in his thesis. [50].
4. Data Collection Methods
In order to create test cycles that were representative of engines in the well-development industry it was necessary to collect data from the prime-movers. These are the major consumers of diesel fuel and have the potential to provide the industry cost savings if this diesel fuel can be displaced with natural gas [1] [2]. Data were collected from diesel engines in OTR trucks and from the engines powering drilling and fracturing equipment.

4.1 Over-the-Road Trucks
Collection of data from OTR trucks servicing the unconventional well completion industry was performed over a seven month period, beginning June 6, 2014 and concluding January 9, 2015. The vehicles targeted for data collection were heavy-duty diesel vehicles that were travelling to and from unconventional natural gas wells in the Marcellus shale region. These heavy-duty vehicles consisted of those hauling water, sand, and gravel. Data were collected from seven different companies located throughout Ohio, Pennsylvania, and West Virginia. Twenty-five vehicles were sampled from these seven companies. Of the 25 vehicles, 18 were water haulers, six were sand haulers and one carried gravel for site preparation. The engines populating the vehicles consisted of eighteen Cummins ISX-15 engines, four Caterpillar C-15 engines, two Volvo D13 engines and one Mack MP8. A summary of the vehicles tested is shown in Table 12.
Table 12: Well Service Vehicles Data Collection Information

<table>
<thead>
<tr>
<th>Truck #</th>
<th>Comp.</th>
<th>Hauling Service</th>
<th>Make</th>
<th>Model</th>
<th>Year</th>
<th>Size (hp)</th>
<th>Chasis</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2012</td>
<td>500</td>
<td>Peterbilt</td>
<td>5-Jun</td>
<td>30-Jun</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2012</td>
<td>500</td>
<td>Peterbilt</td>
<td>7-Jul</td>
<td>16-Jul</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>Water</td>
<td>Mack</td>
<td>MP8</td>
<td>2011</td>
<td>505</td>
<td>Mack</td>
<td>7-Jul</td>
<td>16-Jul</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2012</td>
<td>500</td>
<td>Freightliner</td>
<td>7-Jul</td>
<td>16-Jul</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2005</td>
<td>500</td>
<td>International</td>
<td>7-Jul</td>
<td>16-Jul</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>Gravel</td>
<td>Volvo</td>
<td>D13</td>
<td>2002</td>
<td>435</td>
<td>Volvo</td>
<td>29-Jul</td>
<td>31-Jul</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>Sand</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Kenworth</td>
<td>1-Aug</td>
<td>13-Aug</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>Sand</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Kenworth</td>
<td>1-Aug</td>
<td>13-Aug</td>
</tr>
<tr>
<td>10</td>
<td>C</td>
<td>Sand</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2012</td>
<td>500</td>
<td>Kenworth</td>
<td>1-Aug</td>
<td>13-Aug</td>
</tr>
<tr>
<td>12</td>
<td>D</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Kenworth</td>
<td>15-Aug</td>
<td>26-Aug</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>Sand</td>
<td>Caterpillar</td>
<td>C-15</td>
<td>2006</td>
<td>550</td>
<td>Peterbilt</td>
<td>18-Sep</td>
<td>24-Sep</td>
</tr>
<tr>
<td>15</td>
<td>E</td>
<td>Sand</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2009</td>
<td>500</td>
<td>Peterbilt</td>
<td>26-Sep</td>
<td>22-Oct</td>
</tr>
<tr>
<td>16</td>
<td>F</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Peterbilt</td>
<td>19-Nov</td>
<td>2-Dec</td>
</tr>
<tr>
<td>17</td>
<td>F</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Peterbilt</td>
<td>19-Nov</td>
<td>2-Dec</td>
</tr>
<tr>
<td>18</td>
<td>F</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Peterbilt</td>
<td>19-Nov</td>
<td>2-Dec</td>
</tr>
<tr>
<td>19</td>
<td>F</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Peterbilt</td>
<td>19-Nov</td>
<td>2-Dec</td>
</tr>
<tr>
<td>20</td>
<td>G</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2012</td>
<td>500</td>
<td>Freightliner</td>
<td>17-Dec</td>
<td>5-Jan</td>
</tr>
<tr>
<td>21</td>
<td>G</td>
<td>Water</td>
<td>Caterpillar</td>
<td>C-15</td>
<td>2010</td>
<td>550</td>
<td>Peterbilt</td>
<td>17-Dec</td>
<td>5-Jan</td>
</tr>
<tr>
<td>22</td>
<td>G</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2011</td>
<td>500</td>
<td>Kenworth</td>
<td>17-Dec</td>
<td>5-Jan</td>
</tr>
<tr>
<td>23</td>
<td>G</td>
<td>Water</td>
<td>Volvo</td>
<td>D13</td>
<td>2012</td>
<td>500</td>
<td>Kenworth</td>
<td>17-Dec</td>
<td>5-Jan</td>
</tr>
<tr>
<td>24</td>
<td>G</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2010</td>
<td>500</td>
<td>Peterbilt</td>
<td>5-Jan</td>
<td>9-Jan</td>
</tr>
<tr>
<td>25</td>
<td>G</td>
<td>Water</td>
<td>Cummins</td>
<td>ISX-15</td>
<td>2013</td>
<td>500</td>
<td>Peterbilt</td>
<td>5-Jan</td>
<td>9-Jan</td>
</tr>
</tbody>
</table>

The data were collected at a rate of one Hz using DAWN J1939 Mini Loggers™ manufactured by HEM Data. These loggers, shown in Figure 7, are specifically designed to acquire communication area network (CAN) bus data from heavy-duty trucks and off-road vehicles [53]. The loggers were plugged into the vehicles’ J1939 port, which was typically located under the dash in the cab of the vehicle. The logger then automatically recorded engine parameters broadcasted by the vehicle every time the ignition was powered. The logger records data until a key-off event. The data loggers were used with their compatible software, DawnEdit™, also developed by HEM Data [54]. This software provided a list of 1953 available parameters that could be recorded by the loggers from the vehicles engine control unit (ECU), provided they were broadcasted by the vehicle. DawnEdit™ allowed the user to create a database of the parameters that were desired during data acquisition. With a database of parameters selected, the logger could be
installed in the vehicle. The vehicle was started for approximately 30s, as instructed by the DawnEdit™ manual after which the logger was removed from the vehicle and reconnected to a computer containing the DawnEdit™ software. This procedure allowed the software to determine which of the desired parameters were broadcasted by the engine in question. The desired parameters were then defined in a configuration file which was loaded onto the logger before final instrumentation of the vehicle. Although dozens of parameters were available and ultimately collected, the only parameters relevant to the creation of engine test cycles under this work were Engine Speed and Percent Load at Current Speed. These parameters were broadcasted by all OTR engines used for data collection. The parameters are defined by the International Society of Automotive Engineers (SAE International) under J1939 protocol as follows:

Engine Percent Load at Current Speed – SPN 92: The ratio of actual engine percent torque (indicated) to maximum available at the current engine speed, clipped to zero torque during engine braking [55].

Engine Speed – SPN 190: Actual engine speed which is calculated over a minimum crankshaft angle of 720 degrees divided by the number of cylinders [55].

In addition to J1939 parameters, GPS data were also recorded by the Mini Loggers™ so that vehicle trips could be mapped. Once data were collected from the vehicle, they were converted from raw “.ISO” files to a usable CSV format with the DawnEdit™ software and the governing database of the vehicle from which the data were taken.

Figure 7: HEM J1939 Mini Logger [53]

4.2 Drilling
Drilling activity data were collected during the drilling of two horizontal wells in Westover, WV. The collection of data occurred over a period of 11 days from September 9, 2015 to September 21, 2015. The data were collected from a rig which utilized three Caterpillar 3512C generator units, rated at 1476 hp. These engines were outfitted with DGB kits allowing the engines to run in either dual-fuel or diesel only mode. Data were taken from a single engine that operated continuously during rig setup, preparation, pipe
tripping (PT), and steady state drilling (SSD). The activity logs of the drilling company were obtained to assist with sorting data. A total of 311 hours of data were taken between the beginning of rig setup and the completion of the well. The breakdown of the engine operation activity is presented in Table 13. This breakdown shows the amount of time spent performing each type of activity necessary for the completion of drilling.

**Table 13: Drilling Activity Breakdown**

<table>
<thead>
<tr>
<th>Activity Type</th>
<th>ECU Collection</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hrs</td>
<td>Percent</td>
</tr>
<tr>
<td>Rig Up</td>
<td>4</td>
<td>1%</td>
</tr>
<tr>
<td>Drilling</td>
<td>160</td>
<td>51%</td>
</tr>
<tr>
<td>Reaming</td>
<td>3</td>
<td>1%</td>
</tr>
<tr>
<td>Coring</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Condition &amp; Circulate</td>
<td>16.5</td>
<td>5%</td>
</tr>
<tr>
<td>Trips</td>
<td>21</td>
<td>7%</td>
</tr>
<tr>
<td>Lubricate Rig</td>
<td>10.5</td>
<td>3%</td>
</tr>
<tr>
<td>Repair Rig</td>
<td>9</td>
<td>3%</td>
</tr>
<tr>
<td>Cut off Drilling Line</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>Wireline Logs</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Run casing &amp; cement</td>
<td>27</td>
<td>9%</td>
</tr>
<tr>
<td>Wait on cement</td>
<td>8.5</td>
<td>3%</td>
</tr>
<tr>
<td>Nipple up B.O.P.</td>
<td>5</td>
<td>2%</td>
</tr>
<tr>
<td>Test B.O.P.</td>
<td>12</td>
<td>4%</td>
</tr>
<tr>
<td>Directional Work</td>
<td>13</td>
<td>4%</td>
</tr>
<tr>
<td>Other</td>
<td>20.5</td>
<td>7%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>311</td>
<td>100%</td>
</tr>
</tbody>
</table>

Data were retrieved from the ECU using a nine pin Deutsch connector specifically wired for off-road Caterpillar engines. A serial cable was used to connect the Deutsch connector to a VIA Model HDV100A1 [56]. The connection setup on the engine is shown in Figure 8.
The serial cable was connected to a laptop equipped with WVU CAFEE software known as ECU Slave. A screen capture of the program is shown in Figure 9. This software allows the user to record either SAE J1939 or J1708 data from an engine.

The data collected were exclusively J1939. Converting the data to a usable form was accomplished using CAFEE developed software known as ECU Packet Analyzer, shown in Figure 10. After conversion of data to CSV format it was determined that only 233 of the 311 hours could be considered valid.

Invalid data were sometimes read from the ECU when the engine failed to broadcast messages or broadcast data which were invalid determined by the occurrence of unrealistic values. Other reasons for data loss during the collection time included software issues and power failure. The activity data that were desired for the development of engine cycles included Engine Speed (SPN 190) and Engine Percent Load at Current Speed (SPN 92). The engine was operated at a constant speed so this parameter was not as
important for drilling as for OTR data collection. The engine percent load at current speed was the only parameter that was considered necessary for the cycle development performed by this research. This parameter governed cycle development for off-road engines. Engine speed was always considered to be 100% of rated speed. The engine percent load was broadcasted as a normalized value ranging from 0 to 100%.

![ECU Packet Analyzer Software](image)

Figure 10: ECU Packer Analyzer Software

### 4.3 Hydraulic Fracturing

Hydraulic fracturing activity data were collected during two separate campaigns with two different fracturing fleets. Collection of data was performed in a similar manner to drilling: with a nine pin Deutsch connector, serial cable, VIA Model HDV100A1 and a laptop. ECU Slave was used for data acquisition along with ECU Packet Analyzer for file conversion. The first data collection effort occurred over 21 days between July 19, 2015 and August 8, 2015, from a well pad in central West Virginia. Data were collected from four different hydraulic fracturing pumps during the fracturing of six wells on the pad. These pumps were powered by Cummins QSK50 engines, rated at 2250 hp [57]. These engines were all model year 2010. The majority of the data collected at this location were recorded using an auto-record feature of the data acquisition system. This auto-record feature saved a file every hour. Other data were recorded manually so that an entire fracturing stage was contained in one file.
The second data collection effort occurred at the well pad in Westover, WV from which drilling data were collected. The data were collected from a 2008 Caterpillar 3512B HD engine rated at 2250 hp over a period of 7 days from November 5, 2015 to November 12, 2015. The engine was outfitted with a DGB kit allowing it to operate in either diesel only or dual-fuel mode. Data were only recorded manually and were only taken during fracturing stages. This could have resulted in some lost periods of idle engine activity that occurred while the engine was not under load. This type of activity was deemed rare based on observation, as engines were generally shut down in between fracturing stages. A total of twelve fracturing stages were recorded from a single engine during the collection. Three stages had to be eliminated because the data broadcasted by the engine were considered invalid, due to erratic values. The hydraulic fracturing engines operated at rated speed except for when the engines were idling in between stages, which was limited.

The same data were recorded during both collection campaigns. A summary of the data collected is presented in Table 14.
### Table 14: Hydraulic Fracturing Data Collection Information

<table>
<thead>
<tr>
<th>Stage</th>
<th>Site</th>
<th>Engine Make</th>
<th>Model</th>
<th>Data Rate (Hz)</th>
<th>Length of Stage (s)</th>
<th>Average Speed (RPM)</th>
<th>Average Load (%)</th>
<th>Average Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>5209</td>
<td>1842</td>
<td>66.37</td>
<td>1114</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>4672</td>
<td>1873</td>
<td>52.32</td>
<td>878</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>4001</td>
<td>1937</td>
<td>79.57</td>
<td>1335</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>4914</td>
<td>1866</td>
<td>60.45</td>
<td>1014</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>5226</td>
<td>1936</td>
<td>79.69</td>
<td>1337</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>4638</td>
<td>1936</td>
<td>84.81</td>
<td>1423</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>6271</td>
<td>1842</td>
<td>66.57</td>
<td>1117</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>5134</td>
<td>1823</td>
<td>73.76</td>
<td>1238</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>5234</td>
<td>1936</td>
<td>77.28</td>
<td>1297</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>Caterpillar</td>
<td>3512B</td>
<td>1</td>
<td>4879</td>
<td>1602</td>
<td>73</td>
<td>1223</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>4860</td>
<td>1937</td>
<td>56.31</td>
<td>945</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>5810</td>
<td>1948</td>
<td>50.71</td>
<td>851</td>
</tr>
<tr>
<td>13</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>10770</td>
<td>1466</td>
<td>49.5</td>
<td>831</td>
</tr>
<tr>
<td>14</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>6130</td>
<td>1947</td>
<td>50.47</td>
<td>847</td>
</tr>
<tr>
<td>15</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>5130</td>
<td>1950</td>
<td>49.27</td>
<td>827</td>
</tr>
<tr>
<td>16</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>2380</td>
<td>1948</td>
<td>52.82</td>
<td>886</td>
</tr>
<tr>
<td>17</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>5910</td>
<td>1949</td>
<td>52.5</td>
<td>881</td>
</tr>
<tr>
<td>18</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>5890</td>
<td>1946</td>
<td>48</td>
<td>805</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>1</td>
<td>2507</td>
<td>1949</td>
<td>54.48</td>
<td>914</td>
</tr>
<tr>
<td>20</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>1</td>
<td>3093</td>
<td>1947</td>
<td>53.01</td>
<td>890</td>
</tr>
<tr>
<td>21</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>1</td>
<td>1555</td>
<td>1950</td>
<td>50.09</td>
<td>841</td>
</tr>
<tr>
<td>22</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>6850</td>
<td>1944</td>
<td>50.26</td>
<td>843</td>
</tr>
<tr>
<td>23</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>10</td>
<td>5080</td>
<td>1950</td>
<td>54.73</td>
<td>918</td>
</tr>
<tr>
<td>24</td>
<td>B</td>
<td>Cummins</td>
<td>QSK50</td>
<td>1</td>
<td>3657</td>
<td>1942</td>
<td>51.31</td>
<td>861</td>
</tr>
</tbody>
</table>

Since the engines typically operated at rated speed the most important engine parameter was percent load at current speed. The cycles developed for hydraulic fracturing depended only on load since, like drilling, the engine operated at rated speed. The average power was derived by multiplying the average engine load by the rated power of the engine.
5. Data Processing

After data were collected, steps were taken to convert the data into a format that could be imported to MATLAB®. The engine data were converted into a normalized array and data identified as being invalid was eliminated. Slightly different steps had to be taken for the three different types of data collected.

5.1 Over-the-Road Trucks

Data collected from OTR trucks were imported to MATLAB® using the script *Trucking_Data_Import.m* found in Appendix A. Data were imported from four different folders which categorized the data by the engine from which it came. There was one file from a Mack MP8, 16 files from Volvo D13s, 43 files from Caterpillar C15s, and 191 files from Cummins ISX15 engines. The data were divided because different engine parameters were needed for normalization of engine speed, which was also performed by this script. Normalization of engine data was necessary to compare engine operation between different sizes, makes and models of engines. All normalized data points contain a value between zero and one representing 0 to 100% of the maximum possible value. The parameter Percent Load at Current Speed which was collected is an example of a normalized parameter. This parameter is based on the lug curve of the engine and is broadcasted as a percentage of the maximum possible load at a given engine speed. This parameter was considered similar to percent engine torque in a previous study [46]. This parameter was used to determine the normalized percent torque during cycle development. Percent load was a more reliable parameter than percent engine torque. All files which were obtained and converted contained engine percent load. All of the files did not contain engine percent torque and several contained torque values that simply showed linear transitions to positive or negative infinity. The normalization of engine speed was not as straightforward. This parameter was normalized based on the engine from which the data was taken. The engine speed normalization was based on the equation used for unnormalizing speed from the CFR Title 40 Chapter I Subchapter C Part 86 Subpart N Section 1333. Normalizing engine speed was performed using Equation 1.

\[
\text{Normalized Speed} \ (\%) = \frac{\text{Engine Speed} \ (\text{rpm}) - \text{Idle Speed} \ (\text{rpm})}{\text{Rated Speed} \ (\text{rpm}) - \text{Idle Speed} \ (\text{rpm})}
\]  

Equation 1

The values used for rated speed, shown in Table 15, are advertised by the manufacturer with the engine specifications [58] [59] [60] [61]. Idle speed was more difficult to find in terms of an exact value. The idle speed of the engine was determined from the data by observing the average engine speed when the vehicle experienced zero vehicle speed.
Table 15: Engine Speed Information

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Rated Speed</th>
<th>Idle Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cummins</td>
<td>ISX-15</td>
<td>2000</td>
<td>700</td>
</tr>
<tr>
<td>Caterpillar</td>
<td>C-15</td>
<td>2100</td>
<td>600</td>
</tr>
<tr>
<td>Volvo</td>
<td>D13</td>
<td>2100</td>
<td>550</td>
</tr>
<tr>
<td>Mack</td>
<td>MP8</td>
<td>1950</td>
<td>650</td>
</tr>
</tbody>
</table>

Once all the data were normalized, values that were beyond the range of applicability were eliminated. These values by the normalized definition included those that were less than zero or greater than one, due to the fact that they were not within a range of 0-100%. This led to elimination of speed values that were near idle, but less than the defined idle speed and between the rated and governed speed of the engine. To compensate for this all data which were within 2% of the idle speed were set to 0 (idle speed). These values ranged from -0.02 to 0.02. To accommodate the speed values above the rated speed any value greater than but within 2% of the rated speed were set to 1.0 (rated speed). This included values between 1.0 and 1.02. The total number of points above the rated speeds defined in Table 15 made up only 0.29% of all of the valid speed data imported. These points were eliminated with little concern on the impact of data statistics. Some data points had to be eliminated due to the fact that they were not within the range of values that could be broadcast. There was no known cause or corrective measure, so those data points were eliminated. This was performed on both normalized speed and normalized load. Once all invalid data were eliminated, the values of speed and load were arranged in a two column array and concatenated. The process was completed for all engines so that the final data variable was a single two column array of normalized speed and load. The length of the data was used to calculate the total time of activity available. All data were recorded at a rate of one Hz. The resultant array had a total length of 4,724,800 in total, or over 54 days of usable data.

Once the data array was saved in MATLAB®, a second script was used to bin the data and calculate average speed and load statistics. This script, `Trucking_Transition_Matrix_Formation.m`, was based on the idea that to build a Markov chain, it is necessary to have a distribution of data along with a transition matrix. The data bins were grouped in 5% increments for both speed and load. The bins contained values of speed and load greater than or equal to the governing value of the current bin and less than the governing value of the next highest bin. This created a matrix of binned values in the form of the one shown in Figure 11.
Figure 11: Truck Data Matrix Format

The data were binned by examining each point in the data individually. The distribution of data is shown in Figure 12. Note that nearly 20% of the activity data occurred in a single bin. This was bin (1,3) and contained nearly all of the points that occurred at idle engine speed.

Figure 12: Truck Data Distribution

Once the point under inspection was placed in a bin of the distribution matrix, the next point of data was observed. This data point was placed in the transition matrix of the bin of the previously selected point and in the overall distribution matrix. A multi-layered structure was created in MATLAB® to allow each bin of the overall distribution matrix to also contain a transition matrix of the same size. This was
necessary because of the stochastic logic of Markov chains. Once the entire length of the array was binned, a normalized distribution was created by dividing the size of each data bin by the total length of the array. Statistics of the data set could also be calculated. Average speed and load were found as well as the time spent at idle, which was considered 0% normalized speed. The idle data points were also taken out so that the non-idle speed and load averages could be calculated. These values are shown in Table 16.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Speed (%)</td>
<td>29.2</td>
</tr>
<tr>
<td>Average Load (%)</td>
<td>23.1</td>
</tr>
<tr>
<td>Average Non-Idle Speed (%)</td>
<td>46.9</td>
</tr>
<tr>
<td>Average Non-Idle Load (%)</td>
<td>28.3</td>
</tr>
<tr>
<td>Normalized Idle Time (%)</td>
<td>37.6</td>
</tr>
<tr>
<td>Normalized Non-Idle Time (%)</td>
<td>62.4</td>
</tr>
<tr>
<td>Total Time (s)</td>
<td>4,724,800</td>
</tr>
</tbody>
</table>

Table 16: Trucking Data Statistics

The importance of rates of change of speed and load was understood when considering cycle effects on emissions. However, these values were not taken into account when comparing cycles to the data set statistics. This was due to the fact that the Markov chain logic, discussed in Chapter 7, was meant to imitate not only distributions and statistics, but also rates of change of speed and load, naturally. The logic of the Markov chain was such that created cycles were not capable of transitioning between points that were not seen in real world operation of the engine. This ensured that only realistic rates of change of speed and load occur in the created cycles.

5.2 Drilling

The drilling data were imported in a similar manner using Drilling_Data_Import.m found in Appendix A. These data were all imported from the same folder because all of the data were taken from the same engine, in the same format. There were 235 total CSV files imported to MATLAB®. The only value that was extracted from the files was engine percent load. This was because the drilling engine from which data were extracted operated at a constant speed of 1200 rpm. The speed varied only slightly between about 1150 and 1250 rpm. This meant that drilling cycles only required one variable, load. The load data were imported and concatenated into a single column array. Values below zero and above one were again eliminated and the total length of the data was determined to be the time since data were taken at one hertz. A total of 840,238s or 9.73 days of data were available for use. This was above the minimum expected time of eight days to drill a typical well [11].

41
With data imported as a single array the sorting of data was accomplished with the `Drilling_Transition_Matrix.m` script. The binning logic used was similar to that used for OTR trucks except that only the load variable was binned. This resulted in a matrix with only one column. The structure of the matrix is shown in Table 17. The data were again separated into bins of 5% increments.

Table 17: Drilling and Fracturing Data Matrix Format

<table>
<thead>
<tr>
<th>% Load</th>
<th>≥</th>
<th>&lt;</th>
<th>Bin #</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.05</td>
<td>1</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.1</td>
<td>2</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.15</td>
<td>3</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.15</td>
<td>0.2</td>
<td>4</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.25</td>
<td>5</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.3</td>
<td>6</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.35</td>
<td>7</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.4</td>
<td>8</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.45</td>
<td>9</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.5</td>
<td>10</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.55</td>
<td>11</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.55</td>
<td>0.6</td>
<td>12</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>0.65</td>
<td>13</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.65</td>
<td>0.7</td>
<td>14</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.75</td>
<td>15</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.75</td>
<td>0.8</td>
<td>16</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.85</td>
<td>17</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.85</td>
<td>0.9</td>
<td>18</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>0.95</td>
<td>19</td>
<td>[20x1]</td>
<td></td>
</tr>
<tr>
<td>0.95</td>
<td>1</td>
<td>20</td>
<td>[20x1]</td>
<td></td>
</tr>
</tbody>
</table>

The load array was examined point-by-point and each individual value was placed in its respective bin of the overall distribution matrix. The following value was placed in the appropriate bin of the internal transition matrix of the previously selected bin. This process was repeated until the entire data array was binned and then the distribution matrix was normalized by dividing each bin by the length of the total data array. The data distribution can be seen in Figure 13.
The data were further subdivided into two categories of load, high- and low-load, for the purpose of calculating statistics. High-load was considered any value at or above 40% and low-load was considered anything below 40%. The value of 40% was chosen because it was desired that all of the steady state drilling events be considered high-load. Some of these events, occurred at steady state values below 50%, but none of them occurred at values below 40%. The idle time of the data was also calculated which was considered when the engine ran at 0% load. This value was near zero because the drilling engines operated continuously during data collection. The average value and the normalized time of high- and low-load operation were calculated. The statistics of the data are shown in Table 18.

**Table 18: Drilling Data Statistics**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Load (%)</td>
<td>43.0</td>
</tr>
<tr>
<td>Average High-Load (%)</td>
<td>60.2</td>
</tr>
<tr>
<td>Average Low-Load (%)</td>
<td>19.0</td>
</tr>
<tr>
<td>Normalized High-Load Time (%)</td>
<td>58.4</td>
</tr>
<tr>
<td>Normalized Low-Load Time (%)</td>
<td>41.2</td>
</tr>
<tr>
<td>Normalized Idle Time (%)</td>
<td>0.4</td>
</tr>
<tr>
<td>Total Time (s)</td>
<td>840,238</td>
</tr>
</tbody>
</table>

### 5.3 Hydraulic Fracturing

The hydraulic fracturing engine data were imported from three different folders. The first folder was filled with data that were auto-recorded from a Cummins QSK50 as part of the first collection campaign in central West Virginia. The second folder contained data which were recorded manually so that an entire
stage was captured from the same campaign. The third folder contained data recorded from a Caterpillar 3512B HD at the Westover site. All of the files were in CSV format. The data from all three folders were imported to MATLAB® using the *Fracking_Data_Import.m* script and invalid data points, below zero or above one, were eliminated. The valid data were concatenated into a single array of load points. The total length of the array represented the amount of time of valid data available which was 239,905s or just under 2.78 days.

The hydraulic fracturing data were binned using the exact same methodology as the drilling data with the *Fracking_Transition_Matrix_Formation.m* script. The data were again binned into 5% increments forming an overall distribution matrix by stepping through each point of the array. The inner transition matrices were formed for each bin of the distribution matrix. The structure of the distribution matrix is the same as is shown in Table 16. The distribution of the data is shown in Figure 14.

![Figure 14: Fracturing Data Distribution](image)

Data were sorted into idle (0% load), low-load (points below 40% load), and high-load (points at or above 40% load). The average overall load, average high-, average low-load, normalized idle, low-, and high-load time were all calculated. The calculated statistics are shown in Table 19.
Table 19: Fracturing Data Statistics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Load (%)</td>
<td>35.5</td>
</tr>
<tr>
<td>Average High Load (%)</td>
<td>61.5</td>
</tr>
<tr>
<td>Average Low Load (%)</td>
<td>20.3</td>
</tr>
<tr>
<td>Normalized High Load Time (%)</td>
<td>46.8</td>
</tr>
<tr>
<td>Normalized Low Load Time (%)</td>
<td>33.2</td>
</tr>
<tr>
<td>Normalized Idle Time (%)</td>
<td>19.9</td>
</tr>
<tr>
<td>Total Time (s)</td>
<td>239,905</td>
</tr>
</tbody>
</table>
6. Cycle Formation
Once all of the data were imported and processed, a methodology for creating representative cycles was developed. The technique selected was a stochastic second-by-second concatenation which utilized Markov chain theory. The same basic principle was applied to all three types of cycle development. A starting period of engine idle was used and from there the Markov chain technique was allowed to progress freely based on the transition probability matrix of each of the respective data sets. A time limit of 30 minutes (1800s) was imposed on each of the cycles. The time was defined based on the fact that the FTP transient test was 20 minutes long [24]. Research, however, suggested that longer Markov chains were more representative when attempting to match a distribution [31] [34]. Previous studies on cycle development were noted as having used 30 minutes as the time of the cycle [47]. A time limit was enforced due to the fact that engine testing on a dynamometer is cost intensive.

6.1 Over-the-Road Trucks
OTR cycles were created based on a Markov chain theory which was dependent on both the overall distribution of the data and the transition probability matrix of each bin. The script used to create OTR cycles, Trucking_Markov_Chain_Cycle_Construction.m, can be found in Appendix B. The initial point of the cycle was randomly chosen from data bin (1,1). This was selected because the vehicle started at a low-load idle point when there was an ignition power event. Based on the nature of driving and observations made during data collection from the industry, it was determined that, at a minimum, the first 30s of the cycle should be idle. This was based on the observation that most drivers take some time between a key-on event and the beginning their routes. Therefore, the points in the cycle between 2s and 30s were sampled from the zero speed row of the distribution matrix. The data sample of the next point was weighted based on the transition probability matrix of the bin of the current point in the cycle. The weights were based only on the load distribution for the first 30s. The primary reason for this period of time was to attempt to imitate real world activity in which a typical driver takes time between when the vehicle is started and leaves the lot to sign in and record odometer readings. The value of 30 seconds attempted to simulate this action without inhibiting the Markov chain.

Once 30s of idle were completed, the overall distribution matrix was open for sampling and the Markov chain logic proceeded. This was accomplished by using a weighted sampling technique to pseudo-randomly select the next bin of the cycle. Separate weights were developed for the columns and the rows of the matrix. The weights of each were based on the ratio of the time in a given bin to the total time spent in the column or row. Then, the row of the next bin was selected by sampling 1-20 (the number of rows), based on the weights of the rows developed previously. The column of the next bin is selected in the same manner. Once the row and column of the next bin were selected, the bin was completely defined. With the
bin defined, a data point was randomly selected from the bin population. This point was defined as the next point in the cycle and was added to the end of the current array. The transition matrix of the selected bin then governed the weights used for selection of the next bin. As points were selected the distribution of the cycle was created for comparison to the distribution of the overall data set.

The Markov chain continued to sample using the defined method until the point of 1770 was reached. This point was 30s before the end of the cycle. It was determined that the vehicle should return to a state of idle 30s before the end of the test to simulate real world activity. If the speed bin of the cycle at this point was above 10, meaning normalized speed was over 50%, an intermediate point was required at cycle point 1771, in order to create an acceptable rate of change of speed. The next bin was sampled from the row that was 5 bins lower than the current bin. For example, if the speed row of point 1770 was 17 then the row of point 1771 would be 12. If, as in this example, the current row was still greater than 10 the process was repeated again for point 1772. So if the row of point 1771 was 12, then the point of 1772 would be selected from speed row seven. The weighting of the columns was based on the overall column distribution during this procedure to determine the column of the bin. Once the current speed bin was below 10, the speed row was forced to one, the idle speed row, and the process continued as it did in the first 30s of cycle construction with selection of points based only on the load columns until the cycle reached a length of 1800s. The first and last 30s made up only 3.3% of the total cycle time and an ideal representative cycle would contain over 37% idle time. This ensured that the Markov chain idle periods were not displaced by forced idle. Placing these periods at the beginning and end of the cycle ensured that the engine speed did not jump to a high bin immediately and also did not limit the freedom of the Markov chain.

Once the length of the cycle was reached an addition condition was implemented, making the cycle easier to analyze and more realistic. Each cycle was searched to determine if there were any single points of non-zero speed between points of zero speed. These points were due to the Markov chain jumping out of an idle bin and then back into it for the next point. This type of activity made segmentation of cycles difficult and was not necessary to represent real world engine operation. If any of these points were found in the cycle they were forced to zero speed. This was performed because it made segmentation and recombination of cycles easier during the application of the GA. The reduction of these points did not change the overall distribution of the cycle and had no effect on idle time. This was verified with a randomly generated sample population of 100 cycles. This finalized the creation of individual cycles.

Once a cycle was created, a number of values were calculated to define it. The distribution of each cycle was created by stepping through the cycle and binning its points in the same way that the overall data was binned. The same average statistics were determined from the data and the newly created cycles. These
values included: average speed, average load, average non-idle speed, average non-idle load, and normalized idle time. The distribution and average statistics were compared to those of the data and used to calculate the values which defined the representativeness of the individual cycles. These values were defined as the “Statistical Value” (SV), the “Distribution Value” (DV), and the “Performance Value” (PV). The definitions of these values were all based on absolute differences between the desired data set and the created cycles. The DV was defined by taking the absolute difference between the cycle and data of the time in each bin of the distribution matrix. Each absolute difference was multiplied by 0.05 and the sum of those weighted absolute differences determined the value. The value of 0.05 was used on the distribution so that the column or row distribution could theoretically be taken as a percentage. Since there were 20 of each and 0.05 is 1/20th of 100. The definition of the DV is shown by Equation 2.

\[
DV = \sum_{i=1}^{20} \sum_{j=1}^{29} 0.05 \times |\text{bin}(i,j)_{cycle} - \text{bin}(i,j)_{data}|
\]

Equation 2

The SV was calculated in a similar manner, except the absolute differences of the statistics were used. Each of the five statistics previously defined were given an equal weighting of 0.2 and the sum of the weighted differences was used to calculate the SV. The weighting of 0.2 was used because five statistics from the cycle and data were compared. This allowed for each statistic to be weighted equally and the possibility for the value to represent a percent difference since 1/5th of 1.0 is 0.2. This is shown by Equation 3.

\[
SV = \sum_{i=1}^{20} 0.2 \times |\text{stat}(i)_{cycle} - \text{stat}(i)_{data}|
\]

Equation 3

The PV was calculated by combining the DV and SV with equivalent weights of 0.5 and summing them, as shown by Equation 4.

\[
PV = 0.5 \times SV + 0.5 \times DV
\]

Equation 4

Each truck cycle consisted of a structure of these representation values as well as the statistics, cycle array, time values, and distribution. An example of the structure is shown in Figure 15.
The Markov chain technique allowed for the creation of a nearly infinite number of unique cycles. The nature of this pseudo-random approach to creation leads to the problem of determining which of the nearly infinite possibilities could be deemed most representative. A population of 10,000 cycles was initially created in search of “representative” cycle.

### 6.2 Drilling

The creation of drilling cycles was simpler than that of OTR cycles. The engine speed was constant, therefore only one variable, percent load, had to be evaluated in the cycle creation process. The initial point of a drilling cycle was randomly selected from bin one, which was defined by the limits of 0-5% load. The cycle was forced to stay in the lowest load bin for ten seconds to simulate engine startup and to ensure that the cycle did not jump to a high-load bin immediately. Ten seconds was used instead of 30 due to the nature of operation of the drilling engines. The engines run continuously in the field so they do not experience startup often and the goal was to minimize the cycle time spent at non-rated speed. After the first ten seconds were randomly selected from the first bin of data, the Markov chain was allowed to operate on its own based on the transition probability matrix and overall data distribution.

The transition probability of each bin was defined by a 1x20 matrix because engine speed was constant. Therefore, only a column needed to be selected. The weight of each bin was the time spent in that bin divided by the total time. These weights were defined by the probability matrix of each of the respective bins. For example, if a point in the cycle was selected from bin five, then the next bin was chosen based on a weighted sample of 1-20 based on the transition probability matrix of bin five. Once a bin was selected a data point was randomly chosen from the bin and added to the cycle. This process continued until the cycle reached a length of 1790. Once this point was reached if the current bin was above ten, representing greater than 50% load, then the next bin was chosen to be the bin with an index of five less than the current bin. Point 1791 was then randomly chosen from that bin and if that bin was above ten the
process was repeated for point 1792. If the bin of point 1790, 1791, or 1792 were below ten then the next point was chosen from bin one. The rest of the points of the cycle were chosen from bin one until the cycle reached the desired length of 1800s.

The points of the cycles were then evaluated in the same way as the OTR cycles to determine if there were any singular points of non-zero load between points of zero load. If this was the case, these points were forced to 0%. Also, any point that was selected from bin one that showed a load of less than 2% was forced to 0%. This made no-load segments of activity recognizable by the segmentation process of the GA to be discussed later. These two occurrences were anomalies in drilling cycles due to the fact that cycles rarely returned unforced to bin one.

Statistics, time values, and distributions of created cycles were calculated so that each cycle was represented by values other than the array of load points created by the Markov chain. The distribution was created with the same methodology as was used for OTR trucks, but was only applied to the single variable load array. The statistics that were calculated for drilling included: average load, average high-load, average low-load, normalized idle time, normalized time spent at high-load, and normalized time spent at low-load. The distinction between low- and high-load was defined the same as it was during the calculation of overall data statistics, with high-load being all values at or above 40% load and low-load being values less than 40%. The structure of a created drilling cycle is shown in Figure 16.

![Figure 16: Structure of Drilling and Fracturing Cycles](image)

The three values used to define representativeness of the cycles were defined in a similar manner as they were for OTR trucks. The DV, SV, and PV for drilling were defined in a similar way. The DV was again the weighted sum of the absolute difference in the distribution percentage of the cycle and the data as is shown in Equation 5. There were only 20 bins for drilling and a weighting value of 0.05 was applied to each bin.
Equation 5

\[
DV = \sum_{i=1}^{20} 0.05 \times |\text{bin}(i)_{\text{cycle}} - \text{bin}(i)_{\text{data}}|
\]

The SV was an equally weighted sum of the absolute difference between the cycle and the data of the following statistics: average load, average high- and low-load, time at high-load and time at idle. The SV was calculated in the same manner as is shown in Equation 3. The PV was calculated in the same way as it was in the OTR truck analysis, shown in Equation 4.

6.3 Hydraulic Fracturing

Hydraulic fracturing engine operation was similar to that of drilling in that once the engine was powered beyond idle, the engine speed was constant. Therefore, like drilling, only the load variable was needed to create hydraulic fracturing engine test cycles. The logic employed to build cycles for hydraulic fracturing was identical to that used for drilling. The first ten seconds were forced to remain in bin one and then the Markov logic was allowed to govern the cycle construction process until the final ten seconds. After this the same steps were used to steer the cycle back into bin 1 until the end was reached. One note about the hydraulic fracturing cycles that were created in this manner was that the idle bin of the distribution contained a much larger portion of the data and because of this the reduction of points to zero was more important. The single non-zero points between two zero points and the points less than 2% were more prominent in hydraulic fracturing cycles. They were still reduced to zero for simplicity, with minimal effect on distribution or cycle statistics. The distribution and statistics for hydraulic fracturing were calculated in exactly the same way as they were in drilling, with the same values being defined. The structure of a created hydraulic fracturing cycle is shown in Figure 16. The equations used to define the DV, SV, and PV for hydraulic fracturing were calculated in the exact same way as those used for drilling. The equations are defined by Equations 5, 3, and 4, respectively.
7. Genetic Algorithm for Cycle Optimization
The pseudo-random nature used in the creation of all three types of cycles led to the problem of how to determine whether or not a cycle best represented the overall data set. This fact led to the question of how to select a final cycle. The Markov chain technique allowed for the creation of a plethora of unique cycles. In order to evaluate created cycles, the SV and DV were calculated for each cycle and a weighted combination of the two was used to define the PV. The equations used to calculate these values were defined in the previous section. The PV was the deciding factor in determining which cycle best represented the overall data set. The PV was used because it took into account both the statistical metrics and the distribution of the cycle. While this definition is somewhat arbitrary, any definition of representativeness would be at the mercy of the user.

The use of Markov chain logic should, in theory, have created cycles having a similar distribution to that of the overall data. Sometimes, however, a cycle can get stuck in a certain bin or “space,” or a cycle may move through the space while choosing low probability events. This is one of the side effects of the random nature of the Markov chain. MCMC simulation is the idea of creating a Markov chain that has an equivalent distribution to one that is desired. This can theoretically be accomplished with the creation of one long chain or several smaller chains. Ideally, a cycle constructed under the logic of a Markov chain would exactly match the desired distribution if the cycle was allowed to grow to infinite length. The other option available to best represent the desired distribution would be to create a “large” number of cycles until one is created with the desired distribution. The first option could not be practically applied to this problem due to the fact that cycles must have a reasonable length. They must be completed in the laboratory on an engine dynamometer, which can be cost and time intensive. The second option raised a number of questions. Since an infinite number of cycles could not be created due to time and memory limitations, the main issue was answering the question: “How large must the sample size be for a cycle of optimal representation to be found?” It was not clear that the creation of hundreds of thousands of cycles would produce an optimal candidate cycle. In an initial attempt to find the optimal cycle for each of the three prime-movers, a population of 10,000 cycles was created for each using the respective Markov_Chain_Cycle_Construction.m scripts. The best PV from these populations of OTR truck, drilling, and hydraulic fracturing cycles were 0.0327, 0.0042, and 0.0072, respectively. These cycles could have been used as the final cycles, but a more definitive method, which did not depend on population size was desired. The process of finding “the best possible” cycle required the use of an optimization technique. A GA was chosen as the optimization technique to be used. The GA is an optimization technique that can be used in computing solutions to a variety of complex problems. A GA was used for optimizing all three types of cycles. An initial population of 100 random cycles was selected for each of the GAs. The cycles were randomly selected to ensure that a wide range of genetic material
would be present in the initial population. Previous work on the influence of population size has shown that there is no benefit of having a population size greater than 100 [62].

7.1 Over-the-Road Trucks
Optimization of representativeness was the goal of the GA created for cycle development. As was seen in the review of literature, GAs are open to customization by the user. The primary ways that these algorithms can be altered based on the problem is through chromosome and gene definition, population size, stopping point, genetic operators used, rates of the genetic operators, and selection technique. The definition of these metrics was done after the structure of the GA was developed.

The MATLAB® script Trucking_GA.m defined the GA used for OTR trucks. This script called a number of other scripts which were necessary for the algorithm to run effectively. All of the scripts are found in Appendix C. The first issue encountered was determining which aspects of cycles should be considered chromosomes and how to define genes. The chromosomes or individuals were defined by the overall cycles constructed by the Markov chain technique. The individuals were composed of 1800 point arrays of speed and load pairs. These individuals all had SVs, DVs, and PVs associated with them as well as distribution matrices and average cycle statistics.

Genes were created from these individuals. Typically when using GAs, individuals are converted to binary strings and genes are represented by bits in the string. It was impractical to convert the entire array to binary representation and so a methodology was developed to define genes. Each cycle was “segmented” into portions of different activity. The segments were defined in a similar manner to microtrips from previous studies [45] [46] [52]. Instead of using vehicle speed to define segments, engine speed was used. Periods of idle and non-idle were defined as different segments. A cycle displaying different segment types is shown in Figure 17.
An idle segment was any length of the cycle longer than ten seconds in which the speed remained at 0% of rated. The length of ten seconds was used because it was believed that short segments would have a minimal impact on cycle statistics and distribution. An example of an idle segment is shown in Figure 18.

A non-idle segment was defined as any length of the cycle in which there were more than ten consecutive seconds of non-idle operation. An example of a non-idle segment is shown in Figure 19. The identification of these segments was possible because of the zeroing of the points described earlier. The points that were zeroed included single points of non-zero speed in between two points of zero speed and points with a normal speed value less than 2%.
The segmentation of cycles was performed by the script `Trucking_Segment_Formation.m` shown in Appendix C. The script was used to segment the entire population of a given generation of cycles into genes.

Once genes were defined, selection and implementation of genetic operators were applied. Mutation was applied first. The cycles selected for mutation were randomly chosen based on the mutation rate. Choosing cycles to receive mutation was accomplished by generating a random number for each cycle in the population. If the number generated was less than the rate of mutation then the cycle in question experienced mutation. Mutation was defined as the replacement of a segment of the selected cycle by a segment randomly chosen from the pool of segments. This was performed by completing a number of steps. The first step was to select the type of gene which would be inserted into the cycle. This was achieved with a weighted selection of either idle or non-idle based on the percentage of each type of segment. If the segment was determined to be idle then the script `Trucking_Idle_Segment_Selection.m` was implemented to choose an idle segment. This script randomly selected an idle segment from the pool created based on the current population. The length of that segment was then identified and a starting position was randomly selected from between 60s and 1740s of the current cycle. A 60s period from the start and end of the cycle was used for two reasons. The beginning 30s were excluded because the initial period was intended to simulate engine start and idle in the field. An additional 30s was added to this time in hopes that the cycle would leap out of the idle period in this amount of time. This did not always occur depending on the Markov chain construction. Sixty seconds were excluded at the end with the hope that
the idle period at the end of the cycle would be uninterrupted. Once the starting point was selected, the length of the segment was added to that number to determine the ending position in the cycle. If both of these values were idle points, the gene was inserted into the cycle, if not the process was continued until both the starting and ending position were 0% speed points. Looking for starting and ending points that were idle eliminated the worry of concatenating segments that had different starting and ending speed values. If a non-idle segment was selected as the gene then the script *Trucking_Nonidle_Segment_Selection.m* was used to determine the gene. This script used the same logic as *Trucking_Idle_Segment_Selection.m* but chose a gene from the pool of non-idle segments. Any cycle that experienced mutation was added to the population, but did not replace the older version of itself. This allowed the population to grow within the current generation.

Once each cycle was given a chance to experience mutation, then selection of cycles for crossover was performed. Determining cycles to experience crossover relied on random number generation for each of the cycles in the initial population of the current generation. The cycles that were added to the population due to mutation were not eligible for crossover, but the cycles that they were derived from were eligible. If the random number applied to a given cycle was less than the defined crossover rate then the cycle was selected for crossover. In order to perform crossover, another cycle had to be selected from the population. This cycle was selected randomly from the remaining 99 cycles of the current generation. The script *Trucking_Crossover_Point.m* was initialized to choose a point in the cycles where crossover could occur. This script scanned all points of both cycles between 31s and 1770s and selected idle points. All of the idle points that occurred in the same location of both cycles were saved to an array. That array of common idle points was then randomly sampled to find a point at which the cycles would cross over. If no common point was found then the crossover point was randomly selected from the first 30s of the cycle. This type of selection would have no impact on the resultant cycle speed, but could change the load values and in turn change the statistics. This condition rarely occurred and was only applied to account for anomalies in a generation of the GA and prevent an infinite loop. When two cycles were crossed, both newly created cycles were added to the current population.

The new population size was determined by adding the number of newly created cycles from mutation and crossover to the original population size of the generation which was always 100. The cycle statistics of the newly created cycles were calculated, since newly formed cycles did not contain any information other than the speed and load array. The *Trucking_Cycle_Statistics.m* script was used to calculate statistics of newly created cycles. This script was similar to *Trucking_Markov_Chain_Cycle_Construction.m*. It binned the points so that a cycle distribution could be defined. The script also calculated statistics including: average speed, load, non-idle speed and load, and
normalized idle time. With a distribution defined and statistics calculated, the DV, SV, and PV were found using Equations 2, 3, and 4 respectively. This resulted in newly created cycles having the same structure as the original cycles shown in Figure 15.

The next step in the GA was to sort the cycles based on their respective PVs in order to determine which individuals were most representative. The Sort_Values_PV.m script was used to perform this action. This script took the PV from each cycle and sorted them from least to greatest. This allowed the lowest PV cycle, which was considered most representative, to be the top cycle in the population and others to fall in line according to their respective PVs. Some additional conditional statements were required to eliminate values that were undefined due to a lack of data distribution or valid statistics. This sometimes occurred in created cycles if they were exclusively idle, a rare occurrence. The index of the cycle was saved with the PV array so that sorting of the cycles was simplified. The PV was desired to be as close to zero as possible; however, it was not well scaled between zero and one over the entire population. Many of the cycles within a given population had PVs much closer to zero than one. In order to more evenly distribute the population so that selection of the next population could be implemented “evaluation” and “fitness” functions were defined. These functions were applied in the script Trucking_Evaluation_Function.m and were necessary so that the selection of individuals for the next generation could be achieved with roulette-wheel selection. In order to scale the PV over the range of [0, 1] to create selection probabilities, the largest and smallest PV of the generation were defined as PV_{max} and PV_{min} respectively. These values were then used to create coefficients a and b, which were in turn used to scale the PV into an “Evaluation Function” (EF), as shown in Equations 6, 7, and 8.

\[
a = \frac{1}{P_{V_{max}} - P_{V_{min}}} \tag{Equation 6}
\]

\[
b = -\frac{P_{V_{min}}}{P_{V_{max}} - P_{V_{min}}} \tag{Equation 7}
\]

\[
E_{F_i} = a \times P_{V_i} + b \tag{Equation 8}
\]

The use of these equations allowed an evaluation function to be defined for every cycle. Since a lower PV was desired it was more valuable to define the inverse of the EF desired. An inverse EF value was defined as the EF value calculated subtracted from one as shown in Equation 9.

\[
E_{F_{new}} = 1 - E_{F_{old}} \tag{Equation 9}
\]
These values were summed into a total fitness (TF) value, as shown in Equations 10, and a probability array was then constructed based on each individual’s evaluation function, as is shown in Equation 11.

\[ TF = \sum_{i=1}^{n} EF_i \]  
\[ p_i = \frac{EF_i}{TF} \]

A “q” array was defined which acted as a number line with spacing equivalent to the probabilities defined from Equation 12.

\[ q_i = \sum_{j=1}^{i} p_j \]

Equations 10-12 are defined by Perhinschi et al. [52]. With the “q” array defined, roulette-wheel selection could take place to define the next population. In addition to roulette-wheel selection, elitist selection was also implemented to ensure the cycle with the lowest PV advanced to the next generation. Elitist selection is applied to GAs to improve exploitation and reduces variability between generations. Roulette-wheel selection was used to define the rest of the population. This type of selection implies that each individual owns a piece of the wheel proportional to its overall fitness. The “q” array acts as the wheel. To implement the selection process a random number between zero and one is generated. The number falls somewhere on the “q” number line as described above. The index of the “q” value falling directly to the right of the random number was the index of the cycle that was chosen to advance. It is noted that this was performed without removal of the cycle from the pool. This meant that it was possible for the same cycle to have multiple copies in the next generation. Other cycles which were not selected to advance became extinct. Random numbers were generated until the defined population size of 100 was obtained. The cycles that were selected advanced to the next generation, and became the initial population of that generation. Those cycles which were not chosen to advance were eliminated. This completed the generation of the GA and the next generation began with the same procedure.

7.2 Drilling
The GA logic for drilling was nearly identical to that of OTR trucks with some minor differences. The MATLAB® scripts used for drilling can be found in Appendix C. Several identical scripts were used
from the OTR truck GA. The scripts that are referenced in the drilling GA, but are not found in Appendix C: Drilling, were identical to those used for OTR trucks.

The primary difference in the drilling GA was the definition of genes. Genes, or segments, used for drilling were formed by observing periods of low-load and high-load. A low-load drilling segment was defined as a period of a cycle in which the load remained below 40% for more than ten seconds. A high-load segment was one in which the load remained at or above 40% for more than ten seconds. The selection of genes for mutation was performed in the same way as in the GA used for OTR trucks, except, rather than randomly selecting an idle or non-idle gene, a high- or low-load gene was selected. The same logic was used for insertion of genes into selected cycles, except that instead of start and end points having to be defined by idle points, they were required to be low-load points. The same logic was applied to the points of potential crossover. Instead of the potential crossover points being defined by idle speed, the points at which crossover could occur were defined by low-load.

Once mutation and crossover were applied at their defined rates, the statistics of the newly created cycles were calculated. These statistics were outlined in the Cycle Creation section. The distribution, averages, normalized times, SVs, DVs, and PVs were all calculated in the same way as they were during cycle creation. The exact same logic was used after the calculation of statistics including: sorting the PVs, rescaling using the evaluation and fitness functions as well as elitist and roulette-wheel selection techniques. Equations 6-12 were also applied to the drilling GA. The structure of the GA was nearly identical and is defined by Drilling_GA.m shown in Appendix C.

7.3 Hydraulic Fracturing
The MATLAB® scripts used for the hydraulic fracturing GA are found in Appendix C. The scripts were not distinguished from those used for drilling because the logic was identical to that of drilling. There were no differences in any of the steps of the GA other than initial cycle import. Different naming convention was used to make troubleshooting simpler and to allow for clearer organization of MATLAB® directories. An example of this is shown in the Fracking_GA.m script shown in Appendix C.
8. Results of Genetic Algorithm

The use of a GA allowed for the optimization of cycle representativeness and made selection of final cycles straightforward, because the algorithm was meant to produce the best cycle. One issue in defining the GA was finding the genetic operation rates that would most efficiently find a solution. In other words, it was necessary to select the best form of the GA to use in cycle development. Several iterations of the GA were applied in an attempt to find the best performing genetic operation rates. During these iterations, 30 generations was used as the GA lifespan and the initial population was set to 100. The initial population of cycles were created by the respective Marko_Chain_Cycle_Construction.m scripts for each type of cycle. The genetic operation rates were varied in 5% increments during these iterations while other variables were held constant and each of the iterations was ranked based on the average value of the elite PV that was produced. It is understood that further evaluation of the operators could yield a lower average PV if the genetic operation rates were evaluated with higher precision. The goal of this work was to use the GA as an optimization tool, not to optimize the GA itself. The operators that produced the lowest average elite PV amongst the three cycles were used for the final GA. The results of this analysis are shown in Table 20.

Table 20: Genetic Algorithm Iteration Results

<table>
<thead>
<tr>
<th>GA Rank</th>
<th>Crossover Rate</th>
<th>Mutation Rate</th>
<th>Drilling Elite PV Rank</th>
<th>Hydraulic Fracturing Elite PV Rank</th>
<th>Trucking Elite PV Rank</th>
<th>Total PV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.2</td>
<td>0.0029 1</td>
<td>0.0063 5</td>
<td>0.0301 1</td>
<td>0.0640</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.25</td>
<td>0.0035 2</td>
<td>0.0051 1</td>
<td>0.0336 5</td>
<td>0.0669</td>
</tr>
<tr>
<td>3</td>
<td>0.35</td>
<td>0.2</td>
<td>0.0035 2</td>
<td>0.0073 7</td>
<td>0.0322 2</td>
<td>0.0677</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>0.15</td>
<td>0.0036 3</td>
<td>0.0057 3</td>
<td>0.0340 6</td>
<td>0.0680</td>
</tr>
<tr>
<td>5</td>
<td>0.45</td>
<td>0.2</td>
<td>0.0036 3</td>
<td>0.0070 6</td>
<td>0.0335 4</td>
<td>0.0688</td>
</tr>
<tr>
<td>6</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0048 4</td>
<td>0.0061 4</td>
<td>0.0334 3</td>
<td>0.0690</td>
</tr>
<tr>
<td>7</td>
<td>0.3</td>
<td>0.2</td>
<td>0.0048 4</td>
<td>0.0078 8</td>
<td>0.0322 2</td>
<td>0.0695</td>
</tr>
</tbody>
</table>

Based on the analysis, the final genetic operation rates of 0.4 or 40% for crossover and 0.2 or 20% for mutation yielded the best results. These rates were selected as the rates for the final GA which was applied to all three types of cycles. For each of the three types of cycles an initial population of 100 cycles was applied to the final GA. Each final GA was defined to have a lifespan of 50 generations. The lifespan of the GA was extended with the goal of giving better results than those seen in the initial iterations. Fifty generations was chosen because the improvements of the GA slowed considerably after that point. Shorter lifespans were used in the initial iterations to save computational time. The results of each of the final GAs are shown in the following sections.
8.1 Over-the-Road Trucks
The initial population of 100 created truck cycles had an average PV of 0.0910 and after 50 generations the average PV had been reduced by 61% to 0.0354. The elite PV of the initial population was 0.0388 and was reduced by 18% to 0.0320 by the final generation. Figure 20 shows the improvements of the average and elite PVs as a function of the GA generation. At the 50th generation, the average PV improved by only 0.35% from the previous generation and the elite PV did not improve at all. The final elite OTR truck cycle is shown in Figure 21. The complete normalized cycle is shown in Appendix G. The statistics of the final OTR truck cycle compared to those of the data are shown in Table 21. The differences in statistics and normalized times between the cycle and the data were all less than 1% except for the average load and non-idle load which showed differences of 1.8% and 6.2%, respectively. A similar real world data segment is shown in Appendix H.

![Figure 20: OTR Truck GA Improvements](image-url)
While data derivatives were not included in the comparison with to calculate the “Values,” it is understood that these values were important. The cycle should only be constructed by derivatives that were obtainable due to the nature of the Markov chain. A comparison of the data and cycle average positive and negative derivatives was performed after the final cycle was created for both speed and load. The average difference in the derivatives between the cycle and the data was 10%.

![Normalized Engine Speed (%) vs Time (s)](image1.png)

![Normalized Engine Load (%) vs Time (s)](image2.png)

**Figure 21: Elite OTR Truck Cycle**

**Table 21: OTR Truck Cycle and Data Statistics**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
</tr>
<tr>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td>0.2911</td>
</tr>
<tr>
<td>Load</td>
<td>0.2269</td>
</tr>
<tr>
<td>Non-Idle Speed</td>
<td>0.4665</td>
</tr>
<tr>
<td>Non-Idle Load</td>
<td>0.2649</td>
</tr>
<tr>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Idle</td>
<td>0.3761</td>
</tr>
<tr>
<td>Non-Idle</td>
<td>0.6239</td>
</tr>
<tr>
<td>Values</td>
<td></td>
</tr>
<tr>
<td>Statistical</td>
<td>0.0383</td>
</tr>
<tr>
<td>Distribution</td>
<td>0.0258</td>
</tr>
<tr>
<td>Performance</td>
<td>0.0320</td>
</tr>
</tbody>
</table>
8.2 Drilling
The initial population of 100 created drilling cycles had an average PV of 0.0253. After 50 generations, the average PV had been reduced by 79% to 0.0052. The elite PV of the initial population was 0.0079 and was reduced by 62% to 0.0030 by the end of the 50th generation. Figure 22 shows the improvements of the average and elite PVs as a function of the GA generation. The average PV at the 50th generation actually was worse than it was in the 49th generation and the elite PV showed no improvement signaling that the GA had maximized performance. The final elite drilling cycle is shown in Figure 23. The complete normalized cycle is shown in Appendix G. The statistics of the final drilling cycle compared to those of the data are shown in Table 22. The table shows that all statistical averages and normalized time values were within 2.1% of the respective data values except for idle time. This value was defined by 0% load and was an anomaly during drilling operation since the engine was nearly always loaded. The drilling idle time made up only 0.4% of all time during drilling operation. A similar real world data segment is shown in Appendix H.

![Figure 22: Drilling GA Improvements](image-url)
Table 22: Drilling Cycle and Data Statistics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>Load</td>
<td>0.4295</td>
<td>0.4298</td>
</tr>
<tr>
<td></td>
<td>High-Load</td>
<td>0.5997</td>
<td>0.6024</td>
</tr>
<tr>
<td></td>
<td>Low-Load</td>
<td>0.1934</td>
<td>0.1895</td>
</tr>
<tr>
<td>Time</td>
<td>High-Load</td>
<td>0.5850</td>
<td>0.5837</td>
</tr>
<tr>
<td></td>
<td>Low-Load</td>
<td>0.4067</td>
<td>0.4123</td>
</tr>
<tr>
<td></td>
<td>Idle</td>
<td>0.0083</td>
<td>0.004</td>
</tr>
<tr>
<td>Values</td>
<td>Statistical</td>
<td>0.0025</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Distribution</td>
<td>0.0035</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>0.0030</td>
<td></td>
</tr>
</tbody>
</table>

8.3 Hydraulic Fracturing
The initial population of 100 hydraulic fracturing cycles had an average PV of 0.0511. After 50 generations, the average PV was reduced by 81% to 0.0095. The elite PV of the initial population was 0.0155 and was reduced by 65% to 0.0054 by the end of the 50th generation. Figure 24 shows the improvements of the average and elite PVs as a function of the GA generation. The 50th generation elite PV showed no improvement over the 49th generation and the average PV only improved by 0.55%. The
final elite hydraulic fracturing cycle is shown in Figure 25. The complete normalized cycle is shown in Appendix G. The statistics of the final hydraulic fracturing cycle compared to those of data are shown in Table 23. The table shows the differences in the values of the average statistics and the normalized time values. The all cycle values were less than 3% different from the respective data set values, with the exception of average low load percentage. This value showed a percent difference of 6.6%, however, the real difference was just over 1%. A similar real world data segment is shown in Appendix H.

![Figure 24: Hydraulic Fracturing GA Improvements](image-url)
Table 23: Hydraulic Fracturing Cycle and Data Statistics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Data</td>
</tr>
<tr>
<td>Average</td>
<td>Load</td>
<td>0.3583</td>
</tr>
<tr>
<td></td>
<td>High-Load</td>
<td>0.6174</td>
</tr>
<tr>
<td></td>
<td>Low-Load</td>
<td>0.2167</td>
</tr>
<tr>
<td>Time</td>
<td>High-Load</td>
<td>0.4606</td>
</tr>
<tr>
<td></td>
<td>Low-Load</td>
<td>0.3411</td>
</tr>
<tr>
<td></td>
<td>Idle</td>
<td>0.1983</td>
</tr>
<tr>
<td>Values</td>
<td>Statistical</td>
<td>0.0055</td>
</tr>
<tr>
<td></td>
<td>Distribution</td>
<td>0.0052</td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>0.0054</td>
</tr>
</tbody>
</table>
9. Cycle Denormalization and Verification
Once final cycles were developed they were implemented on a test engine and dynamometer to ensure that they could be performed. To do this, the cycles were denormalized for the engine to be tested. The engine used for the purpose of cycle verification was a 2008 Cummins 8.9L ISLG. This dedicated natural gas engine is rated for 280 HP. Engine specifications are shown in Table 24.

Table 24: Cummins ISLG Engine Specifications [63]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horsepower</td>
<td>280</td>
<td>HP</td>
</tr>
<tr>
<td>Peak Torque</td>
<td>900</td>
<td>lb-ft</td>
</tr>
<tr>
<td>Rated Speed</td>
<td>2200</td>
<td>rpm</td>
</tr>
<tr>
<td>Governed Speed</td>
<td>2400</td>
<td>rpm</td>
</tr>
<tr>
<td>Ignition</td>
<td>Spark</td>
<td></td>
</tr>
<tr>
<td>Arrangement</td>
<td>Inline - 6 Cylinder</td>
<td></td>
</tr>
<tr>
<td>Intake</td>
<td>Turbocharged</td>
<td></td>
</tr>
<tr>
<td>Displacement</td>
<td>540</td>
<td>in$^3$</td>
</tr>
<tr>
<td>Bore</td>
<td>4.49</td>
<td>in</td>
</tr>
<tr>
<td>Stroke</td>
<td>5.69</td>
<td>in</td>
</tr>
<tr>
<td>Oil Capacity</td>
<td>7.3</td>
<td>US gal</td>
</tr>
<tr>
<td>Coolant Capacity</td>
<td>13.1</td>
<td>US qts</td>
</tr>
<tr>
<td>System Voltage</td>
<td>12</td>
<td>V</td>
</tr>
<tr>
<td>Aftertreatment</td>
<td>TWC</td>
<td></td>
</tr>
<tr>
<td>Fuel Used</td>
<td>CNG</td>
<td></td>
</tr>
</tbody>
</table>

Denormalization of the cycle involves converting the normalized speed and load percentages into engine speed and torque. The idle and rated speeds of the engine are used to find the actual speed of the engine in revolutions per minute. The idle and rated speeds used for denormalization were taken from the engine map file shown in Figure 26. The map defined a rated speed of 2207 rpm and an idle speed of 701 rpm. Equation 13 is used for the denormalization of engine speed which is applied by the Matlab® script Denormalization.m shown in Appendix D. This denormalization or “unnormalization” equation used was based on the definition for Otto-cycle engines in the CFR Title 40 Chapter I Subchapter C Part 86 Subpart N Section 1333. It should be noted that the CFR utilized maximum test speed for denormalization of cycle points. This was considered, however, rated speed was used instead due to the fact that the drilling and hydraulic fracturing engines never ran above the rated speed and the fact that only 0.283% off all OTR truck data occurred above the rated speed.

\[
\text{Speed (rpm)} = \text{Normalized } \times (\text{Rated } - \text{Idle}) + \text{Idle} \quad \text{Equation 13}
\]
This equation is applied to each point in the cycle. The speed traces of the drilling and fracturing cycles were applied at this point. The values of the cycles were all set to 1.0 or 100% of rated speed, except for the first and last ten seconds. These speed points were defined as a linear trend from zero to one to allow for smooth start-up and shutdown of the engine. Once the speed of each point was denormalized, the torque of the engine was defined. The percent load output of the engine was defined as a percentage of the maximum available load at the given speed. The percent load was multiplied by the maximum available torque at a given speed to find the torque of that point. The torque was found using a map of the engine, which was obtained after the engine was installed on the dynamometer. The engine map of the Cummins ISLG used for testing was taken on March 18, 2016 and is shown in Figure 26. The process of denormalizing the engine torque was repeated for every point in the cycle so that each final value was given in lb-ft. The denormalized cycles based on the Cummins ISLG for OTR trucks, drilling and hydraulic fracturing engines are shown in Figures 27, 28, and 29 respectively.

![Figure 26: ISLG Lug Curve](image-url)
Figure 27: Denormalized OTR Truck Cycle for Cummins ISLG

Figure 28: Denormalized Drilling Cycle for Cummins ISLG
Before created cycles could be considered final, they had to be verified to ensure that they could be implemented by the dynamometer and engine combination used. Determining whether or not a cycle could be successfully applied on the engine and dynamometer was verified by testing the regression criteria. Regression statistics must be within a certain range according to the CFR for the cycle to be considered valid for the engine and dynamometer combination. If the cycle does not pass one of the criteria, then the cycle must be adjusted in a way that will allow for it to pass all of the criteria. The criteria for validation are defined in the CFR Title 40 Chapter I Subchapter U Part 1065 Subpart F Section 514. The ISLG specific criteria are shown in Table 25.
Table 25: Cycle Regression Validation Criteria for Cummins ISLG

<table>
<thead>
<tr>
<th>Cycle Validation Criteria</th>
<th>Acceptable Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-70.1</td>
</tr>
<tr>
<td>Slope</td>
<td>0.95</td>
</tr>
<tr>
<td>R-Squared</td>
<td>0.97</td>
</tr>
<tr>
<td>SEE</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Torque</strong></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-18.0</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
</tr>
<tr>
<td>R-Squared</td>
<td>0.85</td>
</tr>
<tr>
<td>SEE</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-5.70</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
</tr>
<tr>
<td>R-Squared</td>
<td>0.91</td>
</tr>
<tr>
<td>SEE</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Work</strong></td>
<td>Actual Bhp-hr</td>
</tr>
</tbody>
</table>

Only hot start testing was applied for the purpose of regression criteria. Calculation of regression statistics was completed by comparing the desired set-point values defined by the cycle to the values that were actually seen by the dynamometer. Regression statistics were determined by eliminating points considered invalid by the CFR. Points that could be eliminated when calculating regression statistics are defined in CFR Title 40 Chapter I Subchapter U Part 1065 Subpart F Section 514. The points that are permitted for deletion are as follows:

1. Torque and/or BHP when the engine experiences wide open throttle and torque feedback is less than the reference torque
2. Torque and/or BHP when the engine experiences closed throttle during a point that is not idle and the torque feedback is less than the reference torque
3. Speed and/or BHP when the engine experiences closed throttle during an idle point and the torque feedback is equal to the Curb Idle Transmission Torque (CITT)

The elimination of these points was done with the intention of improving the regression statistics of the tested cycle. A check and balance system was put in place by the total work criterion. This value must be within 5% of the desired value. If points were eliminated from the cycle causing the total work of the cycle to be less than 5%, the cycle would fail regression validation criteria.

The Cummins ISLG was tested on a General Electric (GE) 800 HP Direct Current (DC) Dynamometer. The dynamometer is shown in Figure 30.
A dynamometer calibration was necessary to ensure that the values of torque recorded were accurate. A ten point torque calibration was performed on the GE dynamometer on February 26, 2016. Calibrating the dynamometer was achieved by placing different weights on the side-arm extension of the dynamometer. These known weights were compared to those reported by the dynamometer. Table 26 shows the values obtained during calibration and the calibration curve is shown in Figure 31. The dynamometer calibration file is shown in Appendix D.

Table 26: Dynamometer Side-Arm Calibration Points

<table>
<thead>
<tr>
<th>Point</th>
<th>Actual Value</th>
<th>Computed Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#)</td>
<td>(lbs)</td>
<td>(lbs)</td>
<td>(%)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.4</td>
<td>0.10%</td>
</tr>
<tr>
<td>2</td>
<td>131</td>
<td>131.5</td>
<td>0.43%</td>
</tr>
<tr>
<td>3</td>
<td>306</td>
<td>305.2</td>
<td>-0.24%</td>
</tr>
<tr>
<td>4</td>
<td>481</td>
<td>480.9</td>
<td>-0.02%</td>
</tr>
<tr>
<td>5</td>
<td>656</td>
<td>656.2</td>
<td>0.03%</td>
</tr>
<tr>
<td>6</td>
<td>831</td>
<td>830.9</td>
<td>-0.01%</td>
</tr>
<tr>
<td>7</td>
<td>1006</td>
<td>1005.3</td>
<td>-0.07%</td>
</tr>
<tr>
<td>8</td>
<td>1181</td>
<td>1180.3</td>
<td>-0.05%</td>
</tr>
<tr>
<td>9</td>
<td>1356</td>
<td>1356.4</td>
<td>0.03%</td>
</tr>
<tr>
<td>10</td>
<td>1531</td>
<td>1531.7</td>
<td>0.04%</td>
</tr>
</tbody>
</table>
Regression testing was performed with a Cummins ISLG on the 800 HP GE Dynamometer. The engine testing was performed in WVU’s CAFEE Engines and Emissions Research Center (EERC) heavy-duty test cell. The Cummins ISLG test cell setup on the 800 HP dynamometer is shown in Figure 32.
A configuration file was required by the dynamometer for the specific engine that was tested. The configuration parameters in the file influence the dynamometer’s proportional-integral-derivative (PID) controller settings which were used to control the engine throttle to meet the set points defined by the cycle being performed. These parameters should be defined differently based on how engines respond to the dynamometer setup. The gains defined in the configuration file used for regression testing were defined during a previous testing phase of the Cummins ISLG. The speeds were determined from the engine map. The parameters of the file are defined in Table 27.

Table 27: Parameters of the Engine Configuration File

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportional Gain</td>
<td>0.0044</td>
</tr>
<tr>
<td>Steady State Proportional Gain</td>
<td>0.00025</td>
</tr>
<tr>
<td>Integral Gain</td>
<td>0.0003</td>
</tr>
<tr>
<td>Derivative Gain</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Throttle Position</td>
<td>10</td>
</tr>
<tr>
<td>Throttle Slope</td>
<td>0</td>
</tr>
<tr>
<td>Max Throttle</td>
<td>100</td>
</tr>
<tr>
<td>Dyno Mode</td>
<td>SPEED</td>
</tr>
<tr>
<td>Stop Method</td>
<td>None</td>
</tr>
<tr>
<td>Shutdown Limit</td>
<td>2257</td>
</tr>
<tr>
<td>Look Ahead Rise Time</td>
<td>10</td>
</tr>
<tr>
<td>Look Ahead Fall Time</td>
<td>15</td>
</tr>
<tr>
<td>Look Ahead Else Time</td>
<td>15</td>
</tr>
<tr>
<td>Sigmoid Denominator</td>
<td>300</td>
</tr>
<tr>
<td>Idle Speed</td>
<td>701</td>
</tr>
<tr>
<td>High Idle Speed</td>
<td>2371</td>
</tr>
<tr>
<td>Rated Speed</td>
<td>2207</td>
</tr>
<tr>
<td>Rated Power</td>
<td>285.5</td>
</tr>
<tr>
<td>Peak Torque</td>
<td>902.3</td>
</tr>
<tr>
<td>Intermediate Speed</td>
<td>1356</td>
</tr>
</tbody>
</table>

It was important to ensure that the same engine configuration parameters were used when comparing cycles for regression, as changing them could change how the cycle performed on the dynamometer. Operating the cycle on the dynamometer also required a “.cyc” file containing the information of the cycle in question. These files are custom for CAFEE test software. The “.cyc” files were created from a text file created as part of the MATLAB® script *Denormalization.m*, which is shown in Appendix D. The format of the “.cyc” files for each of the denormalized cycles tested can be found in Appendix E. One adjustment had to be made to the OTR cycle before it could be tested on the dynamometer. The first
point in the denormalized cycle had to be changed from 701 rpm and 10.2498 lb-ft to 700 rpm and 0.000 lb-ft. This allowed the dynamometer to control idle points correctly. The dynamometer interpreted the first point in the cycle as the “idle” setting. Therefore, it would not be active if values below that point were seen in the cycle. So, in order for set points of idle speed with a load above zero to be achieved, the lowest possible speed and torque had to be defined by the first point in the cycle. The cycles were tested on the dynamometer for regression on April 7, 2016. The regression statistics of each of the three original cycles are shown in Tables 28, 29, and 30 for OTR trucks, drilling, and hydraulic fracturing respectively.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Min</th>
<th>Value</th>
<th>Max</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-70.1</td>
<td>≤ 0.509</td>
<td>≤ 70.1</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.95</td>
<td>≤ 0.999</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.97</td>
<td>≤ 0.999</td>
<td>≤ 1.00</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE</td>
<td>0.0</td>
<td>≤ 9.467</td>
<td>≤ 110.4</td>
<td>Pass</td>
</tr>
<tr>
<td>Torque</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-18.0</td>
<td>≤ 6.079</td>
<td>≤ 18.0</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>≤ 0.948</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.85</td>
<td>≤ 0.843</td>
<td>≤ 1.00</td>
<td>FAIL</td>
</tr>
<tr>
<td>SEE</td>
<td>0.0</td>
<td>≤ 79.409</td>
<td>≤ 90.0</td>
<td>Pass</td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-5.7</td>
<td>≤ 1.404</td>
<td>≤ 5.7</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>≤ 0.961</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.91</td>
<td>≤ 0.869</td>
<td>≤ 1.03</td>
<td>FAIL</td>
</tr>
<tr>
<td>SEE</td>
<td>0</td>
<td>≤ 22.99</td>
<td>≤ 28.5</td>
<td>Pass</td>
</tr>
<tr>
<td>Work</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Bhp-hr:</td>
<td>20.80</td>
<td>≤ 22.73</td>
<td>≤ 22.99</td>
<td>Pass</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 28 shows that the OTR truck cycle failed regression based on the coefficient of determination (R²) of torque and power. Generally, if an engine fails regression based on a torque criterion, it will also fail based on the same criteria under power. This is due to the fact that power is derived from torque and both depend on the fueling response of the engine rather than control of the dynamometer. Cycles rarely fail based on speed criteria because the dynamometer controls the speed directly. Since the OTR cycle did not pass regression some alteration had to be made to the cycle so that it could. Section 86.1341-90 of the CFR states “If a dynamometer test run is determined to be statistically or experimentally void, corrective action shall be taken.” The corrective action in this case was to change the points of the cycle to make them more attainable by the dynamometer and the engine. The fact that the cycles were artificially developed made changing the cycles the logical course of action, rather than changing the PID gains of the dynamometer. It was also desired to keep the engine configuration file and PID controller constant across all tests. In order to make the points in the cycle more obtainable by the engine and dynamometer
combination a smoothing technique was applied to the speed and load arrays of the OTR cycle. This is discussed further in Chapter 10: Smoothing.

Table 29: Drilling Cycle Regression Results

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Min</th>
<th>Value</th>
<th>Max</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept:</td>
<td>-70.1</td>
<td>≤</td>
<td>70.1</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope:</td>
<td>0.95</td>
<td>≤</td>
<td>1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared:</td>
<td>0.97</td>
<td>≤</td>
<td>1</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE:</td>
<td>0</td>
<td>≤</td>
<td>110.35</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Torque</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept:</td>
<td>-18.0</td>
<td>≤</td>
<td>18.0</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope:</td>
<td>0.83</td>
<td>≤</td>
<td>1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared:</td>
<td>0.85</td>
<td>≤</td>
<td>1</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE:</td>
<td>0</td>
<td>≤</td>
<td>90</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept:</td>
<td>-5.7</td>
<td>≤</td>
<td>5.7</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope:</td>
<td>0.83</td>
<td>≤</td>
<td>1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared:</td>
<td>0.91</td>
<td>≤</td>
<td>1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE:</td>
<td>0</td>
<td>≤</td>
<td>28.5</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Work</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Bhp-hr:</td>
<td>57.85</td>
<td>≤</td>
<td>63.94</td>
<td>Pass</td>
</tr>
<tr>
<td>Reference:</td>
<td>60.897</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 29 shows that the drilling cycle passed regression based on all criteria. This allowed the drilling cycle to be used as the final cycle as it was originally defined.
Table 30: Hydraulic Fracturing Cycle Regression Results

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Min</th>
<th>Value</th>
<th>Max</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-70.1</td>
<td>≤ -22.419</td>
<td>≤ 70.1</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.95</td>
<td>≤ 1.010</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.97</td>
<td>≤ 0.998</td>
<td>≤ 1</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE</td>
<td>0</td>
<td>≤ 6.113</td>
<td>≤ 110.35</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Torque</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-18.0</td>
<td>≤ 0.335</td>
<td>≤ 18.5</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>≤ 1.000</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.85</td>
<td>≤ 0.962</td>
<td>≤ 1</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE</td>
<td>0</td>
<td>≤ 34.967</td>
<td>≤ 90</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-5.7</td>
<td>≤ 0.346</td>
<td>≤ 5.7</td>
<td>Pass</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>≤ 0.999</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.91</td>
<td>≤ 0.966</td>
<td>≤ 1.03</td>
<td>Pass</td>
</tr>
<tr>
<td>SEE</td>
<td>0</td>
<td>≤ 13.738</td>
<td>≤ 28.5</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Work</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Bhp-hr</td>
<td>48.26</td>
<td>≤ 50.677</td>
<td>≤ 53.34</td>
<td>Pass</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
<td>50.803</td>
</tr>
</tbody>
</table>

Table 30 shows that the hydraulic fracturing cycle passed regression based on its original definition. Therefore, the original cycle was validated as the final cycle.
10. **Smoothing and Final Verification**

The failure of regression by the OTR cycle was most likely the result of the second-by-second concatenation technique. This technique allowed for activity points to sometimes jump by intervals of as much as 10%. This did not allow the engine to respond to the dynamometers signals in time to keep up with the desired points of the cycle. A technique was applied to reduce the rate of change between certain points of both speed and load. Several smoothing and filtering techniques were considered for this application including moving average, Savitsky-Golay (SG), and local regression methods (loess and lowess). The techniques were applied to the cycle using MATLAB® defined functions [64]. Smoothing techniques were applied to normalized cycles as they were originally defined by the Smooth_and_Stat.m and Trucking_Smooth_and_Stat.m scripts shown in Appendix F. These scripts smoothed the speed and load array with the defined function and parameters and calculated the new statistics of the cycle. An analysis was used to determine how the different smoothing techniques affected the cycle. A comparison of the techniques was performed using the OTR cycle to determine how they affected the PV of the cycle. The goal was to apply a technique which smoothed the cycle enough to change the points with minimal effect on the PV. A summary of the smoothing techniques implemented and their respective effects on the PVs of the cycles are shown in Table 31.
Table 31: Smoothing Techniques and Their Effects on PVs

<table>
<thead>
<tr>
<th>Smoothing Parameters</th>
<th>OTR Truck Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method</td>
</tr>
<tr>
<td>Original</td>
<td>(No Smoothing)</td>
</tr>
<tr>
<td>moving</td>
<td>3</td>
</tr>
<tr>
<td>moving</td>
<td>5</td>
</tr>
<tr>
<td>moving</td>
<td>7</td>
</tr>
<tr>
<td>moving</td>
<td>9</td>
</tr>
<tr>
<td>lowess</td>
<td>3</td>
</tr>
<tr>
<td>lowess</td>
<td>5</td>
</tr>
<tr>
<td>lowess</td>
<td>7</td>
</tr>
<tr>
<td>lowess</td>
<td>9</td>
</tr>
<tr>
<td>loess</td>
<td>3</td>
</tr>
<tr>
<td>loess</td>
<td>5</td>
</tr>
<tr>
<td>loess</td>
<td>7</td>
</tr>
<tr>
<td>loess</td>
<td>9</td>
</tr>
<tr>
<td>sgolay</td>
<td>5</td>
</tr>
<tr>
<td>sgolay</td>
<td>7</td>
</tr>
<tr>
<td>sgolay</td>
<td>9</td>
</tr>
<tr>
<td>sgolay</td>
<td>11</td>
</tr>
<tr>
<td>sgolay</td>
<td>7</td>
</tr>
<tr>
<td>sgolay</td>
<td>9</td>
</tr>
<tr>
<td>sgolay</td>
<td>11</td>
</tr>
<tr>
<td>sgolay</td>
<td>13</td>
</tr>
<tr>
<td>sgolay</td>
<td>9</td>
</tr>
<tr>
<td>sgolay</td>
<td>11</td>
</tr>
<tr>
<td>sgolay</td>
<td>13</td>
</tr>
<tr>
<td>sgolay</td>
<td>15</td>
</tr>
<tr>
<td>sgolay</td>
<td>11</td>
</tr>
<tr>
<td>sgolay</td>
<td>13</td>
</tr>
<tr>
<td>sgolay</td>
<td>15</td>
</tr>
<tr>
<td>sgolay</td>
<td>17</td>
</tr>
</tbody>
</table>

The best smoothing techniques based on the PV are highlighted in Table 31. The smoothing techniques were ranked based on their respective effect on the PV of the cycle. The goal was to use the technique that least affected the PV, while at the same time, changing the cycle enough to meet regression criteria. The top performing smoothing techniques appeared to be the loess and lowess filters with a 3-second span. These smoothing techniques are “locally weighted scatter plot smoothings” in which each new smoothed value was determined from a first degree (lowess) or a second-degree (loess) polynomial fit based on the
nearby points within the defined span. The span defined the percentage of data points used in the local weighting. Upon further examination of these two techniques, however, it was determined that they had no effect on any individual points in the cycle which was the reason they had no effect on the PV. For this reason, they were removed from consideration. The loess 5 smoothing technique, ranked 2nd, was determined to have an effect on only 19 of the cycles 1800 points with an average change in speed and load of 0% and 18%, respectively. This technique was also disregarded due to the fact that it had no impact on speed. Due to the fact that the previous techniques had little to no effect on the cycle, the 3rd ranking technique was selected to govern smoothing. Several of the SG filtering techniques produced the 3rd ranked PV of 0.0324. The SG filtering technique used a span of data points and fitted them to a polynomial of degree “n” defined by the user. Two things had to be noted when defining the degree and span of the SG filter. The first was that the span (number of points over which the polynomial was fit) must be odd. The second was that the polynomial degree to which the span was fit had to less than the span [64]. In order to rank the SG techniques with the same PV, the smoothing was applied to drilling and hydraulic fracturing cycles to compare the effects on the respective PVs of these cycles. The MATLAB® scripts Smooth_and_Stat.m, Drilling_Smooth_and_Stat.m, and Fracking_Smooth_and_Stat.m were used for this purpose and are shown in Appendix F. The analysis of all cycles was used to determine the final values for the degree and span of the SG filter. Table 32 shows the results of the SG analysis when performed across all three types of cycles.

Table 32: SG Effect on PV for All Types of Cycles

<table>
<thead>
<tr>
<th>Smoothing Parameters</th>
<th>OTR Trucks</th>
<th>Drilling</th>
<th>Hydraulic Fracturing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PV Rank</td>
<td>PV Rank</td>
<td>PV Rank</td>
<td>PV Rank</td>
</tr>
<tr>
<td>Method</td>
<td>Span</td>
<td>Degree</td>
<td>PV</td>
<td>Rank</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>13</td>
<td>7</td>
<td>0.0324</td>
<td>1</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>17</td>
<td>9</td>
<td>0.0324</td>
<td>1</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>11</td>
<td>9</td>
<td>0.0324</td>
<td>1</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>9</td>
<td>7</td>
<td>0.0324</td>
<td>1</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>7</td>
<td>5</td>
<td>0.0324</td>
<td>1</td>
</tr>
<tr>
<td>sgo1ay</td>
<td>9</td>
<td>5</td>
<td>0.0324</td>
<td>1</td>
</tr>
</tbody>
</table>

The overall PV was used to determine the best technique. Based on the analysis, SG13-7 was the best technique. SG13-7 was chosen as it had the lowest sum of the PVs across all three cycles. When applied to the original OTR truck cycle the SG13-7 smoothing technique had an effect on 1799 of 1800 cycle points. The smoothing technique changed the affected speed points by an average of 4% and the load points by an average of 15%. The average difference in the derivatives of speed and load were calculated after smoothing. The average difference between the derivatives of the data and the smoothed cycle was
20%. A 20 second sample of the original and smoothed points of speed is shown in Figure 33 and the same sample of load is shown in Figure 34. The differences in these points is shown in Appendix F. The final normalized smooth cycle is shown in Figure 35. The complete normalized cycle is shown in Appendix G. The statistics of the final smooth OTR Truck cycle compared to those of data are shown in Table 33.

![Figure 33: Sample of OTR Cycle Speed Smoothing](image1)

![Figure 34: Sample of OTR Cycle Load Smoothing](image2)
The smoothed OTR truck cycle was denormalized for the Cummins ISLG using the Denormalization.m script. A “.cyc” file was created from the text file output and the first point of the “.cyc” file was converted to 700 rpm and 0.0000 lb-ft to prevent the dynamometer from entering “idle” mode. The denormalized cycle is shown in Figure 36 and the “.cyc” file is shown in Appendix E.
The smooth OTR truck cycle was tested for regression with the Cummins ISLG on the 800 HP DC dynamometer. The results of the regression test are shown in Table 34.

Table 34: Smooth OTR Truck Cycle Regression Results

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-70.1</td>
<td>0.666</td>
<td>70.1</td>
</tr>
<tr>
<td>Slope</td>
<td>0.97</td>
<td>0.999</td>
<td>1.03</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.97</td>
<td>0.999</td>
<td>1.00</td>
</tr>
<tr>
<td>SEE</td>
<td>0.0</td>
<td>10.083</td>
<td>110.35</td>
</tr>
<tr>
<td>Torque</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-18.0</td>
<td>0.286</td>
<td>18.0</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>0.973</td>
<td>1.03</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.85</td>
<td>0.919</td>
<td>1.00</td>
</tr>
<tr>
<td>SEE</td>
<td>0.0</td>
<td>54.134</td>
<td>90.0</td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intercept</td>
<td>-5.7</td>
<td>-0.052</td>
<td>5.7</td>
</tr>
<tr>
<td>Slope</td>
<td>0.83</td>
<td>0.983</td>
<td>1.03</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.91</td>
<td>0.934</td>
<td>1.00</td>
</tr>
<tr>
<td>SEE</td>
<td>0.0</td>
<td>15.462</td>
<td>22.1</td>
</tr>
<tr>
<td>Work</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Bhp-hr</td>
<td>20.819</td>
<td>22.186</td>
<td>23.011</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>21.915</td>
<td></td>
</tr>
</tbody>
</table>
The OTR truck cycle smoothed with SG13-7 filtering passed regression on all criteria. This allowed it to be defined as the final OTR truck cycle. With this definition, a final cycle was verified for all three prime-movers.
11. Conclusion
The unconventional well completion industry has been growing in the US and companies are attempting to utilize produced natural gas to meet power demands for cost savings. Several companies are doing this by displacing diesel fuel normally used for engines within the industry. The primary consumers, or prime-movers, of diesel fuel in the unconventional well completion industry are OTR trucks and high horsepower drilling and hydraulic fracturing engines. Displacing diesel fuel with natural gas could have repercussions from a GHG perspective as natural gas is primarily composed of methane, a potent GHG. This has led government agencies such as the DOE to seek information on these issues. As part of a funding opportunity from the US DOE, WVU was tasked with the creation of engine activity cycles representative of the engine activity seen in the industry.

In order to complete this task, in-field engine data were collected from the three prime-movers. Data collected included engine speed and load from OTR trucks traveling to and from well sites hauling water, sand, and gravel, hydraulic fracturing engines used to power pumps for several jobs on two separate pads, and drilling engines used to drill two wells on a single pad. The engine data were filtered and binned to create a matrix of engine activity points. The OTR data were binned into a matrix of speed and load increments of 5% for each, resulting in a 20x20 distribution. Drilling and hydraulic fracturing engines operated at rated speed and, therefore, the data collected from these engines was only sorted by load resulting in 20x1 matrices. The goal of the cycle creation process was to replicate the data as closely as possible within a reasonable period of time. A natural gas engine could be subjected to representative cycles on an engine dynamometer which would allow researchers to determine the natural gas emissions of such operation during future research.

Cycle creation was achieved by applying a MCMC technique. The goal was to replicate data by creating a large number of potential cycles. A Markov chain utilized second-by-second concatenation and a transition probability matrix to determine the next point in the cycle based on the current point. A cycle was started by randomly selecting a point in a low-load or idle bin, depending on the type of cycle creation. Once the initial bin was defined, a short period of idle or low-load was forced and then the Markov chain was allowed to operate freely. This meant that the bin of the next point in the cycle was determined from the transitional probability of the current bin. With the next bin determined by the transitional probabilities, a point from that bin was randomly selected and added to the cycle. The process continued until the desired time of the cycle, 1800s, was reached. The cycle was forced to return to an idle or low-load state shortly before the end of the cycle. The Markov chain technique had previously been used by several other studies in an attempt to represent engine activity in a cycle. The primary difference
between this work and others was the use of second-by-second concatenation, rather than micro-trips or kinematic sequences.

The pseudo-random nature of the cycle creation technique was where the Monte Carlo analysis came into play. This aspect allowed for a nearly infinite number of cycles to be created. A PV was defined to calculate the representativeness of each cycle. A population of 10,000 cycles was initially created for each prime-mover. The elite, lowest, PV of the populations of OTR trucks, drilling, and hydraulic fracturing cycles were 0.0327, 0.0042, and 0.0072, respectively. The fact that a number of cycles could be created with the same method made it necessary to develop a more direct method to determine the best overall cycle. A GA was used to optimize the cycle selection process. A GA is an optimization technique that can be applied to a variety of problems and is based on the Darwin’s theory of evolution, “survival of the fittest.” The GA takes an initial population of created cycles and applies genetic operators (mutation and crossover) to the population to create new, better individuals, which in this case were cycles. A representative cycle was defined with a low PV which compared each cycle to the in-field activity set. The comparison was based on statistics such as average speed, load, high-load, low-load, non-idle speed, and non-idle load as well as the time spent in each of these types of operation and the distribution of the data. The calculation of the PV was based on absolute differences between the cycles and the data. It was defined in such a way that the smaller values meant more representative cycles. The GA employed elitist and roulette-wheel selection to determine the members of the population allowed to advance to the next generation. The final GA used genetic operation rates of 40% for crossover and 20% for mutation. It utilized an initial population of 100 cycles and was executed for 50 generations. The GA was applied to each of the three prime-mover cycles, OTR trucks, drilling and hydraulic fracturing.

The GA was executed for all three types of cycles. After completion of the GA the elite PV of the OTR truck cycle, drilling cycle, and hydraulic fracturing improved by 18%, 62%, and 65%, respectively. Reducing the performance values from 0.0388 to 0.0320 for OTR trucks, 0.0079 to 0.0030 for drilling, and from 0.0155 to 0.0054 for hydraulic fracturing. The final elite cycles were selected to be performed on an engine and dynamometer to confirm that they were acceptable.

Cycles were performed with a Cummins ISLG on an 800 HP GE DC dynamometer in the WVU CAFE EERC test cell. The purpose of verifying the cycles was to ensure that they could meet the regression criteria for transient cycles defined by the CFR Title 40 Chapter I Subchapter U Part 1065 Subpart F Section 514. The cycles were denormalized from percentages of speed and load to engineering units in order to operate them with the engine and dynamometer. Each cycle was denormalized using the engine speed ratings and map of the ISLG. The ISLG was then subjected to the final cycles defined by the GA. The drilling and hydraulic fracturing cycles passed the regression criteria needed as they were initial
defined. The OTR cycle, however, failed regression based on the $R^2$ of torque and power criteria. Failure of regression was determined to be due to the high variability of speed and load simultaneously. The problem was corrected by applying a smoothing technique to the normalized cycle speed and load arrays. A sensitivity analysis was performed to determine the effect of smoothing techniques on the PV of the cycles. A SG filter with a span of 13 and polynomial degree of seven was chosen as the smoothing technique implemented. The smoothing was applied to the speed and load of the OTR cycle resulting in a 1.25% increase in the PV from 0.0320 to 0.0324. The new smoothed OTR truck cycle was again tested for regression on the Cummins ISLG and 800 HP dynamometer and passed all necessary criteria. This cycle was deemed to be the final cycle for the OTR trucks.

The final cycles for OTR trucks, drilling and hydraulic fracturing were fully defined. These cycles were deemed to be the most representative of actual engine operation in the field. Future research could attempt to implement different techniques to ensure that derivatives of cycles are as close to real world operation as possible. This could be done with a micro-trip technique, however, this is difficult to implement for drilling and hydraulic fracturing cycles due to their lack of discontinuities. Cycle development will continue to be necessary for research as emissions standards become more stringent and different industries come under the scrutiny of regulatory institutions. For these reasons, it is important to look for new ways to apply and optimize the techniques of in-field data representation. One way to improve representation of activity for emissions would be to take catalyst temperature into account. Older offroad engines are typically Tier 2 with no aftertreatment, however, newer engines that can be run on dual-fuel have diesel oxidation catalysts (DOCs) and Tier 4 engines have both diesel particulate filters (DPFs) and selective catalytic reduction (SCR). In analyzing newer engines, an additional goal could be to replicate the catalyst temperature history in addition to the engine speed and load, as this may have a significant impact on emissions. Thermal history is also important to understand the effects of certain aftertreatment systems on emissions. This will continue to be a research question as more stringent emissions standards come into play. In order to better understand the effects of real world handling of aftertreatment, in-use data should be collected from aftertreatment systems along with engine exhaust emissions.
References


2015].


92
Appendices

Appendix A: MATLAB® Code for Data Import

The scripts shown in this section were used for data importation of the respective prime-movers.

Over-the-Road Trucks

Trucking_Data_Import.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trucking Data Import
tic
clear all
close all
clc

% Define an Empty Speed Load Array
sl=[];

% Change the Directory Based on one of 4 Engines
engine = 1;
while engine <= 4
    if engine == 1
directory = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\CSV Data\Mack MP8\';
rated_speed = 1950; %rpm
idle_speed = 650; %rpm
NumFile = 1;
name = 'mack';
elseif engine == 2
directory = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\CSV Data\Volvo D13\';
rated_speed = 2100; %rpm
idle_speed = 550; %rpm
NumFile = 16;
name = 'volvo';
elseif engine == 3
directory = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\CSV Data\Cat C-15\';
rated_speed = 2100; %rpm
idle_speed = 600; %rpm
NumFile = 43;
name = 'cat';
elseif engine == 4
directory = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\CSV Data\Cummins ISX 15\';
rated_speed = 2000; %rpm
idle_speed = 700; %rpm
NumFile = 191;
name = 'cummins';
end
% Import all Files in Current Directory
for n=1:Numfile

% Define the current file
currentfile = [name '(',num2str(n),')\.csv'];
file = [directory currentfile];

% Extract the values needed
[values,headings] = xlsread(file);
[~,columns_h]    = size(headings);
[~,columns_v]    = size(values);

for i=1:columns_h
    heading=headings{2,i};
    if strcmp(heading,'Engine Speed (rpm)')
        if columns_v == columns_h
            speed = values(:,i);
        else
            speed = values(:,i-1);
        end
    elseif strcmp(heading,'Engine Percent Load At Current Speed (%)')
        if columns_v == columns_h
            ploadacs = values(:,i);
        else
            ploadacs = values(:,i-1);
        end
    end
end

% Normalize the Speed and Load Values
engine_speed{n}= speed;
engine_load{n}=ploadacs/100;
normal_speed{n} = (engine_speed{n}-idle_speed)/(rated_speed-idle_speed);

% Eliminate Out of Range Values
x=length(engine_speed{n});
for i=x:-1:1
    if normal_speed{n}(i)<-0.02 || normal_speed{n}(i)>1.02 ||
        engine_load{n}(i) < 0 || engine_load{n}(i) > 1
        normal_speed{n}(i)=[ ];
        engine_load{n}(i)=[ ];
    elseif normal_speed{n}(i)<0.02
        normal_speed{n}(i)=0;
    elseif normal_speed{n}(i) > 1.0
        normal_speed{n}(i) = 1.0;
    end
end

% Concatenate Data
speed_load{n}=[normal_speed{n},engine_load{n}];
sl=cat(1,sl,speed_load{n});
end
toc
     engine = engine + 1;
end

% Eliminate the Empty Rows
[row,~]=find(isnan(sl));
sl(row,:)=[];

% Convert the Data to Structure Format
data.sl = sl;

% Calculate the Total Time of Data Imported
data.time.total.s=length(data.sl);
data.time.total.min=length(data.sl)/60;
data.time.total.hr=length(data.sl)/3600;
data.time.total.days=length(data.sl)/(3600*24);

clearvars -except data

Trucking_Transition_Matrix_Formation.m
% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Trucking Data Binning and Matrix Formation

%%
% Binning the Speed and Load Values
clear all
close all
clc

tic
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\Matlab Data from Import\Truck_Data_1_15_16.mat')

% Define the Values that will Govern the Bins
% Each Bin will be a group of 5% of speed and load
group_s=[0:0.05:0.95 1.01];
group_l=[0:0.05:0.95 1.01];

% Define the lengths of the sorting bin values
s=length(group_s);
t=length(group_l);
% Define the length of the data to be sorted
x=length(data.sl);

%%
% Pre-allocation of Matrix
% Predefine the size of the structure
%Bin will be a matrix of structures
data.bin=cell(s-1,t-1);
%Loop through the rows
for i = 1: s-1
  %Loop through the columns
  for j= 1: t-1
    %Create an empty cell array
    data.bin(i,j).data=[];
    %Create a transition matrix of zeros
    data.bin(i,j).tm=zeros(s-1,t-1);
  end
end

%%
%Predefine Distribution of all Data
data.distribution=zeros(s-1,t-1); %Distribution of entire Data set

%Place Data in Bins and create transition matrix for each Bin
for a=1:x-1
  % Scan the Speed Bins
  for i=1:s-1
    % Once Speed Bin is found, Find the Load Bin
    if data.sl(a,1)>=group_s(i) && data.sl(a,1)<group_s(i+1)
      %Scan the Load Bins
      for j=1:t-1
        %Once the Load Bin is found, Bin the Data
        if data.sl(a,2)>=group_l(j) && data.sl(a,2)<group_l(j+1)
          %Concatenate new Data with Data in Bin
          data.bin{i,j}.data=cat(1,data.bin{i,j}.data,data.sl(a,:));
          %Add to the Frequency of that Bin
          data.distribution(i,j)=data.distribution(i,j)+1;
          %With the Current Bin defined, Look at next Data Point
          %Scan the Speed Bins
          for m=1:s-1
            %Place the next Data Point in a Speed Bin
            if data.sl(a+1,1)>=group_s(m) &&
              data.sl(a+1,1)<group_s(m+1)
              %Scan the Load Bins
              for n=1:t-1
                %Place the next Data Point in a Load Bin
                if data.sl(a+1,2)>=group_l(n) &&
                  data.sl(a+1,2)<group_l(n+1)
                  %Add to the Transition Matrix of Bin
                  data.bin{i,j}.tm(m,n)=data.bin{i,j}.tm(m,n)+1;
                end
              end
            end
          end
        end
      end
    end
  end
end
%%
% Determine the Time in each Bin
for i = 1:s-1
    for j = 1:t-1
        % Time spent in each bin is determined by the length of data (1Hz)
        data.bin{i,j}.time=length(data.bin{i,j}.data);
        % The distribution percentage of the data is determined from the
        % number of points in that bin divided by total points
        % Presented as a percentage of total time
        data.distribution(i,j)=data.distribution(i,j)/length(data.sl(:,1));
    end
end

%% Calculate Average Statistics
% Average Speed
data.stats.avg.speed = mean(data.sl(:,1));

% Average Load
data.stats.avg.load = mean(data.sl(:,2));

%% Sort the Data by Idle or Non-Idle
data.time.idle = 0;
data.time.non_idle = 0;
non_idle_speed = 0;
non_idle_load = 0;

% Categorize all Data Points
for i = 1:x
    % Idle will be defined by 0 engine speed
    if data.sl(i,1) == 0
        data.time.idle = data.time.idle + 1;
    else
        data.time.non_idle = data.time.non_idle + 1;
        non_idle_speed = non_idle_speed + data.sl(i,1);
        non_idle_load = non_idle_load + data.sl(i,2);
    end
end

%% Calculate the Non-Idle Averages
% Average Non Idle Speed
data.stats.avg.non_idle_speed = non_idle_speed/data.time.non_idle;

% Average Non Idle Load
data.stats.avg.non_idle_load = non_idle_load/data.time.non_idle;

% Normalize the time values
data.time.idle = data.time.idle/x;
data.time.non_idle = data.time.non_idle/x;
toc
% Clear unneeded variables
clearvars -except data
% Robert Heltzel
% West Virginia University
% Original: September 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Drilling Data Import
tic
clear all
close all
clc

% Basic File Information
directory = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Data\CSV Data\';
Numfile = 235;
name = 'ECU';

% Define an Empty Load Array
data.load=[];

% Import all Files in the Folder
for n= 1:Numfile

    % Define the current file
currentfile = [name ' (' num2str(n) , ')' .csv ];
    file = [directory currentfile];

    % Extract the values needed
    [values,headings] = xlsread(file);
    [~,columns_h] = size(headings);
    [rows_v,columns_v] = size(values);

    for i=1:columns_h
        heading=headings{1,i};
        if strcmp(heading,'Percent load at current speed')
            if columns_v == columns_h
                if rows_v >= 3600
                    ploadacs = values(1:3600,i);
                else
                    ploadacs = values(:,i);
                end
            end
        end
    end

% Normalize the Load Values
normal_load{n}=ploadacs/100;
% Eliminate Out of Range Values
x = length(normal_load{n});
for i=x:-1:1
    if normal_load{n}(i) < 0 || normal_load{n}(i) > 1
        normal_load{n}(i)=[];
    end
end

% Concatenate Data
data.load=cat(1,data.load,normal_load{n});
s = sprintf('File Number %i Complete',n);
disp(s)
end
toc

% Eliminate the Empty Rows
[row,~]=find(isnan(data.load));
data.load(row,:)=[];

% Determine the Total Time of Data Imported
data.time.total.s=length(data.load);
data.time.total.min=length(data.load)/60;
data.time.total.hr=length(data.load)/3600;
data.time.total.days=length(data.load)/(3600*24);
clearvars -except data

Drilling_Transition_Matrix_Formation.m

% Robert Heltzel
% West Virginia University
% Original: September 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Drilling Data Binning and Transition Matrix Formation

%%% % Binning the Load Values
clear all
close all
clc
tic
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Data\Drilling_Raw_Import_11516.mat')

% Define the Values that will govern bins
% Each Bin will be a group of 5% load
group_1 = [0:0.05:0.95, 1.01];

% Define the length of the sorting bin values
t = length(group_1);
% Define the length of the data to be sorted
x = length(data.load);

% Pre-allocation of Matrix
% Predefine the size of the structure

% Bin will be a matrix of structures
data.bin=cell(1,t-1);
% Loop through the columns
for i = 1:t-1
    % Create an empty cell array
data.bin(i).data=[];
    % Create a transition matrix of zeros
data.bin(i).tm=zeros(1,t-1);
end

% Predefine Frequency of all Data
data.distribution=zeros(1,t-1); % Distribution of entire Data set

% Place Data in Bins and create transition matrix for each Bin
for a=1:x-1
    % Scan the Load Bins
    for j=1:t-1
        % Once the Load Bin is found, Bin the Data
        if data.load(a) >= group_l(j) && data.load(a) < group_l(j+1)
            % Concatenate new Data with Data in Bin
            data.bin{j}.data=cat(1,data.bin{j}.data,data.load(a));
            % Add to the Frequency of that Bin
            data.distribution(j)=data.distribution(j)+1;
            % With the Current Bin defined, Look at next Data Point
            % Scan the Load Bins
            for n=1:t-1
                % Place the next Data Point in a Load Bin
                if data.load(a+1) >= group_l(n) && data.load(a+1) < group_l(n+1)
                    % Add to the Transition Matrix of Bin
                    data.bin{j}.tm(n)=data.bin{j}.tm(n)+1;
                end
            end
        end
    end
end

% Determine the Time in each Bin
for j = 1:t-1
    % Time spent in each bin is determined by the length of data (1Hz)
    data.bin{j}.time=length(data.bin{j}.data);
end
% The distribution percentage of the data is determined from the
% number of points in that bin divided by total points
% Presented as a percentage of total time
data.distribution(j)=data.distribution(j)/length(data.load);
Calculate Average Statistics
% Average Load
data.stats.avg.load = mean(data.load);

% Sort the Data by High or Low Load
data.time.high = 0;
data.time.low = 0;
data.time.idle = 0;
high_load = 0;
low_load = 0;

% Categorize all Data Points
for i = 1:x
    % High Load will be defined as Greater than 40 %
    if data.load(i) >= 0.4
        data.time.high = data.time.high + 1;
        high_load = high_load + data.load(i);
    elseif data.load(i) == 0
        data.time.idle = data.time.idle + 1;
    else
        data.time.low = data.time.low + 1;
        low_load = low_load + data.load(i);
    end
end

% Calculate the High and Low Load Averages
% Average High Load
data.stats.avg.high_load = high_load/data.time.high;

% Average Low Load
data.stats.avg.low_load = low_load/data.time.low;

% Normalize the time values
data.time.high = data.time.high/x;
data.time.low = data.time.low/x;
data.time.idle = data.time.idle/x;
toc
% Clear unneeded variables
clearvars -except data

Hydraulic Fracturing

Fracking_Data_Import.m

% Robert Heltzel
% West Virginia University
% Original: October 2015
% Updated: January 2016

% Fracking Data Import
tic
clear all
close all
clc

% Files Imported from 3 Locations
% Location 1: 1 Hr Data Files from Central WV
directory1 = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Fracking\Data\CSV Data SWN\3600s\';
Numfile1 = 37;

% Location 2: Full Length Data Files from Central WV
directory2 = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Fracking\Data\CSV Data SWN\Full Length\';
Numfile2 = 4;

% Location 3: Data File from Westover WV
directory3 = 'C:\Users\Robbie\Documents\Thesis\Cycle Development\Fracking\Data\CSV Data MSEEL\';
Numfile3 = 10;

name = 'ECU';

% Define the Empty Load Array
data.load = [];

%% Import all Files in Location 1
for n= 1:Numfile1

    % Define the current file
    currentfile = [name ' (' ,num2str(n) ,') .csv'];
    file = [directory1 currentfile];

    % Extract the values needed
    [values,headings] = xlsread(file);
    [~,columns_h] = size(headings);
    [rows_v,columns_v] = size(values);

    for i=1:columns_h
        heading=headings{1,i};
        if strcmp(heading,'Percent load at current speed')
            if columns_v == columns_h
                if rows_v >= 3600
                    ploadacs = values(1:3600,i);
                else
                    ploadacs = values(:,i);
                end
            end
        end
    end

    % Normalize the Load Values
    normal_load{n}=ploadacs/100;
end
% Eliminate Out of Range Values
x = length(normal_load{n});
for i=x:-1:1
    if normal_load{n}(i) <0 || normal_load {n}(i) > 1
        normal_load{n}(i)=[];
    end
end

% Concatenate Data
data.load=cat(1,data.load,normal_load{n});
s = sprintf('File Number %i Complete',n);
disp(s)
end
toc

%% Import all Files in Location 2
for n= 1:Numfile2

    % Define the current file
currentfile = [name ' (' ,num2str(n),') .csv' ];
file = [directory2 currentfile ];

    % Extract the values needed
[values,headings] = xlsread(file);
[~,columns_h] = size(headings);
[rows_v,columns_v] = size(values);
for i=1:columns_h
    heading=headings{1,i};
    if strcmp(heading,'Percent load at current speed')
        if columns_v == columns_h
            if rows_v >= 3600
                ploadacs = values(:,i);
            end
        end
    end
end

    % Normalize the Load Values
normal_load{n}=ploadacs/100;

    % Eliminate Out of Range Values
x=length(normal_load{n});
for i=x:-1:1
    if normal_load{n}(i) <0 || normal_load {n}(i) > 1
        normal_load{n}(i)=[];
    end
end

    % Concatenate Data
data.load = cat(1,data.load,normal_load{n});
s = sprintf('File Directory 2 Number %i Complete',n);
disp(s)
end
toc

%% Import all Files in Location 3
for n=1:Numfile3

% Define the current file
currentfile = [name ' (' num2str(n) ',').xls'];
file = [directory3 currentfile];

% Extract the values needed
[values,~] = xlsread(file,'Time Series');
[~,columns_v] = size(values);

ploadacs = values(:,2);

% Normalize the Load Values
normal_load{n} = ploadacs/100;

% Eliminate Out of Range Values
x=length(normal_load{n});
for i=x:-1:1
    if normal_load{n}(i) <0 || normal_load {n}(i) > 1
        normal_load{n}(i)= [ ];
    end
end

% Concatenate Data
data.load = cat(1,data.load,normal_load{n});
s = sprintf('File Directory 3 Number %i Complete',n);
disp(s)
end
toc

%% Eliminate the Empty Rows
[row,~]=find(isnan(data.load));
data.load(row,:)=[];

%% Determine the Total Time of Data Imported
data.time.total.s=length(data.load);
data.time.total.min=length(data.load)/60;
data.time.total.hr=length(data.load)/3600;
data.time.total.days=length(data.load)/(3600*24);

clearvars -except data

**Fracking_Transition_Matrix_Formation.m**

% Robert Heltzel
% West Virginia University
% Original: October 2015
% Updated: January 2016
% Fracking Data Binning and Transition Matrix Formation

%%
% Binning the Load Values
clear all
close all
clc
tic
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Fracking\Data\Fracking_Raw_Import_11516.mat')

% Define the Values that will govern bins
% Each Bin will be a group of 5% load
group_l = [0:0.05:0.95, 1.01];

% Define the length of the sorting bin values
t = length(group_l);
% Define the length of the data to be sorted
x = length(data.load);

%%
% Pre-allocation of Matrix
% Predefine the size of the structure

% Bin will be a matrix of structures
data.bin=cell(1,t-1);
% Loop through the columns
for i = 1:t-1
    % Create an empty cell array
    data.bin{i}.data=[];
    % Create a transition matrix of zeros
    data.bin{i}.tm=zeros(1,t-1);
end

%%
% Predefine Frequency of all Data
% Distribution of entire Data set

% Place Data in Bins and create transition matrix for each Bin
for a=1:x-1
    % Scan the Load Bins
    for j=1:t-1
        % Once the Load Bin is foudn, Bin the Data
        if data.load(a) >= group_l(j) && data.load(a) < group_l(j+1)
            % Concatenate new Data with Data in Bin
            data.bin{j}.data=cat(1,data.bin{j}.data,data.load(a));
            % Add to the Frequency of that Bin
            data.distribution(j)=data.distribution(j)+1;
        end
    end
end

% With the Current Bin defined, Look at next Data Point
% Scan the Load Bins
for n=1:t-1
    % Place the next Data Point in a Load Bin
end
if data.load(a+1) >= group_l(n) && data.load(a+1) <
group_l(n+1)

% Add to the Transition Matrix of Bin
data.bin{j}.tm(n)=data.bin{j}.tm(n)+1;
end
end
end
end

%% Determine the Time in each Bin
for j = 1:t-1
  % Time spent in each bin is determined by the length of data (1Hz)
  data.bin(j).time=length(data.bin{j}.data);
  % The distribution percentage of the data is determined from the
  % number of points in that bin divided by total points
  % Presented as a percentage of total time
  data.distribution(j)=data.distribution(j)/length(data.load);
end

%% Calculate Average Statistics
% Average Load
data.stats.avg.load=mean(data.load);

%% Sort the Data by High or Low Load
data.time.high = 0;
data.time.low = 0;
data.time.idle = 0;
high_load = 0;
low_load = 0;

% Categorize all Data Points
for i = 1:x
  % High Load will be defined as Greater than 40%
  if data.load(i) >= 0.4
    data.time.high = data.time.high + 1;
    high_load = high_load + data.load(i);
  elseif data.load(i) == 0
    data.time.idle = data.time.idle + 1;
  else
    data.time.low = data.time.low + 1;
    low_load = low_load + data.load(i);
  end
end

%% Calculate the High and Low Load Averages
% Average High Load
data.stats.avg.high_load = high_load/data.time.high;

% Average Low Load
data.stats.avg.low_load = low_load/data.time.low;

%% Normalize the time values
data.time.high = data.time.high/x;
data.time.low = data.time.low/x;
data.time.idle = data.time.idle/x;
toc
%Clear unneeded variables
clearvars -except data

Appendix B: MATLAB® Code for Cycle Development

Over-the-Road Trucks
Trucking_Markov_Chain_Cycle_Construction.m

% Robert Heltzel
% West Virginia University
% Original: December 2014
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Markov Chain Cycle Formation for Trucking
% clear all
close all
clc
tic

% Load the Data
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\Trucking_Data_Binned_11516.mat')

% Define the Number of Cycles to Create
pop_size = 100;

% Smoothing: On or Off
Smoothing = 'Off';

% Predefine the size of the cell arrays
cycle=cell(1);

% To Create Multiple Cycles with one run of the code
a=1;
while a <= pop_size

% Predefine Distribution of all Data
cycle(a).distribution=zeros(20,20);

% Randomly Select an Idle Engine Speed Event from idle and low load
initial_point=datasample(data.bin{1,1}.data,1);
% This will be the starting point of the cycle
cycle(a).cycle=initial_point;
% Define the Index of the first point
i=1;
j=1;

% First 30 seconds should be Idle
for c = 2:30
    % Determine the weights of the columns (Should stay in same row)
    weights_c = data.bin{i,j}.tm(i,:)/sum(data.bin{i,j}.tm(i,:));
    % j determines the column of the next bin
    j = datasample(1:20,1,'weights',weights_c);
    % x is a random point selected from the bin (i,j)
    x = datasample(data.bin{i,j}.data,1);
    % The point that is selected is added to the cycle
    cycle{a}.cycle=cat(1,cycle{a}.cycle,x);
    % The point is added to the distribution
    cycle{a}.distribution(i,j) = cycle{a}.distribution(i,j) + 1;
end

% Go through points in the cycle until 30 seconds remain
for c=31:1770
    % Determine the weights of each of the columns and rows
    % Based on the transition matrix of the current bin
    weights_c=data.bin{i,j}.tm(i,:)/sum(data.bin{i,j}.tm(i,:));
    weights_r=data.bin{i,j}.tm(:,j)/sum(data.bin{i,j}.tm(:,j));
    % Randomly select the next bin based on the current transition
    % matrix (weights are controlled by tm)
    % i determines the row of the next bin
    i = datasample(1:20,1,'weights',weights_r);
    % j determines the column of the next bin
    j = datasample(1:20,1,'weights',weights_c);
    % x is a random point selected from the bin (i,j)
    x = datasample(data.bin{i,j}.data,1);
    % The point that is selected is added to the cycle
    cycle{a}.cycle=cat(1,cycle{a}.cycle,x);
    % The point is added to the distribution
    cycle{a}.distribution(i,j) = cycle{a}.distribution(i,j) + 1;
end

last = c+1; % The concatenation should start at next point

% If the last speed bin is above 10, add intermediate second
if i > 10
    i = i - 5;
    % Determine the weights of the columns (Should stay in same row)
    weights_c = data.bin{i,j}.tm(i,:)/sum(data.bin{i,j}.tm(i,:));
    % j determines the column of the next bin
    j = datasample(1:20,1,'weights',weights_c);
    % x is a random point selected from the bin (i,j)
x = datasample(data.bin{i,j}.data,1);

% The point that is selected is added to the cycle
cycle(a).cycle=cat(1,cycle(a).cycle,x);

% The point is added to the distribution
for c = last:1800
weights_c = data.bin{i,j}.tm(i,:)/sum(data.bin{i,j}.tm(i,:));
% j determines the column of the next bin
j = datasample(1:20,1,'weights',weights_c);
% x is a random point selected from the bin (i,j)
x = datasample(data.bin{i,j}.data,1);

% The point that is selected is added to the cycle
cycle(a).cycle=cat(1,cycle(a).cycle,x);

% The point is added to the distribution
cycle(a).distribution(i,j) = cycle(a).distribution(i,j) + 1;
end

x = length(cycle{a}.cycle(:,1));

%% Smoothing if ON
if strcmp(Smoothing,'On')
    % Smooth the Cycle
end

109
cycle(a).cycle(:,1) = sgolayfilt(cycle(a).cycle(:,1),3,11);
cycle(a).cycle(:,2) = sgolayfilt(cycle(a).cycle(:,2),3,11);

% Define the Values that will Govern the Bins
% Each Bin will be a group of 5% of speed and load
group_s=[0:0.05:0.95 1.01];
group_l=[0:0.05:0.95 1.01];

% Define the lengths of the sorting bin values
s = length(group_s);
t = length(group_l);

% Predefine Distribution of all Data
cycle(a).distribution=zeros(20,20);

% Recalculate the Distribution
for k=1:x-1
    % Define a for loop to go through speed groups
    for i=1:s-1
        % Once the speed group is found move on to load grouping
        if cycle(a).cycle(k,1)>=group_s(i) &&
            cycle(a).cycle(k,1)<group_s(i+1)
            % Scan through the load groupings
            for j=1:t-1
                % Once the load grouping is found bin the data
                if cycle(a).cycle(k,2)>=group_l(j) &&
                    cycle(a).cycle(k,2)<group_l(j+1)
                    % Add a point to the cycle distribution
                    cycle(a).distribution(i,j)=cycle(a).distribution(i,j)+1;
                end
            end
        end
    end
end

%%
% Eliminate Spikes between 0 points
for i = 1:x
    if i>1 && i<1800
        if cycle(a).cycle(i,1) > 0 &&...
            cycle(a).cycle(i+1,1) == 0 &&...
            cycle(a).cycle(i-1,1)==0
            cycle(a).cycle(i,1) = 0;
        end
    end
end

% Eliminate Blips
for i = 1:1800
    if cycle(a).cycle(i,1) < 0.02
        cycle(a).cycle(i,1) = 0;
    end
end
if cycle{a}.cycle(i,2) < 0.02
    cycle{a}.cycle(i,2) = 0;
end

%% Normalize the Matrix Distribution
for i = 1:20
    for j = 1:20
        % The distribution percentage of the data is determined from the
        % number of points in that bin divided by total points
        % Presented as a percentage of total time
        cycle{a}.distribution(i,j) = cycle{a}.distribution(i,j)/length(cycle{a}.cycle(:,1));
        distribution_diff(i,j) = 0.05 * abs(cycle{a}.distribution(i,j) -
                             data.distribution(i,j));
    end
end

%% Calculated the Statistics of the Cycle
% Average Cycle Speed
cycle{a}.stats.avg.speed = mean(cycle{a}.cycle(:,1));

% Average Cycle Load
cycle{a}.stats.avg.load = mean(cycle{a}.cycle(:,2));

%% Sort the Data by Idle or Non-Idle
cycle{a}.time.idle = 0;
cycle{a}.time.non_idle = 0;
non_idle_speed = 0;
non_idle_load = 0;

% Go through all of the data and sort by 0 speed (Idle)
for i = 1:x
    if cycle{a}.cycle(i,1) == 0
        cycle{a}.time.idle = cycle{a}.time.idle + 1;
    else
        cycle{a}.time.non_idle = cycle{a}.time.non_idle + 1;
        non_idle_speed = non_idle_speed + cycle{a}.cycle(i,1);
        non_idle_load = non_idle_load + cycle{a}.cycle(i,2);
    end
end

%% Calculate the Non-Idle Averages
% Average Non Idle Speed
non_idle_speed_avg = non_idle_speed/cycle{a}.time.non_idle;

% Average Non Idle Load
non_idle_load_avg = non_idle_load/cycle{a}.time.non_idle;

%% Normalize the time values
cycle{a}.time.idle = cycle{a}.time.idle/x;

cycle{a}.time.non_idle = cycle{a}.time.non_idle/x;

%% Determine the Statistic Value (SV)
cycle{a}.SV = ...  
   0.2*abs(cycle{a}.stats.avg.speed - data.stats.avg.speed)+ ...
   0.2*abs(cycle{a}.stats.avg.load - data.stats.avg.load)+ ...
   0.2*abs(cycle{a}.time.idle - data.time.idle)+ ...
   0.2*abs(cycle{a}.stats.avg.non_idle_speed -
data.stats.avg.non_idle_speed) + ...
   0.2*abs(cycle{a}.stats.avg.non_idle_speed -
data.stats.avg.non_idle_load);

%% Determine the Distribution Value (DV)
cycle{a}.DV = sum(sum(distribution_diff));

%% Calculate the Performance Value (PV)

cycle{a}.PV = ...  
   0.5*cycle{a}.SV + ...
   0.5*cycle{a}.DV;

%% % Comment (starting at figure) if plots are not needed for each cycle
created
% Considerably slows down code

% Plot the normalized speed and normalized torque
figure;
subplot(2,1,1)
plot(cycle{a}.cycle(:,1))
ylabel('Normalized Speed')
xlabel('Time(s)')
ylim([0,1])
subplot(2,1,2)
plot(cycle{a}.cycle(:,2))
ylabel('Normalized Load')
xlabel('Time(s)')
ylim([0,1])

s = sprintf('Cycle Number %i Complete',a);
disp(s)
a=a+1;
end

toc
run_time = toc;
clearvars -except cycle data pop_size generations crossover_rate
mutation_rate Smoothing

Drilling
Drilling_Markov_Chain_Cycle_Construction.m

% Robert Heltzel
% Markov Chain Cycle Formation for Drilling
% clear all
close all
clc
tic

% Load the Data
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Data\Drilling_Data_Binned_12916.mat')

% Define the Number of Cycles to Create
pop_size = 100;

% Smoothing: On or Off
Smoothing = 'Off';

%%
% Predefine the size of the cell arrays
cycle = cell(1);

% To Create Multiple Cycles with one run of the code
a=1;
while a <= pop_size

    % Predefine Distribution of Cycle
cycle{a}.distribution=zeros(1,20);

    % Initial Point should be in Low Load Bin
    initial_point=datasample(data.bin{1}.data,1);
    % This will be the starting point of the cycle
cycle{a}.cycle=initial_point;

    % Define the Index of the first point
    j=1;
    % First 10 seconds should be Low Load
    for c = 2:10
        % Point selected should be in the first bin
        x = datasample(data.bin{1}.data,1);
        % The point that is selected is added to the cycle
        cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

        % The point is added to the distribution
        cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
    end

    % Go through points in the cycle until 10 seconds remain
    for c = 11:1790


113
% Determine the weights of each of the columns
% Based on the transition matrix of the current bin
weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));

% Select Next Bin based on weighted Sample
% j determines the column of the next bin
j = datasample(1:20,1,'weights',weights);

% x is a random point selected from the bin (j)
x = datasample(data.bin{j}.data,1);

% The point that is selected is added to the cycle
cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

% The point is added to the distribution
cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;

end

last = c+1; % The concatenation should start at next point

% If the last speed bin is above 10, add another point
if j > 10
    j = j-5;
    % Determine the weights of the columns (Should stay in same row)
    weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));
    % j determines the column of the next bin
    j = datasample(1:20,1,'weights',weights);
    % x is a random point selected from the bin (j)
    x = datasample(data.bin{j}.data,1);
    % The point that is selected is added to the cycle
    cycle{a}.cycle=cat(1,cycle{a}.cycle,x);
    % The point is added to the distribution
    cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;

    last = last+1;
end

% If the last speed bin is still above 10, add another point
if j > 10
    j = j-5;
    % Determine the weights of the columns (Should stay in same row)
    weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));
    % j determines the column of the next bin
    j = datasample(1:20,1,'weights',weights);
    % x is a random point selected from the bin (j)
    x = datasample(data.bin{j}.data,1);
    % The point that is selected is added to the cycle
    cycle{a}.cycle=cat(1,cycle{a}.cycle,x);
    % The point is added to the distribution
cycle(a).distribution(j) = cycle(a).distribution(j) + 1;

last = last+1;
end

% Cycle should return to Lowest Load for the last 10 seconds
j = 1; % Return to the lowest load bin

% Last 10 seconds
for c = last:1800

% x is a random point selected from the idle bin (1)
x = datasample(data.bin{j}.data,1);

% The point that is selected is added to the cycle
cycle(a).cycle=cat(1,cycle(a).cycle,x);

% The point is added to the distribution
cycle(a).distribution(j) = cycle(a).distribution(j) + 1;
end

x = length(cycle(a).cycle);

%% Smoothing if ON
if strcmp(Smoothing,'On')

% Smooth the Cycle

% Recalculate the Distribution Post Smoothing
cycle(a).distribution = zeros(1,20);
group_l = [0:0.05:0.95, 1.01];
t = length(group_l);
for i = 1:x
    for j = 1:t-1
        if cycle(a).cycle(i) >= group_l(j) && cycle(a).cycle(i) < group_l(j+1)
            cycle(a).distribution(j) = cycle(a).distribution(j) + 1;
        end
    end
end
end
end

%% % Eliminate Spikes between 0 points
for i = 1:x
    if i>1 && i<1800
        if cycle(a).cycle(i,1) > 0 &&...
            cycle(a).cycle(i+1,1) == 0 &&...
            cycle(a).cycle(i-1,1)==0
                cycle(a).cycle(i,1) = 0;
        end
    end
end
% Eliminate Blips
for i = 1:1800
    if cycle{a}.cycle(i) < 0.02
        cycle{a}.cycle(i) = 0;
    end
end

%% Normalize the Matrix Distribution
for j = 1:20
    % The distribution percentage of the data is determined from the
    % number of points in that bin divided by total points
    % Presented as a percentage of total time
    cycle{a}.distribution(j) = cycle{a}.distribution(j)/length(cycle{a}.cycle);
    distribution_diff(j) = 0.05 * abs(cycle{a}.distribution(j) - data.distribution(j));
end

%% Calculated the Statistics of the Cycle
% Average Cycle Load
cycle{a}.stats.avg.load = mean(cycle{a}.cycle);

%% Sort the Data by High or Low Load
cycle{a}.time.high = 0;
cycle{a}.time.low = 0;
cycle{a}.time.idle = 0;
high_load = 0;
low_load = 0;
x = length(cycle{a}.cycle);

% Go through all of the data and sort by Load
for i = 1:x
    % High Load will be defined as Greater than 40%
    if cycle{a}.cycle(i) >= 0.4
        cycle{a}.time.high = cycle{a}.time.high + 1;
        high_load = high_load + cycle{a}.cycle(i);
    elseif cycle{a}.cycle(i) == 0
        cycle{a}.time.idle = cycle{a}.time.idle + 1;
    else
        cycle{a}.time.low = cycle{a}.time.low + 1;
        low_load = low_load + cycle{a}.cycle(i);
    end
end

%% Calculate the High and Low Load Averages
% Average High Load
cycle{a}.stats.avg.high_load = high_load/cycle{a}.time.high;

% Average Low Load
cycle{a}.stats.avg.low_load = low_load/cycle{a}.time.low;
%% Normalize the time values
cycle{a}.time.high = cycle{a}.time.high/x;
cycle{a}.time.low = cycle{a}.time.low/x;
cycle{a}.time.idle = cycle{a}.time.idle/x;

%% Determine the Statistic Value (SV)
cycle{a}.SV =
0.2*abs(cycle{a}.stats.avg.load - data.stats.avg.load)+
0.2*abs(cycle{a}.stats.avg.high_load - data.stats.avg.high_load)+
0.2*abs(cycle{a}.stats.avg.low_load - data.stats.avg.low_load)+
0.2*abs(cycle{a}.time.high - data.time.high)+
0.2*abs(cycle{a}.time.idle - data.time.idle);

%% Determine the Distribution Value (DV)
cycle{a}.DV = sum(distribution_diff);

%% Calculate the Performance Value (PV)
cycle{a}.PV = ...
0.5*cycle{a}.SV + ...
0.5*cycle{a}.DV;

%% Comment (starting at figure) if plots are not needed for each cycle created
% Considerably slows down code

%Plot the normalized speed and normalized torque
figure;
plot(cycle{a}.cycle)
ylabel('Normalized Load')
xlabel('Time(s)')
ylim([0,1])

s = sprintf('Cycle Number %i Complete',a);
disp(s)

    a = a + 1;

end

toc
run_time = toc;
clearvars -except cycle data pop_size generations crossover_rate mutation_rate Smoothing

Hydraulic Fracturing
Fracking_Markov_Chain_Cycle_Construction.m

% Robert Heltzel
% West Virginia University
% Markov Chain Cycle Formation for Fracking
% clear all
close all
clc
tic

% Load the Data
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Fracking\Data\Fracking_Data_Binned_12916.mat')

% Define the Number of Cycles to Create
pop_size = 100;

% Smoothing: On or Off
Smoothing = 'Off';

%%
% Predefine the size of the cell arrays
cycle=cell(1);

% To Create Multiple Cycles with one run of the code
a=1;
while a <= pop_size
  % Predefine Distribution of Cycle
cycle{a}.distribution=zeros(1,20);

  % Initial Point should be in Low Load Bin
  initial_point=datasample(data.bin{1}.data,1);
  % This will be the starting point of the cycle
cycle{a}.cycle=initial_point;

  % Define the Index of the first point
  j=1;
  % First 10 seconds should be Low Load
  for c = 2:10
    % Point selected should be in the first bin
    x = datasample(data.bin{1}.data,1);
    % The point that is selected is added to the cycle
    cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

    % The point is added to the distribution
    cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
  end

  % Go through points in the cycle until 10 seconds remain
  for c = 11:1790
    % Determine the weights of each of the columns

% Based on the transition matrix of the current bin
weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));

% Select Next Bin based on weighted Sample
% j determines the column of the next bin
j = datasample(1:20,1,'weights',weights);

% x is a random point selected from the bin (j)
x = datasample(data.bin{j}.data,1);

% The point that is selected is added to the cycle
cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

% The point is added to the distribution
cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
end

last = c+1; % The concatenation should start at next point

% If the last speed bin is above 10, add another point
if j > 10
  j = j-5;
  % Determine the weights of the columns (Should stay in same row)
  weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));
  % j determines the column of the next bin
  j = datasample(1:20,1,'weights',weights);
  % x is a random point selected from the bin (j)
  x = datasample(data.bin{j}.data,1);

  % The point that is selected is added to the cycle
  cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

  % The point is added to the distribution
cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
end

% If the last speed bin is still above 10, add another point
if j > 10
  j = j-5;
  % Determine the weights of the columns (Should stay in same row)
  weights = data.bin{j}.tm(:)/sum(data.bin{j}.tm(:));
  % j determines the column of the next bin
  j = datasample(1:20,1,'weights',weights);
  % x is a random point selected from the bin (j)
  x = datasample(data.bin{j}.data,1);

  % The point that is selected is added to the cycle
  cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

  % The point is added to the distribution
cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
last = last+1;
end

% Cycle should return to Lowest Load for the last 10 seconds
j = 1; % Return to the lowest load bin

% Last 10 seconds
for c = last:1800

    % x is a random point selected from the idle bin (1)
x = datasample(data.bin{j}.data,1);

    % The point that is selected is added to the cycle
cycle{a}.cycle=cat(1,cycle{a}.cycle,x);

    % The point is added to the distribution
cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
end

x = length(cycle{a}.cycle);

%% Smoothing if ON
if strcmp(Smoothing,'On')
    % Smooth the Cycle
cycle{a}.cycle = sgolayfilt(cycle{a}.cycle,3,11);

    % Recalculate the Distribution Post Smoothing
cycle{a}.distribution = zeros(1,20);
group_l = [0:0.05:0.95, 1.01];
t = length(group_l);
for i = 1:x
    for j = 1:t-1
        if cycle{a}.cycle(i) >= group_l(j) && cycle{a}.cycle(i) < group_l(j+1)
            cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
        end
    end
end
end

%%
% Eliminate Spikes between 0 points
for i = 1:x
    if i>1 && i<1800
        if cycle{a}.cycle(i,1) > 0 &&...
cycle{a}.cycle(i+1,1) == 0 &&...
cycle{a}.cycle(i-1,1)==0
            cycle{a}.cycle(i,1) = 0;
        end
    end
end
% Eliminate Blips
for i = 1:1800
    if cycle{a}.cycle(i) < 0.02
        cycle{a}.cycle(i) = 0;
    end
end

%% Normalize the Matrix Distribution
for j = 1:20
    % The distribution percentage of the data is determined from the
    % number of points in that bin divided by total points
    % Presented as a percentage of total time
    cycle{a}.distribution(j)=cycle{a}.distribution(j)/length(cycle{a}.cycle);
    distribution_diff(j) = 0.05 * abs(cycle{a}.distribution(j) -
                           data.distribution(j));
end

%% Calculated the Statistics of the Cycle
% Average Cycle Load
cycle{a}.stats.avg.load=mean(cycle{a}.cycle);

%% Sort the Data by High or Low Load
cycle{a}.time.high = 0;
cycle{a}.time.low = 0;
cycle{a}.time.idle = 0;
high_load = 0;
low_load = 0;
x = length(cycle{a}.cycle);

% Go through all of the data and sort by Load
for i = 1:x
    % High Load will be defined as Greater than 40 %
    if cycle{a}.cycle(i) >= 0.4
        cycle{a}.time.high = cycle{a}.time.high + 1;
        high_load = high_load + cycle{a}.cycle(i);
    elseif cycle{a}.cycle(i) == 0
        cycle{a}.time.idle = cycle{a}.time.idle + 1;
    else
        cycle{a}.time.low = cycle{a}.time.low + 1;
        low_load = low_load + cycle{a}.cycle(i);
    end
end

%% Calculate the High and Low Load Averages
% Average High Load
cycle{a}.stats.avg.high_load = high_load/cycle{a}.time.high;

% Average Low Load
cycle{a}.stats.avg.low_load = low_load/cycle{a}.time.low;

%% Normalize the time values
cycle{a}.time.high = cycle{a}.time.high/x;
cycle{a}.time.low = cycle{a}.time.low/x;
cycle{a}.time.idle = cycle{a}.time.idle/x;

%% Determine the Statistic Value (SV)
cycle{a}.SV =
    0.2*abs(cycle{a}.stats.avg.load - data.stats.avg.load) + ...%
    0.2*abs(cycle{a}.stats.avg.high_load - data.stats.avg.high_load) + ...%
    0.2*abs(cycle{a}.stats.avg.low_load - data.stats.avg.low_load) + ...%
    0.2*abs(cycle{a}.time.high - data.time.high) + ...%
    0.2*abs(cycle{a}.time.idle - data.time.idle);

%% Determine the Distribution Value (DV)
cycle{a}.DV = sum(distribution_diff);

%% Calculate the Performance Value (PV)
cycle{a}.PV = ... 
    0.5*cycle{a}.SV + ... 
    0.5*cycle{a}.DV;

%%
% Comment (starting at figure) if plots are not needed for each cycle created
% Considerably slows down code

% Plot the normalized speed and normalized torque
figure;
plot(cycle{a}.cycle)
ylabel('Normalized Load')
xlabel('Time(s)')
ylim([0,1])

s = sprintf('Cycle Number %i Complete',a);
disp(s)

a = a + 1;
end
toc
run_time = toc;
clearvars -except cycle data pop_size generations crossover_rate mutation_rate Smoothing

Appendix C: MATLAB® Code for Genetic Algorithm
Over-the-Road Trucks
Trucking_GA.m
% Robert Heltzel
% West Virginia University
% Genetic Algorithm Logic for Trucking

clear all
close all
clc

%%
tic
% All outside information
% Population size
pop_size = 100;
% Define the Number of Generations
generations = 50;
% Define the rate of Crossover
crossover_rate = 0.40;
% Define The rate of Mutation
mutation_rate = 0.20;
% Smoothing: On or Off
Smoothing = 'Off';

%%
% Load the Data
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\Trucking_Data_Binned_11516.mat')

% Load the Cycles for Initial Population
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Cycles\Trucking_10000_13116')

%% Looking for Lowest PV
% Use Something to Choose Initial Population
run('Trucking_Markov_Chain_Cycle_Construction')

%%
The variable cycle is the initial population each cycle represents a chromosome
% Define Array of Zeros for Performance Values
elite_PV = zeros(generations,1);
average_PV = zeros(generations,1);

%%
% GA Needs to start here
for current_gen = 1:generations;

%%
% Segment the Cycles into Idle and Non-Idle operation
% These are the genes of the current generation
run('Trucking_Segment_Formation')

%%
% MUTATION
cycles_from_mutation = 0;
cycle_index = pop_size;
% a is the current cycle
for a = 1:pop_size
    % Determine which members of the population will experience mutation
    mute = rand;
    if mute < mutation_rate
        cycles_from_mutation = cycles_from_mutation + 1;
        cycle_index = cycle_index + 1;
        % Mutation
        gene_type = datasample(1:2,1,'weights',[idle_segment_percentage,
nonidle_segment_percentage]);
        if gene_type == 1
            run('Trucking_Idle_Segment_Selection')
            cycle(cycle_index).cycle = [cycle{a}.cycle(1:start_position,:),
gene ; cycle{a}.cycle(end_position:1800,::)];
        else
            run('Trucking_Nonidle_Segment_Selection')
            cycle(cycle_index).cycle = [cycle{a}.cycle(1:start_position,:),
gene ; cycle{a}.cycle(end_position:1800,::)];
        end
    end
end

%%
% CROSSOVER
added_cycle_1 = cycle_index;
added_cycle_2 = added_cycle_1;
cycles_from_crossover = 0;
cross_no=0;
% a is the current cycle
% b is the other selected cycle for crossover
for a = 1:pop_size
    % Determine which members of the population will experience crossover
    cross = rand;
    if cross < crossover_rate
        % Choose another cycle to crossover with
        b = datasample(1:pop_size,1);
        if cross_no == 0 % If this is the first crossover, next index is +1
            added_cycle_1 = added_cycle_1 + 1;
        else % else index is +2
            added_cycle_1 = added_cycle_1 + 2;
        end
        % A crossover event produces two new cycles
        added_cycle_2 = added_cycle_1 + 1;
        % Number of crossover events
        cross_no = cross_no + 1;
        % Crossover Selection Point
        run('Trucking_Crossover_Point')
        % Crossover
        cycle(added_cycle_1).cycle = [cycle{a}.cycle(1:split_point,:),
cycle(b).cycle(split_point+1:1800,::)];
cycle{added_cycle_2}.cycle = [cycle{b}.cycle(1:split_point,:) ;
                   cycle{a}.cycle(split_point+1:1800,:)];

     cycles_from_crossover = cycles_from_crossover + 2;
end
end

new_pop_size = pop_size + cycles_from_mutation + cycles_from_crossover;

%%% Calculate the Statistics of the new cycles only
%% a represents the current cycle
for a = pop_size+1:new_pop_size
    % Anomaly sometimes comes up one short, if so add a point
    if length(cycle{1,a}.cycle(:,1)) < 1800
        cycle{1,a}.cycle(1800,:) = 0;
    end
run('Trucking_Cycle_Statistics')
end

%%% Ordering of the Population based on the PV
run('Sort_Cycles_PV')

%%% Scaled the PV to an Evaluation Function for Selection
run('Trucking_Evaluation_Function')

% Elitist Selection (Ensure that the Lowest PV advances)
new_pop=cell(1,pop_size);
new_pop{1,1} = cycle{1,1};

% Select Members of the New Population
for i = 2:pop_size
    roulette = rand;
    if roulette < q(1)
        select = 1;
    else
        run('Probability_Selection')
    end
    new_pop{1,i}=cycle{1,select};
end

%%% Plot the average and elite PV values for the GA
% figure(1);
% subplot(2,1,1)
% plot(current_gen,average_PV(current_gen),'go')
% xlabel('Generation')
% ylabel('Average PV')
% ylim([0,1])
% hold on
% subplot(2,1,2)
% plot(current_gen,elite_PV(current_gen),'bx')
% xlabel('Generation')
% ylabel('Elite PV')
```matlab
% ylim([0,1])
% hold on

fprintf('Generation %i Complete \n',current_gen)

% End of the generation

% cycle = cell(1,pop_size);
for i = 1:pop_size
    cycle{1,i}=new_pop{1,i};
end

% Sort the Final Population of Cycles
run('Sort_Cycles_PV')

FINAL_CYCLE = cycle{1,1};

figure;
subplot(2,1,1)
plot(FINAL_CYCLE.cycle(:,1))
ylabel('Normalized Engine Speed (%)')
xlabel('Time (s)')
ylim([0,1])
subplot(2,1,2)
plot(FINAL_CYCLE.cycle(:,2))
ylabel('Normalized Engine Load (%)')
xlabel('Time (s)')
ylim([0,1])

clearvars -except cycle FINAL_CYCLE average_PV elite_PV
toc

Trucking_Segment_Formation.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trucking Cycle Segment Formation
% Based on Idle or Non-Idle

% Determine the number of cycles available
[~,pop_size]=size(cycle);

% Define the initial values
current_cycle = 1;
current_point_in_cycle = 1;
current_nonidle_segment = 1;
current_idle_segment = 1;
```
% Pre-Allocate Idle and Non-Idle Segment
nonidle_segment{1,current_nonidle_segment} = [];
idle_segment{1,current_idle_segment} = [];

% Scan through the cycles
while current_cycle<=pop_size
    % Determine the length of the current cycle
    length_of_current_cycle=length(cycle{current_cycle}.cycle(:,1));
    % Scan through the points of the cycle
    for i=current_point_in_cycle:length_of_current_cycle
        % Look for an initial point of a non-idle segment
        if i<1800
            if cycle{current_cycle}.cycle(i,1)==0 && ...
                cycle{current_cycle}.cycle(i+1,1)>0
                % Define the first point of the segment
                first_point=cycle{current_cycle}.cycle(i,:);
                % Add the point to the segment
                nonidle_segment{1,current_nonidle_segment}=...
                cat(1,nonidle_segment{1,current_nonidle_segment},first_point);
            end
        end
        % If a non-idle segment is found, pre-allocate new
        % idle_segment
        current_idle_segment=current_idle_segment+1;
        idle_segment{1,current_idle_segment}=[];
    end
    % Continue to add points to non-idle segment
    if cycle{current_cycle}.cycle(i,1)>0
        point=cycle{current_cycle}.cycle(i,:);
        nonidle_segment{1,current_nonidle_segment}=cat(1,nonidle_segment{1,current_nonidle_segment},point);
    end
    % Look for an ending point of a non-idle segment
    if i>1
        if cycle{current_cycle}.cycle(i,1)==0 && ...
            cycle{current_cycle}.cycle(i-1,1)>0 || i==1800
            % Pre-allocate a new non-idle segment
            current_nonidle_segment=current_nonidle_segment+1;
            current_point_in_cycle=i;
            nonidle_segment{1,current_nonidle_segment}=[];
        end
    end
end

% Look for a segment of idle
if i>1 && i<1800
if cycle{current_cycle}.cycle(i,1) == 0 && ...
cycle{current_cycle}.cycle(i+1,1) == 0 && ...
cycle{current_cycle}.cycle(i-1,1) == 0
% Define a point of idle
point = cycle{current_cycle}.cycle(i,:);
% Add the point to the idle segment
idle_segment{1,current_idle_segment} = ...
cat(1, idle_segment{1,current_idle_segment}, point);
end
end

% When the end of the cycle is reached, move to next cycle
if i == length_of_current_cycle
    current_cycle = current_cycle + 1;
current_point_in_cycle = 1;
end
end

clear i

% Scan through all of the segments
for i = 1 : current_nonidle_segment - 1
    [length_of_nonidle_segment,~] = size(nonidle_segment{1,i});
    % Eliminate the segments less than 2 Seconds long
    if length_of_nonidle_segment < 10
        nonidle_segment{1,i} = [];
    end
end

% Eliminate Empty Segments
nonidle_segment = nonidle_segment(~cellfun('isempty', nonidle_segment));
[~, nonidle_segment_no] = size(nonidle_segment);

% Scan through all of the idle segments
for i = 1 : current_idle_segment - 1
    [length_of_idle_segment,~] = size(idle_segment{1,i});
    % Eliminate the idle segments less than 2 Seconds long
    if length_of_idle_segment < 10
        idle_segment{1,i} = [];
    end
end

% Eliminate Empty Segments
idle_segment = idle_segment(~cellfun('isempty', idle_segment));
[~, idle_segment_no] = size(idle_segment);

% Calculate the Total Segments and the ratio of idle to non_idle segments
total_segments = nonidle_segment_no + idle_segment_no;
idle_segment_percentage = idle_segment_no / total_segments;
onidle_segment_percentage = 1 - idle_segment_percentage;
Trucking_Idle_Segment_Selection.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Selection of an Idle Segment (Gene Selection)
while 1
    gene_no = datasample(1:idle_segment_no,1);
    gene=idle_segment{1,gene_no};
    gene_length = length(gene);
    start_position = datasample(60:1740,1);
    end_position = start_position + gene_length + 1;
    if end_position < 1800 && cycle{a}.cycle(start_position,1) ==0 &&
        cycle{a}.cycle(end_position,1) ==0
        break
    end
end

Trucking_Nonidle_Segment_Selection.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Selection of a Non-Idle Segment (Gene Selection)
while 1
    gene_no = datasample(1:nonidle_segment_no,1);
    gene=nonidle_segment{1,gene_no};
    gene_length = length(gene);
    start_position = datasample(60:1740,1);
    end_position = start_position + gene_length + 1;
    if end_position < 1800 && cycle{a}.cycle(start_position,1) ==0 &&
        cycle{a}.cycle(end_position,1) ==0
        break
    end
end

Trucking_Crossover_Point.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Crossover Selection Point
%Define an Empty Array for Points
cross_points = [];
% Find all possible points between 30 seconds in and 30 seconds from end
for i=31:1770
    % Ensure both Cycles have 0 Speed
    if cycle{a}.cycle(i,1)==0 && cycle{b}.cycle(i,1)==0
        cross_points = cat(1,cross_points,i);
    end
end

% If the cross point array is empty choose a point in the first 30 seconds
if isempty(cross_points)
    % Split Point
    split_point = datasample(1:30,1);
else
    % Randomly select a split point
    split_point=datasample(cross_points,1);
end

Trucking_Cycle_Statistics.m

% Robert Heltzel
% West Virginia University
% Original: December 2014
% Updated: February 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Cycle Statistics for GA
if strcmp(Smoothing,'On')
    % Smooth the Cycle
    cycle{a}.cycle(:,1) = sgolayfilt(cycle{a}.cycle(:,1),7,13);
    cycle{a}.cycle(:,2) = sgolayfilt(cycle{a}.cycle(:,2),7,13);
end

% Eliminate Blips
for i = 1:1800
    if cycle{a}.cycle(i,1) < 0.02
        cycle{a}.cycle(i,1) = 0;
    end
    if cycle{a}.cycle(i,2) < 0.02
        cycle{a}.cycle(i,2) = 0;
    end
end

% x will be the length of the cycle
x = length(cycle{a}.cycle);

%% Calculated the Statistics of the Cycle
% Average Cycle Speed
cycle{a}.stats.avg.speed=mean(cycle{a}.cycle(:,1));
% Average Cycle Load
cycle{a}.stats.avg.load=mean(cycle{a}.cycle(:,2));
%% Sort the Data by Idle or Non-Idle
    cycle{a}.time.idle = 0;
    cycle{a}.time.non_idle = 0;
    non_idle_speed = 0;
    non_idle_load = 0;

    % Go through all of the data and sort by 0 speed (Idle)
    for i = 1:x
        if cycle{a}.cycle(i,1) == 0
            cycle{a}.time.idle = cycle{a}.time.idle + 1;
        else
            cycle{a}.time.non_idle = cycle{a}.time.non_idle + 1;
            non_idle_speed = non_idle_speed + cycle{a}.cycle(i,1);
            non_idle_load = non_idle_load + cycle{a}.cycle(i,2);
        end
    end

    %% Calculate the Non-Idle Averages
    % Average Non Idle Speed
    cycle{a}.stats.avg.non_idle_speed = non_idle_speed/cycle{a}.time.non_idle;

    % Average Non Idle Load
    cycle{a}.stats.avg.non_idle_load = non_idle_load/cycle{a}.time.non_idle;

    %% Normalize the time values
    cycle{a}.time.idle = cycle{a}.time.idle/x;
    cycle{a}.time.non_idle = cycle{a}.time.non_idle/x;

    %% Determine the Distribution
    group_s=[0:0.05:0.95 1.01];
    group_l=[0:0.05:0.95 1.01];
    s = length(group_s);
    t = length(group_l);
    cycle{a}.distribution=zeros(20,20);

    %Bin all Data
    for k=1:x-1
        %Scan the Speed Bins
        for i=1:s-1
            % Once the Speed Bin is found, move to Load Bin
            if cycle{a}.cycle(k,1)>=group_s(i) &&
                cycle{a}.cycle(k,1)<group_s(i+1)
                % Scan the Load Bins
                for j=1:t-1
                    % Once the load Grouping is found bin the data
                    if cycle{a}.cycle(k,2)>=group_l(j) &&
                        cycle{a}.cycle(k,2)<group_l(j+1)
                        % Add a point to the cycle distribution
                        cycle{a}.distribution(i,j)=cycle{a}.distribution(i,j)+1;
                    end
                end
            end
        end
    end
end
end

%% Normalize the Matrix Distribution
for i = 1:20
    for j = 1:20
        % The distribution percentage of the data is determined from the
        % number of points in that bin divided by total points
        % Presented as a percentage of total time
        cycle{a}.distribution(i,j) = cycle{a}.distribution(i,j)/length(cycle{a}.cycle(:,1));
        distribution_diff(i,j) = 0.05 * abs(cycle{a}.distribution(i,j) - data.distribution(i,j));
    end
end

%% Determine the Statistic Value (SV)
cycle{a}.SV = ...
0.2*abs(cycle{a}.stats.avg.speed - data.stats.avg.speed) + ...
0.2*abs(cycle{a}.stats.avg.load - data.stats.avg.load) + ...
0.2*abs(cycle{a}.time.idle - data.time.idle) + ...
0.2*abs(cycle{a}.stats.avg.non_idle_speed - data.stats.avg.non_idle_speed) + ...
0.2*abs(cycle{a}.stats.avg.non_idle_load - data.stats.avg.non_idle_load);

%% Determine the Distribution Value (DV)
cycle{a}.DV = sum(sum(distribution_diff));

%% Determine the PV

cycle{a}.PV = ...
0.5*cycle{a}.SV + ...
0.5*cycle{a}.DV;

% End of Statistics

Sort_Cycles_PV.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Sort Cycles Based on the PV
[~,new_pop_size]=size(cycle);
% Pre-Allocated PV
PV_values_old=[];

% Extract PV from structures and save in array
for i=1:new_pop_size
    value=[cycle{1,i}.PV, i];
PV_values_old=cat(1,PV_values_old,value);
end

[non_num,~] = find(isnan(PV_values_old));
for i=1:length(non_num)
    c = struct2cell(cycle{non_num(i)});
    c=[];
    cycle{non_num(i)}=c;
end

cycle = cycle(~cellfun('isempty',cycle));
clear PV_values
PV_values_new = [];

for i=1:length(cycle)
    value=[cycle{1,i}.PV, i];
    PV_values_new=cat(1,PV_values_new,value);
end
clear i

% Sort the Array by the PV
[~,row]=sort(PV_values_new(:,1));

%%

% Create a Population Based on the Best Cycles
new_pop_size = length(cycle);
% Transfer all cycle information into the population in order of PV
for j=1:new_pop_size
    new_cycle{1,j}=cycle{1,row(j)};
end

best_PV_cycle = new_cycle{1,1};
clear cycle

for i=1:new_pop_size
    cycle{1,i}=new_cycle{1,i};
end

Trucking_Evaluation_Function.m

% Robert Heltzel
% West Virginia University
% Original: January 2016
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scaled Evaluation Function for Roulette Wheel Selection

EF_max = cycle{1,new_pop_size}.PV;
EF_min = cycle{1,1}.PV;

coeff_a = 1/(EF_max - EF_min);
coeff_b = -EF_min/(EF_max - EF_min);
for i = 1:new_pop_size
    EF(i) = coeff_a * cycle{1,i}.PV + coeff_b;
end

% Formulate a Fitness Function for Rate of Population
TF = 0;
PV_total = 0;
prob = zeros(new_pop_size,1);
for i=1:new_pop_size
    % Formulate the Probability of Advancing
    inverse_EF(i) = 1 - EF(i);
    TF = TF + inverse_EF(i);
    PV_total = PV_total + cycle{1,i}.PV;
end

for i=1:new_pop_size
    % Determine the probability of each cycle
    prob(i) = (inverse_EF(i)) / TF;
end

% Make an Array of Average and Elite PV
elite_PV(current_gen)=cycle{1,1}.PV;
average_PV(current_gen) = PV_total / new_pop_size;

% Determine the q's for the roulette wheel
for i = 1:new_pop_size
    if i==1
        q(i) = prob(i);
    else
        q(i) = q(i-1) + prob(i);
    end
end
clear i

Probability_Selection.m

% Robert Heltzel
% West Virginia University
% Original: January 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Selection of Cycles using Roulette Wheel
j=1;
while j < length(q);
    if roulette > q(j) && roulette <= q(j+1)
        select=j+1;
        break
    else
        j=j+1;
end
Drilling and Hydraulic Fracturing

The drilling and hydraulic fracturing MATLAB® scripts used for the genetic algorithm were identical except for their naming and commenting, because of this the scripts are only presented for drilling.

Drilling_GA.m

% Robert Heltzel
% West Virginia University
% Original: October 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Genetic Algorithm Logic for Drilling
% The variable cycle is the initial population each cycle represents a chromosome
%!c
!
tic
%!c

% All outside information
% Population size
pop_size = 100;
%!c
%!c
% Define the Number of Generations
generations = 50;
%!c
%!c
% Define the rate of Crossover
crossover_rate = 0.40;
%!c
%!c
% Define the rate of Mutation
mutation_rate = 0.20;
%!c
%!c
% Smoothing: On or Off
Smoothing = 'Off';
%!c

%!c
%!c
% Load the Data
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Data\Drilling_Data_Binned_12916.mat')

%!c
%!c
% Load the Cycles for Initial Population
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Cycles\Drilling_10000_13116')

%!c
%!c
% Looking for Lowest PV
run('Drilling_Markov_Chain_Cycle_Construction')

%!c
%!c
%!c
%!c
%!c
%!c
%!c
% GA Needs to start here
for current_gen = 1:generations;

% Segment the Cycles into Low and High Load Operation
% These are the genes of the current generation
run('Drilling_Segment_Formation')

% MUTATION
cycles_from_mutation = 0;
cycle_index = pop_size;

% a is the current cycle
for a = 1:pop_size
    % Determine which members of the population will experience mutation
    mute = rand;
    if mute < mutation_rate
        cycles_from_mutation = cycles_from_mutation + 1;
cycle_index = cycle_index + 1;
    % Mutation
        gene_type = datasample(1:2,1,'weights',[low_segment_percentage,
high_segment_percentage]);
        if gene_type == 1
            run('Drilling_Low_Segment_Selection')
cycle{cycle_index}.cycle = [cycle{a}.cycle(1:start_position) ;
gene ; cycle{a}.cycle(end_position:1800)];
        else
            run('Drilling_High_Segment_Selection')
cycle{cycle_index}.cycle = [cycle{a}.cycle(1:start_position) ;
gene ; cycle{a}.cycle(end_position:1800)];
        end
    end
end

% CROSSOVER
added_cycle_1 = cycle_index;
added_cycle_2 = added_cycle_1;
cycles_from_crossover = 0;
cross_no=0;
% a is the current cycle
% b is the other selected cycle for crossover
for a = 1:pop_size
    % Determine which members of the population will experience crossover
    cross = rand;
    if cross < crossover_rate
        % Choose another cycle to crossover with
        b = datasample(1:pop_size,1);
        if cross_no == 0 % If this is the first crossover, next index is +1
            added_cycle_1 = added_cycle_1 + 1;
        else % else index is +2
            added_cycle_1 = added_cycle_1 + 2;
        end
end
% A crossover event produces two new cycles
added_cycle_2 = added_cycle_1 + 1;
% Number of crossover events
cross_no = cross_no + 1;

% Crossover Point
run('Drilling_Crossover_Point')
% Crossover
    cycle{added_cycle_1}.cycle = [cycle{a}.cycle(1:split_point,:) ;
    cycle{b}.cycle(split_point+1:1800,:)];
    cycle{added_cycle_2}.cycle = [cycle{b}.cycle(1:split_point,:) ;
    cycle{a}.cycle(split_point+1:1800,:)];

    cycles_from_crossover = cycles_from_crossover + 2;
end
end

new_pop_size = pop_size + cycles_from_mutation + cycles_from_crossover;

%%
% Calculate the Statistics of the new cycles only
% a represents the current cycle
for a = pop_size+1:new_pop_size
    % Anomaly sometimes comes up one short, if so add a point
    if length(cycle{1,a}.cycle(:,1)) < 1800
        cycle{1,a}.cycle(1800,:) = 0;
    end
run('Drilling_Cycle_Statistics')
end

%%
% Ordering of the Population based on the PV
run('Sort_Cycles_PV')

%%
% Scale PV to an Evaluation Function for Selection
run('Drilling_Evaluation_Function')

% Elitist Selection (Ensure that the Lowest PV advances)
new_pop = cell(1,pop_size);
new_pop{1,1} = cycle{1,1};

% Select Members of the New Population
for i = 2:pop_size
    roulette = rand;
    if roulette < q(1)
        select = 1;
    else
        run('Probability_Selection')
    end
    new_pop{1,i}=cycle{1,select};
end

% Plot the average and elite PV values for the GA
% figure(1);
% subplot(2,1,1)
% plot(current_gen,average_PV(current_gen),'go')
% xlabel('Generation')
% ylabel('Average PV')
% ylim([0,0.1])
% hold on
% subplot(2,1,2)
% plot(current_gen,elite_PV(current_gen),'bx')
% xlabel('Generation')
% ylabel('Elite PV')
% ylim([0,0.1])
% hold on

fprintf('Generation %i Complete
',current_gen)

% End of the generation

%%
cycle = cell(1,pop_size);
for i = 1:pop_size
    cycle{1,i}=new_pop{1,i};
end
end

% Sort the Final Population of Cycles
run('Sort_Cycles_PV')

FINAL_CYCLE = cycle{1,1};

figure;
plot(FINAL_CYCLE.cycle(:))
ylabel('Normalized Engine Load (%)')
xlabel('Time (s)')
ylim([0,1])

clearvars -except cycle FINAL_CYCLE average_PV elite_PV

toc

Drilling_Segment_Formation.m

% Robert Heltzel
% West Virginia University
% Original: September 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Drilling Cycle Segment Formation
% Based on High or Low Load

% Determine the number of cycles available
[~,pop_size]=size(cycle);
% Define the initial values
current_cycle = 1;
current_point_in_cycle = 1;
current_high_segment = 1;
current_low_segment = 1;

% Pre-Allocate High and Low Load Segments
high_segment{1,current_high_segment} = [];
low_segment{1,current_low_segment} = [];

% Scan through the cycles
while current_cycle <= pop_size
    % Determine the length of the current cycle
    length_of_current_cycle = length(cycle{current_cycle}.cycle);
    % Scan through the points of the cycle
    for i = current_point_in_cycle:length_of_current_cycle
        % Look for an initial point of a non_idle segment
        if i<1800
            if cycle{current_cycle}.cycle(i) < 0.4 &&
                cycle{current_cycle}.cycle(i+1) > 0.4
                % Define the first point of the cycle
                first_point=cycle{current_cycle}.cycle(i);
                % Add the point to the segment
                high_segment{1,current_high_segment} =...
                cat(1,high_segment{1,current_high_segment},first_point);
                % If a high segment is found, pre-allocate new low segment
                current_low_segment = current_low_segment + 1;
                low_segment{1,current_low_segment} = [];
            end
        end
        % Continue to add points to high segment
        if cycle{current_cycle}.cycle(i) > 0.4
            point = cycle{current_cycle}.cycle(i);
            high_segment{1,current_high_segment} = cat(1,high_segment{1,current_high_segment},point);
        end
        % Look for an ending point of the high segment
        if i > 1
            if cycle{current_cycle}.cycle(i) < 0.4 &&
                cycle{current_cycle}.cycle(i-1) > 0.4 || i == 1800
                % Define the last point of the high segment
                last_point = cycle{current_cycle}.cycle(i);
                high_segment{1,current_high_segment}=...
                cat(1,high_segment{1,current_high_segment},last_point);
                % Pre-Allocate a new high segment
                current_high_segment = current_high_segment + 1;
                current_point_in_cycle = i;
                high_segment{1,current_high_segment} = [];
            end
        end
end
% Look for a segment of low load
if i>1 && i<1800
  if cycle{current_cycle}.cycle(i) < 0.4 &&...
    cycle{current_cycle}.cycle(i+1) < 0.4 &&...
    cycle{current_cycle}.cycle(i-1) < 0.4
    % Define a point of low load
    point = cycle{current_cycle}.cycle(i);
    % Add the point to the low segment
    low_segment{1,current_low_segment} = ...
    cat(1,low_segment{1,current_low_segment},point);
  end
end

% When the end of the cycle is reached, move to the next cycle
if i == length_of_current_cycle
  current_cycle = current_cycle + 1;
  current_point_in_cycle = 1;
end
end
end
clear i

% Scan through all of the segments
for i = 1: current_high_segment
  [length_of_high_segment,~] = size(high_segment{1,i});
  % Eliminate the segments less than 2 seconds long
  if length_of_high_segment < 10
    high_segment{1,i} = [];
  end
end

% Eliminate Empty Segments
high_segment = high_segment(~cellfun('isempty',high_segment));
[~,high_segment_no] = size(high_segment);

% Scan through all of the idle segments
for i = 1: current_low_segment
  [length_of_low_segment,~] = size(low_segment{1,i});
  % Eliminate the low segments less than 2 seconds long
  if length_of_low_segment < 10
    low_segment{1,i} = [];
  end
end

% Eliminate Empty Segments
low_segment = low_segment(~cellfun('isempty',low_segment));
[~,low_segment_no] = size(low_segment);

% Calculate the total segments and the ratio of high to low segments
total_segments = high_segment_no + low_segment_no;
low_segment_percentage = low_segment_no/total_segments;
high_segment_percentage = 1 - low_segment_percentage;
% Selection of an Low Load Segment (Gene Selection)
while 1
    gene_no = datasample(1:low_segment_no,1);
    gene=low_segment{1,gene_no};
    gene_length = length(gene);
    start_position = datasample(60:1740,1);
    end_position = start_position + gene_length + 1;
    % Ensure that both poits are low load
    if end_position < 1800 && cycle{a}.cycle(start_position) < 0.4 &&
        cycle{a}.cycle(end_position) < 0.4
        break
    end
end

% Selection of a High Load Segment (Gene Selection)
while 1
    gene_no = datasample(1:high_segment_no,1);
    gene=high_segment{1,gene_no};
    gene_length = length(gene);
    start_position = datasample(60:1740,1);
    end_position = start_position + gene_length + 1;
    % Ensure that both points are low load
    if end_position < 1800 && cycle{a}.cycle(start_position) < 0.4 &&
        cycle{a}.cycle(end_position,1) < 0.4
        break
    end
end

% Crossover Selection Point
%Define an Empty Array for Points
cross_points = [];
% Find all possible points between 30 seconds in and 30 seconds from end
for i=31:1770
    % Ensure both Cycles have Low Load
    if cycle{a}.cycle(i,1) < 0.4 && cycle{b}.cycle(i,1) < 0.4
        cross_points = cat(1,cross_points,i);
    end
end

% If the cross point array is empty choose a point in the first 10 seconds
if isempty(cross_points)
    % Split Point
    split_point = datasample(1:10,1);
else
    % Randomly select a split point
    split_point=datasample(cross_points,1);
end

Drilling_Cycle_Statistics.m
% Robert Heltzel
% West Virginia University
% Original: September 2015
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cycle Statistics for GA

if strcmp(Smoothing,'On')
    % Smooth the Cycle
    cycle{a}.cycle = sgolayfilt(cycle{a}.cycle,7,13);
end

% Eliminate Small Blips
for i = 1:1800
    if cycle{a}.cycle(i,1) < 0.02
        cycle{a}.cycle(i,1) = 0;
    end
end

% Determine the Distribution
group_1 = [0:0.05:0.95, 1.01];
t = length(group_1);
x = length(cycle{a}.cycle);
cycle{a}.distribution = zeros(1,20);

% Bin all Data
for i = 1:x
    for j = 1:t-1
        if cycle{a}.cycle(i) >= group_1(j) && cycle{a}.cycle(i) < group_1(j+1)
            cycle{a}.distribution(j) = cycle{a}.distribution(j) + 1;
        end
    end
end
%%% Normalize the Matrix Distribution
for j = 1:t-1
% The distribution percentage of the data is determined from the
% number of points in that bin divided by total points
% Presented as a percentage of total time
cycle{a}.distribution(j) = cycle{a}.distribution(j)/x;
distribution_diff(j) = 0.05 *abs(cycle{a}.distribution(j) -
data.distribution(j));
end

%%% Calculate Average Cycle Load
cycle{a}.stats.avg.load=mean(cycle{a}.cycle);

%%% Sort the Data by High or Low Load
cycle{a}.time.high = 0;
cycle{a}.time.low = 0;
cycle{a}.time.idle = 0;
high_load = 0;
low_load = 0;
% Go through all of the data and sort by Load
for i = 1:x
% High Load will be defined as Greater than 40 %
if cycle{a}.cycle(i) >= 0.4
    cycle{a}.time.high = cycle{a}.time.high + 1;
    high_load = high_load + cycle{a}.cycle(i);
elseif cycle{a}.cycle(i) == 0
    cycle{a}.time.idle = cycle{a}.time.idle + 1;
else
    cycle{a}.time.low = cycle{a}.time.low + 1;
    low_load = low_load + cycle{a}.cycle(i);
end
end

%%% Calculate the High and Low Load Averages
% Average High Load
cycle{a}.stats.avg.high_load = high_load/cycle{a}.time.high;

% Average Low Load
cycle{a}.stats.avg.low_load = low_load/cycle{a}.time.low;

%%% Normalize the time values
% cycle{a}.time.high = cycle{a}.time.high/x;
cycle{a}.time.high = cycle{a}.time.high/x;
cycle{a}.time.low = cycle{a}.time.low/x;
cycle{a}.time.idle = cycle{a}.time.idle/x;

%%% Determine the Statistic Value (SV)
cycle{a}.SV = ...
    0.2*abs(cycle{a}.stats.avg.load - data.stats.avg.load)+...
0.2*abs(cycle{a}.stats.avg.high_load - data.stats.avg.high_load)+...
0.2*abs(cycle{a}.stats.avg.low_load - data.stats.avg.low_load)+...
0.2*abs(cycle{a}.time.high - data.time.high)+...
0.2*abs(cycle{a}.time.idle - data.time.idle);

%% Determine the Distribution Value (DV)
cycle{a}.DV = sum(distribution_diff);

%% Calculate the Performance Value (PV)
cycle{a}.PV = ...
  0.5*cycle{a}.SV + ...
  0.5*cycle{a}.DV;

% End of Statistics

Drilling_Evaluation_Function.m

% Robert Heltzel
% West Virginia University
% Original: January 2016
% Updated: January 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scaled Evaluation Function for Roulette Wheel Selection

EF_max = cycle{1,new_pop_size}.PV;
EF_min = cycle{1,1}.PV;

coeff_a = 1/(EF_max - EF_min);
coeff_b = -EF_min/(EF_max - EF_min);

for i = 1:new_pop_size
  EF(i) = coeff_a * cycle{1,i}.PV + coeff_b;
end

%% Formulate a Fitness Function for Rate of Population
TF = 0;
PV_total = 0;
prob = zeros(new_pop_size,1);
for i=1:new_pop_size
  % Formulate the Probability of Advancing
  inverse_EF(i) = 1 - EF(i);
  TF = TF + inverse_EF(i);
  PV_total = PV_total + cycle{1,i}.PV;
end

for i=1:new_pop_size
  % Determine the probability of each cycle
  prob(i) = (inverse_EF(i)) / TF;
end

%% Make an Array of Average and Elite PV
elite_PV(current_gen)=cycle{1,1}.PV;
average_PV(current_gen) = PV_total / new_pop_size;

%%%                  
% Determine the q’s for the roulette wheel          
for i = 1:new_pop_size                             
    if i==1                                         
        q(i) = prob(i);                            
    else                                           
        q(i) = q(i-1) + prob(i);                   
    end                                           
end clear i

Appendix D: Denormalization and Dynamometer Calibration  
The denormalization script as shown was for the original unsmoothed OTR truck cycle.

MATLAB® Code                            
Denormalization.m

% Robert Heltzel                          
% West Virginia University                 
% Original: March 2015                     
% Updated: April 2016                      
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Engine Denormalization
clc
close all
clear all

%%%                  
% Directory of Deliminated Excel Engine Map
directory = 'C:\Users\Robbie\Documents\Thesis\Denormalization\Engine
Maps\ISLG\';

% Filename of Deliminated Excel Engine Map
filename = 'new_islg_map';

% Load the Engine Map of the Desired Engine
% Ensure that the file defined in Lug Curve Script is Deliminated Properly
% run('Import_Lug_Curve')

%%%                  
% Choose the desired Cycle
load('C:\Users\Robbie\Documents\Thesis\Cycle
Development\3_Final_stuff\Cycles\Finals.mat')
cycle = Trucking_Cycle; % CHANGE

%%%                  
% Define the Directory and File for Output of Cycle Text File
output_directory = 'C:\Users\Robbie\Documents\Thesis\Regression\Text Files\';
output_filename = 'Trucking_New.txt'; % CHANGE

%% Run Through All Cycle Points and Denormalize
for j=1:1800
    % Denormalize the Speed
    engine.speed(j) = ceil( (cycle.cycle(j,1)*(rated_speed-idle_speed)) ...
    +idle_speed); %rpm

    % Find the Torque based on the Lug Curve
    for i = 1:length(engine_lug_speed)
        if engine_lug_speed(i) == engine.speed(j)
            current_lug_torque = engine_lug_torque(i);
        end
    end

    engine.torque(j) = cycle.cycle(j,2)*current_lug_torque;

    engine.power.hp(j) = engine.speed(j)*engine.torque(j)/5252;
    engine.power.kw(j) = (engine.power.hp(j)/745.7)*1000;

    %Values for Output File
    output(j,1) = j;
    output(j,2) = engine.speed(j);
    output(j,3) = engine.torque(j);
end

output(1,2) = idle_speed - 1;
output(1,3) = 0;

%% Plot the Denormalized Cycle
figure;
subplot(2,1,1)
plot(output(:,2))
ylabel('Engine Speed (RPM)')
xlabel('Time(s)')
ylim([idle_speed - 10,rated_speed + 10])
subplot(2,1,2)
plot(output(:,3),'g')
ylabel('Engine Torque (lb-ft)')
xlabel('Time(s)')
ylim([0,1000])

% Extract Time Information for Output File
[c = clock; year = c(1)-2000; month = c(2); day = c(3); hr = c(4); minute = c(5);]
%% Print to Text File for the Dynamometer
file = [output_directory output_filename];
fileID = fopen(file, 'w');
fprintf(fileID,'Generated: %i/%i/%i %i:%i:00 AM\r\n', month, day, year, hr, minute);
fprintf(fileID,'Map Time: %s %s %s\r\n',date, time, am_pm);
fprintf(fileID,'Barometric Pressure: %5.1f\r\n', pressure);
fprintf(fileID,'----------------------------\r\n');
for i = 1:1800
    fprintf(fileID, '%i,%i,%6.4f\r\n', output(i,:));
end
fclose(fileID);
The dynamometer side arm calibration file was taken using the data acquisition computer of the EERC test cell.
Appendix E: Denormalized Cycle “.cyc” File Formats
The first 30 seconds of the normalized cycles are shown.

<table>
<thead>
<tr>
<th>Number</th>
<th>RPM</th>
<th>Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>700</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>701</td>
<td>35.1866</td>
</tr>
<tr>
<td>3</td>
<td>701</td>
<td>30.1599</td>
</tr>
<tr>
<td>4</td>
<td>701</td>
<td>35.1866</td>
</tr>
<tr>
<td>5</td>
<td>701</td>
<td>45.2399</td>
</tr>
<tr>
<td>6</td>
<td>701</td>
<td>30.1599</td>
</tr>
<tr>
<td>7</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>8</td>
<td>701</td>
<td>75.3998</td>
</tr>
<tr>
<td>9</td>
<td>701</td>
<td>60.3198</td>
</tr>
<tr>
<td>10</td>
<td>701</td>
<td>55.2932</td>
</tr>
<tr>
<td>11</td>
<td>701</td>
<td>75.3998</td>
</tr>
<tr>
<td>12</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>13</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>14</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>15</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>16</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>17</td>
<td>701</td>
<td>55.2932</td>
</tr>
<tr>
<td>18</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>19</td>
<td>701</td>
<td>75.3998</td>
</tr>
<tr>
<td>20</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>21</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>22</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>23</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>24</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>25</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>26</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>27</td>
<td>701</td>
<td>70.3731</td>
</tr>
<tr>
<td>28</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>29</td>
<td>701</td>
<td>65.3465</td>
</tr>
<tr>
<td>30</td>
<td>701</td>
<td>65.3465</td>
</tr>
</tbody>
</table>

Barometric Pressure: 97484.5

Generated: 4/7/16 10:20:00 AM
Map Time: 3/18/16 04:12:53 PM

149
Appendix F: Smoothing
The smooth and stat MATLAB® scripts for drilling and hydraulic fracturing were identical except for the naming, therefore only drilling is shown. A 20 second sample of the original and final smoothed cycles is shown for comparison.

MATLAB® Code

Smooth_and_Stat.m

% Robert Heltzel
% West Virginia University
% Original: March 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Smooth the Selected Cycle

clear all
close all
clc

% Load the Final Cycles
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\3_Final_stuff\Cycles\Finals.mat')

% Define Desired Cycle
FINAL_CYCLE = Trucking_Cycle;

%% Define Smoothing Characteristics
method = 'sgolay';
% Options:
% moving - moving average
% lowess - local regression w/ 1st degree model
% loess - local regression w/ 2nd degree model
% sgolay - Savitsky-Golay filter

span = 13; % Span is percent of points or number of points (sgolay)
degree = 7; % Degree of the polynomial for sgolay (must be odd)

if isequal(FINAL_CYCLE,Drilling_Cycle)
    % Apply Smoothing
    Smooth_Drilling_Cycle.cycle(:,1) = FINAL_CYCLE.cycle(:,1);
    Smooth_Drilling_Cycle.cycle(:,2) = smooth(FINAL_CYCLE.cycle(:,2),span,method,degree);
    % Smooth and Stat
    run('Drilling_Smooth_Stat')

elseif isequal(FINAL_CYCLE,Fracking_Cycle)
    % Apply Smoothing
    Smooth_Fracking_Cycle.cycle(:,1) = FINAL_CYCLE.cycle(:,1);
    Smooth_Fracking_Cycle.cycle(:,2) = smooth(FINAL_CYCLE.cycle(:,2),span,method,degree);
    % Smooth and Stat
    run('Fracking_Smooth_Stat')
% Define Smoothing of Cycle
clearvars -except Fracking_Cycle Smooth_Fracking_Cycle method span degree
elseif isequal(FINAL_CYCLE,Trucking_Cycle)
% Apply Smoothing
Smooth_Trucking_Cycle.cycle(:,1) =
smooth(FINAL_CYCLE.cycle(:,1),span,method,degree);
Smooth_Trucking_Cycle.cycle(:,2) =
smooth(FINAL_CYCLE.cycle(:,2),span,method,degree);
% Smooth and Stat
run('Trucking_Smooth_Stat')
% Define Smoothing of Cycle
clearvars -except Trucking_Cycle Smooth_Trucking_Cycle method span degree
else
display('Please Choose a Valid Cycle Option')end

Trucking_Smooth_and_Stat.m

% Robert Heltzel
% West Virginia University
% Original: February 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Smooth and Calculate Statistics (Trucking)

% Load the Data for Comparison
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Trucking\Data\Trucking_Data_Binned_11516.mat')

% Eliminate spikes above 0 between 0 points
for i = 1:1800
    if i>1 && i<1800
        if Smooth_Trucking_Cycle.cycle(i,1) > 0 &&...
            Smooth_Trucking_Cycle.cycle(i+1,1) == 0 &&...
            Smooth_Trucking_Cycle.cycle(i-1,1)==0
                Smooth_Trucking_Cycle.cycle(i,1) = 0;
        end
    end
end

% Eliminate Blips
for i = 1:1800
    if Smooth_Trucking_Cycle.cycle(i,1) < 0.02
        Smooth_Trucking_Cycle.cycle(i,1) = 0;
    elseif Smooth_Trucking_Cycle.cycle(i,1) > 1.0
        Smooth_Trucking_Cycle.cycle(i,1) = 1.0;
    end
    if Smooth_Trucking_Cycle.cycle(i,2) < 0.02
        Smooth_Trucking_Cycle.cycle(i,2) = 0;
    elseif Smooth_Trucking_Cycle.cycle(i,2) > 1.0
        Smooth_Trucking_Cycle.cycle(i,2) = 1.0;
    end
end
% x will be the length of the cycle
x = length(Smooth_Trucking_Cycle.cycle);

%% Calculated the Statistics of the Cycle
% Average Cycle Speed
Smooth_Trucking_Cycle.stats.avg.speed=mean(Smooth_Trucking_Cycle.cycle(:,1));

% Average Cycle Load
Smooth_Trucking_Cycle.stats.avg.load=mean(Smooth_Trucking_Cycle.cycle(:,2));

%% Sort the Data by Idle or Non-Idle
Smooth_Trucking_Cycle.time.idle = 0;
Smooth_Trucking_Cycle.time.non_idle = 0;
non_idle_speed = 0;
non_idle_load = 0;

% Go through all of the data and sort by 0 speed (Idle)
for i = 1:x
    if Smooth_Trucking_Cycle.cycle(i,1) == 0
        Smooth_Trucking_Cycle.time.idle = Smooth_Trucking_Cycle.time.idle + 1;
    else
        Smooth_Trucking_Cycle.time.non_idle = Smooth_Trucking_Cycle.time.non_idle + 1;
        non_idle_speed = non_idle_speed + Smooth_Trucking_Cycle.cycle(i,1);
        non_idle_load = non_idle_load + Smooth_Trucking_Cycle.cycle(i,2);
    end
end

%% Calculate the Non-Idle Averages
% Average Non Idle Speed
Smooth_Trucking_Cycle.stats.avg.non_idle_speed = non_idle_speed/Smooth_Trucking_Cycle.time.non_idle;

% Average Non Idle Load
Smooth_Trucking_Cycle.stats.avg.non_idle_load = non_idle_load/Smooth_Trucking_Cycle.time.non_idle;

%% Normalize the time values
Smooth_Trucking_Cycle.time.idle = Smooth_Trucking_Cycle.time.idle/x;

Smooth_Trucking_Cycle.time.non_idle = Smooth_Trucking_Cycle.time.non_idle/x;

%% Redefine Distribution of Cycle
% Define the Values that will Govern the Bins
% Each Bin will be a group of 5% of speed and load
group_s=[0:0.05:0.95 1.01];
group_l=[0:0.05:0.95 1.01];

% Define the lengths of the sorting bin values
s = length(group_s);
t = length(group_l);
% Predefine Distribution of Cycle
Smooth_Trucking_Cycle.distribution=zeros(20,20);

% Look at each point in the cycle
for k=1:x-1
    % Scan the Speed Bins
    for i=1:s-1
        % Once the Speed Bin is found, look at Load Bins
        if Smooth_Trucking_Cycle.cycle(k,1)>=group_s(i) && Smooth_Trucking_Cycle.cycle(k,1)<group_s(i+1)
            % Scan the Load Bins
            for j=1:t-1
                % Once the Load Bin is found, Bin the Point
                if Smooth_Trucking_Cycle.cycle(k,2)>=group_l(j) && Smooth_Trucking_Cycle.cycle(k,2)<group_l(j+1)
                    % Add a point to the cycle distribution
                    Smooth_Trucking_Cycle.distribution(i,j)=Smooth_Trucking_Cycle.distribution(i,j)+1;
                end
            end
        end
    end
end

% Normalize the Matrix Distribution
for i = 1:20
    for j = 1:20
        % The distribution percentage of the data is determined from the number of points in that bin divided by total points
        % Presented as a percentage of total time
        Smooth_Trucking_Cycle.distribution(i,j)=Smooth_Trucking_Cycle.distribution(i,j)/length(Smooth_Trucking_Cycle.cycle(:,1));
        distribution_diff(i,j) = 0.05 * abs(Smooth_Trucking_Cycle.distribution(i,j) - data.distribution(i,j));
    end
end

% Determine the Statistic Value (SV)
Smooth_Trucking_Cycle.SV =...
    0.2*abs(Smooth_Trucking_Cycle.stats.avg.speed - data.stats.avg.speed) + ...
    0.2*abs(Smooth_Trucking_Cycle.stats.avg.load - data.stats.avg.load) + ...
    0.2*abs(Smooth_Trucking_Cycle.stats.time.idle - data.stats.time.idle) + ...
    0.2*abs(Smooth_Trucking_Cycle.stats.avg.non_idle_speed - data.stats.avg.non_idle_speed) + ...
    0.2*abs(Smooth_Trucking_Cycle.stats.avg.non_idle_load - data.stats.avg.non_idle_load);

% Determine the Distribution Value (DV)
Smooth_Trucking_Cycle.DV = sum(sum(distribution_diff));

% Calculate the PV
Smooth_Trucking_Cycle.PV = ...
    0.5*Smooth_Trucking_Cycle.SV + ...
Drilling_Smooth_and_Stat.m

% Drilling Smooth and Stat.m

% Robert Heltzel
% West Virginia University
% Original: February 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Smooth and Calculate Statistics (Drilling)

% Load the Data for Comparison
load('C:\Users\Robbie\Documents\Thesis\Cycle Development\Drilling\Data\Drilling_Data_Binned_12916.mat')

% Eliminate spikes above 0 between 0 points
for i = 1:1800
    if i>1 && i<1800
        if Smooth_Drilling_Cycle.cycle(i,2) > 0 &&
           Smooth_Drilling_Cycle.cycle(i+1,2) == 0 &&
           Smooth_Drilling_Cycle.cycle(i-1,2)==0
            Smooth_Drilling_Cycle.cycle(i,2) = 0;
        end
    end
end

% Eliminate Blips
for i = 1:1800
    if Smooth_Drilling_Cycle.cycle(i,2) < 0.02
        Smooth_Drilling_Cycle.cycle(i,2) = 0;
    elseif Smooth_Drilling_Cycle.cycle(i,2) > 1.0
        Smooth_Drilling_Cycle.cycle(i,2) = 1.0;
    end
end

%% Distribution of the Cycle
group_1 = [0:0.05:0.95, 1.01];
t = length(group_1);
x = length(Smooth_Drilling_Cycle.cycle);
Smooth_Drilling_Cycle.distribution = zeros(1,20);
for i = 1:x
    for j = 1:t-1
        if Smooth_Drilling_Cycle.cycle(i,2) >= group_1(j) &&
           Smooth_Drilling_Cycle.cycle(i,2) < group_1(j+1)
            Smooth_Drilling_Cycle.distribution(j) =
            Smooth_Drilling_Cycle.distribution(j) + 1;
        end
    end
end
for j = 1:t-1
  % Normalize the Distribution of the cycle
  Smooth_Drilling_Cycle.distribution(j) = Smooth_Drilling_Cycle.distribution(j)/x;
  % Calculate the distribution differential compared to the normalized distribution of all the data
  distribution_diff(j) = 0.05* abs(Smooth_Drilling_Cycle.distribution(j) - data.distribution(j));
end

%% Calculate Average Cycle Load
Smooth_Drilling_Cycle.stats.avg.load=mean(Smooth_Drilling_Cycle.cycle(:,2));

%% Sort the Data by High or Low Load
Smooth_Drilling_Cycle.time.high = 0;
Smooth_Drilling_Cycle.time.low = 0;
Smooth_Drilling_Cycle.time.idle = 0;
high_load = 0;
low_load = 0;

% Go through all of the data and sort by Load
for i = 1:x
  % High Load will be defined as Greater than 40 %
  if Smooth_Drilling_Cycle.cycle(i,2) >= 0.4
    Smooth_Drilling_Cycle.time.high = Smooth_Drilling_Cycle.time.high + 1;
    high_load = high_load + Smooth_Drilling_Cycle.cycle(i,2);
  elseif Smooth_Drilling_Cycle.cycle(i,2) == 0
    Smooth_Drilling_Cycle.time.idle = Smooth_Drilling_Cycle.time.idle + 1;
  else
    Smooth_Drilling_Cycle.time.low = Smooth_Drilling_Cycle.time.low + 1;
    low_load = low_load + Smooth_Drilling_Cycle.cycle(i,2);
  end
end

%% Calculate the High and Low Load Averages
% Average High Load
Smooth_Drilling_Cycle.stats.avg.high_load = high_load/Smooth_Drilling_Cycle.time.high;

% Average Low Load
Smooth_Drilling_Cycle.stats.avg.low_load = low_load/Smooth_Drilling_Cycle.time.low;

%% Normalize the time values
Smooth_Drilling_Cycle.time.high = Smooth_Drilling_Cycle.time.high/x;
Smooth_Drilling_Cycle.time.low = Smooth_Drilling_Cycle.time.low/x;
Smooth_Drilling_Cycle.time.idle = Smooth_Drilling_Cycle.time.idle/x;

%% Determine the Statistic Value (SV)
Smooth_Drilling_Cycle.SV =...
\begin{verbatim}
0.2*abs(Smooth_Drilling_Cycle.stats.avg.load - data.stats.avg.load) +
0.2*abs(Smooth_Drilling_Cycle.stats.avg.high_load -
data.stats.avg.high_load) +
0.2*abs(Smooth_Drilling_Cycle.stats.avg.low_load -
data.stats.avg.low_load) +
0.2*abs(Smooth_Drilling_Cycle.time.high - data.time.high) +
0.2*abs(Smooth_Drilling_Cycle.time.idle - data.time.idle);

\% Determine the Distribution Value (DV)
Smooth_Drilling_Cycle.DV = sum(distribution_diff);

\% Calculate the Performance Value (PV)
Smooth_Drilling_Cycle.PV = ...
    0.5*Smooth_Drilling_Cycle.SV + ...
    0.5*Smooth_Drilling_Cycle.DV;
\end{verbatim}

% End of Statistics

20 Second Sample comparison of Original and Smooth OTR Cycle

<table>
<thead>
<tr>
<th>Point (#)</th>
<th>Speed</th>
<th>Load</th>
<th>Difference</th>
<th>Original</th>
<th>Smooth</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Smooth</td>
<td>Difference</td>
<td>Original</td>
<td>Smooth</td>
<td>Difference</td>
</tr>
<tr>
<td>302</td>
<td>0.09</td>
<td>0.06</td>
<td>33%</td>
<td>0.18</td>
<td>0.18</td>
<td>1%</td>
</tr>
<tr>
<td>303</td>
<td>0.10</td>
<td>0.08</td>
<td>13%</td>
<td>0.18</td>
<td>0.17</td>
<td>5%</td>
</tr>
<tr>
<td>304</td>
<td>0.05</td>
<td>0.08</td>
<td>45%</td>
<td>0.16</td>
<td>0.15</td>
<td>6%</td>
</tr>
<tr>
<td>305</td>
<td>0.06</td>
<td>0.06</td>
<td>3%</td>
<td>0.10</td>
<td>0.13</td>
<td>30%</td>
</tr>
<tr>
<td>306</td>
<td>0.08</td>
<td>0.06</td>
<td>28%</td>
<td>0.13</td>
<td>0.11</td>
<td>14%</td>
</tr>
<tr>
<td>307</td>
<td>0.06</td>
<td>0.08</td>
<td>41%</td>
<td>0.10</td>
<td>0.10</td>
<td>1%</td>
</tr>
<tr>
<td>308</td>
<td>0.10</td>
<td>0.09</td>
<td>9%</td>
<td>0.10</td>
<td>0.09</td>
<td>9%</td>
</tr>
<tr>
<td>309</td>
<td>0.10</td>
<td>0.09</td>
<td>8%</td>
<td>0.06</td>
<td>0.08</td>
<td>31%</td>
</tr>
<tr>
<td>310</td>
<td>0.08</td>
<td>0.08</td>
<td>3%</td>
<td>0.08</td>
<td>0.07</td>
<td>14%</td>
</tr>
<tr>
<td>311</td>
<td>0.06</td>
<td>0.07</td>
<td>21%</td>
<td>0.08</td>
<td>0.08</td>
<td>1%</td>
</tr>
<tr>
<td>312</td>
<td>0.07</td>
<td>0.06</td>
<td>17%</td>
<td>0.08</td>
<td>0.09</td>
<td>8%</td>
</tr>
<tr>
<td>313</td>
<td>0.06</td>
<td>0.06</td>
<td>7%</td>
<td>0.09</td>
<td>0.08</td>
<td>9%</td>
</tr>
<tr>
<td>314</td>
<td>0.07</td>
<td>0.07</td>
<td>1%</td>
<td>0.08</td>
<td>0.07</td>
<td>7%</td>
</tr>
<tr>
<td>315</td>
<td>0.06</td>
<td>0.06</td>
<td>3%</td>
<td>0.07</td>
<td>0.09</td>
<td>25%</td>
</tr>
<tr>
<td>316</td>
<td>0.06</td>
<td>0.06</td>
<td>7%</td>
<td>0.10</td>
<td>0.10</td>
<td>1%</td>
</tr>
<tr>
<td>317</td>
<td>0.05</td>
<td>0.06</td>
<td>10%</td>
<td>0.13</td>
<td>0.11</td>
<td>17%</td>
</tr>
<tr>
<td>318</td>
<td>0.06</td>
<td>0.06</td>
<td>1%</td>
<td>0.10</td>
<td>0.11</td>
<td>8%</td>
</tr>
<tr>
<td>319</td>
<td>0.06</td>
<td>0.06</td>
<td>8%</td>
<td>0.08</td>
<td>0.09</td>
<td>14%</td>
</tr>
<tr>
<td>320</td>
<td>0.06</td>
<td>0.06</td>
<td>5%</td>
<td>0.09</td>
<td>0.08</td>
<td>11%</td>
</tr>
<tr>
<td>321</td>
<td>0.06</td>
<td>0.06</td>
<td>5%</td>
<td>0.09</td>
<td>0.09</td>
<td>2%</td>
</tr>
<tr>
<td>322</td>
<td>0.06</td>
<td>0.06</td>
<td>6%</td>
<td>0.09</td>
<td>0.10</td>
<td>14%</td>
</tr>
</tbody>
</table>
Appendix G: All Cycle Points
The entire normalized cycles are shown.
<table>
<thead>
<tr>
<th>OTR Trucks</th>
<th>Drilling</th>
<th>Hydraulic Fracturing</th>
<th>Smooth OTR Trucks</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>speed</td>
<td>torque</td>
<td>number</td>
</tr>
<tr>
<td>1</td>
<td>0.02</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.07</td>
<td>2</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.06</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.07</td>
<td>4</td>
<td>0.3</td>
</tr>
<tr>
<td>5</td>
<td>0.09</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>6</td>
<td>0.06</td>
<td>6</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>0.14</td>
<td>7</td>
<td>0.6</td>
</tr>
<tr>
<td>8</td>
<td>0.15</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>9</td>
<td>0.12</td>
<td>9</td>
<td>0.8</td>
</tr>
<tr>
<td>10</td>
<td>0.11</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>11</td>
<td>0.15</td>
<td>11</td>
<td>1.0</td>
</tr>
<tr>
<td>12</td>
<td>0.14</td>
<td>12</td>
<td>1.0</td>
</tr>
<tr>
<td>13</td>
<td>0.14</td>
<td>13</td>
<td>1.0</td>
</tr>
<tr>
<td>14</td>
<td>0.13</td>
<td>14</td>
<td>1.0</td>
</tr>
<tr>
<td>15</td>
<td>0.14</td>
<td>15</td>
<td>1.0</td>
</tr>
<tr>
<td>16</td>
<td>0.14</td>
<td>16</td>
<td>1.0</td>
</tr>
<tr>
<td>17</td>
<td>0.11</td>
<td>17</td>
<td>1.0</td>
</tr>
<tr>
<td>18</td>
<td>0.13</td>
<td>18</td>
<td>1.0</td>
</tr>
<tr>
<td>19</td>
<td>0.15</td>
<td>19</td>
<td>1.0</td>
</tr>
<tr>
<td>20</td>
<td>0.14</td>
<td>20</td>
<td>1.0</td>
</tr>
<tr>
<td>21</td>
<td>0.14</td>
<td>21</td>
<td>1.0</td>
</tr>
<tr>
<td>22</td>
<td>0.13</td>
<td>22</td>
<td>1.0</td>
</tr>
<tr>
<td>23</td>
<td>0.14</td>
<td>23</td>
<td>1.0</td>
</tr>
<tr>
<td>24</td>
<td>0.15</td>
<td>24</td>
<td>1.0</td>
</tr>
<tr>
<td>25</td>
<td>0.13</td>
<td>25</td>
<td>1.0</td>
</tr>
<tr>
<td>26</td>
<td>0.14</td>
<td>26</td>
<td>1.0</td>
</tr>
<tr>
<td>27</td>
<td>0.15</td>
<td>27</td>
<td>1.0</td>
</tr>
<tr>
<td>28</td>
<td>0.13</td>
<td>28</td>
<td>1.0</td>
</tr>
<tr>
<td>29</td>
<td>0.13</td>
<td>29</td>
<td>1.0</td>
</tr>
<tr>
<td>30</td>
<td>0.13</td>
<td>30</td>
<td>1.0</td>
</tr>
<tr>
<td>31</td>
<td>0.13</td>
<td>31</td>
<td>1.0</td>
</tr>
<tr>
<td>32</td>
<td>0.13</td>
<td>32</td>
<td>1.0</td>
</tr>
<tr>
<td>33</td>
<td>0.12</td>
<td>33</td>
<td>1.0</td>
</tr>
<tr>
<td>34</td>
<td>0.15</td>
<td>34</td>
<td>1.0</td>
</tr>
<tr>
<td>35</td>
<td>0.13</td>
<td>35</td>
<td>1.0</td>
</tr>
<tr>
<td>36</td>
<td>0.14</td>
<td>36</td>
<td>1.0</td>
</tr>
<tr>
<td>37</td>
<td>0.12</td>
<td>37</td>
<td>1.0</td>
</tr>
<tr>
<td>38</td>
<td>0.13</td>
<td>38</td>
<td>1.0</td>
</tr>
<tr>
<td>39</td>
<td>0.12</td>
<td>39</td>
<td>1.0</td>
</tr>
<tr>
<td>40</td>
<td>0.16</td>
<td>40</td>
<td>1.0</td>
</tr>
<tr>
<td>41</td>
<td>0.16</td>
<td>41</td>
<td>1.0</td>
</tr>
<tr>
<td>42</td>
<td>0.16</td>
<td>42</td>
<td>1.0</td>
</tr>
<tr>
<td>43</td>
<td>0.19</td>
<td>43</td>
<td>1.0</td>
</tr>
<tr>
<td>44</td>
<td>0.19</td>
<td>44</td>
<td>1.0</td>
</tr>
<tr>
<td>45</td>
<td>0.16</td>
<td>45</td>
<td>1.0</td>
</tr>
<tr>
<td>46</td>
<td>0.16</td>
<td>46</td>
<td>1.0</td>
</tr>
<tr>
<td>47</td>
<td>0.16</td>
<td>47</td>
<td>1.0</td>
</tr>
<tr>
<td>48</td>
<td>0.16</td>
<td>48</td>
<td>1.0</td>
</tr>
<tr>
<td>49</td>
<td>0.19</td>
<td>49</td>
<td>1.1</td>
</tr>
<tr>
<td>50</td>
<td>0.18</td>
<td>50</td>
<td>1.3</td>
</tr>
<tr>
<td>51</td>
<td>0.67</td>
<td>51</td>
<td>1.4</td>
</tr>
<tr>
<td>52</td>
<td>0.09</td>
<td>52</td>
<td>1.2</td>
</tr>
<tr>
<td>53</td>
<td>0.07</td>
<td>53</td>
<td>1.3</td>
</tr>
<tr>
<td>54</td>
<td>0.08</td>
<td>54</td>
<td>1.5</td>
</tr>
<tr>
<td>55</td>
<td>0.09</td>
<td>55</td>
<td>1.5</td>
</tr>
<tr>
<td>56</td>
<td>0.05</td>
<td>56</td>
<td>1.2</td>
</tr>
<tr>
<td>57</td>
<td>0.06</td>
<td>57</td>
<td>1.4</td>
</tr>
<tr>
<td>58</td>
<td>0.09</td>
<td>58</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Prob.</td>
<td>Year</td>
<td>Prob.</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>1739</td>
<td>0.21</td>
<td>1740</td>
<td>0.23</td>
</tr>
<tr>
<td>1741</td>
<td>0.23</td>
<td>1742</td>
<td>0.23</td>
</tr>
<tr>
<td>1743</td>
<td>0.20</td>
<td>1744</td>
<td>0.17</td>
</tr>
<tr>
<td>1745</td>
<td>0.15</td>
<td>1746</td>
<td>0.18</td>
</tr>
<tr>
<td>1747</td>
<td>0.17</td>
<td>1748</td>
<td>0.17</td>
</tr>
<tr>
<td>1749</td>
<td>0.17</td>
<td>1750</td>
<td>0.17</td>
</tr>
<tr>
<td>1751</td>
<td>0.17</td>
<td>1752</td>
<td>0.15</td>
</tr>
<tr>
<td>1753</td>
<td>0.17</td>
<td>1754</td>
<td>0.17</td>
</tr>
<tr>
<td>1755</td>
<td>0.17</td>
<td>1756</td>
<td>0.17</td>
</tr>
<tr>
<td>1757</td>
<td>0.18</td>
<td>1758</td>
<td>0.17</td>
</tr>
<tr>
<td>1759</td>
<td>0.18</td>
<td>1760</td>
<td>0.18</td>
</tr>
<tr>
<td>1761</td>
<td>0.27</td>
<td>1762</td>
<td>0.26</td>
</tr>
<tr>
<td>1763</td>
<td>0.27</td>
<td>1764</td>
<td>0.33</td>
</tr>
<tr>
<td>1765</td>
<td>0.22</td>
<td>1766</td>
<td>0.23</td>
</tr>
<tr>
<td>1767</td>
<td>0.21</td>
<td>1768</td>
<td>0.29</td>
</tr>
<tr>
<td>1769</td>
<td>0.28</td>
<td>1770</td>
<td>0.29</td>
</tr>
<tr>
<td>1771</td>
<td>0.11</td>
<td>1772</td>
<td>0.00</td>
</tr>
<tr>
<td>1773</td>
<td>0.00</td>
<td>1774</td>
<td>0.00</td>
</tr>
<tr>
<td>1775</td>
<td>0.00</td>
<td>1776</td>
<td>0.00</td>
</tr>
<tr>
<td>1777</td>
<td>0.00</td>
<td>1778</td>
<td>0.00</td>
</tr>
<tr>
<td>1779</td>
<td>0.00</td>
<td>1780</td>
<td>0.00</td>
</tr>
<tr>
<td>1781</td>
<td>0.00</td>
<td>1782</td>
<td>0.00</td>
</tr>
<tr>
<td>1783</td>
<td>0.00</td>
<td>1784</td>
<td>0.00</td>
</tr>
<tr>
<td>1785</td>
<td>0.00</td>
<td>1786</td>
<td>0.00</td>
</tr>
<tr>
<td>1787</td>
<td>0.00</td>
<td>1788</td>
<td>0.00</td>
</tr>
<tr>
<td>1789</td>
<td>0.00</td>
<td>1790</td>
<td>0.00</td>
</tr>
<tr>
<td>1791</td>
<td>0.00</td>
<td>1792</td>
<td>0.00</td>
</tr>
<tr>
<td>1793</td>
<td>0.00</td>
<td>1794</td>
<td>0.00</td>
</tr>
<tr>
<td>1795</td>
<td>0.00</td>
<td>1796</td>
<td>0.00</td>
</tr>
<tr>
<td>1797</td>
<td>0.00</td>
<td>1798</td>
<td>0.00</td>
</tr>
<tr>
<td>1799</td>
<td>0.00</td>
<td>1800</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Appendix H: Real World Activity Segments Similar to Cycles
The segments shown are taken from raw data collected from in use engines. These segments are similar to created cycles. It should be noted that the drilling and hydraulic fracturing segments are of longer length than cycles. The nature of the activity is such that a cycle must be scaled to capture all forms of activity.

Over the Road Trucks